# IBM

**International Technical Support Centers**

**OS/2 Version 2.0**

**Volume 2: DOS and Windows Environment**

# OS/2 Version 2.0
# Volume 2: DOS and Windows Environment

```
┌─── Take Note! ──────────────────────────────────────────────────────────────────┐
│                                                                                  │
│ Before using this information and the product it supports, be sure to read the general information under │
│ "Special Notices" on page xvii.                                                  │
│                                                                                  │
└──────────────────────────────────────────────────────────────────────────────────┘
```

**First Edition (April 1992)**

This edition applies to OS/2 Version 2.0.

Order publications through your IBM representative or the IBM branch office serving your locality. Publications are not stocked at the address given below.

A form for reader's comments appears at the back of this publication. If the form has been removed, address your comments to:

IBM Corporation, International Technical Support Center
Dept. 91J, Building 235-2 Internal Zip 4423
901 NW 51st Street
Boca Raton, Florida 33432 USA

When you send information to IBM, you grant IBM a non-exclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

# Abstract

This document describes both the Multiple Virtual DOS Machines (MVDM) component of OS/2 Version 2.0 and the implementation of Microsoft Windows application support under OS/2 Version 2.0. It forms Volume 2 of a five volume set; the other volumes are:

- *OS/2 Version 2.0 - Volume 1: Control Program* GG24-3730

- *OS/2 Version 2.0 - Volume 3: Presentation Manager and Workplace Shell* GG24-3732

- *OS/2 Version 2.0 - Volume 4: Application Development* GG24-3774

- *OS/2 Version 2.0 - Volume 5: Print Subsystem* GG24-3775

The entire set may be ordered as *OS/2 Version 2.0 Technical Compendium*, GBOF-2254.

This publication is intended for IBM customers, IBM system engineers, and IBM authorized dealers, and other individuals who require a knowledge of the features, functions, and implementation of DOS and Microsoft Windows application support under OS/2 Version 2.0.

This document assumes that the reader is generally familiar with the DOS operating system and Microsoft Windows, and with the function provided in previous releases of OS/2.

PS                                                                    (290 pages)

# Acknowledgements

The project leader and editor for this project was:

Hans J. Goetz
International Technical Support Center, Boca Raton

The authors of this document are:

Robert Beck
IBM United Kingdom

Herman Benders
IBM Netherlands

Bo Falkenberg
IBM Denmark

Dorle Hecker
IBM Germany

Patrick Lee
IBM Australia

Robert Penrose
IBM Canada

Dwight Ronquest
ISM South Africa

Laurie Rose
IBM UK

Karl-Peter Schweder
IBM Germany

Tim Sennitt
IBM UK

Neil Stokes
IBM Australia

Bernd Westphal
IBM Germany

This publication is the result of a series of residencies conducted at the International Technical Support Center, Boca Raton.

Thanks to the following people for the invaluable advice and guidance provided in the production of this document:

Terri Beck
IBM Programming Center, Boca Raton.

# Contents

# Figures

# Tables

# Special Notices

This publication is intended to help customers and system engineers to under-
stand and utilize the new features in Version 2.0 of OS/2. The information in this
publication is not intended as the specification of any programming interfaces
that are provided by OS/2 Version 2.0. See the PUBLICATIONS section of the
IBM Programming Announcement for OS/2 Version 2.0 for more information
about what publications are considered to be product documentation.

References in this publication to IBM products, programs or services do not
imply that IBM intends to make these available in all countries in which IBM
operates. Any reference to an IBM product, program, or service is not intended
to state or imply that only IBM's product, program, or service may be used. Any
functionally equivalent program that does not infringe any of IBM's intellectual
property rights may be used instead of the IBM product, program or service.

Information in this book was developed in conjunction with use of the equipment
specified, and is limited in application to those specific hardware and software
products and levels.

IBM may have patents or pending patent applications covering subject matter in
this document. The furnishing of this document does not give you any license to
these patents. You can send license inquiries, in writing, to the IBM Director of
Commercial Relations, IBM Corporation, Purchase, NY 10577.

The information contained in this document has not been submitted to any
formal IBM test and is distributed AS IS. The information about non-IBM
("vendor") products in this manual has been supplied by the vendor and IBM
assumes no responsibility for its accuracy completeness. The use of this infor-
mation or the implementation of any of these techniques is a customer responsi-
bility and depends on the customer's ability to evaluate and integrate them into
the customer's operational environment. While each item may have been
reviewed by IBM for accuracy in a specific situation, there is no guarantee that
the same or similar results will be obtained elsewhere. Customers attempting to
adapt these techniques to their own environments do so at their own risk.

The following document contains examples of data and reports used in daily
business operations. To illustrate them as completely as possible, the examples
contain the names of individuals, companies, brands, and products. All of these
names are fictitious and any similarity to the names and addresses used by an
actual business enterprise is entirely coincidental.

**xvii**

The following terms, which are denoted by an asterisk (*) in this publication, are trademarks of the International Business Machines Corporation in the United States and/or other countries:

AIX
C/2
IBM
Micro Channel
Operating System/2
OS/2
Personal System/2
Presentation Manager
SAA
Systems Application Architecture
Workplace Shell

The following terms, which are denoted by a double asterisk (* *) in this publication, are trademarks of other companies.

Adobe is a trademark of Adobe Systems Inc.
AST is a trademark of AST.
CP/M is a trademark of Digital Research Inc.
CompuServe is a trademark CompuServe Inc.
DR-DOS is a trademark of Digital Research Inc.
Excel is a trademark of Microsoft Corporation.
Helvetica is a trademark of Linotype Company.
HP and Hewlett-Packard are trademarks of Hewlett-Packard Corporation.
Intel is a trademark of Intel Corporation.
LaserJet is a trademark of Hewlett-Packard Corporation.
Lotus and Lotus 1-2-3 are trademarks of Lotus Development Corporation.
Microsoft is a trademark of Microsoft Corporation.
MS-DOS is a registered trademark of Microsoft Corporation.
SPF/2 is a trademark of Command Technology Corporation.
Times New Roman is a trademark of Monotype Corporation Limited.
Windows is a trademark of Microsoft Corporation.
WordPerfect is a trademark of Wordperfect Corporation.
386, 486, SX are trademarks of Intel Corporation.
80286, 80386 and 80486 are trademarks of Intel Corporation.

# Preface

This document provides an understanding of the architecture and function of the Multiple Virtual DOS Machines (MVDM) component of OS/2 Version 2.0, which allows concurrent execution of multiple DOS applications, each in its own virtual DOS machine. Further, this document describes the support for Windows applications under OS/2 Version 2.0.

This document contains information on the MVDM architecture and components, including the use of device drivers by DOS applications, and support for expanded and extended memory. Other MVDM-related topics discussed in this document include the DOS Settings feature, which allows the user to determine the way in which a DOS application runs and the resources available to it, and Virtual Machine Boot, which allows the user to load any version of DOS into a virtual DOS machine to support the execution of version-dependent DOS applications.

Support for Windows applications on the OS/2 Version 2.0 platform is another important topic examined in this document. The document includes a discussion of Windows device drivers, inter-process communication between Windows, DOS, and OS/2 applications (including DDE and clipboard capabilities), and compatibility issues as they relate to Windows applications in the OS/2 Version 2.0 environment.

This document is intended for:

- Customer planners and technical support personnel who require an understanding of DOS and Windows implementation in OS/2 Version 2.0.

- IBM and IBM authorized dealer technical support personnel.

- Programmers of DOS and Windows applications who wish to ensure compatibility of their applications with the OS/2 Version 2.0 platform.

The information contained in this document assumes that readers have a general familiarity with the DOS and Windows environments and the applications which run in these environments.

The code examples used in this document are available in electronic form via CompuServe** or through a local IBM Support Bulletin Board System (BBS), as package RB3731.ZIP. IBM employees may obtain the code examples from the GG243731 PACKAGE on OS2TOOLS.

**The document is organized as follows:**

- *Chapter 1, "Overview"* provides a brief introduction to the topics covered in this document.

  This chapter is recommended for all readers of the document.

- *Chapter 2, "MVDM Architecture"* describes the architecture of the Multiple Virtual DOS Machines component of OS/2 Version 2.0, including information regarding the creation and management of virtual DOS machines.

  This chapter is recommended for those readers who require an understanding of the way in which OS/2 Version 2.0 manages virtual DOS machine resources and an understanding of how MVDM implementation differs from

the implementation of the DOS Compatibility Box in previous versions of OS/2.

- *Chapter 3, "8086 Emulation"* discusses 8086 emulation under OS/2 Version 2.0.

  This chapter is recommended for those readers who desire an overview of the 8086 emulation capabilities of the Intel 80386 processor, which are exploited by OS/2 Version 2.0, and who wish to compare the functions this environment provides to DOS applications with those available to DOS applications under previous versions of OS/2.

- *Chapter 4, "MVDM DOS Emulation"* describes the way in which DOS emulation is achieved by the MVDM component, and compares the functions available in a virtual DOS machine to those available in native DOS 5.0. Considerations for running a DOS application under OS/2 Version 2.0 versus running it under native DOS are also discussed.

  This chapter is recommended for those readers who wish to compare the VDM environment under OS/2 Version 2.0 with that of DOS 5.0.

- *Chapter 5, "Device Drivers"* discusses MVDM device drivers. It describes the device drivers which are supported in a virtual DOS machine under OS/2 Version 2.0 and explains how device drivers are implemented, differentiating between physical device drivers and virtual device drivers. Virtual DOS machine interrupt support is also discussed.

  This chapter is intended primarily for programmers who plan to write device drivers for DOS applications that will run under OS/2 Version 2.0 and for technical support personnel who wish an in-depth understanding of virtual DOS machine device driver support.

- *Chapter 6, "Memory Extender Support"* describes the support for DOS memory extenders provided in the MVDM component. This chapter explains expanded and extended memory support.

  This chapter is recommended for those readers who wish to understand the way in which MVDM supports applications which make use of more than 640KB of conventional memory.

- *Chapter 7, "Installing and Migrating Applications"* describes installing and migrating DOS and Windows applications to OS/2 V2.0 It also discusses the use of the utility for creating a customized migration database.

  This chapter is recommended for system administrators responsible for setting up applications for OS/2 V2.0 users.

- *Chapter 8, "Windows Applications"* describes the implementation of Windows application support under OS/2 Version 2.0.

  This chapter is intended for those readers who wish to run Windows applications under OS/2 Version 2.0.

- *Chapter 9, "DOS Protected Mode Interface"* describes the implementation of the *DOS Protect Mode Interface*, DPMI.

  This chapter is intended for those readers who wish to run Windows applications under OS/2 Version 2.0 and who also need a deeper understanding of the technical implications of this programming interface.

- *Chapter 10, "Running DOS Applications"* describes the way to define, configure and start DOS applications under OS/2 Version 2.0.

This chapter is recommended for those readers who wish to run DOS applications under OS/2 Version 2.0, and who wish to define and configure their application environment for optimum compatibility and performance.

- *Chapter 11, "DOS Settings"* describes the DOS Settings feature of MVDM. This feature allows the user to customize parameters which affect how an application runs in a VDM and the resources available to it.

  This chapter is recommended for every reader who plans to run DOS applications under OS/2 Version 2.0.

- *Chapter 12, "Virtual Machine Boot"* describes the Virtual Machine Boot feature of MVDM, which allows a specific version of DOS to be started within a virtual DOS machine, thereby providing full compatibility for those applications which require version-specific DOS features.

  This chapter is recommended for readers who need to run such applications in a VDM.

**The following appendixes are included in this document:**

1. *Appendix A, "Running Personal Communications/3270 Version 2 for Windows"* explains how to set up and run Personal Communications/3270 Version 2 for Windows in a WIN-OS/2 window.

2. *Appendix B, "Running DOS PC Support/400 in OS/2 V2.0"* explains how to set up and run DOS PC Support/400 in a Virtual Machine Boot session.

3. *Appendix C, "Running Lotus 1-2-3 in a VDM"* explains how to set up and run Lotus 1-2-3 in a virtual DOS machine session with EMS or DPMI support.

4. *Appendix D, "Memory Extender Architectures"* provides a brief overview of the Lotus-Intel-Microsoft (LIM) Expanded Memory Specification (EMS) Version 4.0 and LIMA Extended Memory Specification (XMS) Version 2.0, for those readers who desire an understanding of these specifications in the context of their support by MVDM.

5. *Appendix E, "Multiple Virtual DOS Machines Lab Sessions"* provides a series of lab exercises designed to illustrate the new functions and features of the Multiple Virtual DOS Machines component of OS/2 Version 2.0. The exercises cover such topics as virtual DOS machine  configuration, use of the OS/2 clipboard, virtual DOS machine device drivers, and virtual DOS machine video mode restrictions.

# Related Publications

The following publications are considered particularly suitable for a more detailed discussion of the topics covered in this document.

## Prerequisite Publications

- *OS/2 Version 2.0 Installation Guide*
- *OS/2 Version 2.0 Overview Guide*
- *OS/2 Version 2.0 Online Documentation.*

## Additional Publications

- *OS/2 V1.2 Standard Edition Internals and Evaluation*, GG24-3466
- *OS/2 V1.3 Volume 1: New Features*, GG24-3630
- *OS/2 V1.3 Volume 2: Print Subsystem*, GG24-3631
- *OS/2 Version 2.0 - Volume 1: Control Program*, GG24-3730
- *OS/2 Version 2.0 - Volume 3: Presentation Manager and Workplace Shell*, GG24-3732
- *OS/2 Version 2.0 - Volume 4: Application Development*, GG24-3774
- *OS/2 Version 2.0 - Volume 5: Print Subsystem*, GG24-3775
- *OS/2 Version 2.0 Remote Installation and Maintenance*, GG24-3780
- *IBM DOS 5.0, Windows 3.0, Windows Connection 2.0, Personal Communications/3270 2.0*, GG24-3612
- *Intel 80386 System Software Writer's Guide*, ISBN 1-55512-023-7
- *Virtual Control Program Interface (VCPI) Specification Version 1.0*
- *DOS Protect Mode Interface (DPMI) Specification, Version 0.9*
- *Expanded Memory Specification (EMS), Version 4.0*
- *Extended Memory Specification (XMS), Version 2.0*
- *IBM Personal System Technical Solutions Journal*
- *IBM Personal System Developer Journal*
- *Microsoft Systems Journal*
- *Microsoft Windows Programming Reference*
- *DOS 5.0 User's Guide and Reference*
- *DOS 5.0 Technical Reference.*

# Chapter 1. Overview

A significant new feature of IBM* OS/2* Version 2.0 is the ability to execute multiple DOS applications concurrently, with pre-emptive multitasking and full memory protection for each application. Microsoft** Windows** applications are also supported in the same way. These capabilities allow the use of OS/2 Version 2.0 as an integration platform for DOS applications, Windows applications, and OS/2 applications in a seamless, fully functional environment.

This chapter provides a brief overview of the OS/2 Version 2.0 product and the Multiple Virtual DOS Machines component which provides support for DOS and Windows applications. DOS and Windows application support is then described in more detail in the remainder of the document.

## 1.1 OS/2 Version 2.0

While this document focuses on Multiple Virtual DOS Machines and the support of Windows applications under OS/2 Version 2.0, it is useful to briefly review the highlights of OS/2 Version 2.0.

### 1.1.1 OS/2 Version 2.0 Overview

OS/2 Version 2.0 is an advanced multitasking, single-user operating system for IBM Personal System/2* computers and other computers equipped with the Intel** 80386**, 80486**, or compatible processors. It exploits a rich set of features from previous versions of OS/2, such as support for multitasking, multithreading, dynamic linking, interprocess communication, a graphical user interface, and the High Performance File System. OS/2 Version 2.0, however, provides significant enhancements over previous versions of OS/2.

The following *new features* have been implemented in OS/2 Version 2.0:

- Support for the Intel 80386 32-bit microprocessor instruction set.
- 32-bit memory management.
- Enhanced hardware exploitation.
- Multiple Virtual DOS Machines.
- Support for Microsoft Windows applications.
- 32-bit programming environment.
- Object-code compatibility with previous versions of OS/2, allowing 16-bit applications written for previous versions to execute under Version 2.0 without modification.
- Enhanced Presentation Manager* user shell, which implements the 1991 SAA** CUA Workplace Environment.

### 1.1.2 Memory and Task Management

The foundation for OS/2 Version 2.0 capabilities is its support for the Intel 80386 microprocessor. This support means that a powerful set of 32-bit features now becomes available to the operating system and applications, including enhanced memory management and more sophisticated multitasking.

OS/2 Version 2.0 requires the features of the Intel 80386 or compatible 32-bit microprocessors, and therefore does not run on computers that use the Intel 80286** processor, or its predecessors. In order to maintain compatibility, however, OS/2 Version 2.0 supports applications written for previous versions of OS/2 by providing both a 16-bit as well as a 32-bit application programming interface, allowing 16-bit applications to execute under OS/2 Version 2.0 without modification. The Intel 80386 can address 4 gigabytes of physical memory and up to 64 terabytes of virtual memory.

OS/2 Version 2.0 supports execution of the following types of applications:

- DOS applications, in full-screen mode or in window-mode on the Presentation Manager desktop.

- Microsoft Windows applications, in windows on the Presentation Manager desktop.

- 16-bit OS/2 applications developed for previous versions of OS/2.

- 32-bit applications developed for OS/2 Version 2.0.

All applications execute as protected mode processes under OS/2 Version 2.0, and are therefore provided with pre-emptive multitasking and full memory protection between processes.

*Memory management* is the way in which the operating system allows applications to access the system's memory. The operating system must check how much memory is available to an application, and handle the event when there is no longer any real memory left to satisfy an application's requests.

In OS/2 Version 2.0, memory management has been enhanced to provide a **flat memory model**, which takes advantage of the 32-bit addressing scheme provided by the Intel 80386 architecture. This means that through memory management, the system's memory is seen as one large linear address space of 4GB. This 32-bit programming environment is free from the limitations and inherent complexity of the segmented memory model used by DOS and previous 16-bit versions of OS/2. Memory management within applications is greatly simplified, allowing applications to be developed faster, with better performance due to reduced memory manipulation overhead. Through the use of the flat memory model, applications may be more easily ported to or from other operating system platforms.

*Task Management*, the management of processes and threads executing in a system, is greatly simplified and streamlined under OS/2 Version 2.0. This is due primarily to the fact that support for processes executing in real mode (such as the DOS Compatibility Box in previous versions of OS/2) is no longer required, since the execution of DOS applications is supported using virtual DOS machines which run as protected mode processes under OS/2 Version 2.0.

Interrupt handling under OS/2 Version 2.0 is simplified by removal of the need to handle real mode software interrupts. Interrupts issued by DOS and Windows applications are trapped by a **virtual programmable interrupt controller (VPIC)** which translates the interrupts to the appropriate device access commands for the protected mode environment. The virtual programmable interrupt controller is described in Chapter 5, "Device Drivers."

### 1.1.3 User Interface

OS/2 Version 2.0 also provides an enhanced user shell, known as the **Workplace Shell\*\***, through enhancements to Presentation Manager. The Workplace Shell is object-based and implements the 1991 SAA CUA Workplace Environment. This shell is more intuitive than the Presentation Manager shell implemented in previous versions of OS/2, and allows users to become familiar with the system more quickly.

In the Workplace Shell, system resources, such as files and printers, are regarded as objects, and represented by graphical icons on the screen. Users may manipulate objects (open them for editing, copy them, and print them, for example) by direct manipulation of the icons. For example, a file is copied from one location to another by pointing to it with the mouse, *dragging* the object's icon to the required destination, and *dropping* the icon by releasing the mouse button. This is known as **drag and drop** manipulation.

Presentation Manager and the Workplace Shell are described in more detail in *OS/2 Version 2.0 - Volume 3: Presentation Manager and Workplace Shell.*

## 1.2 Multiple Virtual DOS Machines

OS/2 Version 2.0 provides the user with the ability to run *multiple* concurrent DOS applications, and to multitask these applications with OS/2 applications. In previous versions of OS/2, support for DOS applications was limited to a single DOS session, known as the DOS Compatibility Box, in which the amount of memory available to the DOS application was restricted. Applications running in the DOS Compatibility Box could operate in full-screen mode only, and were suspended when switched to the background.

Support for DOS applications has been completely redesigned in OS/2 Version 2.0, which now provides for the execution and management of multiple concurrent DOS applications, where each application is executed as a single-threaded, protected mode OS/2 program. This capability is provided by a component of OS/2 Version 2.0 known as **Multiple Virtual DOS Machines (MVDM)**.

MVDM introduces powerful DOS application support to OS/2 by exploiting the **virtual 8086 (V86) mode** of the Intel 80386 processor. This mode of operation allows the emulation of an Intel 8086 processor and associated hardware devices within a protected mode 80386 task. OS/2 Version 2.0 uses the virtual 8086 mode to allow the creation of multiple instances of independent **virtual DOS machines**. Through this technique, a virtual interface is provided to each virtual DOS machine, giving the impression that the application running in that machine owns all the required resources, both hardware and software.

Each virtual DOS machine runs as a protected mode process, in a manner similar to an OS/2 application. The use of protected mode allows pre-emptive multitasking of DOS applications and provides a protected system environment in which DOS applications can execute. This means that system memory and all other applications (both DOS and OS/2), are protected from ill-behaved applications, and the user can terminate a DOS application which is "hung." An errant DOS application can affect only its own virtual DOS machine; other applications in the system will not be affected.

Flight Simulator

Boot DR DOS 5.0

Laden...

DR DOS Release 5.0
Copyright (c) 1976,8

A:\>DOS-Vergnügen

123HJG.BAT

DBASE.EXE

Records   Organize   Fields   Go To   Exit

CLIEN_ID
L00001
C00001
L00002
R00005
C00002
B12000
A10025

► Create new index
► Modify existing index
  Order records by index
  Activate .NDX index file
  Include .NDX index file
  Remove unwanted index tag

  Sort database on field list
  Unmark all records
  Erase marked records

Lotus  (R)  1-2-3  (R)  Release 3.1*
6200007-9617124
Copyright 1990 Lotus Development Corporation. All rights reserv

Username:        Hans Goetz
Company name:    IBM

DOS W

[.]
FILE.TXT
[MUGLIB]
[SPOOL]          START
20 file(s)

Boot PC-DOS 3.3

54 Datei(en)         77312 Byte frei
A>ver

IBM Personal Computer DOS-Version  3.30

A>

C:\>ver

The Operating System/2 Version is 2.00

C:\>

Figure 1. Concurrent DOS Applications under the Workplace Shell

Each virtual DOS machine has a great deal more available memory than did the DOS Compatibility Box implemented in previous versions of OS/2. Depending on the use of DOS device drivers and TSR programs, it is possible to have as much as 630KB of available memory for application execution. In addition, OS/2 Version 2.0 supports the use of the Lotus**-Intel-Microsoft (LIM) Expanded Memory Specification (EMS) and the Lotus-Intel-Microsoft-AST** (LIMA) Extended Memory Specification (XMS) to provide additional memory for those DOS applications which are capable of using such memory extenders. OS/2 Version 2.0 maps this expanded or extended memory into the system's linear memory address space, and manages it in the same manner as any other memory.

Each virtual DOS machine may run either in full-screen mode or within a Presentation Manager window. A window containing a DOS application may be sized and manipulated in the same manner as any other Presentation Manager window, and other Presentation Manager desktop features are readily available such as the ability to cut/copy/paste information between applications using the clipboard, or the ability to change fonts.

From the user's perspective, DOS applications behave exactly like VIO applications. DOS applications have the following characteristics:

• They may run in either full-screen mode or in window-mode.

• They can run in the background if doing text screen output.

- Windowed DOS applications have all the same system menu controls as do OS/2 windowed applications, including font adjustment and clipboard functions such as *mark, copy* and *paste.*

Furthermore, DOS applications under OS/2 Version 2.0 have advantages over VIO applications:

- They may be switched between windowed and full-screen while running.

- A full-screen graphics-mode DOS application may be switched into a window and the graphics bitmap will be rendered in the window. This allows the user to copy graphics to the Presentation Manager clipboard and gives the viewer more flexibility when running multiple applications.

- For single-plane graphics modes (CGA, and VGA 320 x 200), DOS graphics applications will execute in a window and continue to update while in the background.

The sole restriction for DOS applications running in a virtual DOS machine when compared with VIO applications is that DOS applications in virtual DOS machines cannot be used in process subtrees. That is, VDMs cannot be run as child processes of either an OS/2 session or another VDM session.

There are some DOS applications and products that cannot be supported by DOS emulation, due to the nature of the emulation code and the multitasking and protection demands of OS/2 Version 2.0. Unsupported products/functions include:

- DOS applications which have internal DOS structure dependencies, such as Windows 1.0x and MS/PC Net.

- DOS applications which do not work in a multitasking environment, such as Norton Disk Utilities**, DOS block device drivers, and Fastback**.

- DOS network drivers, because DOS emulation uses an implementation different from DOS to control its I/O. However, DOS applications running in VDMs may access network services through the normal OS/2 network driver.

Some of these applications may be run under OS/2 Version 2.0 by booting a specific version of DOS in a virtual DOS machine, using the **Virtual Machine Boot** feature of MVDM. This feature is described in detail in Chapter 12, "Virtual Machine Boot."

Application compatibility in the virtual DOS machine is also enhanced over previous versions of OS/2. A virtual DOS machine can be used to execute DOS-based communications applications and other applications which address hardware I/O devices, through the use of **virtual device drivers**, which map device driver calls from DOS applications to the appropriate physical device driver within the operating system. Applications using hardware devices which do *not* have to be shared with DOS applications in the same system may access these devices using the standard DOS device drivers, without the need for a virtual device driver. Certain restrictions still apply with respect to communications line speed and time-critical interrupt handling.

A powerful new feature called **DOS Settings** allows an individual to easily tailor, via Presentation Manager windows, the resources, such as video and memory, available to an application running in a virtual DOS machine, and thus optimize the way in which a DOS application runs.

## 1.2.1 MVDM Architecture

MVDM is designed to exploit the virtual 8086 (V86) mode of the 80386 processor, which allows operating systems such as OS/2 Version 2.0 to execute multiple DOS applications within the 80386 protected mode environment. Under OS/2 Version 2.0, each DOS application executes in a virtual DOS machine (VDM), which runs as a single-threaded protected mode process. The operating system scheduler controls task switching of VDMs in the same way as it does OS/2 application processes.
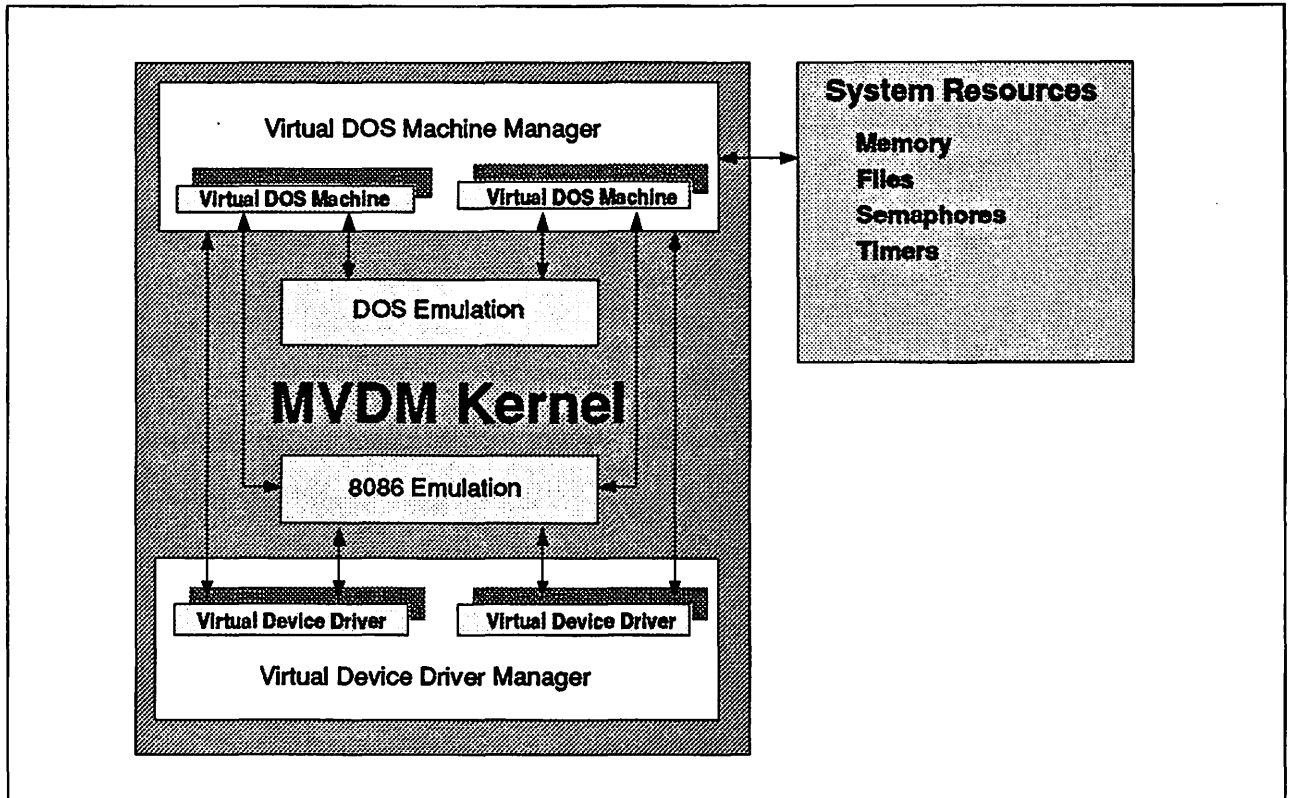


*Figure 2. MVDM Architecture*

The **MVDM kernel** controls the state and operation of concurrent VDMs, and is composed of the following four major components as shown in Figure 2.

1. The **Virtual DOS Machine Manager** (**VDMM**) contains the mechanism to start and interact with DOS applications. It creates, initializes, and terminates VDMs. The VDMM manages system resources such as memory, timers, semaphores, and files for all VDMs active in the system. The VDMM is responsible for loading and initializing all virtual device drivers, in conjunction with the Virtual Device Driver Manager. The VDMM is described in more detail in Chapter 2, "MVDM Architecture."

2. **8086 Emulation** manages communication between 8086 instruction streams and virtual device drivers. This emulation performs 8086 instruction decoding, controls the 80386 processor's I/O Privilege Map for each VDM, manages the reflection of software interrupts for each VDM, routes IN/OUT instruction traps to the appropriate virtual device driver, and manages various control structures required by each virtual device driver. 8086 emulation is described in more detail in Chapter 3, "8086 Emulation."

3. **DOS Emulation** emulates the function and operation of the DOS operating system on a per-VDM basis. Each VDM emulates an entirely independent

instance of DOS. DOS services are emulated within the MVDM kernel, or by invoking protected-mode services provided by the OS/2 kernel. For example, most DOS file I/O functions are provided by the OS/2 file system. DOS 5.0 compatibility is provided. DOS emulation is described in more detail in Chapter 4, "MVDM DOS Emulation."

4. The **Virtual Device Driver Manager (VDDM)** loads, initializes, and communicates with virtual device drivers. Virtual device drivers are required to virtualize the hardware and ROM BIOS, thereby allowing DOS applications to access hardware devices and BIOS without affecting other VDMs or other non-V86 mode processes in the system. The VDDM provides various system services, known as **Virtual Device Helper (VDH)** services, to virtual device drivers. The Virtual Device Driver Manager is described in more detail in Chapter 5, "Device Drivers."

These four components interact with one another and with the OS/2 Version 2.0 operating system kernel to provide requested services to DOS applications executing in VDMs.

## 1.2.2 Virtual Device Drivers

In order for multiple DOS applications, each running in its own VDM, to access physical hardware devices, each VDM must be provided with a set of "virtual interfaces" to these devices, so that the actions of one application do not affect the state of the device as perceived by applications in other VDMs.
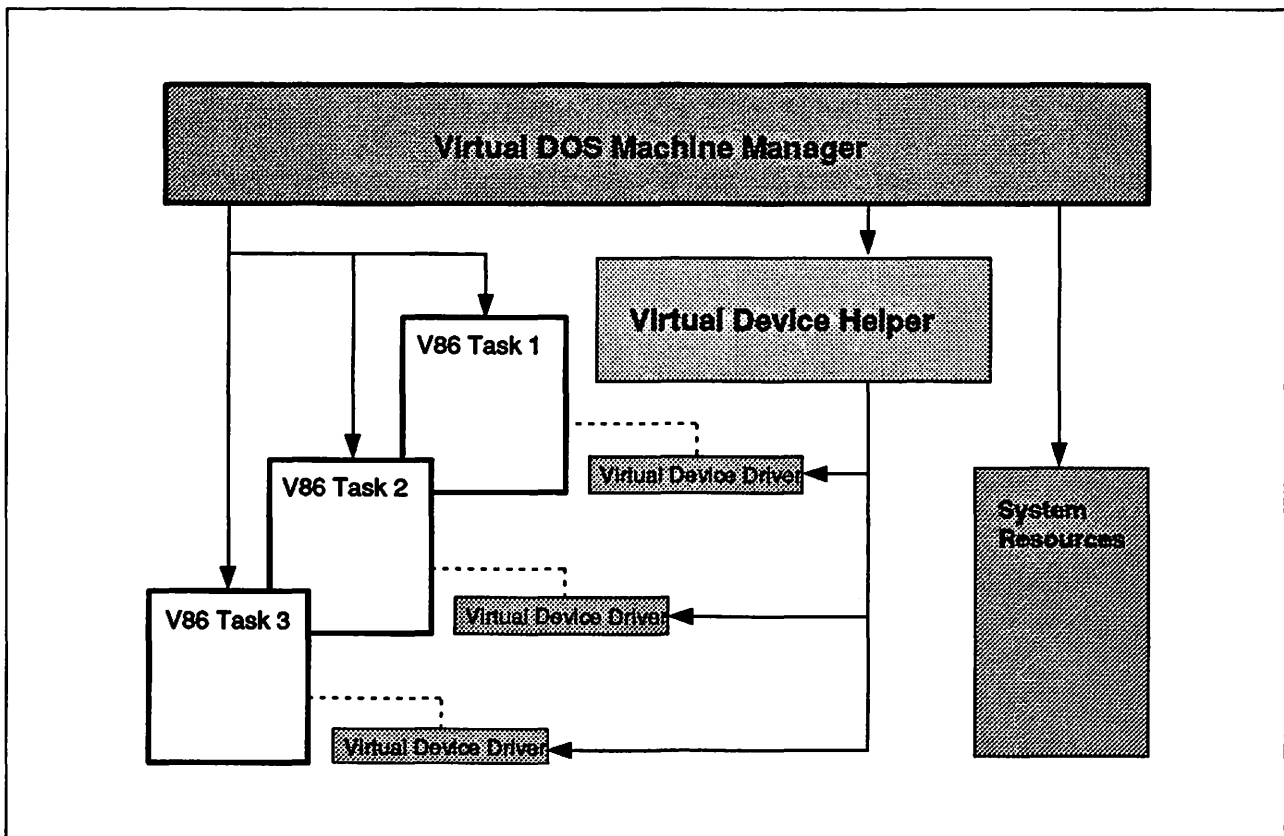


*Figure 3. MVDM System Structure Overview*

These virtual interfaces are provided by OS/2 Version 2.0 using **virtual device drivers**. Virtual device drivers are installable modules responsible for virtualizing the hardware and ROM BIOS aspects of the DOS environment for

virtual DOS machines. A virtual device driver manages shared access to hardware I/O devices for multiple VDMs, allowing an application running in a VDM to act as though it exercised sole control over I/O devices. Virtual device drivers are implemented whenever possible, allowing the BIOS in the system to perform its functions without interference from DOS applications. Virtual device drivers are used to control hardware such as the keyboard, mouse, and serial and parallel ports.

Virtual device drivers are responsible for the following functions:

- Maintaining a virtual hardware state for each virtual DOS machine (VDM)

- Preventing a VDM from corrupting the state of another VDM, or the system as a whole

- Supporting fast screen I/O

- Supporting fast communications I/O.

The virtual device driver architecture implemented in OS/2 Version 2.0 provides support for all standard hardware utilized by DOS applications and supports installable virtualization, so that new hardware may be added and supported by VDMs without requiring an upgrade to the operating system.

Virtual device drivers obtain and release system resources via the **Virtual Device Helper (VDH)** services, provided by the MVDM kernel. A virtual device driver typically performs I/O through a physical device driver using a direct call interface. However, a virtual device driver may directly access an I/O control device. The virtual video device driver performs such direct access under OS/2 Version 2.0 for performance purposes. A virtual device driver may also simulate hardware interrupts into one or many VDM processes.

Under OS/2 Version 2.0, a virtual device driver is inherently protected from a VDM because it is not visible in the VDM address space, although a virtual device driver must be careful to check all parameters coming in from a VDM to ensure that it does not damage itself or some other part of the system by executing an invalid instruction.

## 1.2.3 Expanded and Extended Memory Support

Many popular DOS applications use EMS (expanded) and/or XMS (extended) memory extenders to gain access to memory in protected mode on the 80286, 80386, or 80486 processors. These extenders allow DOS applications to access memory above the 1MB real-mode addressing limit, in order to have total code and data space larger than the available base memory, and to have very large code or data objects loaded into memory for enhanced function and performance. The standard configuration of OS/2 Version 2.0 provides both EMS and XMS functions as part of MVDM.

Under MVDM, EMS and XMS memory allocations are managed as OS/2 pageable virtual memory in the same way as any other memory allocated under OS/2 Version 2.0, and not as fixed physical memory as is the case under DOS. As such, the total expanded/extended memory allocated can exceed the amount of physical memory installed in the system.

### 1.2.3.1 LIM EMS Version 4.0 Support

The LIM (Lotus-Intel-Microsoft) Expanded Memory Specification (EMS) Version 4.0 provides a standard interface for the use of expanded memory with Intel 80x86 computers. The specification allows for up to 32MB of expanded memory, with up to 255 expanded memory objects. Regions within these objects can be mapped into the 8086 address space (below 1MB) as required, allowing DOS applications to access large address spaces.

Under OS/2 Version 2.0, EMS emulation provides the following function:

- Implements all the required functions in the LIM 4.0 EMS.

- Provides each VDM with a logically separate EMS emulation. Each VDM has its own set of expanded objects so that features like interprocess communication work as they would if each VDM were running on a different physical 8086. A VDM cannot affect the availability of objects in other VDMs or access expanded memory "owned" by other VDMs.

- Provides for remapping of conventional memory (below 640KB) for use by programs such as Windows 2.0.

- Provides configurable limits for how much expanded memory is available for all VDMs, as well as a limit per VDM. An installed program in the start list allows the user to override the per-VDM limit, subject to the constraints imposed by the overall limit.

- Supports multiple physical to single logical mappings. Different 8086 addresses can map to the same expanded memory object address. This is required by programs like Lotus 1-2-3**.

EMS emulation is provided in MVDM by the **Virtual Expanded Memory Manager (VEMM)**. VEMM is a virtual device driver offering a separate EMS emulation for each VDM. This is accomplished by placing most EMS control structures for a VDM in a per-VDM instance data area outside the V86 application's address space.

Unlike most virtual device drivers, VEMM does not have a corresponding physical device driver. Rather, VEMM traps software interrupts for EMS services using a system call and manages the appropriate memory. VEMM depends upon the memory management capabilities of the OS/2 Version 2.0 operating system kernel. Allocation, reallocation, or deallocation of EMS memory objects is accomplished by requesting corresponding services from VDH services.

### 1.2.3.2 LIMA XMS Version 2.0 Support

The LIMA Extended Memory Specification (XMS) V2.0 provides a standard interface for the use of extended memory on Intel 80286, 80386, and 80486 computers. XMS functions allow for the moving of code and data objects from base memory into extended memory, and from extended memory to base memory. Two of the best known programs using XMS functions are print spoolers and virtual disks (RAM disks).

The XMS specifications manage three distinct regions of memory:

- The **High Memory Area (HMA)** is located immediately above 1MB and is exactly 65520 bytes (64KB - 16 bytes) in size.

- The **Upper Memory Area (UMA)**, consisting of **Upper Memory Blocks (UMBs)**, is located between 640KB and 1MB.

- The **Extended Memory Blocks (EMBs)** are used only for data storage.

Under OS/2 Version 2.0, LIMA XMS emulation provides the following function:

- Implements all LIMA V2.0 XMS functions.

- Provides each VDM with a separate XMS emulation. Each VDM has its own high memory area, upper memory area, and extended memory blocks, so that features like interprocess communication work as they would if each VDM were running on a different physical 8086 processor. VDMs cannot affect the availability of objects in other VDMs or access memory "owned" by other VDMs.

- Provides configurable limits for how much extended memory is available across all VDMs, as well as a limit per VDM. An installed program in the start list can override the per-VDM limit, subject to the constraint given by the overall limit, and can disable XMS support altogether for a particular VDM if its installation conflicts with the program being run in the VDM.

The **Virtual Extended Memory Manager (VXMS)** is a virtual device driver that provides a separate XMS emulation for each VDM. As with VEMM, this is accomplished by placing most VXMS control structures for a VDM in a per-VDM instance data area outside the V86 application's address space. The amount of memory available to a VDM, the number of handles, and the existence of upper memory blocks are all configurable parameters which may be altered on a per-VDM basis.

Like the VEMM virtual device driver, VXMS does not have a corresponding physical device driver, and utilizes the memory management capabilities of the operating system kernel. XMS object allocation, reallocation and deallocation are accomplished by requesting corresponding services from the memory manager.

## 1.2.4 DOS Settings

MVDM provides the user with the ability to customize the operation of DOS applications via a feature called DOS Settings. This feature allows the user to control special properties which affect the behavior of DOS applications running in a VDM.

The DOS Settings feature further enhances the DOS compatibility of a VDM because it allows a user to configure the VDM for DOS applications which might otherwise not work well (or not work at all) with the default settings for a VDM. The DOS Settings feature also gives the user more control over the consumption of system resources by a DOS application. Help is provided for each setting to assist users in tuning their applications' operation.

DOS sessions have many more customizable properties than OS/2 sessions. MVDM provides a common mechanism that supports both a standard complement of settings, and allows virtual device drivers to register custom settings. The standard settings are a subset of the configuration settings available in the CONFIG.SYS file, plus some additional settings required for MVDM. The primary reason for the existence of the option to alter these settings is that DOS applications are typically not careful about consuming system resources, such as memory and processor time. Hence, MVDM itself must provide a flexible environment for these applications in order to preserve the integrity and performance of the system as a whole.

DOS settings are managed on a per-VDM basis and are accessed through Presentation Manager windows. The dialog boxes presented allow the user to change a setting while the VDM is running. Only those settings that can be

changed for that VDM are presented. There are many settings which can be tuned. One parameter, for example, the *Idle Detection Threshold*, detects idle DOS applications and allows the user to configure the system such that processor time is not wasted by idle DOS applications.

Note that while the DOS Settings feature provides significantly enhanced control over the behavior and capabilities of a virtual DOS machine, this level of control is not necessarily obvious to the end user. Most DOS applications will execute quite satisfactorily with the default VDM settings, and the user is therefore not required to use the DOS Settings feature. This approach therefore provides the increased functionality without necessarily increasing the complexity of user interaction.

## 1.3 Windows Application Support

OS/2 Version 2.0 provides the capability for Microsoft Windows applications to run under OS/2 Version 2.0. This support allows applications written for Windows 3.0 and previous versions of Windows (except V1.x) to coexist with OS/2 and DOS applications under OS/2 Version 2.0.

Each Windows application executes in a virtual DOS machine, and is thus a protected mode process. As such, Windows applications are subject to the same application protection facilities provided to other protected mode processes under OS/2 Version 2.0. Windows applications are protected from other Windows applications and from DOS and OS/2 applications executing in the system. This is in contrast to the native Windows 3.0 environment, where limited protection is provided for Windows 3.0 applications, and none at all for DOS applications unless Windows is running in enhanced mode.

The execution of Windows applications as protected mode tasks also allows these applications to take full advantage of the pre-emptive multitasking capabilities of OS/2 Version 2.0, with full pre-emptive multitasking between Windows applications, OS/2 applications, and DOS applications. This is again in contrast to the native Windows 3.0 environment, where pre-emptive multitasking is available only when Windows 3.0 is running in enhanced mode, thereby impacting performance and preventing many applications written for previous versions of Windows from executing. OS/2 Version 2.0 has no such restriction.

Windows applications running under OS/2 Version 2.0 will run in a mode equivalent to the *real* or *standard* modes of Windows 3.0; the *enhanced* mode of Windows 3.0 is not required since the OS/2 Version 2.0 operating system itself provides equivalent function.

Support for Microsoft Windows applications under OS/2 Version 2.0 is discussed in more depth in Chapter 8, "Windows Applications."

## 1.4 Summary

OS/2 Version 2.0 provides significantly enhanced DOS application support capability over previous versions of OS/2, using a component known as Multiple Virtual DOS Machines. MVDM offers many significant functions and features, including:

- Ability to run multiple DOS sessions concurrently, with full pre-emptive multitasking and memory protection

- Ability to run DOS applications in Presentation Manager windows

- Ability to switch between DOS applications via Presentation Manager

- High amount of available free memory for DOS applications (630KB and more)

- Expanded (EMS) and extended (XMS) memory emulation support

- Clipboard support.

OS/2 Version 2.0 provides DOS 5.0 compatibility within the virtual 8086 mode of the 80386 processor, and allows execution of multiple concurrent DOS applications, each operating in its own 1MB virtual address space. This brings true multiprogramming to the DOS environment under OS/2. A user may run multiple DOS applications in much the same way as they run multiple OS/2 applications. DOS and OS/2 applications can be started in the same ways:

1. From a desktop group window.
2. From the Drives folder.
3. From a command prompt.
4. From the OS/2 START command.

The DOS environment is more DOS-compatible than the DOS Compatibility Box implemented under previous versions of OS/2, since OS/2 Version 2.0 encapsulates the entire DOS environment in a virtual machine. This provides better protection for other processes in the system and for the operating system environment itself. With MVDM, an errant DOS application can only "hang" its own virtual DOS machine, which may then be terminated by the user without affecting other applications in the system.

DOS applications may be run full-screen, windowed, or iconized in the background. Besides being better protected, providing better compatibility and more concurrent DOS applications, the OS/2 Version 2.0 MVDM environment leaves applications with more than 630KB of memory in which to execute.

OS/2 Version 2.0 also provides support for the execution of Microsoft Windows applications (written for Windows 3.0 and/or previous versions of Windows, except version 1.0x) to execute under the control of OS/2 Version 2.0. Each Windows application executes as a separate protected mode task, and is therefore provided with the same pre-emptive multitasking and memory protection as other protected mode tasks under OS/2 Version 2.0.

Support for both the Expanded Memory Specification (EMS) and the Extended Memory Specification (XMS) is provided. DOS asynchronous communications applications can support communication speeds of up to 9600 baud. Since the DOS environments are swappable, starting many DOS sessions will not increase requirements for more physical (real) memory.

The MVDM environment also provides an extendable architecture that allows the environment to be custom tailored to emulate any DOS environment. The virtual device driver architecture supports this flexible environment. All of the low-level DOS support, which in previous versions of OS/2 resided in physical device drivers, has been moved into virtual device drivers. In virtual 8086 mode, all interrupt processing is done in protected mode; bimodal device drivers are no longer needed. The new driver architecture provides physical device drivers for basic device support and virtual device drivers for supporting virtual devices to the DOS environments. DOS settings allow the user to tailor the functioning of DOS applications in the VDM environment.

# Chapter 2. MVDM Architecture

The Multiple Virtual DOS Machines component of OS/2 Version 2.0 is itself comprised of a number of subcomponents, which interact with one another and with the OS/2 Version 2.0 operating system kernel to provide services to DOS applications running in virtual DOS machines. This chapter describes each subcomponent of MVDM, and explains the interaction between subcomponents.

## 2.1 Introduction

OS/2 Version 2.0 is designed to fully exploit the advanced features of the Intel 80386 processor. A major innovation of the 80386 is its support for the execution of multiple 8086 tasks within the 80386 protected mode environment. An 8086 task in this environment is called a virtual 8086 (V86) task. Under OS/2 Version 2.0, V86 tasks are implemented as **virtual DOS machines (VDMs)**, and each runs as a single-threaded, protected mode process. See also Figure 3 on page 7. The OS/2 scheduler controls task switching for V86 processes in a way similar to the manner in which it controls other OS/2 application processes. When a task switch occurs, the VM bit in EFLAGS contained in the V86 process's task state segment indicates the type of the current process. If the VM bit is set, indicating that the process is a VDM, the processor switches to V86 mode.

In this area, performance has been improved over previous versions of OS/2, since the processor is *never* switched to real mode. Switching from protected mode to real mode takes a long time since all CPU register contents must be saved and paging must be disabled before DOS registers are loaded. Switching to real mode is often accomplished by resetting the CPU, which is very time consuming. The V86 mode of the processor allows the system to run both OS/2 and DOS applications in protected mode.

Compared to previous versions of OS/2, DOS applications running in VDMs may:

- Run full-screen or in a window
- Run in a background session and not be suspended
- Use the clipboard
  - Copy text
  - Copy graphics as bitmaps
- Run graphics in full-screen mode
- Switch between full-screen and windowed mode
- Use LIM EMS Version 4.0 expanded memory services
- Use LIMA XMS Version 2.0 extended memory services.

Full-screen graphics applications may be switched to windowed mode where the graphics will be displayed as a bitmap. Switching between modes can be done via the system icon menu when in windowed mode. If in full-screen mode, the user must first switch to the Presentation Manager screen group. Selecting *Windowed* in the menu of the DOS program icon switches the application to windowed mode. To facilitate mode switching, the hot-key combination *Alt + Home* may be used.

While in full-screen mode, the user may copy only the entire contents of the screen to the clipboard. Switching to windowed mode enables the user to copy parts of the screen to the clipboard by selecting areas with the mouse.

**13**

DOS compatibility is achieved through a combination of hardware and software which ensure the successful execution of DOS applications. Since DOS compatibility is something of a "moving target," MVDM has been architected to provide the maximum possible flexibility. When attempting to ensure the proper execution of a DOS application, typical variable factors to be considered are the hardware and ROM BIOS of the machine, as well as DOS and the application itself.

```
  DOS Operating System
+ Hardware
+ DOS Application
----------------------
= Compatibility
```

The following DOS functions are supported by virtual DOS machines:

- All *documented* DOS system interfaces

- Most direct ROM BIOS interfaces

- Memory extenders

    - LIM EMS Version 4.0
    - LIMA XMS Version 2.0

- Direct manipulation of common hardware devices.

VDMs have certain restrictions:

- Single tasking only; no child processes
- No active background graphics
- No DOS block device drivers; such device drivers are not written for a multi-tasking environment, and may compromise the integrity of other VDMs, or of other processes in the system.
- No direct manipulation of hard disk data (that is, bypassing, in this case, the OS/2 file system); this may also compromise the integrity of other processes in the system.
- No DOS network device drivers, due to differences in the internal implementation of DOS I/O. However, DOS applications running in VDMs may access LAN resources through the normal OS/2 network drivers, and therefore no function is lost.

Figure 4 on page 15 provides an overview of the OS/2 Version 2.0 system structure, showing the MVDM kernel and virtual device drivers in relation to key components of the operating system kernel and physical device drivers which provide services to MVDM.

*Figure 4. MVDM System Structure and Control Flow*

Note that virtual device drivers typically access hardware devices through a physical device driver. Direct communication between virtual device drivers and the hardware (as shown in Figure 4) is used only in exceptional circumstances. One such case is the virtual video device driver, VVIDEO.SYS, which communicates directly with hardware in order to achieve the highest possible level of performance.

## 2.2 Virtual DOS Machine Manager (VDMM)

OS/2 Version 2.0 enables the user to start multiple DOS applications in multiple concurrent VDMs, all of which are controlled by the **Virtual DOS Machine Manager (VDMM)**.

*Figure 5. MVDM Protected Mode processes*

The VDMM creates a VDM by:

1. Allocating and mapping the required memory

2. Loading and initializing the virtual device drivers required to virtualize the hardware and ROM BIOS

3. Access to the virtual device helper services and system resources, such as memory, semaphores, timers, and files, is provided by the VDMM for all VDMs.
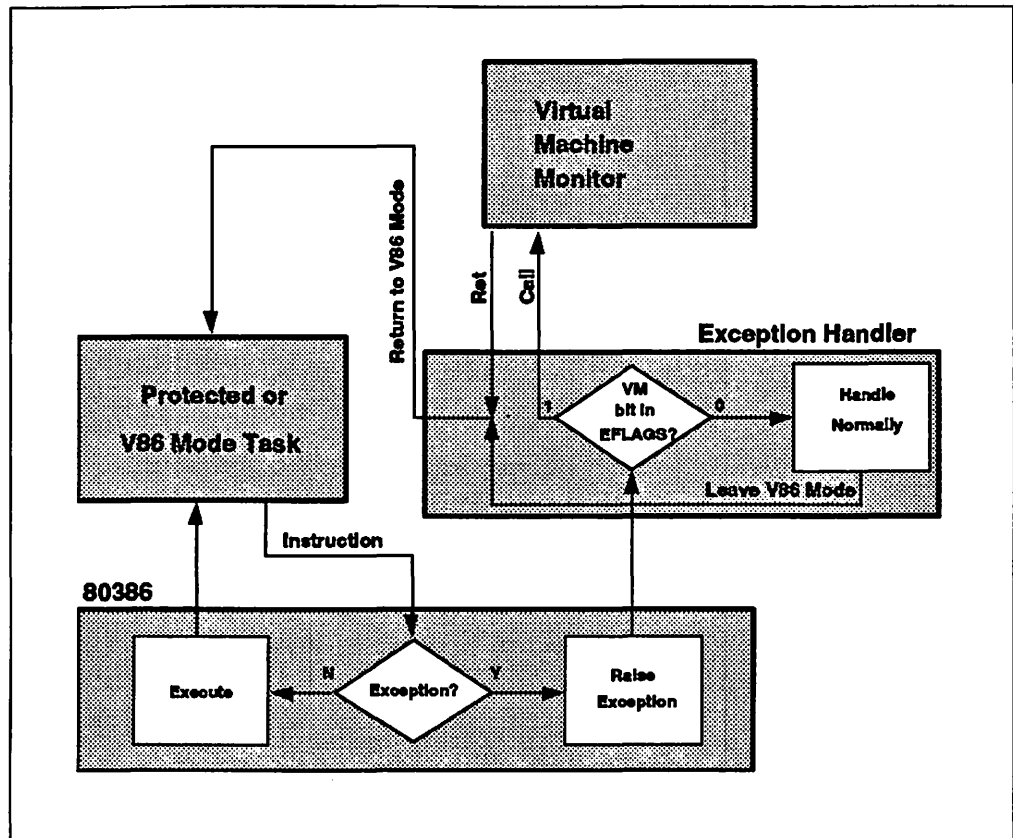
*Figure 6. VDM Exception Handling*

The VDMM is responsible for handling those 8086 instructions which cannot be executed in V86 mode, such as interrupts. Such instructions cause an exception to be generated by the 80386 processor; this exception is passed to an exception handler within the operating system, which checks the VM bit in the current process's EFLAGs control area. If the VM bit is set, control is passed to the VDMM. The VDMM therefore runs as a protected mode system level process at privilege level 0.

## 2.2.1 VDM Address Space Management

At initialization, each VDM has a linear address space of 4MB, because a single page table is assigned for each VDM. The page table itself is a single 4KB page, and can hold a maximum of 1024 page table entries. Each of these entries is a doubleword and points to a page with a size of 4KB, hence the 4MB size of the initial address space. The size of the address space can be expanded beyond 4MB if required; for instance, an application running in the VDM may request the allocation of a large amount of expanded memory, in which case the VDMM will allocate the memory as needed.

Note that when memory is allocated OS/2 V2.0 merely reserves a linear address range. The range is not backed by physical memory until the memory is *committed*. Memory may not actually be committed until a later time, and it may be committed in portions. Allocation without commitment does not use any physical memory and therefore does not waste resources.

A typical VDM address space map is shown in Figure 7 on page 18.
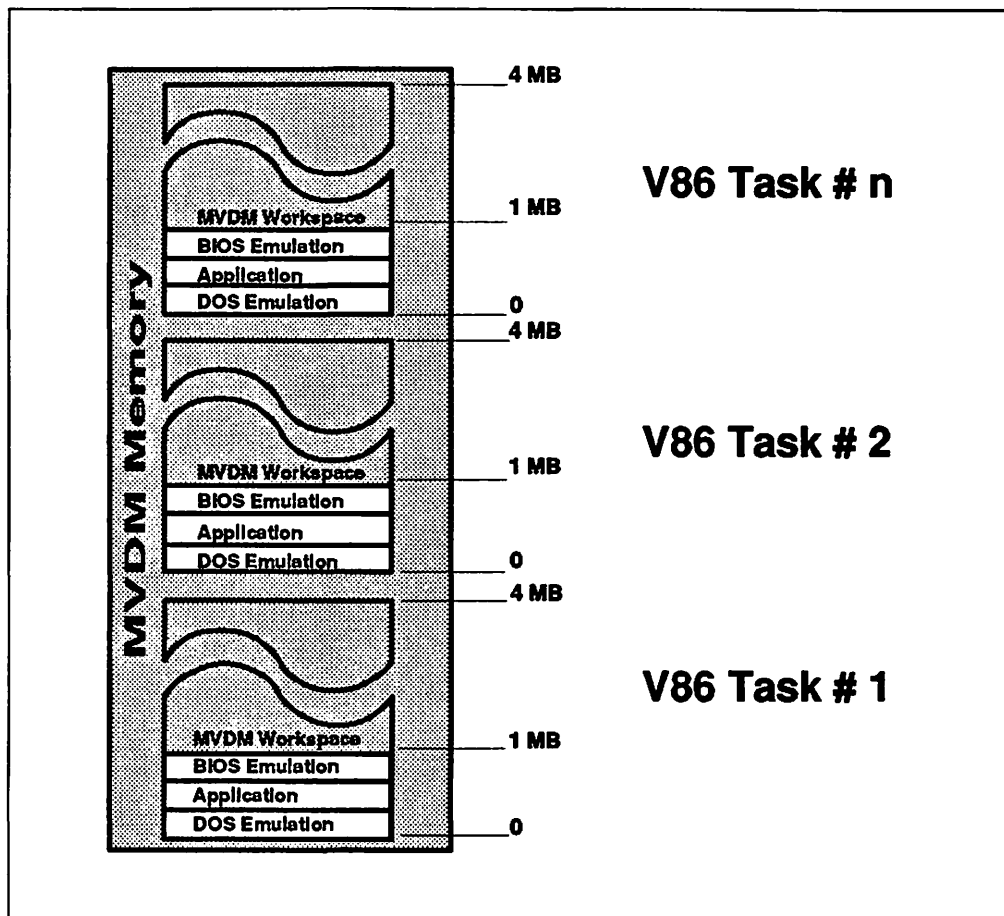
*Figure 7. Typical VDM Address Space Map*

Each VDM task executes in the first megabyte of the linear address space, thereby allowing the physical addresses used within the DOS applications to be mapped directly to the process address space of the VDM. DOS system areas such as the ROM BIOS area and the Interrupt Vector table, are mapped from physical memory into the VDM's address space by the virtual device driver VBIOS.SYS.

Page 0 contains, for example, the ROM BIOS area, the Interrupt Vector table, and the DOS communication area. ROM areas used by hardware devices are mapped by the virtual device drivers into the linear address space of the VDM. The same applies to other linear memory objects, such as EMS objects and display memory.

A virtual device driver uses the **VDHQueryFreePages()** device helper service to find a region where no memory in the linear address space has been allocated or mapped. If a free region is found, the **VDHReservePages()** service is used to reserve the memory region. Mapping cannot take place where memory has already been allocated. On the other hand, memory may not be allocated where mappings already exist. The operating system's memory manager takes care of this. Page faults are generated if a process attempts to access memory which has previously been invalidated because it was unmapped, or if the page has been swapped out to disk.

The 64KB memory area above 1MB (also known as the **high memory area**) must be treated in a special way. This is described in further detail in 3.4, "A20 Line

## 2.2.2 VDM Creation

A number of tasks are performed when a VDM is initialized. The four main tasks managed by the Virtual DOS Machine Manager are:

1. Task state segment (TSS) initialization

2. Virtual DOS machine process initialization

3. 8086 Emulation initialization

4. DOS Emulation initialization.

### 2.2.2.1 Task State Segment Initialization

The **task state segment** describes the current machine state, including CPU register contents, and the current software state of a task, such as file descriptors and priority information. The following steps are necessary to initialize the TSS:
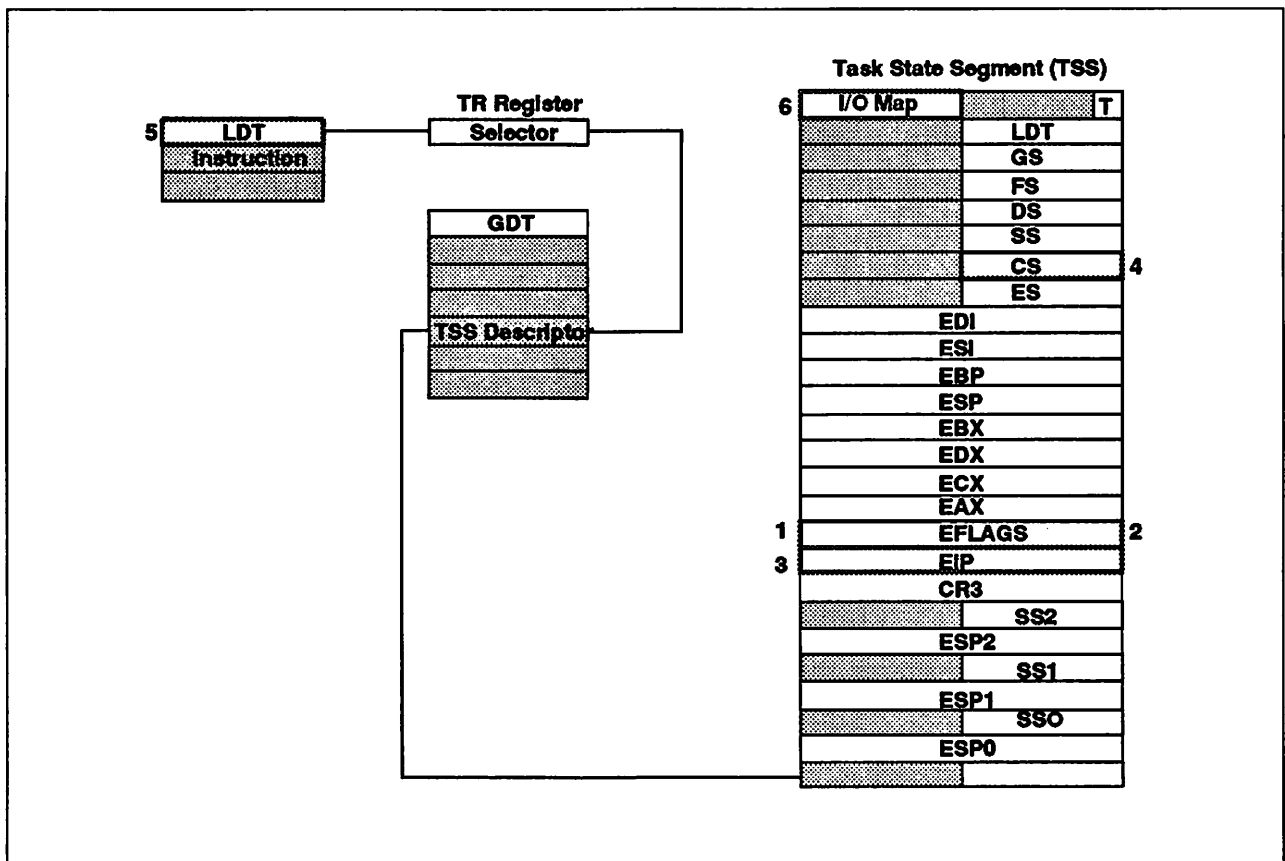


*Figure 8. VDM Initialization*

1. The VM bit in EFLAGS is set to 1, indicating that the process is a virtual DOS machine.

2. The IOPL indicator in EFLAGS is set to 3

3. The instruction pointer is set to the task's entry point

4. The code segment (CS) register is set to the linear base address of the task's initial code segment

5. An LDT is not used in V86 mode, but one must be initialized since interrupt or exception routines might use an LDT

6. I/O access rights are defined in the I/O permission map.

### 2.2.2.2 VDM Process Initialization

The *VDM process initialization* is very similar to the creation of a protected mode session:

1. The Per-Task Data Area (PTDA) is created

2. A process slot and a process ID are allocated

3. The operating system's memory manager provides a 4MB linear address space

4. A 4MB global Page Directory Entry, which references the 4MB linear space, is created

5. The VDM process is started.

### 2.2.2.3 8086 Emulation (V86) Initialization

The *8086 Emulation initialization* then proceeds as follows:

1. The 8086 Emulation code is loaded

2. VDM kernel data is allocated in per-VDM memory below 4MB

3. Virtual device driver instance data is allocated in per-VDM memory below 4MB

4. All known VDM creation hooks (registered by VDHInstallUserHook) are invoked to initialize virtual device drivers on a per-VDM basis

5. The VEMM (expanded memory) virtual device driver (if used) allocates a block of memory in a mappable window either between 640KB and 1MB or in the area between 256KB and RMSIZE (as specified in CONFIG.SYS), and maps it into the VDM address space.

### 2.2.2.4 DOS Emulation Initialization

The *DOS Emulation initialization* then proceeds as follows:

1. VDM-related kernel structures are initialized

2. DOS Emulation kernel (DOSKRNL) is loaded

3. Virtual processor mode is started

4. Standard file handles are opened

5. Virtual device driver and DOS device driver "stubs" are initialized

6. DOS device drivers are initialized

7. DOS shell is loaded and executed.

DOS Emulation initialization and VDM creation are discussed in greater detail in Chapter 4, "MVDM DOS Emulation."

Once DOS Emulation is complete, the Virtual DOS Machine Manager then issues the VDM_CREATE_DONE event handler to install default I/O hooks, page fault hooks, or INT hooks. Control is then passed to the DOS Emulation kernel to initialize and start the user application program.

Once the VDM creation process is complete, the name of the DOS program and other customization parameters are passed to the new VDM. Customization parameters for a specific DOS application can be specified in the Workplace Shell folder in which the name of the DOS program is contained. To change these parameters select the *DOS Settings* pushbutton.

DOS Settings allow the user to tune the VDM environment in which a DOS application runs in order to achieve optimal execution of the program.

## 2.2.3 VDM Termination

A VDM is terminated when the DOS application running within it terminates, or when a virtual device driver terminates the VDM due to some illegal operation. Termination is carried out in the following way:

1. All registered VDM termination hooks are called

2. The 8086 emulation releases all its per-VDM resources

3. The VDMM releases all its per-VDM resources

4. The VDM is destroyed in a way similar to an OS/2 process.

The criteria for abnormal termination of a VDM by a virtual device driver are discussed in 5.6, "VDM Termination" on page 89.

## 2.3 8086 Emulation

OS/2 Version 2.0 MVDM 8086 Emulation provides "pure" 8086 emulation support for the V86 mode of the Intel 80386 SX and DX processors.[1] In a native 8086 processor environment, an application can access all available memory up to 1MB and can execute all 8086 processor instructions when running as a single task in an unprotected environment. However, virtual DOS machines run only in protected mode, and an application running in the V86 mode environment therefore has limited access to system memory and cannot execute all CPU instructions; only the OS/2 Version 2.0 operating system itself has full access to memory and the processor.

Each VDM runs as a single V86 mode process, emulating all DOS operating system functions and providing a compatible DOS environment unaffected by other VDMs in the system. In the VDM environment, therefore, an application behaves as if it has complete control over system processor and memory resources. In this way, full application function is provided while preserving the integrity of other applications and of the system as a whole.

The 8086 Emulation component provides the following:

- Software interrupt reflection support
- IOPL sensitive instruction emulation support
- I/O port trapping
- I/O simulation support
- Context hook services
- Hook software interrupt virtual device driver service
- Change VDM execution flow services.

---

[1] The Intel 80486 SX and DX processors also provide a compatible V86 mode, and are therefore supported by MVDM.

8086 Emulation is described in more detail in Chapter 3, "8086 Emulation."

Note that the 8086 Emulation component *does not* provide DOS or hardware-specific emulation. These emulations are provided by the DOS Emulation component (described in more detail below and in Chapter 4, "MVDM DOS Emulation") and virtual device drivers (described in more detail below and in Chapter 5, "Device Drivers") respectively.

## 2.4 DOS Emulation

Provision of DOS compatibility requires a combination of hardware, operating system, and application software support. The MVDM DOS Emulation component of OS/2 Version 2.0 addresses only the software aspects of providing DOS compatibility; the VDM Manager, 8086 Emulation and virtual device drivers work together to provide hardware and ROM BIOS compatibility.

DOS Emulation provides DOS services to DOS applications running in a virtual DOS machine in such a way as to provide maximum compatibility with the same services provided to DOS applications running under native DOS 5.0.

DOS Emulation is implemented by running a very small portion of the DOS Emulation kernel in V86 mode and a much larger portion of code in protected mode outside the VDM. In OS/2 Version 2.0, physical device drivers are loaded above 1MB and only the DOS Emulation kernel resides below 1MB; hence, any user-installed OS/2 device drivers will not affect the amount of application space available to a DOS application running in a VDM. Similarly, adding LAN drivers to the OS/2 configuration to support the network server or redirector functions does not take up DOS application space, even though DOS applications may make use of these network devices. Virtual device drivers are also loaded outside the VDM address space, and therefore do not reduce the amount of memory available to a DOS application.

In this way, the MVDM architecture makes available to DOS applications the maximum amount of memory. In fact, up to 630KB is free for multiple DOS sessions; this represents an increase of about 100KB over memory available to the single DOS session that was available under OS/2 Version 1.3. Note that this amount may be reduced if DOS device drivers and/or TSR programs are loaded in a VDM.

DOS Emulation supports all documented DOS interrupts and features. In addition, some undocumented aspects of these functions (especially INT 21h) are supported because a large number of significant DOS applications rely upon these interfaces.

DOS Emulation is described in more detail in Chapter 4, "MVDM DOS Emulation."

## 2.5 Virtual Device Drivers

Virtual device drivers are installable modules responsible for virtualizing the hardware and ROM BIOS aspects of the DOS environment for virtual DOS machines. A virtual device driver manages shared access to hardware I/O devices for multiple VDMs, allowing an application running in a VDM to act as though it exercised sole control over I/O devices. See also Figure 3 on page 7.

Virtual device drivers are implemented whenever possible, allowing the BIOS in the system to perform its functions without interference from DOS applications. Virtual device drivers are used to control hardware, such as the keyboard, mouse, and serial and parallel ports.

The virtual device driver architecture implemented in OS/2 Version 2.0 provides compatible support for all standard hardware utilized by DOS applications and supports installable virtualization, allowing new hardware to be added in the field and supported by VDMs without requiring an upgrade to the operating system.

Virtual device drivers are responsible for the following functions:

- Maintaining a virtual hardware state for each virtual DOS machine (VDM)

- Preventing a VDM from corrupting the state of another VDM, or the system as a whole

- Supporting fast screen I/O

- Supporting fast communications I/O.

Since DOS may be emulated more than once in OS/2 Version 2.0, virtual device drivers must virtualize the following features to maintain a separate hardware state for each VDM:

- ROM BIOS services
- Direct manipulation of ROM BIOS data area
- Direct manipulation of video RAM
- Direct programming of I/O ports
- Direct manipulation of device memory
- Hardware interrupts
- Software interrupts.

A virtual device driver typically performs I/O through a physical device driver, using a direct call interface. However, a virtual device driver may directly access an I/O control device; this technique is used by the virtual video device driver, VVIDEO.SYS, for performance reasons. A virtual device driver may simulate hardware interrupts into one or many VDM processes.

Virtual device drivers use a minimal amount of memory. Virtual device drivers do not have to reserve arrays of data structures for each VDM (as did device drivers under previous versions of OS/2). They may be made swappable, so that each device driver does not increase the amount of conventional (or real) memory space consumed. Conventional memory is used only for code and data that must be accessible at hardware interrupt time (for example, when calling a physical device driver).

Under OS/2 Version 2.0, a virtual device driver is inherently protected from a VDM because it is not visible in the VDM address space, although the device driver must be careful to check all parameters coming in from a VDM to ensure that it does not damage itself or some other part of the system by executing an invalid instruction.

Virtual device drivers obtain and release system resources via the **Virtual Device Helper (VDH)** services provided by the MVDM kernel. These helper services are accessed via a published programming interface so that manufacturers of hardware devices may develop virtual device drivers for their own devices. Virtual device drivers are installed using the DEVICE= statement in CONFIG.SYS.

Note that a virtual device is required *only* if a device will be shared with other virtual DOS machines. If a particular device is to be used exclusively by one DOS application, a normal DOS device driver (that is, one that is written for DOS) may be used, and a virtual device driver is not required.

## 2.6 VDM Page Faults

A page fault exception may occur in a VDM where a particular page in real memory has been swapped to disk. When a page fault occurs for a linear region which has been initialized as an **alias** by a virtual device driver, the exception is routed to an exception handler, which has been registered previously by the virtual device driver for the linear address region in which the page fault occurred. The exception handler may cause the page to be loaded or may allow the memory reference to default into a temporary data page, several of which are provided by the MVDM kernel at initialization time.

Exception handlers are registered by virtual device drivers at initialization time using the **VDHInstallFaultHook()** helper function. If no exception handler is registered for the linear region in which the page fault occurred, the page is mapped to temporary data pages in memory.

The start address and size of each aliased region, and the exception handler address for each aliased region, is kept in a table, which is set up via the **VDHInstallFaultHook()** helper service. When a page fault occurs in the VDM address space, this table is searched for a matching region, and the exception handler for that address is called. The page address and the type of fault which occurred are passed to the exception handler.

## 2.7 VDM Window Management

OS/2 Version 2.0 provides for the ability to run a V86 task in a window. The PMShield manages windowed VIO sessions input/output and windowed VDM input/output.

The video VDD (Virtual Device Driver) intercepts all I/O instructions to the video adapter, and all screen updates will be redirected/mapped to a logical video buffer (LVB) maintained by the Video VDD. When the Video VDD detects changes in a VDM's video state, it notifies a new PMShield thread, providing it with update information for a specified VDM window. This thread is a high-priority thread that serves the needs of all windowed VDMs.

For keyboard and mouse input, the PMShield must simply transform keystroke and mouse event messages into calls to the keyboard and mouse VDDs.

## 2.7.1 Virtual Display Management
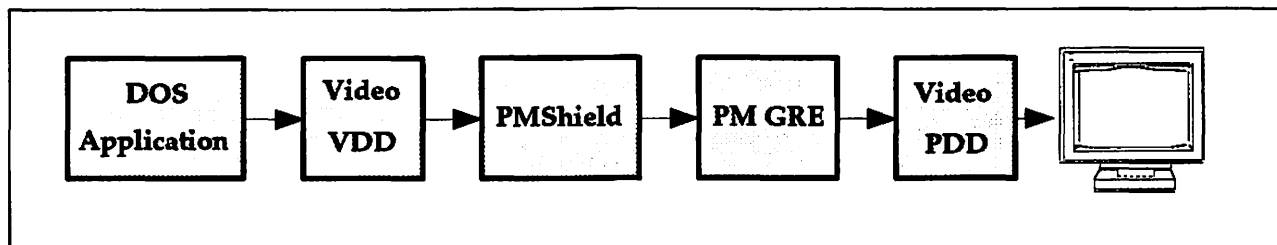


Figure 9. Virtual Display Management

The PMShield is notified by the Video VDD of different changes, which are prioritized, for example "change in video mode," "change in palette," "change in LVB," "scroll of LVB," "string output," "change in cursor," "input notification" when the window scroll has to be adjusted based on the cursor position, "paste notification."

Having received notification of one of these changes, it is the PMShield's responsibility to make appropriate changes to the VDM window, usually through the use of one or more PM Graphics Engine call(s).

When a windowed VDM is found with a dirty page, the PMShield thread is notified of a LVB change, along with rectangles describing which areas of the VDM's LVB may be dirty. The PMShield finds the smallest rectangle(s) of change, and updates the window using appropriate Graphics Engine services.

The PMShield obtains access to the VDM's LVB indirectly, by requesting the Video VDD to copy some rectangular portion of it into a "shield buffer." The PMShield compares the "shield buffer" against a "shadow buffer," which contains the previously displayed contents of the VDM window, to find the smallest areas of change. The roles of the two buffers are then interchanged in preparation for the next update, as it is now the "shield buffer," which contains the last data displayed.

If the VDM changes video modes before the PMShield is able to copy the VDM's LVB, an error will be returned to the PMShield on the copy request. The PMShield's action will be to query the new mode and recopy the LVB.

## 2.7.2 Virtual Keyboard Management
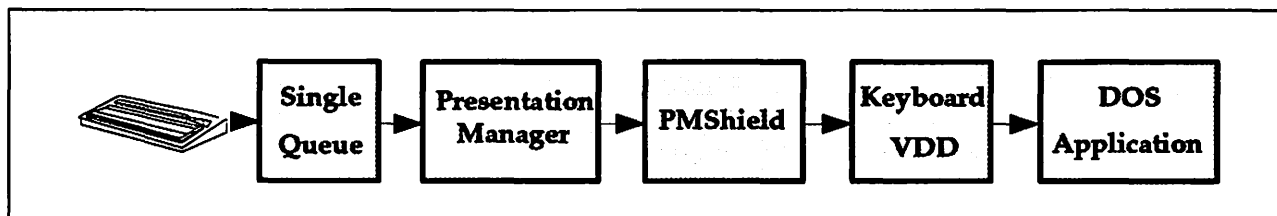


Figure 10. Virtual Keyboard Management

The PMShield transforms keystroke messages into calls to the Keyboard VDD. No buffering by the PMShield is necessary, since the Keyboard VDD already maintains its own virtual keyboard buffer.

The only keystrokes that require special handling by the PMShield are those that affect shift states.

There are two operations that a VDM can perform on its virtual keyboard:

- Change repeat rate
- Change shift states.

When a VDM changes the repeat rate, whether it is in the foreground, background or windowed at that time, the repeat rate is changed for the whole system. Since the repeat rate is not a per-session attribute under OS/2, the PMShield need not be involved. The shift state information is per-VDM and is managed by the Keyboard VDD, in conjunction with the physical keyboard driver.

## 2.7.3 Virtual Mouse Management
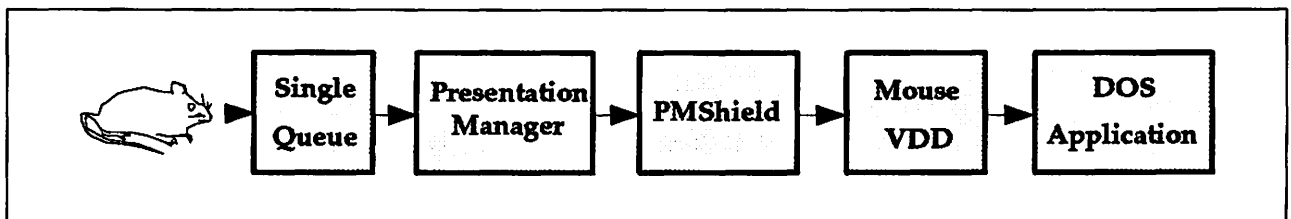


*Figure 11. Virtual Mouse Management*

For mouse input, the PMShield will transform mouse event messages into calls to the Mouse VDD. The conversation will require a mapping of the physical mouse position reported by Presentation Manager to a corresponding position within the VDM's screen dimensions.

A VDM can change the state of its virtual pointer in various ways:

- Change pointer position
- Change pointer shape
- Change pointer scaling factors.

These special changes are of no interest to PMShield.

## 2.8 VDM Interprocess Communication

Communication between processes is valuable in a multitasking operating system to enable concurrent processes to work together. Pipes are one of three forms within OS/2 of interprocess communication (IPC) and the only form available for IPC from a DOS application running in a VDM to an OS/2 application.

This chapter describes how to create, manage, and use named pipes for IPC of a DOS application in a VDM to an OS/2 program. Pipes enable two or more processes to communicate as if they were reading from and writing to a file.

## 2.8.1  About Pipes

A *pipe* is a named or unnamed buffer used to pass data between processes. A process writes to or reads from a pipe as if the pipe were standard input or standard output. A parent process can use pipes to control the input that a child process receives and to receive the output that the child process produces. There are two types of pipes - named and unnamed. The only supported pipe to be used between a DOS application in a VDM and an OS/2 program is the **named pipe**.

## 2.8.2  Named Pipes

Named pipes enable related or unrelated processes on the same computer system or different systems to communicate with each other. Any process that knows the name of a pipe can open and use a named pipe. In addition, named pipe data can be transparently redirected across a network, such as a local area network (LAN).

---
**Note**

The current file system implementation for VDMs does not support named pipes across the LAN. (This limitation will go away with future versions of OS/2 V2.0). To use named pipes within a VDM and across the LAN, you would have to boot a real DOS session (see also Chapter 12, "Virtual Machine Boot" on page 227). You would also have to load the LAN Support Program and the DOS LAN Requestor within such a VMBOOT session. If you do that, your network adapter will be used by that DOS session exclusively and cannot be shared with any other session on this machine.

---

One process (*server process*) creates the pipe and connects to one end of it. Other processes that access the named pipe are called *client processes*; they connect to the other end of the pipe. The server and client processes can then pass data back and forth by reading from and writing to the pipe. The server process controls access to the named pipe.

The client process can be either local or remote. A local client process is one that runs on the same computer system as the server process. A remote client process runs on a different system and communicates with the server process across a local area network (LAN).

The DOS applications running in different VDMs can only work as client processes. The OS/2 application for this kind of IPC has to be the server process. That is because there are no equivalent pipe APIs in DOS to create a named pipe, etc. there are only the standard I/O commands. This means that the DOS client process can read and write from and to the pipe, but it cannot create one. To do this the DOS client has to know the name of the named pipe to be able to use it like it would use a flat file to open it and process the I/O calls.

There is an exercise included in the appendixes that demonstrates a VDM IPC and shows you the sample code (see E.2.5, "Lab Session 5: VDM Interprocess Communications" on page 281 for more information).

## 2.9 Summary

MVDM is composed of four major components:

1. The Virtual DOS Machine Manager is responsible for creating and initializing VDMs.

2. 8086 Emulation provides an emulation of the 8086 hardware environment, supporting actions such as direct hardware manipulation and hardware interrupts.

3. DOS Emulation provides an emulation of the DOS operating system environment, supporting actions such as software interrupts and INT 21h services.

4. Virtual device drivers provide virtualization of hardware devices, allowing these devices to appear to a DOS application as though the application has direct control over the device. Devices may thus be shared by multiple DOS applications and/or protected mode (OS/2) applications.

These components operate in conjunction with the hardware and the OS/2 Version 2.0 operating system kernel to provide support for DOS applications under OS/2 Version 2.0.

MVDM allows OS/2 Version 2.0 to exploit the capabilities of the virtual 8086 mode of the 80386 processor. MVDM provides the capability to run multiple concurrent DOS applications in virtual DOS machines. Since each VDM is a protected mode task, MVDM provides pre-emptive multitasking and full memory protection for DOS applications, protecting other applications in the system and the operating system itself from interference by an ill-behaved application. Executing VDMs in protected mode (as opposed to real mode) also improves the performance of DOS applications, since the processor is never required to switch to real mode.

Each VDM executes in its own 4MB protected mode address space, with the DOS application having access to the lower 1MB of the linear address space. The remainder of the space is occupied by the MVDM code, device drivers, and extended or expanded memory objects. VDMs may run in window or full-screen mode, and a user can dynamically switch between the two.

All documented DOS system interfaces, most direct ROM BIOS interfaces, and memory extenders, such as LIM EMS Version 4.0 and LIMA XMS Version 2.0, are supported by the MVDM architecture. Direct manipulation of common hardware devices is also supported by MVDM. In addition, certain undocumented but commonly used INT 21h services are supported. DOS Emulation provides DOS 5.0 compatible support.

# Chapter 3. 8086 Emulation

A significant capability of the Intel 80386 and 80486 processor families is their ability to emulate the Intel 8086 processor. This emulated state is known as **virtual 8086 (V86) mode**. The Multiple Virtual DOS Machines component of OS/2 Version 2.0 makes use of this mode to provide emulation of a native DOS environment for applications executing in virtual DOS machines. The 8086 processor hardware emulation is provided by the 8086 Emulation component of MVDM. Other aspects of the DOS environment, such as program loading and execution and device driver support, are provided by the DOS Emulation component, which is described in Chapter 4, "MVDM DOS Emulation," and by virtual device drivers, described in Chapter 5, "Device Drivers."

This chapter provides an overview of V86 mode, and describes the operation of the 8086 Emulation component.

## 3.1 Virtual 8086 Mode

In a native 8086 environment, the processor does not constrain an application in any way. The application may access all available memory and execute all processor instructions, since it is running as a single task in an unprotected, real mode environment. Operating systems and applications written for the 8086 (such as DOS) typically take advantage of this freedom to exercise direct control over hardware and system resources.

The 80386 processor exhibits its best performance when running in protected mode. However, in the protected mode environment, applications are restricted to a subset of the system memory and processor instruction set, and real mode applications written for the 8086 processor can violate the protection rules imposed by the processor.

The virtual 8086 mode of the 80386 processor allows an operating system to integrate existing applications, written for the 8086 processor, into the protected mode multitasking environment of the 80386, and to execute such applications concurrently with other 8086 applications and protected mode applications.

V86 mode tasks execute in the 80386 processor at privilege level 3, and are compatible with the virtual memory and paging facilities of the 80386. A V86 mode task may execute most 8086 instructions, including those which reference memory mapped or I/O mapped devices, or which access the 80386 interrupt enable flag. These instructions may be executed by the 80386 processor directly, or the 80386 operating system may trap and emulate such instructions.

Certain 8086 instructions may not be executed in V86 mode; these include instructions which generate interrupts or exceptions, some of which are not valid in a normal protected mode task. However, such instructions may be valid for a V86 mode task. For example, an application running in V86 mode may issue an interrupt 21h operating system call. An 80386 operating system may register an exception handler to trap and emulate such instructions at a higher privilege level.

**29**
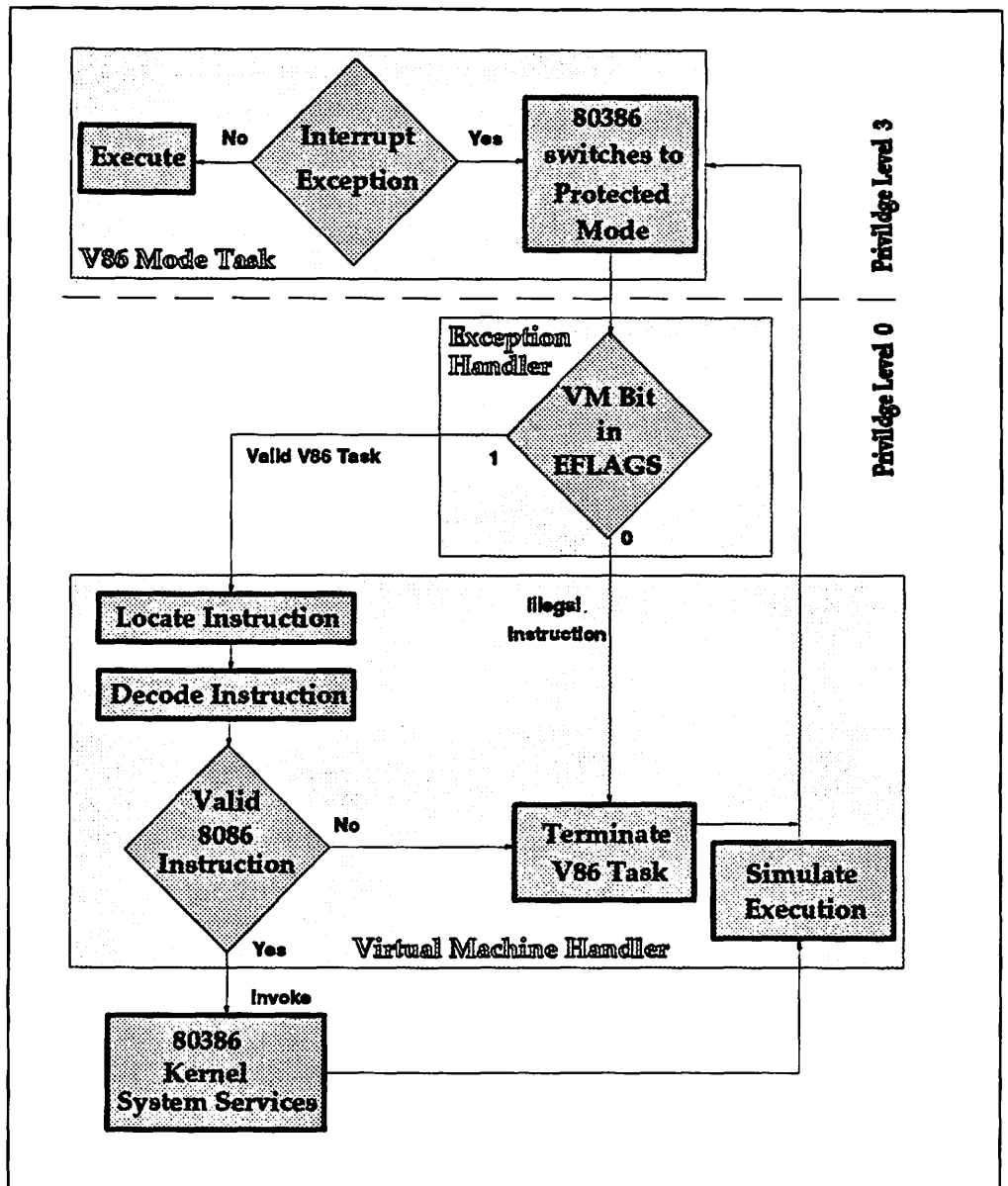
*Figure 12. VDM Exceptions and Interrupt Handling in a V86 Mode Task*

An interrupt or exception in the V86 mode task causes the processor to switch
from V86 mode to protected mode. An exception handler running as a protected
mode task at privilege level 0 is then invoked by the 80386 operating system.
This handler first determines that the task which issued the interrupt or excep-
tion instruction is a valid V86 mode task. It does this by checking the VM bit in
the EFLAGS register. Two possible states are possible:

- *If this bit is set*, the current task is a V86 mode task. The exception handler
  then invokes a *virtual machine monitor*.

  The virtual machine monitor locates the instruction which caused the inter-
  rupt or exception, decodes the instruction and, if it is a valid 8086 instruction,
  simulates its execution by invoking appropriate 80386 operating system ser-
  vices. If the instruction is not valid (for example, a privileged 80386 instruc-
  tion), the virtual machine monitor terminates the V86 mode task.

- *If the bit is not set*, the task has issued an illegal instruction, and is termi-
  nated by the virtual machine monitor.

Once the instruction which caused the interrupt or exception has been proc-
essed, the virtual machine monitor transforms the result into the expected format
for the V86 mode task. It then advances the V86 mode task's EIP (Extended
Instruction Pointer) to the next instruction, and issues an IRET instruction which
causes the processor to switch back to V86 mode and continue execution of the
V86 mode task.

The *8086 Emulation component of MVDM is a virtual machine monitor.*

## 3.2 DOS Software Interrupt Handling

Upon creation of a VDM, the IOPL field in the EFLAGS register within the VDM
process's task state segment is set to 3. This has two major effects:

- It allows the VDM to access the interrupt enable flag (IF), thus permitting
  compatibility with DOS applications which temporarily disable interrupts
  while performing critical operations.

- It means that an interrupt issued from a VDM does not necessarily cause a
  general protection exception; certain interrupt handlers may execute at privi-
  lege level 3 within the VDM.

If the VDM process is running with IOPL < 3, every interrupt causes a general
protection exception; in such cases the operating system would need to
virtualize the interrupt at all times, and to emulate all IOPL-sensitive instructions
(CLI, STI, LOCK, PUSHF, POPF, INT n, and IRET). This would result in increased
mode switching (between V86 and protected mode) and higher interrupt latency,
and would therefore reduce performance.

Thus, under OS/2 Version 2.0, a VDM runs with IOPL=3 for maximum perform-
ance. Interrupts are virtualized and, where possible, handled within the V86
mode task.

## 3.2.1 Virtualizing Interrupts

The behavior of 8086 Emulation in response to an interrupt is dependent upon
the descriptor privilege level (DPL) field for the interrupt handler. This is shown
in Figure 13 on page 32.

When a software interrupt occurs in a DOS application running in a VDM, the
interrupt is vectored to an interrupt handler determined by an entry in the VDM's
Interrupt Vector table. This interrupt handler has its own descriptor privilege
level (DPL). If the DPL is 3, the interrupt handler code is simply executed, and
control returns to the DOS application.

| IOPL < 3 | IOPL = 3 | |
|---|---|---|
| GP fault handler emulates the DOS interrupt and points back to the VDM through the V86 Interrupt Vector Table | Interrupt Descriptor Table (IDT) | |
| | DPL < 3 | DPL = 3 |
| | GP fault handler emulates the DOS interrupt and points back to the VDM through the V86 Interrupt Vector Table | The DOS interrupt vectors through the corresponding IDT entry |

*Figure 13. Descriptor Privilege Levels*

If the DPL of the interrupt handler is less than 3, a general protection exception is generated by the processor and passed to the OS/2 Version 2.0 kernel. The general protection exception handler then determines that the exception occurred from a V86 mode task, and invokes 8086 Emulation to simulate execution of the interrupt handler. Depending upon the type of interrupt, 8086 Emulation may perform the emulation within itself, or call another component of MVDM to handle the emulation.

## 3.2.2 Disabling Interrupts

Disabling interrupts from within a DOS application running in a VDM may cause severe problems. If an error or program loop occurs while interrupts are disabled, the condition cannot be handled correctly and the system may crash.

To prevent a DOS application running in V86 mode from disabling interrupts for an extended period of time, a hardware timer is provided by the 80386 processor known as the **watchdog timer**. Such lengthy disabling may cause unrecoverable system errors. The watchdog timer is programmable and generates non-maskable interrupts (NM) after a specified period of time which allow an operating system to detect an errant 8086 application and terminate it. OS/2 Version 2.0 provides a hardware interrupt manager. This maintains a time counter for every VDM. All interrupts, except NMI, are checked by this hardware interrupt manager, which resets the time counter with every occurrence of an interrupt for the corresponding VDM.

If a counter exceeds a predefined limit (a typical value is 60 milliseconds), interrupts are automatically re-enabled. The Virtual DOS Machine Manager is notified of the ill-behaved application program, and will then terminate the VDM.

## 3.3 I/O Port Trapping

A DOS application running in a VDM may access I/O ports directly using the normal 8086 I/O instructions (such as, IN and OUT). These instructions are not considered IOPL-sensitive and do not normally generate a general protection exception; the operating system checks the I/O privilege map in the VDM's task state segment to determine whether to allow the instruction to execute or to generate a general protection exception. This allows DOS applications to access hardware devices using the normal DOS device drivers from within a VDM.

When access to a device must be shared with other applications, however, a virtual device driver is required, and the VDM may not directly access the I/O port. At initialization time, the virtual device driver issues a call to the **VDHSetIOHookState()** device helper function, which sets the appropriate bit in the I/O privilege map.

When a DOS application subsequently issues an instruction for the I/O port:

1. A general protection exception is generated.

2. The operating system's exception handler routes the exception to 8086 Emulation.

3. 8086 emulation then invokes the virtual device driver.

## 3.4  A20 Line Services (64KB Wraparound)

The region from 1MB to 1MB+64KB is known as the *A20 wrap area*. Due to the segmented scheme for generating 20-bit physical addresses on an 8088, it is possible for a DOS program to generate physical addresses in the range from 1MB to 1MB + 64KB. On an 8088 system, these addresses wrap to the low 64KB of physical memory.

In the 16-bit version of OS/2, OS/2 V1.x, 80286 physical addresses are 24 bits. The *twenty-first* address line of the 80286 is called the *A20 line*, and its setting determines whether real-mode programs wrap low physical memory, or directly access the range from 1MB to 1MB + 64KB. When an 80286 is started, the A20 line is disabled, causing the 80286 to emulate the 8088 environment. When the 80286 is switched to protected mode, the A20 line is enabled, since the protected mode of the 80286 generates 24-bit physical addresses. However, the A20 wrap area can be addressed in real mode in OS/2 V1.x if the A20 line is enabled manually. OS/2 V1.x can thus use the memory in the A20 wrap area for bimodal code (for example, OS/2 V1.x device drivers) by managing the state of the A20 line. When running a DOS application in real mode, OS/2 V1.x disables the A20 line to force the 8088 segment wrapping semantics on DOS applications. When accessing bimodal code in the range from 1MB to 1MB + 64KB in real mode, the OS/2 V1.x kernel enables the A20 line.

In OS/2 V2.0 however, the area between 1MB and 1MB + 65519 bytes, cannot be accessed by a DOS program in V86 mode using 20 address lines (that is lines 0 to 19). For example, addressing 1MB + 1 byte from within a DOS application in V86 mode would access physical memory at address 0; in other words, the memory addressing would wrap around to access the extra byte of memory. This is known as *wraparound*. Under OS/2 Version 2.0, the addressing of memory beyond 1MB would result in an addressing exception because although address line 20 is activated, it is not valid for a V86 mode process. This is shown in Figure 14 on page 34.

In order to avoid addressing exceptions in OS/2 Version 2.0, the wraparound feature must be simulated with **aliased pages**. The 1MB V86 address space requires a page table with 256 entries (256 x 4KB = 1MB). Sixteen additional page table entries are used for the 65519 bytes above 1MB. These 16 entries are identical to the first 16 entries for the 1MB area, thereby mapping the wraparound area to the address range 0 to 65519.
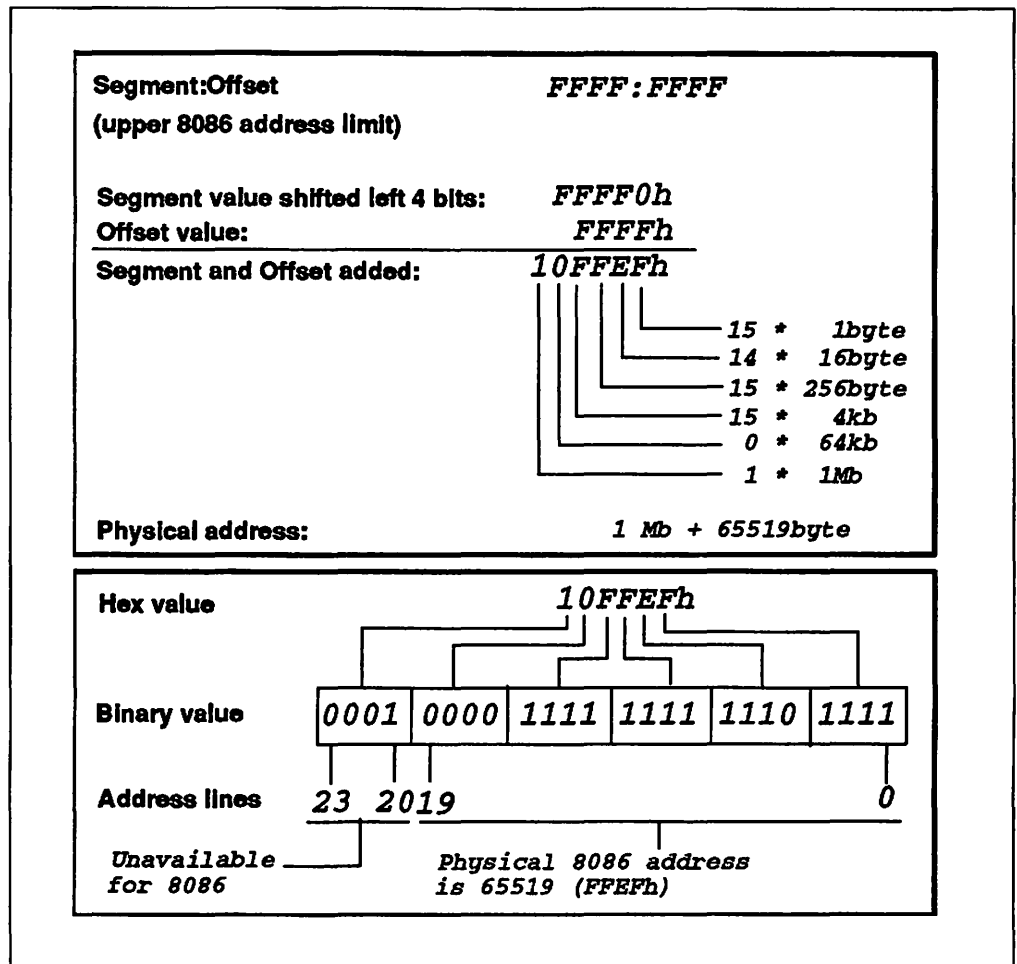
Figure 14. A20 Line Service (64KB Wraparound)

The wraparound feature requires some additional housekeeping by the OS/2 Version 2.0 kernel. When an aliased page is swapped to disk, both page table entries must be flagged as not present.

## 3.5 Summary

The 8086 Emulation component makes use of the virtual 8086 mode of the 80386 processor to provide an emulation of the 8086 hardware environment for DOS applications executing in virtual DOS machines. Application instructions which cause interrupts or which cannot be executed in V86 mode are trapped by the OS/2 Version 2.0 operating system kernel and routed to 8086 Emulation, which may process the instruction within itself or re-route it to the appropriate component of MVDM.

For maximum performance, not all interrupts are trapped and routed in this manner. MVDM makes use of the IOPL field in the VDM process's task state segment, and the Descriptor Privilege Level (DPL) field in the interrupt handler's code segment, to allow certain interrupts to be processed in V86 mode. Only when the interrupt handler must execute at a higher privilege level are the interrupts trapped and routed by the operating system to 8086 Emulation. This improves performance by reducing the number of processor mode-switches required.

8086 Emulation also supports the use of the Intel 8086 64KB wraparound feature, allowing access to the 64KB of memory immediately above the 1MB address line. This capability is used by some DOS software such as the LIMA XMS memory extender, which implements its high memory area (HMA) in this region. More detailed information regarding HMA can be found in Chapter 6, "Memory Extender Support" on page 93.

# Chapter 4. MVDM DOS Emulation

MVDM DOS Emulation provides DOS services to applications running in Multiple Virtual DOS Machines. The environment provided is highly compatible with those same services that are provided to DOS applications running under native DOS 5.0. It addresses only DOS compatibility aspects; processor and other hardware aspects of the DOS environment are addressed by the 8086 Emulation component and virtual device drivers.

This chapter describes the implementation of the DOS Emulation component, and the DOS software services provided to support DOS applications running in virtual DOS machines.

## 4.1 DOS Emulation Overview

DOS Emulation is implemented by running a very small portion of the DOS Emulation kernel in V86 mode and a much larger portion of this code in protected mode outside the VDM. In OS/2 Version 2.0, physical device drivers are loaded above 1MB and only the DOS Emulation kernel resides below 1MB. Any user-installed OS/2 device drivers will not affect the amount of application space available to a DOS application running in a virtual DOS machine. Similarly, adding LAN drivers to the OS/2 configuration to support a network server or redirector does not impact DOS application space, even though DOS applications may make use of these OS/2 network devices. Virtual device drivers are also loaded outside the VDM address space, and therefore do not reduce the amount of memory available to a DOS application.

In this way, the MVDM architecture makes the maximum amount of memory available to DOS applications. In fact, up to 630KB are free for use by DOS applications; this represents an increase of approximately 100KB over memory available to the single DOS session that was available under OS/2 Version 1.3. Note that this amount may be reduced if DOS device drivers and/or TSR programs are loaded in a VDM. The following DOS features are implemented by the DOS Emulation:

- DOS console device driver
- DOS device driver loading/support
- DOS program loading and execution
- DOS FCB I/O support
- DOS memory manager
- DOS NLS support
- DOS PDB process support
- VDM entering and exiting kernel mode support
- Special DOS compatibility mechanisms.

DOS Emulation supports all documented DOS interrupts and features. In addition, some undocumented aspects of these functions (especially INT 21h) are supported. This support is incorporated in the DOS Emulation component because a large number of popular DOS applications rely upon these interfaces.

The following list shows the *documented* DOS system interrupt services which are implemented by DOS Emulation and which are available to DOS applications running in VDMs:

| INT 20h | Program terminate interface |
| --- | --- |
| INT 21h | System call interface |
| INT 22h/INT 23h | Terminate and Ctrl-Break address interfaces |
| INT 24h | Critical error handler interface |
| INT 25h/INT 26h | Absolute disk read/write interfaces |
| INT 27h | Terminate and stay resident (TSR) interface |
| INT 28h | Idle loop interface |
| INT 2Fh | Print spool interface. |

Other DOS compatibility functions are implemented by the 8086 Emulation component of MVDM, and by virtual device drivers. The functions provided by these components are explained in Chapter 3, "8086 Emulation" and in Chapter 5, "Device Drivers," respectively.

## 4.2 DOS Emulation Implementation

A primary design goal for MVDM and the DOS Emulation component was to provide a DOS compatible environment in which a VDM could not negatively affect other VDMs or OS/2 protected mode processes, while at the same time providing the greatest amount of free memory for DOS applications. This goal was achieved by allowing as much of the DOS Emulation code as possible to execute in protected mode, outside the VDM address space. This provides improved protection, leaves more memory available for DOS application use, and enhances overall performance.

## 4.2.1 Initialization and VDM Creation

Initialization of the DOS Emulation component is divided into two stages. The first occurs during OS/2 system initialization. The second stage occurs during creation of each virtual DOS machine.

### 4.2.1.1 OS/2 Initialization

DOS Emulation is involved in OS/2 system initialization because it requires access to information contained in CONFIG.SYS. As the OS/2 initialization procedure processes the CONFIG.SYS file, it records parameters relevant to DOS Emulation. These parameters include those specified in the FCBS and RMSIZE statements, and any DEVICE statements which specify DOS device drivers. These parameters become the default DOS settings for all VDMs.

**Note:** Virtual device drivers are not loaded or initialized at this stage. Initialization of DOS settings occurs prior to loading virtual device drivers, since these default settings may be required by the device drivers during VDM initialization (creation).

### 4.2.1.2 VDM Creation Stage

Upon creation of a VDM, the Virtual DOS Machine Manager calls the creation-time initialization routines for virtual device drivers and then passes control to the DOS Emulation kernel. At this point, the V86 memory organization appears as shown in Figure 15 on page 39.
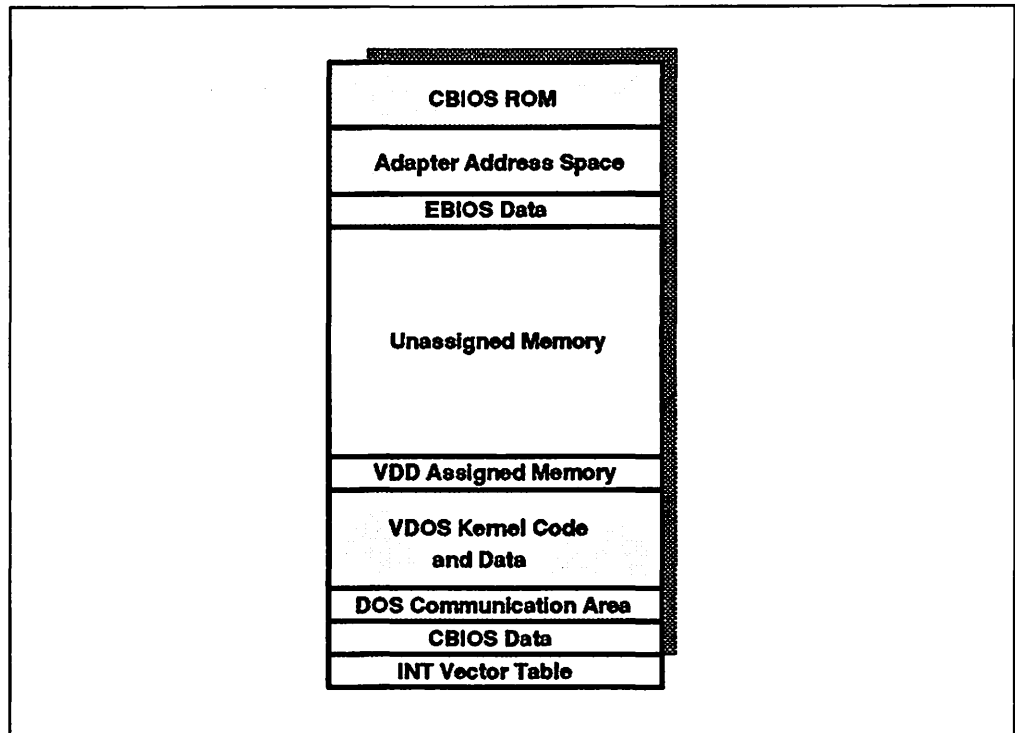
*Figure 15. V86 Memory Map Prior to DOS Emulation Initialization*

During VDM creation, DOS Emulation performs the following steps:

1. *Initialize VDM-Related Kernel Structures*

   Certain structures in the OS/2 Version 2.0 kernel are initialized in preparation for processing VDM requests. The System File Table (SFT) structures, for example, which are used for FCB I/O, are allocated and initialized.

2. *Load DOS Emulation Kernel (DOSKRNL.COM)*

   The portion of the DOS Emulation kernel which runs in V86 virtual memory is loaded at the high end of the VDM memory address space.

3. *Start Virtual Processor Mode*

   The protected mode initialization routine returns control to the Virtual DOS Machine Manager, which then invokes the initialization code within the V86 mode DOS Emulation kernel. This represents the first transition to V86 mode; at this point, memory is organized as in Figure 16 on page 40.

```
        CBIOS ROM
   Adapter Address Space
        EBIOS Data
       VDOS Init Code

       DOS Memory Area

      VDD Assigned Memory
       VDOS Kernel Code
          and Data
    DOS Communication Area
        CBIOS Data
       INT Vector Table
```
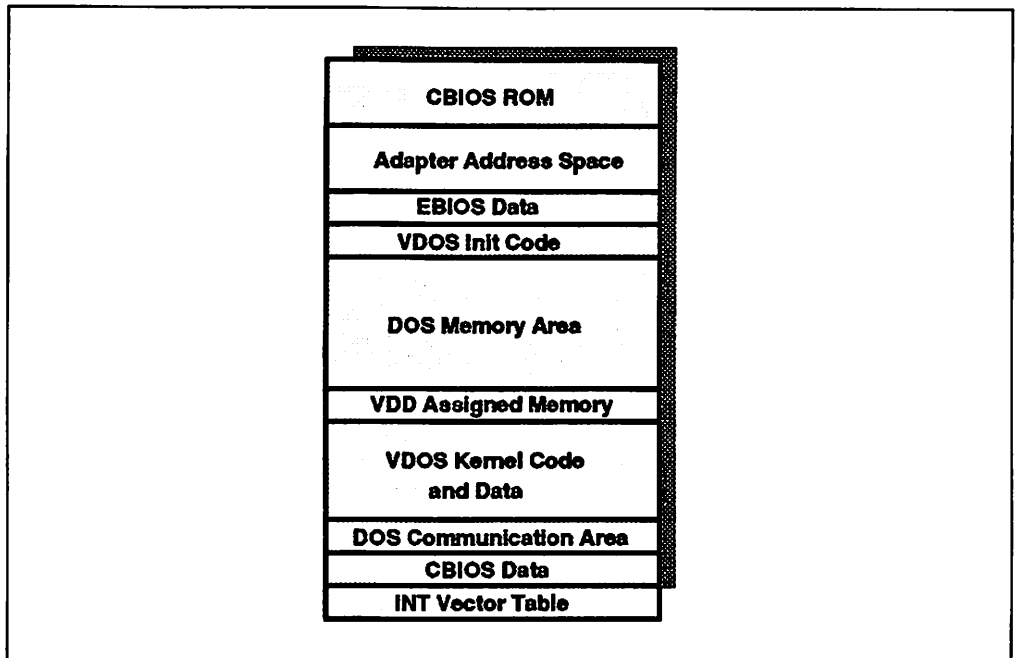
*Figure 16. V86 Memory Map at Initial V86 Mode Entry. This diagram shows the VDM's memory map when the V86 mode DOS Emulation kernel is first invoked.*

4. *Open Standard Devices*

   The initial five file handles (stdin, stdout, stderr, stdaux, and stdprn) are opened.

5. *Initialize Virtual Device Driver DOS Device Driver "stubs"*

   Some virtual device drivers provide a DOS device driver "stub"; these stubs are inserted into the V86 address space prior to initialization of DOS Emulation. As such, this step involves calling the inserted initialization code and linking the devices into the device chain. Unlike real DOS device drivers, however, the return from the initialization does not allow reducing the size of the driver code. See Chapter 5, "Device Drivers" for further information.

6. *Initialize DOS Device Drivers*

   Each DOS device driver specified in the CONFIG.SYS file is loaded into the VDM and initialized. Any VDM-specific DOS device drivers passed in the DosCreateProcess() function call, or configured via the DOS Settings option *DOS Device Drivers*, are then loaded and initialized. This is performed one device driver at a time to allow the device drivers to consume only the memory that they require or to de-install themselves entirely. As each device is initialized, it is added to the chain of devices in the VDM.

   During initialization, device drivers may issue a limited set of INT 21h system calls (functions 01h through 0Ch, 25h, 30h, and 35h). This restores functionality that had been removed from previous versions of OS/2.

   **Note:** The result is undefined when a DOS device driver issues an INT 21h system call other than those mentioned above. This is consistent and compatible with DOS. Issuing an unsupported INT 21h system call will crash the VDM.

   After all device drivers have been initialized, the initialization code is discarded.

7. *Load and Execute DOS Shell*

The shell specified in the SHELL command in CONFIG.SYS is loaded, the initialization code in the V86 address space is discarded, and control is passed to the shell program. The SHELL specified in CONFIG.SYS can be overridden in the **DosCreateProcess()** function call. This is a useful feature if, for example, a software developer wishes to allow different versions of COMMAND.COM, for such reasons as alternative national language support.

Upon invocation of the shell program, the VDM's memory map is organized as shown in Figure 17.
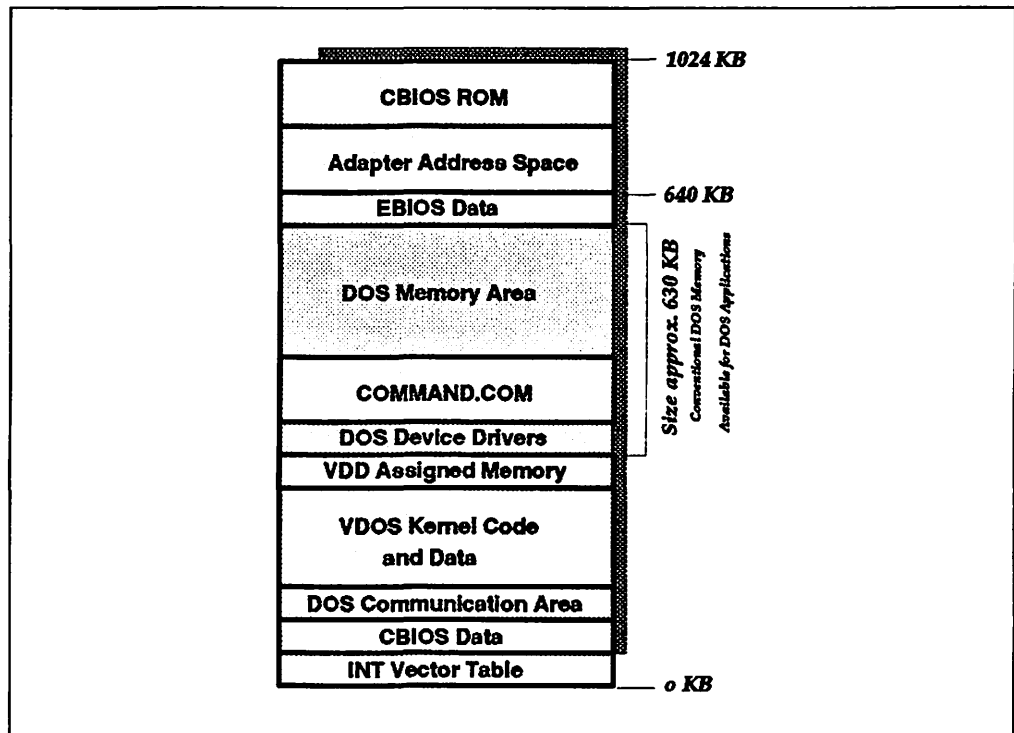


*Figure 17. V86 Memory Map after Initialization. This diagram shows the VDM's memory map after initialization of the DOS environment and prior to loading a DOS application.*

Before passing control, the **Program Descriptor Block** (PDB) of the shell is initialized with the command line parameters as specified in CONFIG.SYS. As an extension to the native DOS environment, an additional string is appended after these parameters, separated from the command line string by a NULL byte. This string specifies the drive and directory of the virtual DOS environment after the shell completes its initialization. This extension provides a default working drive and directory for real mode applications, as is provided for protect mode applications using the Presentation Manager.

## 4.2.2 Requesting System Services

As previously mentioned, MVDM isolates some VDM-specific code and places it into the DOS Emulation kernel in the VDM's address space. This code provides those DOS services which can be supported in V86 mode, such as memory management services. Other services which require protected mode execution are provided by additional code which runs in protected mode.

When the DOS Emulation kernel requires protected mode services, it specifies the kernel procedure via an index, and executes a processor instruction which causes a general protection exception. The OS/2 Version 2.0 general protection

exception handler traps the exception and invokes the 8086 Emulation component, which in turn transfers control to the specified kernel procedure. This is explained in detail in Chapter 3, "8086 Emulation."

## 4.2.3 System Service Call Behavior

DOS system services provided within VDMs are generally compatible with their implementation under DOS 5.0. Some differences do exist, however, and are described below.

**INT 20h** This service forwards the request to the INT 21h function 00h service in order to abort the application.

**INT 21h** All INT 21h services provided in DOS 5.0 are also provided in VDMs. However, the internal behavior and error processing of some functions may be different. Where such changes are significant, they are listed here:

- **Function 00h (ABORT)**

  If the CS register does not reference the current PDB, the VDM is terminated. In certain previous DOS versions, the effect of such a call was undefined.

- **FCB Functions**

  Although OS/2 only provides file I/O access externally through file handles, it supports these handles internally through the System File Table (SFT). MVDM allows file handles to be bypassed and SFT entries to be manipulated directly using a special set of reserved SFT entries, in a manner similar to previous versions of OS/2. However, since multiple VDMs are supported, these SFT entries are allocated dynamically upon creation of a VDM

  FCB functions may now be called from device drivers during initialization; this functionality was not available in previous versions of OS/2.

- **Function 38h (International)**

- **Function 44h (IOCTL)**

  IOCTL requests which are destined for device drivers within the VDM are processed internally. IOCTL requests which are destined for OS/2 device drivers, however, are treated specially by their respective device drivers. Such requests may contain pointers to data within the VDM. In these cases, it is the responsibility of the OS/2 device driver to perform the necessary translation from V86 virtual addresses into addresses that are meaningful to the device driver.

- **Function 5Dh (Internal)**

  - *Subfunction 0Ah (Set Extended Error Information)*

    This subfunction allows the calling program to set the register values that are returned from a call to function 59h. This provides functionality that was not present in previous versions of OS/2.

    This subfunction allows terminate and stay resident programs (TSRs) to save and restore extended error information when they are invoked.

- **Function 66h - Get/Set Code Page**

- **Function 67h - Set Handle Count**

  This function restricts the maximum number of open device handles to 254, including the four standard devices.

### INT 25h/INT 26h

Absolute Disk Read/Write

The read function operates in the same way as in a DOS 5.0 system. The write function however, is restricted to removable media only, and reports a hard error on non-removable media.

## 4.2.4 System Callback Procedures

The system callback procedures are "hooks" into DOS (in the case of VDMs, into the DOS Emulation kernel) which allow application programs to change the default processing action taken for certain system events. These hooks are specified in the V86 mode interrupt vector table as trap service routines. By default, the vectors reference a single IRET instruction if an application does not register its own hook procedure.

The vectors used to specify the callback routines are:

### INT 22h - Terminate Address

The DOS Emulation kernel stores the termination return address at this vector location.

### INT 23h - Control-C Exit Address

The method used to invoke this callback procedure works the same way as in a DOS 5.0 system.

### INT 24h - Critical Error Handler

Hard errors are normally generated from within the OS/2 kernel. When such an error is detected, the file system checks to see if the requesting task is a VDM. If so, the error indication is returned to the protected mode portion of DOS Emulation, which determines if the INT 24h vector has been changed.

If it has not been changed, the normal OS/2 hard error monitor is called to display the hard error information and to prompt for a reply. If it has been changed, the specified critical error handler is invoked.

### INT 28h - Idle Loop

The method used to call the INT 28h callback routine is similar to that used in DOS 5.0, but takes into account the fact that the DOS session is running in a multitasking environment.

The OS/2 scheduler maintains a flag in the VDM address space which indicates if another process in the system is ready to run. While in the idle loop, the DOS Emulation kernel repeatedly examines this flag. If no other OS/2 tasks are ready, the loop proceeds as normal. If the flag indicates that other tasks are waiting, DOS Emulation yields control to the operating system, which dispatches the waiting task.

In all other respects, callback procedures operate under MVDM in an identical manner to that experienced under DOS 5.0.

## 4.2.5 VDM Termination

When a VDM is terminated, DOS Emulation closes all handles that have been assigned to all PDB processes within the VDM. All resources associated with open FCBs are also closed.

Since the VDM may have been terminated due to an error in a DOS application within the VDM, no data within the VDM is relied upon when closing resources and cleaning up. DOS Emulation explicitly closes both the files and the OS/2 devices opened by the VDM.

## 4.2.6 Standard Devices

As in DOS, DOS Emulation includes device drivers for the three "standard" devices, CON, AUX, and PRN. While DOS packages these device drivers in IBMBIO.COM (IO.SYS), DOS Emulation links these into the DOS Emulation kernel (DOSKRNL) to avoid the need for another file.

Unlike DOS, the AUX and PRN drivers do not include support for COM1, COM2, LPT1, LPT2, and LPT3. These latter devices are supported directly by the OS/2 asynchronous (COM.SYS) and printer (PRINT0n.SYS) device drivers.

This approach allows the CON, AUX, and PRN drivers to behave in a highly compatible manner. These drivers issue ROM BIOS calls (INT 10, 14, and 17, respectively) in order to perform their required tasks. At the same time, using the OS/2 device drivers directly for the numbered I/O devices provides higher performance than through the ROM BIOS interfaces, and allows a numbered I/O device to be easily redirected to a remote device on a network, using the underlying OS/2 mechanisms. Hence there is no INT 17 issued when printing on the numbered I/O devices (for example, LPT1, LPT2, etc.) as long as there is only the virtual printer device driver VLPT.SYS installed as the device driver for LPTn devices. If INT 17 is required, for example by a DOS TSR (Terminate and Stay Resident) that intercepts INT 17, the LPTDD.SYS device driver must be installed as well. You can find LPTDD.SYS in the subdirectory \os2\mdos.

```
┌─ Note ──────────────────────────────────────────────────────┐
│                                                              │
│ Installing LPTDD.SYS will cause printing from a VDM to slow down. │
│                                                              │
│ Remember that only PRN redirection is supported. This is achieved by the │
│ virtual printer device driver VLPT.SYS, which routes INT 17 and direct hard- │
│ ware printing to LPT1, LPT2, or LPT3 using the OS/2 file system. This is │
│ explained in more detail in Chapter 5, "Device Drivers." │
│                                                              │
└──────────────────────────────────────────────────────────────┘
```

## 4.3 Maximizing VDM Memory

The OS/2 Version 2.0 CONFIG.SYS file specifies the operating system configuration, and installs device drivers and other memory-resident programs. The OS/2 AUTOEXEC.BAT file is specific to the functioning of the VDM environment. More base memory can be made available to programs running in a VDM by removing unnecessary commands from these files.

## 4.3.1 CONFIG.SYS

Virtual device drivers utilized by VDMs consume very little or no memory below the 640KB limit. These device drivers reside outside the V86 mode address space. However, a user may install device drivers that are required by and are specific to certain applications which will run in a VDM. If the commands to load these device drivers (or other memory-resident programs) are added to CONFIG.SYS, these device drivers (or programs) will be loaded into *every* VDM when it is created, and will reduce the amount of conventional memory available to DOS applications in every VDM. To ensure the maximum amount of memory is available in each VDM, it is recommended that:

- DOS device drivers specific to a particular DOS application should be loaded via the *DOS_DEVICE* option of DOS Settings. DEVICE= statements for DOS device drivers should be eliminated from CONFIG.SYS unless the device driver is required for *every* VDM.

- The number of buffers specified in the *Buffers* command in CONFIG.SYS should be minimized. Each buffer consumes about 500 bytes. Be careful to not reduce this number too much because some programs might not run properly if there are too few buffers. The default number of buffers is 30; the number should *not* be reduced to fewer than 10 or 15 buffers.

- If CONFIG.SYS includes the LASTDRIVE command, this should be set to a letter such as J or K, rather than Z. Each additional drive uses about 100 bytes.

- If the CONFIG.SYS file contains an FCBS command, set FCBS to 1.

The order of the DEVICE and DEVICEHIGH commands in CONFIG.SYS is important since it can affect both the efficient use of memory and the proper operation of the various programs started from CONFIG.SYS.

The CONFIG.SYS statements and options related to VDM operation, and which can be selected during installation, are shown below:

- Load the DOS Command Interpreter and load it resident into memory
  - SHELL=C:\OS2\MDOS\COMMAND.COM C:\OS2\MDOS /P

- Where to load DOS in memory
  - DOS=HIGH, UMB

- Optional: extended keyboard and display features
  - DEVICE=C:\OS2\MDOS\ANSI.SYS

- Expanded Memory Manager
  - DEVICE=C:\OS2\MDOS\VEMM.SYS

- Extended Memory Manager
  - DEVICE=C:\OS2\MDOS\VXMS.SYS /UMB

- Mouse support
  - DEVICE=C:\OS2\MDOS\VMOUSE.SYS

- DOS graphics support
  - DEVICE=C:\OS2\MDOS\Vxxx.SYS

  Where xxx depends on video adapter:

  - VCGA.SYS     (if CGA or EGA w/ 64KB)

- VEGA.SYS    (if EGA w/ 128KB)
- VVGA.SYS    (if VGA or 8514)
- V8514.SYS    (if 8514)

- Direct I/O serial communication support

  - DEVICE = C:\OS2\MDOS\VCOM.SYS

The following list shows the order in which device drivers should be specified in CONFIG.SYS:

1. VEMM.SYS

2. Any device drivers that use expanded memory

3. VXMS.SYS

4. Any device drivers that use extended memory

5. Any device drivers that use upper memory blocks.

This is the optimal order in which the CONFIG.SYS file should start device drivers. Whether these statements are included at all depends on the amount of memory installed, the hardware configuration, and on the DOS applications to be run.

## 4.3.2 AUTOEXEC.BAT

AUTOEXEC.BAT is specific to the virtual DOS machine environment and has no effect on the OS/2 Version 2.0 operating system. The AUTOEXEC.BAT file starts memory-resident programs, such as network programs, and sets up environment variables. In addition, the AUTOEXEC.BAT file may also define the command prompt.

The default AUTOEXEC.BAT file for all VDMs in OS/2 Version 2.0 is shown in Figure 18.

```
PATH C:\OS2;C:\OS2\MDOS;C:\;
LOADHIGH APPEND C:\OS2;C:\OS2\SYSTEM
PROMPT $I$P$G
```

*Figure 18. Default AUTOEXEC.BAT File*

In the above AUTOEXEC.BAT file, the LOADHIGH command loads the APPEND TSR into the High Memory Area, thus making available more memory to all DOS applications running in every VDM. This example assumes that the function performed by the SET COMSPEC command, which is often found in AUTOEXEC.BAT, has been moved to CONFIG.SYS.

**Note:** In order to maximize the amount of base memory available to applications, any unnecessary commands should be removed from AUTOEXEC.BAT. Commands should only be included in AUTOEXEC.BAT if they are required for every VDM. Commands which are required only for a specific DOS application to be run in a VDM should be placed into a batch file. This batch file should be explicitly entered into the *path and file name* field of the "parameters field" in the DOS Settings notebook on the "program page" for that specific application.

## 4.4 Command Compatibility

For maximum compatibility, OS/2 Version 2.0 now supports commands previously unique to DOS, including new commands and enhancements recently introduced with IBM DOS 5.0. Commands new to OS/2 Version 2.0 are:

**MEM**            Display memory availability and contents
**FC**            Intelligent file compare utility
**DOSKEY**            Command-line enhancer
**DEBUG**            Program debugger
**UNDELETE**            Recover erased files
**QBASIC**            Enhanced BASIC Interpreter

UNDELETE runs in both an OS/2 session and in a VDM. The others run in DOS MVDM mode only.

In addition, several enhancements introduced in DOS 5.0 are also supported:

**DIR**            Several new output formatting options
**ATTRIB**            Now supports *hidden* and *system* file attributes
**RESTORE**            Option to list backup diskette's contents
**FIND**            Case-insensitive search option

These command enhancements are available in both MVDM and OS/2 protect mode.

**Note:** The following DOS 5.0 enhancements are not provided in OS/2 Version 2.0:

- UNFORMAT command

- Quick FORMAT

A brief summary follows for those unfamiliar with DOS 5.0 enhancements.

## 4.4.1 MEM

MEM displays the amount of used and free memory in the DOS environment. It lists information about allocated memory areas, free memory areas, and programs that are currently loaded into memory.

The format of the MEM command is:

```
mem [/program | /debug | /classify]
```

where:

**/p(rogram)**            displays the status of programs that are currently loaded into memory.
**/d(ebug)**            displays the status of currently loaded programs, internal drivers and other programming information.
**/c(lassify)**            displays the status of programs loaded into conventional memory and the upper memory area.

MEM only runs in a DOS session.

## 4.4.2 FC (File Compare)

FC compares two files and displays their differences. The format of the FC command to compare ASCII (text) files is:

FC [/A][/C][/L][/LBn][/N][/T][/W][/n] <file1> <file2>

FC /B [drive1:][path1] <file1> <file2>

where:

| | |
|---|---|
| /A | Displays only first and last lines for each set of differences |
| /B | Performs a binary comparison |
| /C | Disregards the case of letters |
| /L | Compares files as ASCII text |
| /LBn | Sets the maximum consecutive mismatches to "n" lines |
| /N | Displays the line numbers on an ASCII comparison |
| /T | Does not expand tabs to spaces |
| /W | Compresses white space (tabs and spaces) for comparison |
| /n | Specifies the number of consecutive lines that must match after a mismatch |

Unlike the COMP command, FC attempts to resynchronize after a mismatch. It recognizes inserted or deleted character sequences, then resumes the comparison.

## 4.4.3 DOSKEY

DOSKEY enhances command line editing, recalls DOS commands, and creates macros. The format of the DOSKEY command is:

doskey [/reinstall] [/bufsize=size] [/macros] [/history]
       [/insert | /overstrike] [macroname=[text]]

where:

| | |
|---|---|
| /r(einstall) | installs a new copy of the DOSKEY program |
| /b(ufsize)=size | size of buffer (in bytes) to store commands and macros |
| /m(acros) | displays list of all macros |
| /h(istory) | displays list of all commands stored in memory |
| /i(nsert)\|/o(verstrike) | selects insert or overtype mode |

DOSKEY stacks and recalls commands from a LIFO buffer using the up and down cursor keys. The command line can be intuitively edited via left and right cursor keys, with insert and non-destructive backspace. DOSKEY provides similar command line editing to that of an OS/2 prompt with KEYS=ON specified.

## 4.4.4 DEBUG

DEBUG is used to assist in testing and debugging of executable files. The format of the DEBUG command is:

debug <filename> [parameters]

where <filename> is the file to test.

parameters are any command line parameters to be passed to the program being debugged, not to DEBUG itself.

DEBUG allows the user to examine memory and registers, assemble and disassemble programs, set breakpoints, and step through programs one instruction at a time.

**Note:** DEBUG supports 8086 and 8087 instructions only. It will not assemble or disassemble 80286/7 or later opcodes, and can only access the low 16 bits of 32 bit registers.

## 4.4.5 UNDELETE

The UNDELETE utility recovers files that have been deleted or erased. When the DEL or ERASE command is issued from any session type, the file is moved to a specified hidden directory, along with details of where the file originated. If no space is available, the oldest files are automatically purged to provide more space.

The DELDIR environment variable is used to specify the name and maximum size of the directories where files targeted for deletion are to be stored. Normally this will be set in CONFIG.SYS, and takes the form:

```
SET DELDIR = drive:\path, maxsize [;drive:\path, maxsize]
```

The string is composed of a directory path specifier and maximum size value for each supported logical drive. These parameters are separated by commas. The definitions for each drive are separated by semicolons.

The path portion of the string consists of a drive letter and the fully qualified path of the directory that will be used for storing deleted files.

The maxsize portion of the string defines the maximum size of the storage directory, in kilobytes.

In order to keep track of deleted files, the system also maintains a control file in each storage directory.

When UNDELETE is specified and the file is still recoverable, it is restored to its specified path. If the file already exists, the user is prompted to rename the file, or to skip the current entry.

Space occupied by recoverable files is included in DIR and CHKDSK output reports.

The format of the UNDELETE command is :

```
undelete <filename> [/list | /all] [/s] [/force]
```

where:

**<filename>** is the path and name of the file to recover or purge. It may be wildcarded.

**/l(ist)** lists deleted files that are available to be recovered without recovering the files.

**/a(ll)** recovers all deleted files if they are still present without prompting for confirmation on each file.

**/s** include all files in the specified directory and all subdirectories

**/f(orce)** removes files so they cannot be recovered.

UNDELETE runs in both protect mode and in a DOS VDM.

## 4.4.6 DIR

DIR lists files and subdirectories: New options on the DIR command are:

**/a** *attribute* Lists only those files with the chosen attribute:

| | |
|---|---|
| **a** | files not yet backed up |
| **h** | hidden files |
| **r** | read-only files |
| **s** | system files |
| **d** | directories |

*attribute* may be preceded by a minus sign to select items which do **not** have that attribute.

**/o** *sortorder* Lists items in the chosen order:

| | |
|---|---|
| **n** | sorted by name |
| **e** | sorted by extension |
| **d** | sorted by date |
| **s** | sorted by size |
| **g** | directories grouped before files |

*sortorder* may be preceded by a minus sign to reverse the output order.

| | |
|---|---|
| **/b** | Displays "bare" file information (no date or size) |
| **/f** | Displays fully qualified file and directory names. |
| **/l** | Displays output in lowercase letters. |
| **/n** | Displays the listing in "HPFS" style format. |
| **/s** | Includes all subdirectories in the search |

Other enhancements to the DIR command include:

- The total byte count of matched files is given
- /p (pause) repeats the directory name after each screen is full
- /w (wide) distinguishes directory entries from file entries via square brackets.

## 4.4.7 ATTRIB

ATTRIB now supports manipulation of "hidden" and "system" file attributes.

| | |
|---|---|
| **attrib +s <file>** | sets the system attribute. |
| **attrib -s <file>** | clears the system attribute. |
| **attrib +h <file>** | sets the hidden attribute. |
| **attrib -h <file>** | clears the hidden attribute. |

ATTRIB may also be used to set or clear the "Archive" and "Read-only" attributes as before.

## 4.4.8 RESTORE

A new parameter "/d" displays a list of files on the backup diskette which match the file's specified filename, without restoring any files.

## 4.4.9 FIND

A new parameter "/I" will ignore the case of characters when searching for the string.

## 4.5 Summary

The DOS Emulation component provides an emulation of the DOS operating system environment, including DOS features and system services, for applications running in virtual DOS machines. DOS Emulation uses the parameters specified in CONFIG.SYS and virtual device drivers to create an application-compatible DOS environment within the VDM address space.

During execution, DOS Emulation receives requests from DOS applications. Depending upon the type of request, it either processes the request itself or causes a general protection exception which results in the request being routed by the operating system kernel to the 8086 Emulation component, which processes the request at a higher privilege level.

DOS Emulation also allows DOS applications to hook interrupts, in order to modify the default behavior in response to particular events. This is achieved by providing a virtual interrupt vector table within the V86 mode address space, and routing interrupts through this table.

At VDM termination time, DOS Emulation is responsible for closing all files opened by the VDM, and cleaning up the environment to ensure that no devices are still held or memory objects allocated after termination.

DOS Emulation also provides support for applications which access the standard devices CON, AUX, and PRN. These devices are emulated within DOS Emulation itself, which processes the interrupts for these devices and returns the results to the DOS application.

# Chapter 5. Device Drivers

In order to provide the maximum level of hardware independence for the OS/2 Version 2.0 operating system, device driver are used to communicate with hardware devices. This concept is not new, and has been implemented in previous versions of OS/2 and in the DOS operating system. However, the implementation of device drivers under OS/2 Version 2.0 differ in several significant ways from that seen in previous versions. This chapter describes the implementation of device drivers under OS/2 Version 2.0.

## 5.1 Device Driver Architecture

The architecture and structure of a device driver differs considerably, depending on whether the device driver is physical or virtual.

OS/2 Version 2.0 makes use of two distinct types of device driver to communicate between the operating system and hardware devices:

- **Physical device drivers** are considered as *true* device drivers in the sense that they have a unique and rigid file structure, and interface directly with the hardware. They operate in protected mode, and are accessed by protected mode processes and by virtual device drivers.

- **Virtual device drivers** are essentially a dynamic link library conforming to the EXE32 Load Module format, and generally do not interface directly with hardware devices. Instead, virtual device drivers are responsible for presenting a virtual copy of a hardware resource to DOS applications running in virtual DOS machines, and for coordinating physical access to that resource. DOS applications typically address hardware devices directly using interrupts; the virtual device driver allows the VDM environment to appear to the DOS application as though the application had direct control over the hardware. Virtual device drivers include a stub which executes in V86 mode within each VDM, while the main portion of the virtual device driver executes in protected mode.

The relationship between applications, physical and virtual device drivers, and hardware devices is shown in Figure 19 on page 54.
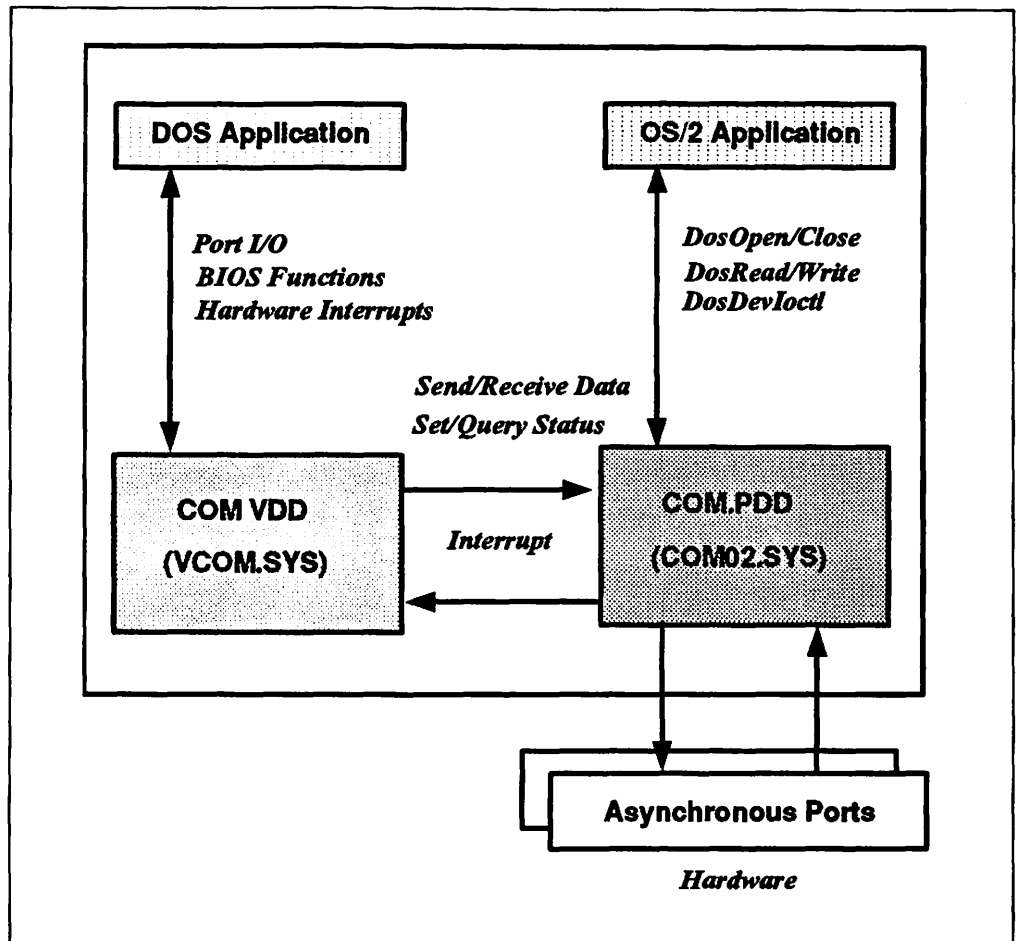
*Figure 19. Physical and Virtual Device Drivers under OS/2 Version 2.0. This example shows the asynchronous communications port.*

## 5.2  Physical Device Drivers

The concept of a device driver is not new to OS/2 Version 2.0; previous versions of OS/2 and DOS have employed device drivers to communicate with hardware devices. Under previous versions of OS/2, however, many device drivers were required to run in both real mode and protected mode in order to accommodate the requirements of applications running in the DOS Compatibility Box. **This** complicated the design of device drivers under previous versions of OS/2, since they were required to be written in a bi-modal manner. Figure 20 on page 55 shows the structure of bi-modal OS/2 device drivers.
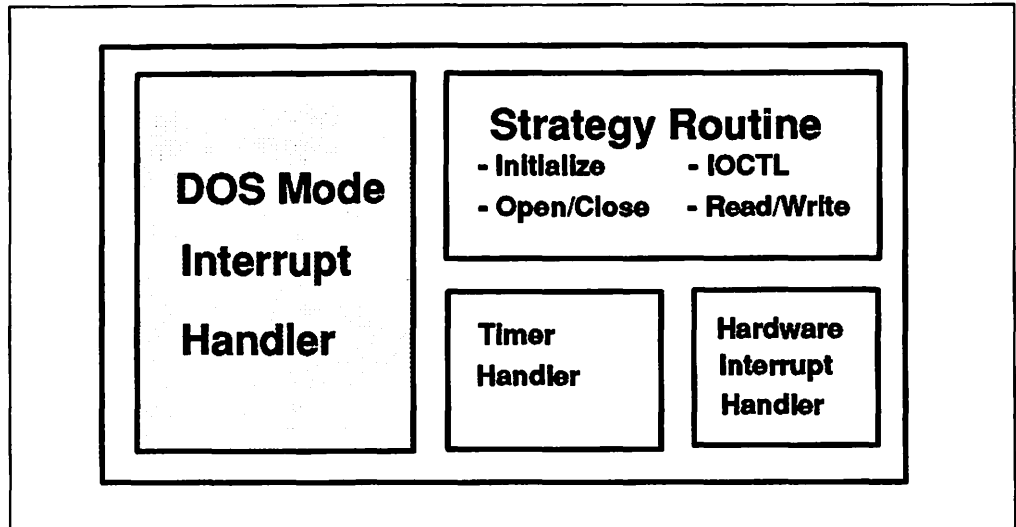
*Figure 20. Structure of Bi-Modal Device Drivers in OS/2 V1.x*

MVDM removes the need for real mode device drivers, since DOS applications run in virtual DOS machines, where each VDM is a protected mode task. Real mode device interrupts issued by DOS applications are trapped by the MVDM kernel and routed to device drivers which execute in protected mode. Hence device drivers for OS/2 Version 2.0 need not include real mode sections, and existing device drivers may be updated to remove the real mode components.

Physical device drivers communicate directly with hardware devices, and are installed at system initialization time using DEVICE= statements in CONFIG.SYS, as shown in Figure 21.

```
DEVICE=C:\OS2\MOUSE.SYS TYPE=PDIMOU$          (Mouse PDD)
DEVICE=C:\OS2\COM.SYS                    (COM port PDD)
```

*Figure 21. Physical Device Driver Statements in CONFIG.SYS*

The advantage of physical device drivers, as with device drivers in previous versions of OS/2, is that the operating system itself need not be changed if a new hardware device or adapter is installed. The corresponding device driver may be installed with the device, and used by applications which require access to the device. The major enhancement in Version 2.0 over previous versions of OS/2 lies in the removal of the real mode component, which simplifies device driver development and improves performance by avoiding the need for processor mode switching.

## 5.3 Virtual Device Drivers

As mentioned previously, DOS applications typically address hardware devices directly using interrupts. This is permissible in a single tasking DOS environment, since it can be safely assumed that the application has complete and exclusive control over the hardware. However, in the OS/2 Version 2.0 environment where multiple applications may be executing concurrently in virtual DOS machines, it is clearly not allowable, since applications could interfere with one another to the detriment of application and system integrity.

Multiple DOS applications accessing the same hardware devices from within VDMs require those hardware devices to be *virtualized*; virtualization implies separate simulation of the physical hardware (I/O ports, device memory, and ROM BIOS) for each virtual DOS machine. This virtualization is accomplished by installable virtual device drivers, which in turn communicate with physical device drivers to address hardware devices.

The following virtual device drivers are provided with the OS/2 Version 2.0 operating system:

| VDD | Description |
|---|---|
| **VBIOS** | ROM BIOS support (1) |
| **VCMOS** | CMOS data area + Real Time Clock support (1) |
| **VDMA** | Direct Memory Access (1) |
| **VDSK** | Disk, only for INT 13 copy-protection (1) |
| **VFLPY** | Floppy Disk interface (1) |
| **VKBD** | Keyboard (1) |
| **VLPT** | Parallel Port Printer (1) |
| **VNPX** | Numeric Coprocessor Extension (80387) (1) |
| **VPIC** | Programmable Interrupt Controller (1) |
| **VTIMER** | Timer (1) |
| **VCOM** | Asynchronous communication ports |
| **VDPMI** | DOS Protected Mode Interface |
| **VDPX** | DOS Protected Mode Extender |
| **VXMS** | Extended Memory Support |
| **VEMM** | Expanded Memory Support |
| **VWIN** | WIN-OS/2 windows |
| **VMOUSE** | Mouse |
| **VCDROM** | CD-ROM support |
| **VVIDEO** | Video (VCGA, VEGA, VMONO, VVGA, VXGA, V8514, VSVGA). |

These virtual device drivers are described in 5.4, "Standard Virtual Device Drivers" on page 61. Those indicated with an (1) form the base set of OS/2 V2.0 and are automatically loaded at system initialization time. The others are installed at operating system initialization time, using DEVICE= statements in CONFIG.SYS, as shown in Figure 22. The first two virtual device drivers in Figure 22 communicate with the corresponding physical device drivers from Figure 21 on page 55.

```
DEVICE=C:\OS2\MDOS\VMOUSE.SYS                    (Mouse VDD)
DEVICE=C:\OS2\MDOS\VCOM.SYS                      (COM port VDD)
DEVICE=C:\OS2\MDOS\VEMM.SYS                      (EMS VDD)
```

*Figure 22. Virtual Device Driver Statements in CONFIG.SYS*

Virtual device drivers perform four main functions:

- Maintain the hardware state for each VDM

- Prevent an application in one VDM from corrupting another VDM

- Support fast screen I/O

- Support fast communications I/O.

## 5.3.1 Loading Virtual Device Drivers

Virtual device drivers are loaded into memory when the initialization phase of the physical device drivers has completed. Upon loading, the virtual device driver verifies the communication path to the corresponding physical device driver, and registers hooks with the Virtual DOS Machine Manager for VDM events such as creation, destruction, and foreground/background switching.

Upon creation of a VDM, the virtual device driver is notified by the Virtual DOS Machine Manager, and the creation routine of the virtual device driver is invoked. This causes a stub device driver to be loaded into the VDM's V86 mode address space. This stub driver accepts device requests from DOS applications within the VDM, and routes them to the virtual device driver outside the V86 mode address space.

This is typically achieved by having the stub device driver issue an instruction which causes a general protection exception. This exception is passed to the operating system's general protection exception handler, which in turn passes it to the Virtual DOS Machine Manager, and finally to the appropriate virtual device driver. The virtual device driver then communicates with the corresponding physical device driver in order to access the hardware device.

When a hardware interrupt occurs, the physical device driver is notified and communicates the event to the virtual device driver, which then takes the appropriate action to inform the DOS application. This occurs even if the VDM is not currently executing in the foreground, since the virtual device driver can access its instance data directly.

Note that certain virtual device drivers do not have a corresponding physical device driver. For example, the VEMM.SYS virtual device driver is used to provide support for the LIM Expanded Memory Specification Version 4.0; this virtual device driver communicates directly with the operating system's memory manager to allocate and manipulate expanded memory objects.

Virtual device drivers communicate with the OS/2 Version 2.0 kernel using **virtual device helper (VDH)** services. The use of these services is required because virtual device drivers execute at privilege level 0, and are thus prevented from issuing normal privilege level 3 function calls to the operating system kernel. VDH services are also used to communicate with physical device drivers, and for communication between virtual device drivers.

Note that there is no fixed communication protocol between a virtual device driver and a physical device driver. The programmer may use any protocol that suits the needs of the driver. A shutdown protocol is recommended in case the virtual device driver has to be shut down.

## 5.3.2 Virtual Device Driver Structure

A virtual device driver is a 32-bit EXE file which runs in protected mode and supports the flat memory model. Figure 23 shows the structure of a virtual device driver and the interfaces to a physical device driver.

```
┌─────────────────────────────────────────────────────────────────┐
│                                                                   │
│   Virtual Device Driver        Physical Device Driver             │
│                                                                   │
│  ┌──────────────────────────┐  ┌──────────────────────────────┐  │
│  │  ┌────────────────────┐  │  │  ┌────────────────────────┐  │  │
│  │  │                    │  │  │  │  Strategy Routine        │  │  │
│  │  │ Virtual DOS Machine│  │  │  │  - Initialize  - IOCTL   │  │  │
│  │  │                    │  │  │  │  - Open        - Read    │  │  │
│  │  │ Event Handler      │  │  │  │  -Close        - Write   │  │  │
│  │  │                    │  │  │  │                          │  │  │
│  │  └────────────────────┘  │  │  └────────────────────────┘  │  │
│  │                          │←─┤                              │  │
│  │  ┌───────┐ ┌──────────┐  │  │  ┌───────┐ ┌──────────────┐  │  │
│  │  │ I/O   │ │          │ optional│ │       │ │  Timer       │  │  │
│  │  │ Trap  │ │          │  │  │  │       │ │  Handler     │  │  │
│  │  │Handler│ │ VDD      │  │  │  │ VDD   │ │              │  │  │
│  │  │       │ │          │──┼─►│  │       │ ├──────────────┤  │  │
│  │  ├───────┤ │ Interface│  │  │  │Interface│ Hardware     │  │  │
│  │  │Software│ │         │  │  │  │       │ │ Interrupt    │  │  │
│  │  │Interrupt│ │        │  │  │  │       │ │ Handler      │  │  │
│  │  │Handler │ │         │  │  │  │       │ │              │  │  │
│  │  └───────┘ └──────────┘  │  │  └───────┘ └──────────────┘  │  │
│  └──────────────────────────┘  └──────────────────────────────┘  │
│                                                                   │
│   0:32 Protect Mode Code       16:16 Protect Mode Code            │
│                                                                   │
└─────────────────────────────────────────────────────────────────┘
```
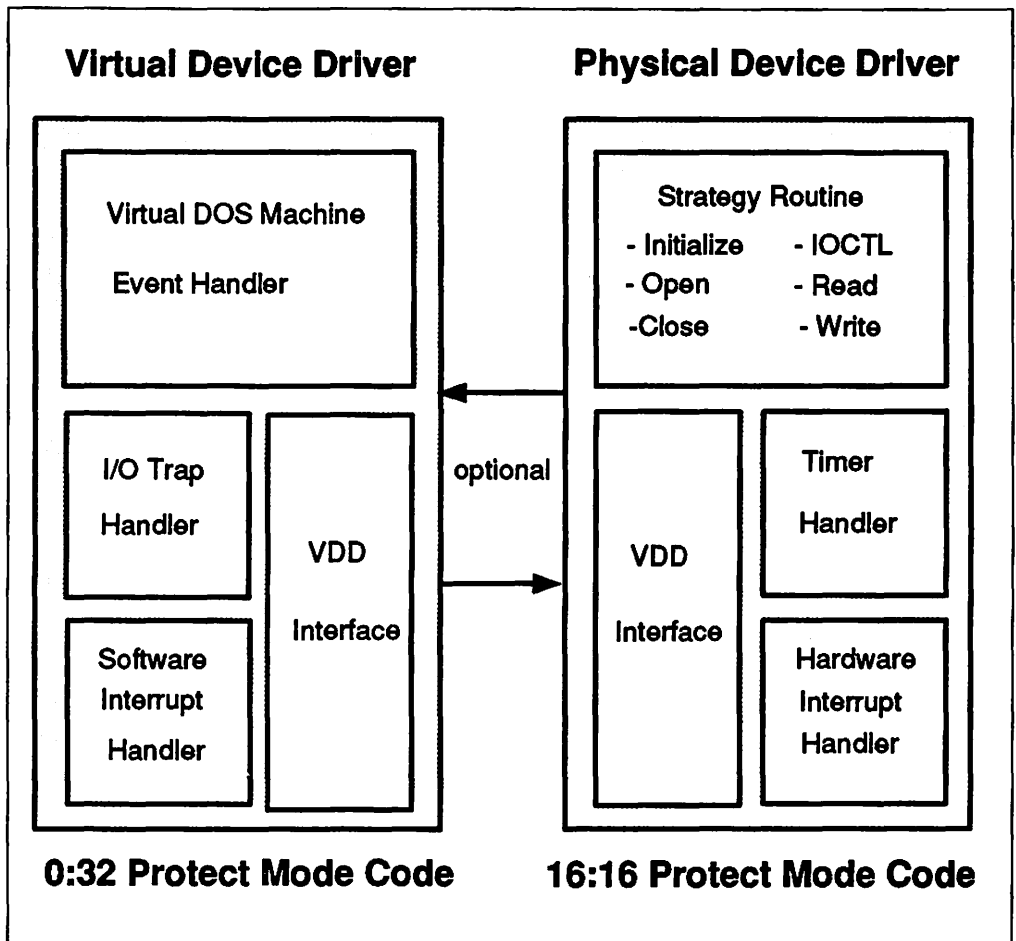
*Figure 23. Virtual COM and Physical COM Device Drivers*

Nearly all virtual device drivers provided in OS/2 V2.0 are written in a high-level language ("C"), although several have portions that were developed using assembler language. Since software interrupts and hooked I/O port operations cause a trap to privilege level 0, time critical code for these operations should be written in assembler language to achieve the maximum possible performance.

A virtual device driver is a 32-bit EXE file that can contain some, all, or none of the following types of objects:

- Initialization code
- Initialization data
- Swappable global code.

A virtual device driver must have at least one object of the following types:

- Swappable global data
- Swappable instance data
- Resident global code

- Resident global data

- Resident instance data.

A virtual device driver should contain resident objects for code and data which must be accessed at physical hardware interrupt time, that is, when a physical device driver calls the virtual device driver. A virtual device driver which does not interact with a physical device driver needs no resident objects. Examples of such device drivers are VEMM and VXMS.

A virtual device driver locates its instance data above the 1MB + 64KB line, but below 4MB. The instance data is therefore outside the VDM's V86 mode address space. This linear address range is the same for all VDMs; that is, all VDMs have the instance data for a particular virtual device driver at the same linear address. The offset from the VDM's linear base address to the virtual device driver's instance data is returned by the OS/2 kernel when the device driver is initialized.

A virtual device driver may need to access its instance data area at physical hardware interrupt time. This may be required even when the VDM is not currently executing in the foreground. Since the instance data is system data located above the V86 addressing range, the virtual device driver may address the per-VDM buffer regardless of which process is currently running. VDM instance data is accessed by adding the VDM's handle + instance data area offset + data offset within the instance data area.

Note that memory objects created by a virtual device driver for passing to a physical device driver must not exceed 64KB in size. This limitation results from the fact that many physical device drivers are still written using the 16:16 segmented memory model, and cannot therefore support memory objects greater than 64KB in size.

## 5.3.3 ROM BIOS Compatibility

ROM BIOS provides many function calls which are typically used by a DOS application program. To maintain the virtualization concept and ensure compatibility with applications which access BIOS or its functions, these functions are emulated by a virtual device driver.

The VBIOS virtual device driver contains a mechanism to map and initialize the system ROMs (including the ROM and EBIOS data areas) and the interrupt vector table into memory within each virtual DOS machine. This is done at VDM creation time, before any other virtual device drivers are loaded. This allows other virtual device drivers to hook interrupt vectors and use VBIOS services. Certain BIOS interfaces are not emulated directly, but are passed to other routines which provide improved performance or functionality. For example, the video interface routines provided by ROM BIOS are powerful but extremely slow. In order to increase the performance of the video output, the video virtual device driver intercepts the ROM BIOS video interrupt (INT 10h) and performs the requested operations directly, providing improved performance.

## 5.3.4  Hardware Interrupt Simulation

A virtual device driver typically establishes communications with a physical device driver and receives events at hardware interrupt time. Based on the event received from the physical device driver and the VDM's current state, the virtual device driver may need to send a hardware interrupt to the VDM. However, the virtual device driver cannot simply call the VDM's interrupt handler, since the interrupt handler may currently be paged out, and page faults cannot be taken on the VDM's interrupt handler code at hardware interrupt time.

The solution is to "simulate" the hardware interrupt to the VDM by delaying it until the VDM process becomes active. This is done in three steps:

1. The VDM's interrupt request flag for the particular interrupt level (IRQ) is set, and a global context hook is set to pass control to the virtual device driver as soon as *any* VDM becomes active.

2. A VDM context hook is set, which increases the priority of the target VDM, based on the priority of the interrupt, thereby enabling fast interrupt processing by the VDM.

3. When the VDM is scheduled and the interrupt request flag is noted, the VDM's interrupt handler code is invoked.

The VDM's interrupt handler typically issues a request for the interrupt data or an EOI instruction. When the virtual device driver receives either of these, it calls the **VDHClearVIRR()** helper service to clear the interrupt request flag. If the interrupt request flag is not cleared before the VDM issues an EOI instruction, the virtual device driver immediately simulates another interrupt to the VDM. For example, the virtual timer device driver may leave the interrupt request flag set when it receives the EOI from a previously simulated interrupt, if it has another hardware timer interrupt pending for that VDM.

### 5.3.4.1  Clearing Interrupts

Note that a virtual device driver must call the **VDHClearVIRR()** helper service when the VDM issues an EOI instruction. Otherwise, the application may receive spurious interrupts because the interrupt request flag is not cleared. For this reason, unknown device interrupts are not supported for VDMs, since there is typically no virtual device driver to clear the interrupt request flag.

Interrupts must be simulated to VDMs as quickly as possible. It is not advisable for a virtual device driver to have too many interrupts pending since the physical device driver's buffers may overflow.

A virtual device must also be very careful when it simulates an interrupt to a VDM because too many nested interrupts may cause the application's stack to overflow. A virtual device driver should wait until the IRET instruction has been executed in the VDM's interrupt handler before it simulates the next interrupt; the virtual device driver may gain control immediately upon IRET being issued, via an IRET handler registered using the **VDHOpenVIRQ()** helper service.

### 5.3.4.2 Shared Interrupts

Personal System/2 machines equipped with the IBM Micro Channel* bus architecture support multiple hardware devices on the same IRQ level. Hence, support may also be required for virtual device drivers to share interrupts. This support is provided through the **VDHOpenVIRQ()** helper function, which accepts a flag indicating that a virtual device driver is willing to share its IRQ. Note that *all* virtual device drivers using the same IRQ must pass the sharing flag; otherwise, an error is returned.

Each virtual device driver receives an IRQ handle, which points to an IRQ data block specific to that device driver. Data not specific to the virtual device driver is contained in a shared IRQ data structure.

When an interrupt is received for a VDM, the virtual interrupt request flag is set and a device request mask is updated. This device request mask is specific to each VDM, and contains a bit for every virtual device driver which has requested a virtual interrupt for the IRQ. When the interrupt is cleared, the device mask bit for the virtual device driver is cleared. However, the virtual interrupt request flag is not cleared until all virtual device drivers have cleared the interrupt.

Note that EOI and IRET handler routines are called when the device mask bit is cleared, allowing virtual device drivers to perform correct interrupt termination handling.

## 5.3.5 Protection

An application within a VDM cannot corrupt or destroy a virtual device driver, since the virtual device driver operates in protected mode outside the V86 mode address space, and is thus not accessible by the application. However, the parameters passed to a virtual device driver from the VDM must be carefully checked before being acted upon, in order to ensure that the service request is valid. Failure to do so may result in failure of a virtual device driver due to an invalid instruction or invalid data.

System registers must not be modified by a virtual device driver. Only certain flags may be modified. These are as follows:

- Arithmetic bits
- Interrupt bit; note that this must be handled with extreme care
- Direction bit.

Failure to observe these rules by a virtual device driver may result in failure of the VDM's parent process, and possible corruption of the virtual device driver itself.

## 5.4 Standard Virtual Device Drivers

The following pages provide brief descriptions of each of the standard virtual device drivers provided with the OS/2 Version 2.0 operating system.

## 5.4.1 VBIOS Device Driver

The VBIOS virtual device driver is like any other base virtual device driver except that it is loaded before any other virtual device drivers. This driver is loaded and initialized first, so that other virtual device drivers can use services provided by VBIOS.

The system BIOS reserves physical memory for the interrupt vector table, ROM and EBIOS data areas. This reservation is done by an indication in the *arena info* data structure passed to the kernel. These physical pages are treated as "unavailable" by the virtual memory manager.

During virtual device driver initialization, the physical interrupt vector table and ROM data area (previously allocated/reserved by the BIOS) are mapped with the **VDHMapMem()** function. VBIOS also installs hooks which cause its own VDM creation handler to be invoked, and an I/O hooking routine to be invoked when all virtual device drivers have been initialized for a particular VDM.

Space is also reserved for the EBIOS data area (if the machine is a PS/2) and the system ROM linear address ranges. This allows virtual device drivers to use and modify this information globally (affecting all VDMs created thereafter).

The following steps are taken when initializing the BIOS for a newly created VDM:

1. Map the CBIOS system area to the VDM address space, using the **VDHMapMem()** service.
2. Allocate memory for the interrupt vector table and ROM BIOS data area.
3. Map and copy the physical interrupt vector table and ROM BIOS data area into the VDMs linear address space.
4. Allocate memory for the extended BIOS data area in the VDM's linear address space (only on PS/2s).
5. Map and copy the physical extended BIOS data area into the linear address space.

When VBIOS's VDM_CREATE_DONE handler is called (after all virtual device drivers' VDM_CREATE handlers have been invoked), VBIOS attempts to install I/O hook routines for the serial and parallel ports COMx and LPTx. These hook routines will take effect *only* if the virtual COM device driver or the virtual printer device driver have not successfully hooked the I/O ports. VBIOS I/O hook routines are used only to display pop-up messages when the device is not virtualized, and to terminate the VDM on user request.

## 5.4.2 Virtual CMOS Device Driver

The virtual CMOS device driver VCMOS.SYS provides support for virtualization of the CMOS battery backed-up RAM, the real time clock (RTC) and the non-maskable interrupt (NMI) disable logic. It provides virtual access to CMOS addresses and data latches through virtual I/O ports.

### 5.4.2.1 CMOS Memory Access

The CMOS portion of the CMOS/RTC may be read or written. Virtual CMOS memory is initialized to the contents of the physical CMOS memory upon VDM initialization. Values written to CMOS memory by DOS applications are written in a buffer local to the VDM. Unlike the physical CMOS memory, however, the contents of the virtual CMOS buffer are lost when the VDM is terminated.

### 5.4.2.2 I/O Port Support

The virtual CMOS device driver component monitors all accesses to its two VDM I/O ports. The two ports are a write-only address latch and a read/write data latch. The address latch port has two functions:

- NMI disable

- CMOS/RTC device address selection.

The data latch is a register for holding a byte being transferred to or from the CMOS/RTC device.

### 5.4.2.3 NMI Disable

The NMI-disable portion of the address latch may be set or reset by a DOS application, but changes to enable or disable NMI are otherwise ignored by VCMOS.

### 5.4.2.4 Real Time Clock and Interrupt Access

The real time clock consists of a time-of-day clock, an alarm interrupt, and a periodic interrupt. Accesses to the real time clock to change the time of day, the timing mode or to set an alarm or periodic interrupt are disallowed. Thus, the CMOS/RTC registers related to the real time clock are supported for read-only access.

Since interrupts can only be supported through write access to the ports, real time clock interrupts are not supported by VCMOS.

## 5.4.3 Virtual DMA Device Driver

The PC AT has eight DMA channels, each of which can be hard-wired to a slave device on the bus. Assignments, therefore, cannot change unless the device adapter is reconfigured by changing jumpers. Because of this one-to-one relationship, there is no separate device driver for the DMA controller. Each device requiring DMA services programs the corresponding DMA channel directly in its device driver, as though the DMA channel were part of its own hardware.

In the PS/2, *virtual DMA channels* may also be assigned; two of the eight channels, channels 0 and 4, are virtual channels which can be dynamically assigned to any device. These channels can, therefore, be multiplexed to service more than one device. ABIOS serializes channel accesses to avoid contention.

Device drivers for hardware devices access the DMA channels directly, with no device driver required for the DMA controller itself. OS/2 Version 2.0, therefore, does not use a physical device driver for DMA access. The virtual DMA device driver VDMA.SYS supports access to DMA ports from DOS applications in virtual machines.

VDMA consists of port trap handlers which ignore IN/OUT commands. The supported DMA services are:

- Memory address and page address registers for all DMA channels

- Transfer count registers for all DMA channels

- Status registers

- Mask registers

- Mode registers

- Byte pointer flip flop port

- Extended function/execute ports (for PS/2 only)

- Master clear ports

- Command register (for PC/AT only)

- Write request register (for PC/AT only).

Note that VDMA *does not* support direct access to the DMA controller by DOS applications.

## 5.4.4 Virtual Disk Device Driver

The virtual disk device driver VDSK.SYS supports access to disk via the INT 13h CBIOS service. Since the CBIOS accesses the hardware ports directly and may therefore cause problems for other processes in the system, VDSK traps the INT 13h interrupt and emulates the processing of this interrupt. Note that VDSK *does not* provide I/O port level access to disk controllers.

The processing of an INT 13h request typically proceeds as follows:

1. The DOS application accesses the disk using INT 13h interface; the INT 13h request is trapped by VDSK.

2. VDSK builds a device driver request packet and sends it to the physical disk device driver. The VDM is then blocked, waiting for the request to complete.

3. The physical disk device driver processes the request packet. If the disk is currently busy, the request is queued.

4. When the request is completed, the physical disk device driver notifies VDSK, which unblocks the VDM.

Protected mode applications access disks via a programming interface which goes through the kernel's device routing mechanism and finally to the physical disk device driver. The physical device driver receives an access request packet similar to that sent by VDSK.

## 5.4.5 VFLPY Device Driver

The virtual diskette device driver (VFLPY) intercepts virtual DOS machine access to the diskette drive directly or through INT 13H calls in order to serialize and coordinate diskette I/O between multiple virtual DOS machines.

## 5.4.6 Virtual Keyboard Device Driver

The Virtual Keyboard Device Driver VKBD.SYS provides virtualization support for the keyboard. It allows keystrokes to be passed from the keyboard to virtual DOS machines. It also allows for text to be pasted into the VDM as key strokes.

Upon creation of the VDM, VKBD establishes communication with the physical keyboard device driver and initializes the portions of the CBIOS data area associated with the keyboard. Subsequently, the physical device driver notifies VKBD of each scan code that is bound for the VDM.

To allow monitoring of I/O activity, VKBD registers itself with the 8086 Emulation component. 8086 Emulation then notifies VKBD when a DOS application in a VDM accesses a virtual keyboard port.

VKBD supports two virtual I/O ports:

- Port 64h - Controller Status/Command

- Port 60h - Controller Input/Output Buffer.

These two ports may respond to requests in a variety of ways, depending on the state of the controller at the time of the request.

### 5.4.6.1 Read Output Buffer

An I/O read request to port 0x60 reads the contents of the controller output buffer. If the "output-buffer-full" status was set before the read request, a timer (T1) is started. The output-buffer-full status is then cleared. When the T1 timer expires, VKBD determines if another byte is ready to be placed into the output buffer. If so, the byte is placed into the output buffer and the "output-buffer-full" status is set.

### 5.4.6.2 Write Output Buffer

An I/O write request to port 0x60 writes the specified byte to the controller input buffer. The previous contents of the input buffer are lost. The port number (0x60) is saved and the byte written is processed, depending on the current state of the keyboard controller. The "input-buffer-empty" status is never set.

### 5.4.6.3 Status Read

An I/O read request to port 0x64 simply returns the contents of the controller internal status register. Reading this port has no effect on the state of the virtual keyboard hardware.

### 5.4.6.4 Write Controller Command

An I/O write request to port 0x64, like a write request to port 0x60, writes the specified byte to the controller input buffer. Since the port number (0x64) is saved here, the system distinguishes between a command byte and a data byte. As above, the byte written is processed, depending on the state of the controller, and the "input-buffer-empty" status is never set.

### 5.4.6.5 Physical Device Driver Notification

Since keystrokes are external events, it is the responsibility of the physical device driver to notify VKBD when keystrokes are available for processing. In particular, the physical device driver calls VKBD when hardware scan codes arrive, and passes each scan code received to VKBD. This occurs whenever the keyboard's current focus is a virtual DOS machine.

When called, VKBD places the scan code in a queue. If the queue was previously empty, the controller "output-buffer-full" status condition is set and if interrupts are enabled, the Virtual Programmable Interrupt Controller is called to simulate the interrupt to the VDM. If the queue was not previously empty, the scan code is added without any other processing. If the queue is full, the speaker is sounded.

### 5.4.6.6 INT 09h Processing

In a "real" DOS environment, the IRQ1 interrupt request is translated by the interrupt controller, causing the INT 09h interrupt service routine to be invoked. This interrupt vector normally points to a routine in the CBIOS. This manner of processing is not desirable in a VDM since the CBIOS only performs U.S. key translation; such processing would complicate the task of national language support. Instead, VKBD simulates this CBIOS function, and may thus use whatever key translation is appropriate for the current country and code page.

The INT 09h emulation code within VKBD performs all functions that the CBIOS would normally perform. This includes:

- Key and scan code enqueueing
- INT 05h (Print Screen) processing
- INT 15h processing for monitoring scan codes and handling the SysReq key
- INT 1Bh for Ctrl + Break and Pause key processing
- Key translation
- Update of keyboard LEDs
- Update of the CBIOS data area status.

Upon termination, VKBD relinquishes access to the keyboard.

## 5.4.7 Virtual Printer Device Driver

The virtual line printer device driver VLPT.SYS supports access to three virtual parallel port controller devices from DOS applications running in virtual DOS machines.

VLPT hooks INT 17h and processes requests for INT 17h services itself, rather than allowing these requests to be handled by CBIOS. INT 17h support includes support for function 02h (Read Status).

VLPT does not support virtual hardware interrupts. If a DOS application attempts to enable interrupts (that is, it attempts to set control port bit 4, "IRQ EN"), that I/O operation is ignored, and the application will not receive interrupts from the parallel port hardware.

VLPT buffers the print data which is subsequently directed to the OS/2 spooler using file system services provided by the Virtual DOS Machine Manager. The spooler may be configured for output on each printer device (LPTx) that will be accessed by DOS applications from a VDM. Figure 24 on page 67 shows the various operations performed by the virtual printer device driver.
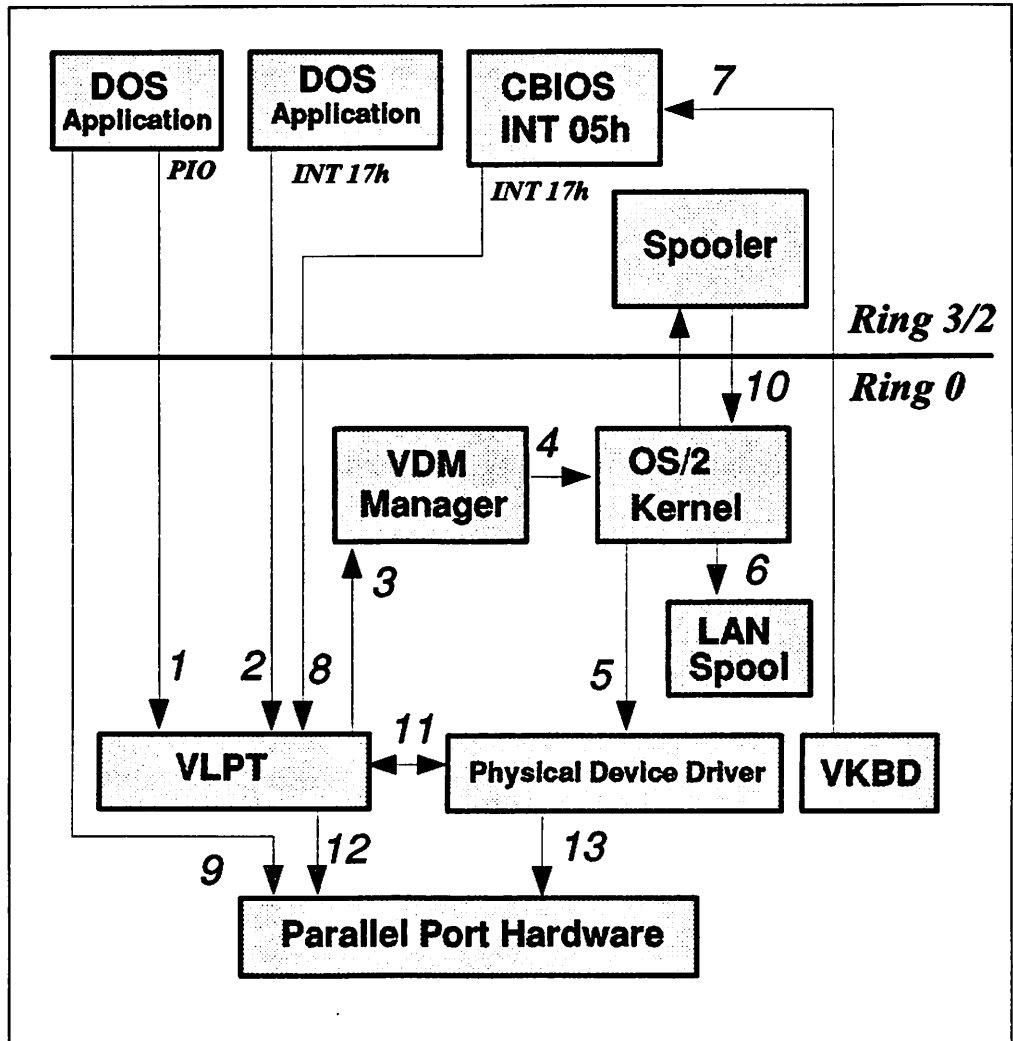
Figure 24. Virtual Printer Device Driver Operation

If VLPTDD.SYS is the only virtual printer device driver installed, no INT 17 will be issued when printing is done on numbered I/O devices (for example, LPT1, LPT2, etc.). However, if an application such as a TSR program must catch all INT 17 interrupts, the LPTDD.SYS device driver must be installed as well. You can find LPTDD.SYS in the subdirectory \os2\mdos.

When LPTDD.SYS is available, a request from the DOS file system issuing INT 21 is converted by LPTDD.SYS into INT 17. INT 17 is then forwarded to VLPT.SYS and from this point on the print request proceeds as described above in Figure 24. Note, however, that installing LPTDD.SYS in addition to VLPT.SYS will cause the printing from a VDM to slow down.

### 5.4.7.1 Spooling
In order to support spooled print output, the OS/2 spooler must be installed. A new IOCTL interface is defined in order to allow the spooler to identify itself to the physical printer device driver. The new IOCTL functions *Set Spooler Status* and *Get Spooler Status* are called by the spooler and the Virtual DOS Machine Manager.

The spooler invokes the *Set Spooler Status* function (Category 5, Function 4Ch) at spooler monitor registration time to inform the physical device driver that a

particular port is actively configured as the output device for a particular spool queue. It also invokes this interface whenever the user manipulates the spooler queue setup by invoking the Print Manager's *Setup Printers/Change Printers* dialog.

The Virtual DOS Machine Manager invokes the *Get Spooler Status* function (Category 5, Function 6Ch) during an implicit open of a print device whenever VLPT processes the first print output (INT 17h) from a VDM. This allows the Virtual DOS Machine Manager to determine if the spooler is active so that it can return the spool queue file handle to VLPT to continue printing.

### 5.4.7.2  Print Screen

VLPT also supports the *Print Screen* function by hooking INT 05h so that it is notified before the CBIOS INT 05h handler. The CBIOS INT 05h handler invokes INT 17h functions, and the VLPT INT 17h emulation in turn sends this data to a spool file, if the spooler is active. When the CBIOS INT 05h handler returns, VLPT also receives control so that it may close the spool file.

### 5.4.7.3  File Transfer

To support DOS file transfer programs which use the parallel ports (such as the IBM Data Migration Facility), and which typically program the parallel port controller directly, VLPT provides a mode known as **direct I/O access**. In this mode, the physical printer device driver grants temporary exclusive ownership of the parallel port hardware to the VDM in which the application is running. This mode allows the application to have direct access to the parallel port's data, status and control registers.

If the port is not currently active (printing) under control of the physical printer device driver, the physical device driver will grant VLPT exclusive access to the port, and continue to service incoming file system I/O requests. Any incoming monitor requests from the spooler are blocked until exclusive access is released (no error or status monitor packets are sent to the monitor chain).

If the physical printer device driver is actively processing a hardware I/O operation, when VLPT makes a request for exclusive access, the physical device driver will return an error code to VLPT. VLPT will then display a pop-up message (via the **VDHPopUp()** helper service), allowing the user to select the most appropriate system action ("End program" or "Retry").

**Note:**  Due to the multitasking nature of OS/2 V2.0, data communications using this type of application have an increased probability of error when multiple processes are concurrently active and/or when the virtual DOS machine is switched to the background.

### 5.4.7.4  PS/2 Register Virtualization

On PS/2 systems, the physical printer device driver ensures that all LPT ports which support extended mode (read/write 8-bit parallel I/O) will be enabled for extended mode at system initialization time. On any PS/2 models which do not enable this mode by default, the physical printer device driver enables extended mode via the system's Programmable Option Select (POS) function. This ensures that all PS/2 LPT ports will support manipulation of the control port *Direction Control* bit.

On all PS/2 models (but *not* on IBM PC/AT systems), VLPT will virtualize the adapter enable/setup register. All bits of this register are virtualized for read operations, but *only bit 7 of the enable/setup register is virtualized for write oper-*

*ations.* DOS applications may modify bit 7 of this register in order to gain access to the system board's POS Register 2, thereby enabling or disabling extended mode operation of the parallel ports. When bit 7 is set to 0 (the default state), the parallel port is configured as an 8-bit, parallel, bidirectional interface. When bit 7 is set to 1, the parallel port bidirectional mode is disabled. As described above, the physical printer device driver ensures that all PS/2 models have this bit set to 0 (extended mode enabled) during system initialization.

Note that only bit 7 is virtualized and may be manipulated; attempting to manipulate any other bits of this register will result in termination of the VDM. As the behavior is virtualized, the true state of the hardware register is not affected by any operations of a DOS application running in a virtual DOS machine.

### 5.4.7.5 Printer Close

VLPT exports the **VDHPrintClose()** service. This interface may be called by another virtual device driver such as VKBD to force any open printers in a VDM to close. This technique is used to handle a forced End-of-Job (Ctrl+Alt+PrintScreen) character, and is required because some DOS applications do not explicitly close or disable the printer when their print activity is completed.

VLPT may also close open print files whenever a VDM is terminated. VLPT registers an event hook with the Virtual DOS Machine Manager, and is therefore notified upon termination of a VDM. All open print files in the terminating VDM are closed, after any buffered data has been sent to the spooler.

When operating in direct I/O access mode, VLPT can detect application termination in one of three ways:

- PDB is changed or destroyed (default)
- VDM is terminated
- User hot-key (Ctrl+Alt+PrintScreen).

When this termination event is detected, direct I/O access mode is cancelled and VLPT relinquishes the VDM's exclusive control of the parallel port hardware. The physical printer device driver then reclaims ownership of the port and resumes normal I/O operations.

Note that VLPT will not always close an open print file when the DOS application terminates. Depending on the DOS application's behavior, the VDM may remain active when the program ends, and the spooler print file may therefore remain open. If so, the user can cause the open print file(s) to close by using the Ctrl-Alt-PrintScreen control key sequence. Alternatively, the user can leave it to the system by setting the *PRINT_TIMEOUT* value in the *DOS settings* to the time in seconds that the operating system should wait before forcing the print job to the printer. Consequently there is no need to exit these DOS programs to have the print job released from the print spooler. For more information on PRINT_TIMEOUT see Chapter 11, "DOS Settings" on page 205.

## 5.4.8 Virtual Numeric Coprocessor Device Driver

The virtual numeric coprocessor device driver VNPX.SYS provides virtualization of the 80287/80387 numeric coprocessor hardware, allowing access to numeric coprocessor facilities by multiple DOS applications running in virtual DOS machines.

OS/2 Version 2.0 provides a physical device driver for the numeric coprocessor, within the operating system kernel. At system initialization time, VNPX registers a number of hooks with the physical device driver, so that VNPX is informed whenever a numeric coprocessor exception or emulation trap occurs. Handling routines are also registered with the Virtual DOS Machine Manager, and are invoked upon VDM creation and termination. Coprocessor I/O ports visible to V86 mode applications are hooked by VNPX.

VNPX is informed by the physical device driver at initialization time, whether VDMs are permitted to use the coprocessor. Upon VDM creation, VNPX sets the equipment summary flag for the numeric coprocessor according to the information received at initialization time. In the event of the coprocessor being unavailable to DOS applications, the equipment summary flag is turned off. An application interrogating the flag will therefore assume that no coprocessor is present, and take appropriate action.

The first time an application in a VDM executes a coprocessor instruction within a particular timeslice, an exception condition (Trap 0007) occurs. The exception handler sets a flag within the physical device driver before allowing processing of the instruction to continue. This flag is checked at task switch time and if it is set, the coprocessor state is saved by the physical device driver. Note that the save operation takes place *only* if the coprocessor is used by an application during its timeslice; for those applications which do not use the coprocessor, no action is taken. This allows optimum performance during task switching.

## 5.4.9  Virtual Programmable Interrupt Controller

The virtual programmable interrupt controller VPIC.SYS is a virtual device driver responsible for virtualization of hardware interrupts to virtual DOS machines. This device driver simulates interrupts to virtual DOS machines by providing a virtual interface to the 8259 Programmable Interrupt Controller (PIC).

The virtual PIC device driver supports the hardware interrupt-related services needed by virtual device drivers and DOS Sessions. The services include setting handlers to trap EOI and IRET events, simulating interrupts to DOS Sessions, and handling PIC I/O accesses by DOS Sessions. The virtual PIC device driver maintains a per-DOS Session virtual PIC state so that each DOS Session appears to have its own independent 8259 Programmable Interrupt Controller.

This per-DOS Session virtual PIC state contains items such as the current mask, the current IR (interrupt request) and IS (interrupt service) registers, base interrupt vector and initialization mode for a particular VDM. A per-VDM state machine is used to track the initialization control word (ICW) or operation control word (OCW) for which VPIC is waiting. This module also invokes the virtual device driver's EOI handler when it receives EOI commands from a VDM.

The interrupt simulation mechanism is similar to the way signals are handled for OS/2 applications. The virtual PIC device driver can be broken up into two major parts:

1. The virtualization of the PIC ports, and

2. The simulation of hardware interrupts to VDMs.

Figure 25 on page 71 below shows this breakdown and the interfaces to other components, it shows the VPIC architecture and the role it plays in virtualizing hardware interrupts to virtual DOS machines.
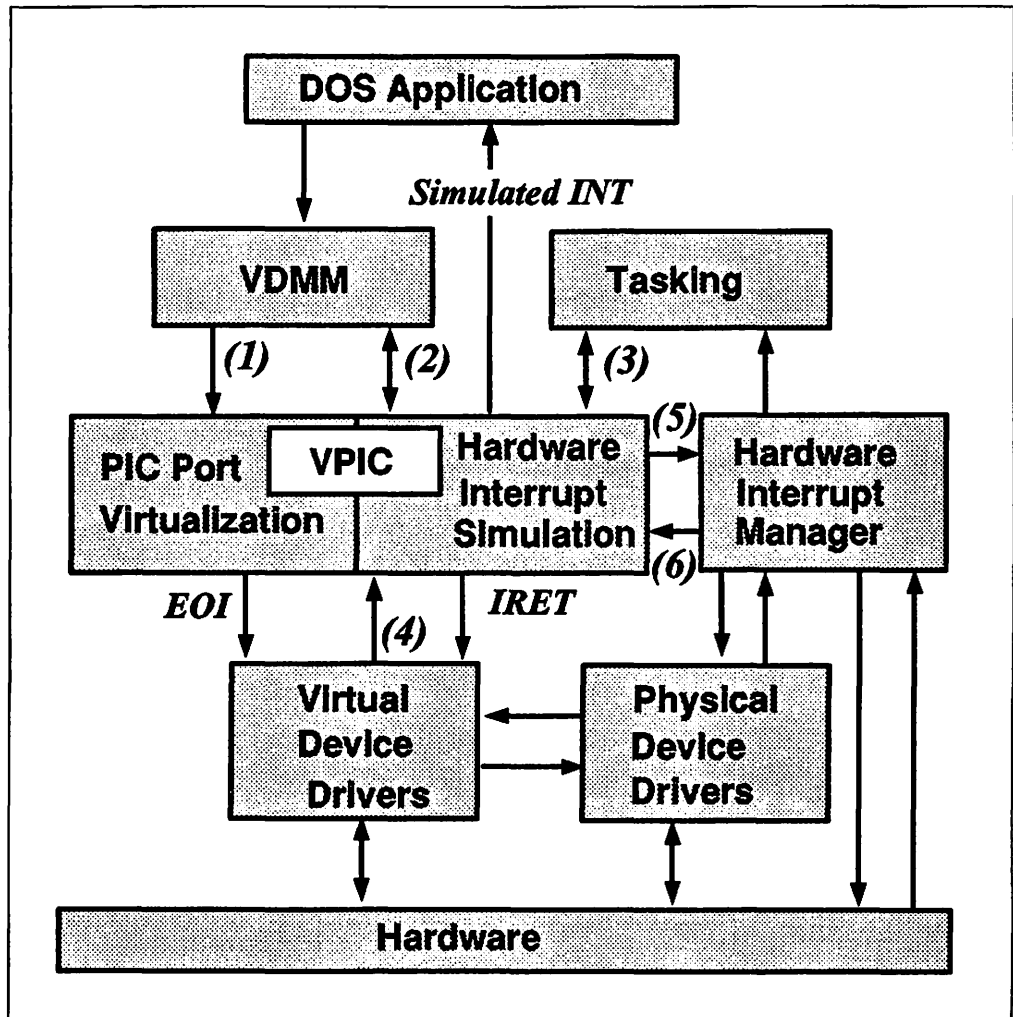
*Figure 25. Virtual Programmable Interrupt Controller*

Notes to the numbers in the above figure:

1. • 8259 PIC port accesses
   • VPIC initialization entry point (VDDInit)
   • VPIC VDM creation entry point (vpicCreateVDM)

2. • Call when interrupts enabled service (VDHArmSTIHook)
   • Return to VDM interrupt code service (VDHPushInt)

3. • Set the global and the VDM context hook service (VDHArmContextHook)
   • Start the timer service (VDHArmTimerHook)
   • Set the VDM priority service (VDHSetPriority)

4. • Set the IRET and EOI handlers service (VDHOpenVIRQ)
   • Set the virtual IRR request service (VDHSetVIRR)
   • Clear the virtual IRR request service (VDHClearVIRR)
   • Get virtual IRQ status service (VDHQueryVIRQ)
   • Send virtual EOI service (VDHSendEOI)
   • Wait for simulated interrupt service (VDHWaitVIRRs)
   • Wake up the VDM waiting for a simulated interrupt (VDHWakeVIRRs)

5. • Physical PIC hardware interrupt requests and EOI
   • VPIC requests an IRQ level will be dispatched to VPIC if no physical device driver services the interrupt (INTSetVDMIRQ)
   • VPIC notifies the interrupt manager of the end of the interrupt processing

(INTEOIVDMIRQ)

6. • Hardware Interrupt dispatching
   • The interrupt manager transfers control to the VPIC for hardware interrupts
     that the VPIC has set, and no physical device driver has serviced
     (VPICIntHdlr)

Not all combinations of ICWs or OCWs are supported. Some seldom used initialization modes and operation commands are ignored. These unsupported features are:

• Slave PIC on any IRQ other than IRQ2
• Level-triggered initialization
• Special fully nested mode
• Auto EOI mode
• 8080/8085 mode
• Buffered mode
• Special mask mode
• Set IRQ priority command
• Poll command
• Rotate on specific and non-specific EOI commands.

The following figures show in detail which 8259 PIC initialization and operation commands from a VDM are supported by the VPIC. Table 1 shows the supported and unsupported initialization control words.

| Table 1 (Page 1 of 2). PIC Initialization Control Words | | | |
|---|---|---|---|
| **ICW** | **Field** | **Explanation** | **Supported** |
| ICW1 | D0 | 1 = ICW4 needed | Supported |
| | | 0 = no ICW4 needed | Supported |
| | D1 | 1 = single PIC | Ignored |
| | | 0 = cascade mode (slaves) | Supported |
| | D2 | 1 = call address interval of 4 | Ignored |
| | | 0 = call address interval of 8 | Ignored |
| | D3 | 1 = level triggered mode | Ignored |
| | | 0 = edge triggered mode | Supported |
| | D4 | 1˙ | |
| | D5-D7 | A7-A5 of int vector in 8080 mode | Ignored |
| ICW2 | D0-D2 | | Ignored |
| | D3-D7 | base int vector number in 8086 mode | Supported |
| ICW3 (Master) | | Ignore if slave is not IRQ2 | |
| | D0-D7 | 1 = IRQ has a slave connected | |
| | | 0 = IRQ does not have a slave connected | |
| ICW3 (Slave) | | Ignore if slave is not IRQ2 | |
| | D0-D2 | slave ID | |
| | D3-D7 | 0 | |
| ICW4 | D0 | 1 = 8086/8088 mode | Supported |

| Table 1 (Page 2 of 2). PIC Initialization Control Words | | | |
|---|---|---|---|
| **ICW** | **Field** | **Explanation** | **Supported** |
| | | 0 = 8080/8085 mode | Ignored |
| D1 | 1 = auto EOI | Ignored | |
| | | 0 = normal EOI | Supported |
| | D2-D3 | 0x = non-buffered mode | Supported |
| | | 10 = buffered mode/slave | Ignored |
| | | 11 = buffered mode/master | Ignored |
| | D4 | 1 = special fully nested mode | Ignored |
| | | 0 = not special fully nested mode | Supported |
| | D5-D7 | 0 | |

| Table 2 (Page 1 of 2). PIC Operation Control Words | | | |
|---|---|---|---|
| **OCW** | **Field** | **Explanation** | **Supported** |
| OCW1 | D0-D7 | IMR (interrupt mask register) | Supported |
| | | 1 = IRQ masked (disabled) | |
| | | 0 = IRQ unmasked (enabled) | |
| OCW2 | D0-D2 | IRQ level to be acted upon | |
| | D3-D4 | 0 | |
| | D5-D7 | EOI command | |
| | | 000 = rotate in auto EOI mode (clear) | Ignored |
| | | 001 = non-specific EOI | Supported |
| | | 010 = no operand | Supported |
| | | 011 = specific EOI | Supported |
| | | 100 = rotate in auto EOI mode (set) | Ignored |
| | | 101 = rotate on non-specific EOI | Ignored |
| | | 110 = set priority command | Ignored |
| | | 111 = rotate on specific EOI | Ignored |
| OCW3 | D0-D1 | read register command | |
| | | 00 = no action | Supported |
| | | 01 = no action | Supported |
| | | 10 = read IR register | Supported |
| | | 11 = read IS register | Supported |
| | D2 | 1 = poll command | Ignored |
| | | 0 = no poll command | Supported |
| | D3 | 1 | |
| | D4 | 0 | |
| | D5-D6 | Special mask mode | |
| | | 00 = no action | Supported |
| | | 01 = no action | Supported |
| | | 10 = reset special mask | Ignored |

| Table 2 (Page 2 of 2). PIC Operation Control Words | | | |
|---|---|---|---|
| OCW | Field | Explanation | Supported |
| | | 11 = set special mask | Ignored |
| | D7 | 0 | |

# 5.4.10 Virtual Timer Device Driver

The virtual timer device driver VTIMER.SYS provides virtualization of timers used by DOS applications running in VDMs. Three timer ports are supported in order to allow reprogramming of interrupt frequency and speaker tone frequency. VTIMER also distributes timer ticks to VDMs and maintains a count of timer ticks.

All accesses to the timer are simulated by VTIMER. Therefore, unlike most other virtual device drivers, VTIMER has no communication with the physical timer. VTIMER derives its timer tick directly from the system clock.

VTIMER keeps a per-VDM cumulative timer count in the VDM's data area. When a periodic timer interrupt occurs, VTIMER receives control. It adds the periodic interval count value to the cumulative timer count of the VDM and compares it with the VDM's programmed count. If the cumulative timer count exceeds the VDM's programmed count, an interrupt is simulated to the VDM and the programmed count is subtracted from the cumulative timer count.

### 5.4.10.1 Timer 0

This timer is normally used to provide periodic interrupts to DOS applications. A periodic system timer with an interval close to 18.2 milliseconds is set up so that VTIMER can accumulate virtual ticks and simulate tick interrupts if necessary.

If a VDM attempts to program an interrupt period less than 18.2 milliseconds, the periodic timer interval will be changed to four times the normal frequency of 18.2 Hz, regardless of the VDM's programmed value. A *HighRate* reference count is also kept to record the number of VDMs using a higher interrupt rate. As long as there is at least one VDM using a higher interrupt rate, the periodic rate is maintained at approximately four times 18.2 Hz. Otherwise, the timer frequency is reset to 18.2 Hz.

Accesses to timer count registers from within a VDM are trapped by VTIMER. Read accesses to the count registers should be preceded by a counter latch command written to the control word register, in which case a random value derived from the system time is latched and stored in the virtual latch registers. Its purpose is to provide the applications with a sense of elapsed time, although the count value itself is meaningless. Read access to the count register is simulated by reading the virtual latch value. Write accesses to the count registers are stored in the virtual count registers.

### 5.4.10.2 Timer 1

Timer 1 is used in the PC AT[2] as the memory refresh request timer. Since the correct operation of this timer is vital to the system, no known software reprograms it for uses other than reading it as the random number generator speed.

---

[2] Although the IBM PC AT is based on the Intel 80286 processor and therefore is not supported by OS/2 Version 2.0, many PC AT machines exist which have been fitted with processor upgrades from various manufacturers, which may enable them to run OS/2 Version 2.0. Information on the PC AT architecture is therefore included herein for completeness.

Therefore, VTIMER does not support any access except counter read to this timer by DOS applications. Any accesses other than counter reads are trapped and ignored.

### 5.4.10.3 Timer 2

Timer 2 is the speaker tone generator. It is accessed by DOS applications directly, via programming interfaces or DevHlp services, or by the **VDHBeep()** function. Serialization of the speaker usage can be achieved by using a semaphore in the kernel.

When a DOS application accesses the speaker, it typically programs timer 2 for the appropriate frequency and then enables the speaker by programming the speaker enable bits in system control port B. These programming operations are trapped by VTIMER and the frequency value is stored for the VDM. When Control Port B is programmed to enable the speaker, the kernel beep service is called to generate the beep. This call may be blocked due to the fact that another process is beeping and thus owns the speaker semaphore. After the semaphore is obtained, the stored frequency value is programmed into timer 2 and the speaker is enabled. The semaphore is released when the DOS application disables the speaker by programming the System Control Port B.

There is one exception to the speaker serialization, and this occurs during interrupt time. For example, when a process is generating a speaker tone, the keyboard buffer may be full and the keyboard interrupt handler needs to generate a warning beep immediately. Therefore, the kernel also provides an interrupt time beep service which can pre-empt any ongoing beeps and use the speaker to generate its own beep regardless of which process currently owns the speaker semaphore.

## 5.4.11 Virtual COM Device Driver

The virtual COM device driver VCOM.SYS provides virtual support for the serial communication I/O ports and for the serial channel-related CBIOS entry points. It provides support in each VDM for up to two communication channels on the IBM PC AT and compatible systems, and up to four on the IBM PS/2 Model 80 and compatible PS/2 systems.

VCOM only supports access to communication channels which are physically present on a given system. It does not include support for accessing communication devices which may be redirected by network software. For each supported port, VCOM searches for the port's base address in the CBIOS data area. If the address is zero, indicating that the device is not present or is owned by a physical device driver other than the COM.SYS device driver, VCOM does not attempt to support that serial device.

If the port's base address is found, VCOM verifies that the physical device driver has installed itself for that serial device. If the physical device driver indicates it is not installed for that port, VCOM does not attempt to support that serial device.

Many DOS applications that support asynchronous communications include hardware interrupt handler routines. These routines typically perform I/O directly to the COM port hardware. To support these DOS applications and to allow them to run in both background and foreground, VCOM simulates hardware interrupts in the task-time context of the V86 mode process. VDMs are scheduled and dispatched using the same preemptive task dispatching method that drives OS/2

processes. Hardware interrupts on the other hand, occur asynchronously to this scheduling process. By simulating hardware interrupts and presenting a virtual hardware state, the interrupt handling logic of the DOS application does not execute on the physical interrupt thread. This means that switching to V86 mode is not done at interrupt time, but is deferred until the scheduler dispatches the VDM task.

The advantage of simulated interrupts is that mode switching and hardware virtualization need not be done at interrupt time. In addition, the DOS application does not gain control at interrupt time, which helps to maintain system integrity. A potential disadvantage of this approach is that DOS applications with time-critical routines may not operate correctly under some system load configurations.

### 5.4.11.1 Port/Channel Contention

After VDM creation, the CBIOS data area provides a logical link between the virtual communication channels in the VDM and the physical serial channel hardware. Since VCOM does not support a COM port if its address is not in the CBIOS area, it can handle its own errors and/or terminate in its usual fashion if the DOS application does not find the device address.

If the CBIOS area indicates that the device is present, however, VCOM determines if the device is already in use in another VDM when the DOS application makes its first access to that device. If so, VCOM does not attempt to send the OPEN command to the physical device driver. Instead, VCOM will issue a pop-up message that informs the user of the resource contention and allows for a user-selected action. As a result of the user action, the conflict is resolved or the VDM may be terminated.

If the port is not already in use, VCOM calls the physical device driver with the OPEN command. This command will succeed provided the port is not currently in use by a protected mode process.

Once the device is opened, VCOM communicates directly with the physical device driver to perform all virtual hardware operations. These include sending and receiving data, detection and simulation of hardware interrupts, and setting or querying the status and control registers.

### 5.4.11.2 CBIOS Access

VCOM also supports access to CBIOS COM port services through software interrupt 14h. Rather than allowing the CBIOS to perform the functions by accessing the virtual I/O ports directly, VCOM emulates CBIOS functions. The CBIOS access emulation supports six functions:

- **Initialize**

  The initialize function establishes the communication speed and framing options for the channel, and returns the modem and line status. To specify bit rates of greater than 9600 bits per second, the *extended initialize* function must be used.

- **Extended Initialize**

  The extended initialize function, like the initialize function, establishes the communication speed and framing options for the channel and returns the modem and line status. This function is used if a bit rate of 19,200 bits per

second (or greater) is desired, or if *space* or *mark* parity selection is desired.

- **Send Character**

  The send character function sends a character to the communication channel.

- **Receive Character**

  The receive character function waits for a character from the communication channel and returns the character.

- **Read Status**

  The read status function returns the modem and line status.

- **Extended Port Control**

  The extended port control function sets or reads the modem control register.

DOS applications running in VDMs may access these functions using the standard INT 14h service, in a manner identical to that used in a native DOS environment.

### 5.4.11.3  Virtual Interrupt Indication

The DOS application must properly enable interrupts and the specific interrupt type before the interrupt will be simulated to the VDM. The following INS8250 interrupt types are virtualized by VCOM:

- Line Status Interrupt

- Receive Data Available Interrupt

- Transmitter Holding Register Empty Interrupt

- Modem Status Interrupt.

When the physical device driver notifies VCOM of an interrupt, VCOM passes a virtual interrupt to the Virtual Programmable Interrupt Controller, which in turn simulates the interrupt to the VDM.

To maintain high performance on the physical serial channel, the COM physical device driver typically does not notify the VCOM on every interrupt. Rather, VCOM receives notification of certain events, and determines whether to begin or continue simulating interrupts to the VDM.

### 5.4.11.4  Coexistence with Other Serial Device Drivers

Since there is always the potential for other device drivers to own a serial port, VCOM does not assume ownership of any devices for which the physical COM device driver has not been installed. For example, a serial mouse device driver may be installed and may own the COM1 serial port. The COM physical device driver will not install for this port, and VCOM will therefore support only the higher numbered serial ports (if installed).

If a physical device driver installs itself and zeros the COM port base addresses in the CBIOS data area, VCOM does not attempt to initialize for that COM port and will not assume any responsibility to virtualize the serial device hardware for that port. This may result in problems for certain DOS applications which rely on the CBIOS data area in order to access multiple serial ports.

## 5.4.12 VDPMI Device Driver

The virtual DOS Protected Mode Interface(VDPMI) device driver provides Version 0.9 DPMI support for virtual DOS machines. DPMI applications run in protected mode, not V86 mode. VDPMI allows the user to specify how much protected mode memory should be made available to a DOS session using the *DPMI_MEMORY_LIMIT* setting in the *settings* notebook.

## 5.4.13 VDPX Device Driver

The virtual DOS Protected Mode Extender(VDPX) device driver provides address translation from protected mode to V86 mode for DPMI applications running in virtual DOS machines. This translation is necessary because DPMI applications run in protected mode but issue interrupt requests in V86 mode to perform systems services.

The VDPX device driver registers the *DOS Setting* called *DPMI_DOS_API*. The available settings are:

**Enabled**    VDPX always translates the interrupts it supports from protected mode to V86 mode

**Auto**    The application must issue an INT 2FH in protected mode to begin the translation

**Disabled**    VDPX does not perform any address translation.

## 5.4.14 VXMS Device Driver

The Extended Memory Specification (XMS) Version 2.0 provides a standard interface for the use of three regions of extended memory on 80286 and 80386 machines:

- High Memory Area (HMA), accessible in real mode if the A20 address line is enabled
- Extended Memory Blocks(EMBs), up to 64MB that is used for data storage
- Upper Memory Blocks(UMBs), located between 640KB and 1MB in conventional memory.

The Virtual Extended Memory Specification(VXMS) device driver

- Implements all the functions of XMS Version 2.0
- Provides each virtual DOS machine with its own separate XMS emulation
- Provides configurable limits on the amount of extended memory for each virtual DOS machine separately.

The VXMS device driver is installed via CONFIG.SYS, with the following syntax and options:

Syntax

```
DEVICE=\OS2\MDOS\VXMS.SYS [options]
```

Options

**/XMMLIMIT=g,i**

Set the global maximum memory usage of VXMS to *g*KB, and the per virtual DOS machine limit to *i*KB. The default is *16384,2048*.

**/HMAMIN=d**
> Minimum request size in KB for an HMA request to succeed. The default is *0*, the maximum is *63*.

**NUMHANDLES=n**
> Number of handles available to each virtual DOS machine. Handles are used to access EMBs. The maximum is *128* and the default is *32*.

**/UMB**      Create UMBs. The default is not to create any.

**/NOUMB**    Do not create UMBs. This is the default setting.

## 5.4.15  VEMM Device Driver

The Lotus-Intel-Microsoft (LIM) Expanded Memory Specification (EMS) Version 4.0 provides a standard interface for the use of expanded memory on 8086 and 8088 machines. The specification offers up to 32MB of expanded memory divided into up to 255 objects that can be mapped into conventional memory.

The Virtual Expanded Memory Specification (VEMM) virtual device driver:

- Implements all the INT 67H functions in LIM 4.0 EMS, except those for DMA registers
- Provides each virtual DOS machine with its own separate EMS emulation
- Supports remapping of conventional memory for use by DOS programs
- Provides a configurable limit to EMS memory for each virtual DOS machine
- Supports multiple physical to single logical memory mappings so that different 8086 addresses can be mapped to the same expanded memory object address.

The VEMM device driver is installed via CONFIG.SYS, with the following syntax and options:

Syntax

```
DEVICE=\OS2\MDOS\VEMM.SYS [options]
```

Options

**/S=n**      EMS memory limit per virtual DOS machine, with the default being *2048*. This can also be set with the *EMS_MEMORY_LIMIT* setting in the *settings* notebook for the virtual DOS machine.

**/L=n**      Size of the remappable conventional memory. This can also be set with the *EMS_LOW_OS_MAP_REGION* setting in the *settings* notebook for the virtual DOS machine.

**/H=n**      Size of the extra mappable memory, with the default being *32*. This can also be set with the *EMS_HIGH_OS_MAP_REGION* setting in the *settings* notebook for the virtual DOS machine.

**/F=xxxx**   Frame location for remapping expanded memory into conventional memory. The default is AUTO. This can also be set with the *EMS_FRAME_LOCATION* setting in the *settings* notebook for the virtual DOS machine. The frame location may need to be moved if there is a conflict with the mapping address of a physical device driver that does not have a corresponding virtual device driver. Refer to Chapter 11, "DOS Settings" on page 205 on how to use the *MEMORY_INCLUDE_REGIONS* and *MEMORY_EXCLUDE_REGIONS* settings when memory conflicts occur.

## 5.4.16 VWIN Device Driver

"Seamless" WIN-OS/2 support is the ability to run Windows applications in a window right on top of the Workplace Shell desktop. This requires the Windows video device driver and the PM video device driver communicate and coordinate their access of the video hardware. Each device driver effectively *owns* its piece of the screen. Allowing the Windows display device driver to access the video hardware directly avoids the more cumbersome process of a thorough task switch. However this hardware access poses integrity problems in the areas of simultaneous access of hardware, rectangle invalidation handling, window management, and the exchange of window state information between PM and seamless VDMs supported by separate video device drivers.

To address these problems, a high performance, interprocess communication, virtual device driver (VWIN.SYS) was created. It serializes the simultaneous accesses to the hardware, oversees the exchange of window state information between PM and seamless VDMs, and establishes the addressability of PM resources (either directly or indirectly) by the Windows display device driver.

## 5.4.17 Virtual Mouse Driver

Given the diversity of mouse hardware, and the complexity of dealing with all possible combinations of mouse hardware and video equipment, few if any applications talk directly to mouse hardware. Most applications which support a mouse do so through the INT 33h interface. Virtualization of the INT 33h interface is provided by the virtual mouse device driver VMOUSE.SYS.

The INT 33h interface performs the following:

- Position and button tracking

- Position and button event notification

- Selectable pixel and mickey mappings

- Video mode tracking

- Video pointer management (location and appearance)

- Light pen emulation.

The interface between the physical mouse driver and VMOUSE is straightforward. The physical mouse driver is aware at all times of which session owns the mouse; thus, when a foreground VDM owns the mouse, it notifies VMOUSE of mouse events. The events themselves remain buffered by the physical device driver until VMOUSE requests them. Normally, VMOUSE asks for each event immediately, and updates the INT 33h data structures for the foreground VDM, unless the application running in the VDM has registered a mouse event subroutine.

If a mouse event subroutine has been registered by the application, VMOUSE may have to notify the routine of a mouse event. This depends on the events for which the subroutine has requested notification. When a registered subroutine must be called, VMOUSE requests a fake interrupt to be simulated into the VDM. A fake interrupt service routine (loaded into each VDM upon creation) immediately emulates the interrupt and then proceeds to notify the VDM's registered subroutine. In order to pick an IRQ that is guaranteed to not conflict with any other virtual device driver, VMOUSE queries the physical mouse driver at initialization time for the physical IRQ used by the mouse. This ensures that no conflict occurs.

### 5.4.17.1  Position and Button Data Tracking

Position and button events are interrupt time events that are communicated to VMOUSE by the physical device driver via a "data ready" entry point. If the VDM is not already processing a previous event, VMOUSE calls the physical device driver to get the new event; otherwise, VMOUSE waits until previous event processing is complete. This avoids the need for buffering within VMOUSE.

Normally, the VDM will not be processing any events, so position and button information can be immediately retrieved and stored for later query by a VDM application, via INT 33h.

### 5.4.17.2  Position and Button Event Notification

As events are requested and supplied by the physical mouse driver, VMOUSE peeks ahead to the next event (if any) and, if it is a movement-only event, extracts it and overlays the current event. This is continued until there are no more events, or the next event is not a movement-only event. This reduces the amount of interrupt-simulation overhead during periods of rapid mouse movement.

### 5.4.17.3  Selectable Pixel-to-Coordinate and Mickey-to-Pixel Mappings

Since pixel-to-virtual-coordinate mappings are often used by DOS applications but are not supported for protected mode applications; VMOUSE manages such mappings. Since mickey-to-pixel mappings *are* supported for protected mode, VMOUSE relies on the physical mouse driver to perform these mappings. Thus physical mouse driver interfaces are provided to set the mickey-to-pixel mapping and the dimensions of the screen in pixels.

## 5.4.18  VCDROM Device Driver

The VCDROM virtual device driver enables audio support for CD-ROM applications running in virtual DOS machines. In native DOS, audio and other IOCTL support is provided by the pass-through function of the CD-ROM file system driver, MSCDEX. VCDROM provides two features necessary to support DOS CD-ROM applications:

- It emulates the presence of the MSCDEX
- It translates the DOS style IOCTLs into requests that the physical CD-ROM device driver can understand.

VCDROM provides only audio IOCTL support and not a full emulation of MSCDEX, as most DOS CD-ROM applications use the standard DOS interface for file system services and the MSCDEX interface for audio services only. Any application that calls MSCDEX for file system services will not run in a virtual DOS machine.

## 5.4.19  Virtual Video Device Driver

VDM video activity consists of both port I/O and memory operations. Virtual video device drivers are provided to support concurrent operations by multiple VDMs. A number of virtual video device drivers are provided by OS/2 Version 2.0, and are installed depending upon the physical display adapter types present in the system:

- VCGA.SYS provides support for CGA devices

- VEGA.SYS provides support for EGA devices

- VVGA.SYS provides support for VGA devices
- VSVGA.SYS provides support for SVGA devices
- VEGB.SYS provides support for VGA devices when configured to operate in EGA modes only
- V8514.SYS provides support for the display adapter 8514/A
- VXGA.SYS provides support for the XGA display adapter.

In the following discussion, the term *VVIDEO* will be used generically to refer to all virtual video device drivers.

By trapping all I/O operations and mapping a virtual video memory buffer to the region where a VDM expects physical video memory, VVIDEO insulates the physical hardware from background VDM activity. Only a foreground VDM's video operations are allowed to write directly to the physical hardware.

The major IBM adapter types (MONO, CGA, EGA, VGA, XGA and 8514) are fully supported by VVIDEO, in that it supports all standard modes of operation for multiple VDMs (concurrently, if all VDMs are running in text modes).

The following is a list of the video configurations supported and their default mode of operation:

| Table 3 (Page 1 of 2). List of Supported Video Configurations | | | | | | |
|---|---|---|---|---|---|---|
| Example Dis- plays | Size (in.) | Color | Adapter | Resolution | Memory | Colors/Grays |
| Mono | - | Mono | Mono | Supported as secondary display only | - | - |
| CGA | - | B&W | CGA | 640x200 | - | - |
| ECD Monitor | - | B&W | EGA | 640x350 | 64K | no 4-color support, uses EGA/Mono driver |
| ECD | - | Color | EGA | 640x350 | 128KB | 16 colors |
| - | - | - | - | 640x350 | 192KB | 16 colors |
| - | - | - | - | 640x350 | 256KB | 16 colors |
| Mono | - | B&W | EGA | 640x350 | any mem. size | no reverse video, blink or intense support |
| RGB | - | EGA | Color | 640x200 | any mem. size | 16 Colors |
| EGA | - | - | EGA w/2xx port config. | - | - | Not supported |
| 8503 | 12 | Mono | VGA | 640x480 | - | 16 Grays |
| 8512 | 14 | Color | VGA | 640x480 | - | 16 Colors |
| 8513 | 12 | Color | VGA | 640x480 | - | 16 Colors |

| Table 3 (Page 2 of 2). List of Supported Video Configurations | | | | | | |
|---|---|---|---|---|---|---|
| Example Displays | Size (in.) | Color | Adapter | Resolution | Memory | Colors/Grays |
| 8514 | 16 | Color | VGA | 640x480 | - | 16 Colors |
| 8503 | 12 | Mono | 8514/A | 640x480 | - | 16 and 64 Grays |
| 8512 | 14 | Color | 8514/A | 640x480 | - | 16 and 256 Colors |
| 8513 | 12 | Color | 8514/A | 640x480 | - | 16 and 256 Colors |
| 8514 | 16 | Color | 8514/A | 1024x768 | - | 16 and 256 Colors |
| 8503 | 12 | Mono | XGA | 640x480 | 512KB | 64 Grays |
| | | | | | 1MB | 64 Grays |
| 8513 | 12 | Color | XGA | 640x480 | 512KB | 256 Colors |
| | | | | | 1MB | 256 Colors |
| 8512 | 14 | Color | XGA | 640x480 | 1MB | 65536 Colors |
| 8515 | 14 | Color | XGA | 640x480 | 512KB | 256 Colors |
| | | | | 1024x768 | 512KB | 16 Colors |
| | | | | 640x480 | 1MB | 256/65536 Colors |
| | | | | 1024x768 | 1MB | 16/256 Colors |
| 8604 | 15 | Mono | XGA | 640x480 | 512KB | 64 Grays |
| | | | | 640x480 | 1MB | 64 Grays |
| 8507 | 19 | Mono | XGA | 1024x768 | 512KB | 16 Grays |
| | | | | 1024x768 | 1MB | 16/64 Grays |
| 8514 | 16 | Color | XGA | 640x480 | 512KB | 256 Colors |
| | | | | 1024x768 | 512KB | 16 Colors |
| | | | | 640x480 | 1MB | 256/65536 Colors |
| | | | | 1024x768 | 1MB | 16/256 Colors |
| 8514 Compatible | - | Color | XGA | 1024x768 | 512KB | 16 Colors |
| | | | | 640x480 | 1MB | 65536 Colors |
| | | | | 1024x768 | 1MB | 16/256 Colors |

As in a native DOS environment, the default setting of the equipment flags determines which display adapter is the primary display. VVIDEO examines this to determine which will be the primary display. The video signal for secondary displays is initially disabled, until such time as a MODE command or user application explicitly enables it.

### 5.4.19.1 VDM Screen-Switching

The OS/2 Version 2.0 session manager informs VVIDEO whenever a VDM's display state changes, ensuring that no more than one foreground VDM exists at any point in time, and that no VDMs are regarded as foreground processes while any other protected mode process, including the operating system shell, is in foreground. This mutually exclusive activity relationship between VVIDEO and OS/2 display drivers ensures screen integrity.

VDMs in background are switchable at any time since their state is maintained in memory by VVIDEO, rather than in the device itself. 32KB of virtual video buffer memory is allocated by VVIDEO for each VDM upon creation. This is the maximum buffer size addressable in any text mode. When the VDM is switched to foreground, the video buffer is reallocated to the maximum size supported by the adapter, with a limit of 256KB.

The act of switching a VDM from foreground to background or vice versa requires that the calling routine yield control, and hence there may be a time delay during the switch. In order to preserve the integrity of the video buffer, the VDM is suspended for the duration of the screen switch, to avoid any portion of the video state that was already copied to or from the hardware being changed before the switch is complete.

### 5.4.19.2 Foreground VDM Support

There are literally no limits to what a VDM can do with video hardware when running in foreground, since it has complete access to all ports and device memory through VVIDEO. I/O trapping is still operative, but only to "shadow" changes and ensure the ability to switch screens.

### 5.4.19.3 Background VDM Support

VDMs running in the background must always use virtual video memory, which is actually normal system memory that has been mapped into the VDM's video address space. In cases where the selected video mode (typically a graphics mode) requires multiple planes of video memory, normal system memory is inadequate to effectively virtualize video memory.

Whenever a VDM running the in background places the video hardware in a multi-plane graphics mode, virtual video memory is invalidated and if touched, results in the VDM being "frozen." If the VDM returns to a single-plane video mode (implying that it never accessed video memory), then its virtual video memory is validated once more. This approach allows VDMs to switch between different text modes entirely in background, without the risk of being frozen.

To support graphics operation in the background, VVIDEO must trap all video memory references and remap them to a set of simulated planes, or use some form of hardware-assisted virtualization that Presentation Manager and the other OS/2 processes know nothing about.

*Suspended Background VDM:* There are three cases in which DOS graphics applications may be suspended (receive no processor time) when running in the background:

1. A DOS multiplane graphics application that uses advanced graphics, such as 640x480x16 or 640x350x16, will be suspended, regardless of the graphics adapter installed, if any other DOS application is running in the foreground in a full-screen session.

2. A DOS multiplane graphics application that uses advanced graphics, such as 640x480x16 or 640x350x16, will be suspended when a Presentation Manager session is running in the foreground in XGA mode. Currently, this situation occurs even if you have an Extended Graphics Array (XGA) and a Video Graphics Array (VGA) adapter connected to your system.

3. A DOS multiplane graphics application that uses 1024x768x16 graphics mode will be suspended when a Presentation Manager session is running in the foreground in 8514/A mode.

Note that suspending DOS applications running in the background generally poses no problems unless the applications are timing-dependent, such as communications or process-control applications. In these cases, suspending them may cause them to fail. Avoid this situation by running these applications in the foreground in full-screen sessions only. If they are graphics applications, run them only in a single-plane mode, such as 640x200x2, 320x200x256, or 640x480x2, in full-screen sessions.

Note also that for WIN-OS/2 sessions, set the VIDEO_SWITCH_NOTIFICATION DOS setting to ON to avoid having Windows programs suspended when running in the background.

*Graphical Applications Programs Support:* OS/2 Version 2.0 supports a broad variety of display-adapter hardware as you can see from Table 4 on page 86. This allows OS/2 programs, DOS programs, and Windows programs to run in both windowed and full-screen sessions. OS/2, DOS, and Windows programs can run successfully in both the foreground and the background. Normally, the OS/2 user need not be concerned with the graphics modes that are used within a program, or whether a program will run successfully in a background session.

Some types of display adapters do, however, place limits on the ability of the OS/2 operating system to run certain classes of DOS and Windows graphics programs in the background. The limits exist because of the difficulty of providing virtual access to the display-adapter hardware without disturbing either the foreground session or other background sessions.

Under certain conditions, DOS applications that use graphical display modes will become suspended in background sessions when they attempt to write to the display.

Table 4 on page 86 gives you an overview of what happens with graphical applications programs in combination with different display adapters.

To determine under what conditions your applications will run in a background session in Table 4 on page 86 as described now:

1. Find your graphical display hardware in the "Type of Video" column.

2. Find your System Desktop Mode.

3. Read across the table to your application column.

For example, assume you have a DOS application using VGA mode on a system with VGA video. The application is in full-screen. To determine if the application will be suspended:

1. Find your type of video (VGA) in the "Type of Video" column.

2. Find your System Desktop Mode (VGA).

3. Read across the table to your application column (DOS Application, Matches Desktop Mode, Full-Screen).

The PF indicates that the DOS application runs when PM has control of the screen or when the application is in a foreground session.

| | | | | Windows Apps | | | | | DOS Apps | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | Full Screen (FS) | | | | | | | | |
| | | | | Matches Desktop Mode | | Using VGA Mode | | WIN-OS/2 Window | Matches Desktop Mode | | Using VGA Mode | |
| Type of Video | System Desktop Mode | PM Apps | VIO OS/2 Apps | A | B | A | B | (Win) | FS | Win | FS | Win |
| XGA | XGA | R | F | R | F | R | F | N/A | F | X | F | X |
| XGA | VGA | R | F | R | PF | R | PF | R | PF | PF' | PF | PF' |
| VGA | VGA | R | F | R | PF | R | PF | R | PF | PF' | PF | PF' |
| 8514 | VGA | R | F | R | PF | R | PF | R | PF | PF' | PF | PF' |
| 8514 | 8514 | R | F | R | F | R | R | N/A | F | X | R | R |
| EGA | EGA | R | F | R | PF | N/A | N/A | N/A | PF | PF' | N/A | N/A |
| CGA | CGA | R | F | R | R | N/A | N/A | N/A | R | R | N/A | N/A |

Table 4. Graphical Applications Programs Support under OS/2 Version 2.0

Legend:
R        Runs
PF       Runs when PM has control of the screen, or when the application is in a foreground session
PF'      Runs only when PM has control of the screen
F        Runs only when the application is in a foreground session
X        Not supported
N/A      Not applicable

**Note:** Column A indicates the use of a Windows display driver that suppresses background output. Column B indicates the use of a Windows display driver that does not suppress background output.

### 5.4.19.4 Device-Independent Pointer Services

VVIDEO provides services which allow the virtual mouse driver to define a pointer image and specify its position. Since the position must always be given relative to the physical dimensions of the VDM's screen, and since coordinates may change whenever the video mode is reset, the virtual mouse driver provides an entry point which is notified of such changes by VVIDEO. These interfaces are device-independent because dimensions are always given in terms of pixels or character cells, and not in predefined video mode identifiers. By separating the pointer-drawing code from the virtual mouse driver, mouse support becomes automatically available on any video adapter.

### 5.4.19.5 Font Support

At VDM creation time only a single font exists; this is either a default font contained in video ROM, or one specified by OS/2 if code page support is active. In the latter case, VVIDEO automatically loads the OS/2 code page font whenever the VDM restores the default ROM font by resetting the video mode.

Note that since the process of loading a font is essentially the same as entering a graphics mode, background VDMs will "freeze" if they attempt this.

### 5.4.19.6 Clipboard Support

To transfer VDM screen contents to the clipboard, VVIDEO provides two services: one to return the VDM's video configuration, and a second to copy the video contents to a shell-supplied buffer address. The shell then handles the transfer from this buffer to the clipboard.

## 5.5 Virtual Device Helper Services

In order to allocate, free and reallocate memory, virtual device drivers use the **virtual device helper** (VDH) memory management services. These services help the virtual device driver maintain a linear heap for each VDM. This heap is maintained as a linked list; each entry in this linked list refers to a linear region with attributes such as:

- Reserved
- Allocated
- Mapped
- Page granular
- Byte granular.

Memory allocation is always done in chunks of 4KB (page granular), but byte granular services are provided for handling instance data reservations and for memory allocation in the DOS environment.

### 5.5.1 Memory Management

VDH services provide the following support for memory management within virtual device drivers:

- Allocation/reallocation/freeing services for:

  - Global and per-VDM objects
  - Page and byte granular objects
  - Options for fixed, swappable allocations.

- Allocation of memory in DOS environment

- Reserve specified linear space

- V86-mode stack manipulation

- Mapping services:

  - Map to physical address
  - Map to linear address
  - Map to invalid address
  - Map to black holes (don't care) pages

- Copy/exchange services

- Block management services (pools of equal-size memory blocks)

- Query services:

  - Query the biggest linear space in a specified range
  - Query dirty bits for set of pages
  - Query amount of free virtual memory.

## 5.5.2 Semaphore Services

These services are used to synchronize a virtual device driver with another OS/2 process. If a virtual device driver blocks a VDM process, that VDM will not receive any simulated hardware interrupts until it becomes unblocked. VDH semaphore services are used to handle the following:

- Mutual exclusion and event semaphores
- OS/2 process to physical device driver communication
- Virtual device driver/physical device driver communication
- VDM event list management.

## 5.5.3 Freeze/Thaw Services

The virtual video device driver uses these services to freeze and unfreeze the operation of a VDM. This is typically required in response to a video mode switch in a background VDM, which would place the VDM in a video mode not supported when running in the background.

## 5.5.4 Timer/Priority Services

Timer services are provided to support the virtual programmable interrupt controller in the event of a time out occurring in an interrupt handler. Priority services are also used by VPIC to handle VDM scheduling priority.

## 5.5.5 Page Fault Services

A virtual device driver may register its own handler for page fault exceptions, in order to handle such events in an orderly manner. VDH services are provided in order to support this registration.

## 5.5.6 Other Services

- Error message and display
- Terminate VDM service

## 5.5.7 VDH Functions

The following list summarizes most of the VDH functions:

| *VDH API* | *Description* |
|---|---|
| **VDHAllocDosMem** | Reserve memory for stub DOS device driver |
| **VDHAllocMem** | Allocate small buffers |
| **VDHAllocPages** | Allocate linear space and commit backing storage |
| **VDHArmReturnHook** | Used to catch return from a VDHPushFarCall |
| **VDHArmSTIHook** | Receive control when current DOS session enables simulated interrupts |
| **VDHClearVIRR** | Clear interrupt request flag |
| **VDHClearSem** | Used to protect global structures |
| **VDHCloseVDD** | Terminate communication between virtual device driver and physical device driver |
| **VDHCopyMem** | Used by the EMM copy service and to copy device driver stub to VDM |
| **VDHExchangeMem** | Used by the EMM exchange service |
| **VDHFindFreePages** | Find a region of free linear space below 1MB + 64KB |
| **VDHFreeMem** | Deallocate small buffers |
| **VDHFreePages** | Deallocate memory objects |
| **VDHGetDirtyPageInfo** | Read and clear dirty-page bits (Dirty bits indicate whether a page has been written to) |

| | |
|---|---|
| **VDHInstallFaultHook** | Install hook for page faults |
| **VDHInstallIntHook** | Used to hook INT 67h (EMS interrupt) |
| **VDHInstallUserHook** | Register to get notification about VDM creation and termination |
| **VDHLockMem** | Verifies that a specified memory region is available and locks it |
| **VDHMapPages** | Used to map EMS windows to EMS objects or to unmap pages |
| **VDHNotIdle** | Resets VDHPostIdle |
| **VDHOpenVDD** | Establish communication between virtual device driver and physical device driver |
| **VDHOpenVIRQ** | Returns an IRQ handle for use with the other VPIC services |
| **VDHPopInt** | Remove ROM return address from user's CS:IP |
| **VDHPostIdle** | Put VDM into sleeping state. |
| **VDHPushFarCall** | Used by the EMM map and call service |
| **VDHPushInt** | Change control to the V86-interrupt handler |
| **VDHQueryConfigString** | Used to retrieve configuration data strings |
| **VDHQueryFreePages** | Determine amount of free virtual memory |
| **VDHQuerySysValues** | Determine VDM conventional memory size |
| **VDHReallocPage** | Change previous page allocation |
| **VDHRequestSem** | Used to protect global data |
| **VDHRequestVDD** | Requests the operation of a VDD |
| **VDHReservePage** | Reserve region of linear space below 1MB + 64KB |
| **VDHSetDOSDevic** | Register DOS device driver |
| **VDHSetVIRR** | Set interrupt request flag |
| **VDHUnreservePages** | Unreserve region of linear space below 1MB + 64KB |
| **VDHWakeIdle** | Awake VDM from sleeping state |
| **VDHYield** | Yield the processor to any other thread of equal or higher priority |

These functions are only valid when issued from within a module executing at privilege level 0; they cannot be issued by normal protected mode application processes.

## 5.6  VDM Termination

Virtual device drivers are responsible for a number of actions upon termination of a VDM. The nature of these actions is largely dependent on whether the VDM terminates normally or abnormally.

### 5.6.1  Normal Termination

A virtual device driver typically registers a VDM_TERMINATE hook with the Virtual DOS Machine Manager, which causes the virtual device driver to be informed when a VDM is terminated. When the VDM_TERMINATE hook is called, the virtual device driver is responsible for freeing all resources allocated on behalf of the terminating VDM.

## 5.6.2 Abnormal Termination

Virtual device drivers may experience a number of different error conditions, and must be able to act in order to recover from such errors where possible.

### 5.6.2.1 Errors Returned from VDH Services

Requests for VDH services may be refused by the operating system or may fail due to lack of resources. For example, a call to **VDHAllocMem()** may return 0, indicating that the memory allocation request cannot be satisfied.

During initialization of the virtual device driver or creation of a VDM such an error would cause the initialization or creation to be terminated. During execution of a DOS application, the virtual device driver should return control to the application, indicating the failure of the requested application service. If this cannot be done, the VDM must be terminated.

### 5.6.2.2 Bad Parameter Passed to VDH Service

A virtual device driver may make a service request with bad data, typically due to a bug in the device driver code; such events are likely during development and testing. For example, the virtual device driver may attempt to issue a **VDHFreeMem()** function call specifying an address which was not previously allocated using **VDHAllocMem()**.

Such errors are costly; since virtual device drivers operate at privilege level 0 and hence have access to all code and data in the system, it is impossible to localize the effect to a single VDM, or to be certain that the event has not corrupted data or control structures in the operating system kernel. In such cases, the Virtual DOS Machine Manager will halt the system.

### 5.6.2.3 Virtual Device Driver Consistency Failures

A virtual device driver may detect inconsistencies within its own state information. Such inconsistencies may be experienced in either global or instance state data. The virtual device driver must inform the user of the error. If the error can be isolated to the instance data of a single VDM, that VDM must be terminated. If the error is in global state data, it will be necessary to halt the system.

Note that halting the entire system is highly unfriendly behavior on the part of a virtual device driver. Very rarely, if ever, should such action become necessary. A virtual device driver should take all possible steps to isolate any state inconsistencies to a single VDM only.

### 5.6.2.4 Illegal Operation by a DOS Application

DOS applications running in VDMs may issue illegal instructions. For example, a DOS application may issue an OUT instruction to a port controlled by the virtual disk device driver, which does not support direct hardware control of the disk controller.

In such cases, the virtual device driver must inform the user of the error condition and either ignore the error or terminate the VDM and the application within it.

## 5.7  Summary

OS/2 Version 2.0 provides device drivers to handle the interface between the operating system and the hardware.  Physical device drivers are used by normal protected mode processes running OS/2 applications, while virtual device drivers are used by DOS applications running in virtual DOS machines.

Virtual device drivers provide a means of representing hardware devices to a DOS application in a virtual DOS machine, such that the devices appear to the application as though the application had sole control over the device.  In this way, MVDM allows DOS applications to issue instructions which directly manipulate hardware devices or the DOS system environment, while maintaining full protection of other applications in the system.  Virtual device drivers typically access hardware by requesting services from physical device drivers.

Virtual device drivers are used not only for shared hardware devices, but also for other aspects of the machine environment, such as BIOS, CMOS, and the (physical) programmable interrupt controller.  Through the use of virtual device drivers for these components, DOS applications may freely access and manipulate them without affecting other DOS applications or OS/2 applications in the system.

OS/2 Version 2.0 provides a number of standard virtual device drivers for the DOS system environment and common hardware devices.  Hardware vendors may develop virtual device drivers for their own hardware adapters.  Note that if a hardware device will be dedicated to one application (that is, sharing of the hardware is not required) then a virtual device driver is not needed; the normal DOS device driver will allow the application to access the hardware device as in a native DOS environment.

A virtual device driver operates at privilege level 0, and therefore cannot access operating system services via the normal application programming interfaces provided by OS/2 Version 2.0.  Instead, a set of virtual device helper services is provided to enable virtual device drivers to access system services.  Virtual device drivers may be written in a high-level language such as "C."

# Chapter 6. Memory Extender Support

Many popular DOS applications use memory extenders such as EMS and/or XMS to gain access to memory above the 1MB real mode addressing limit on the 80286, 80386, or 80486 processors. Such extenders allow DOS applications to have total code and data spaces larger than the available base memory, and to have very large code or data objects loaded into memory for enhanced execution speed. The standard configuration of OS/2 Version 2.0 provides both LIM EMS Version 4.0 (which includes backward compatibility with LIM Version 3.X) and LIMA XMS Version 2.0 functions for DOS applications running in virtual DOS machines.
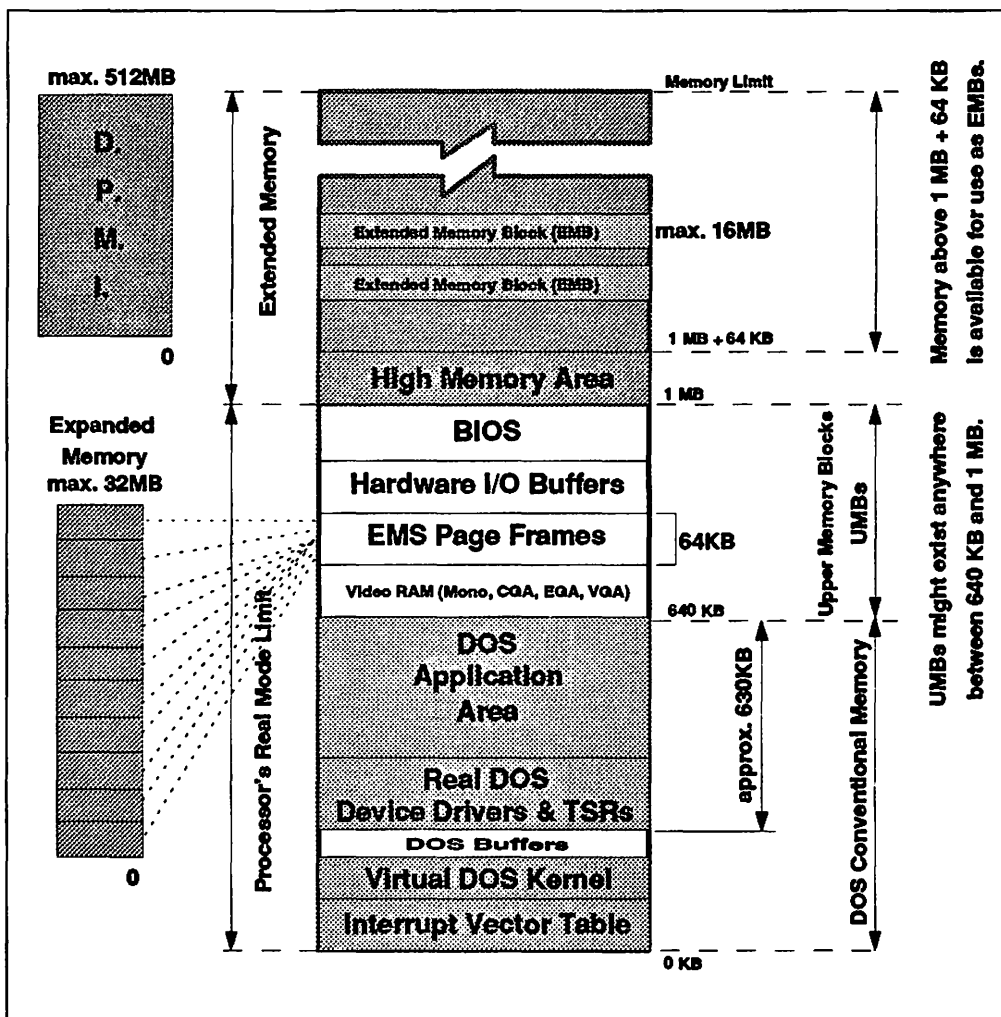


*Figure 26. General Overview of Different Types of Memory for DOS Applications*

This chapter describes the implementation of EMS and XMS support for virtual DOS machines. For those readers not already familiar with the architecture of these memory extenders, an overview is provided in Appendix D, "Memory Extender Architectures."

## 6.1 Expanded Memory Support

OS/2 Version 2.0 supports expanded memory according to the LIM EMS Version 4.0. Under DOS, special hardware is normally required to support EMS, although a number of software-based EMS emulation packages exist. MVDM supports EMS by mapping memory allocation requests into the linear process address space, using normal system memory. Hence no special hardware or software is required.

The OS/2 Version 2.0 LIM EMS emulation provides the following function:

- Implements all the required functions in the LIM EMS Version 4.0.

- Provides each VDM with a separate EMS emulation. Each VDM has its own set of expanded objects so that features like interprocess communication work as they would if each VDM were running on a different real 80386. Each VDM cannot affect the availability of objects in other VDMs or access memory in other VDMs.

- Provides for remapping of conventional memory (below 640KB) for use by programs like Windows 2.0.

- Provides configurable limits for how much EMS memory is available across VDMs, as well as a limit per VDM. The DOS Settings feature allows the user to override the per-VDM limit, subject to the constraint given by the overall limit.

- Supports multiple physical to single logical mappings. Different 8086 addresses can map to the same expanded memory object address. This is required by programs like Lotus 1-2-3.

- EMS can be removed and the operating system can run without loading EMS in any VDM session.

Memory objects are mapped into the V86 mode address space (below 1MB), so DOS applications can access very large address spaces. Applications access EMS services using the DOS interrupt INT 67h.

## 6.1.1 Virtual Expanded Memory Manager

EMS services are implemented under MVDM using a virtual device driver known as the **Virtual Expanded Memory Manager (VEMM)**, which offers a separate EMS emulation to each VDM. This implementation is accomplished by placing most VEMM control structures in a per-VDM data area outside the V86 mode address space. Each VDM has up to 255 handles, 15 alternate register sets, remappable conventional memory for operating system use, and a 16KB "raw" block size.

VEMM prehooks interrupt vector 67h through a VDH service to catch software interrupts for EMS services. Prehooking means that VEMM gains control before the V86 mode interrupt vector is called. VEMM also provides a V86 mode stub driver used to indicate to DOS applications that EMS is available. This stub must hook INT 67h so that applications can find a particular string in the header to determine if EMS is available. When, as in the typical case, applications have not also hooked INT 67h, VEMM handles service requests at prehook time. When INT 67 has been hooked, VEMM handles requests when the stub's hook calls it by doing an INT 67h from inside the stub.

To prevent VDM's from grabbing large amounts of EMS memory, there is a configurable default per VDM limit. VEMM depends heavily upon the memory

manager. EMS object allocation, reallocation, or deallocation is accomplished by requesting corresponding services from VDH services. Most VEMM creation time setup is postponed until the first INT 67h service request is made. Figure 27 on page 95 shows the flow of control when a DOS application makes an EMS service request from within a VDM:

1. INT 67h service requests are trapped by the Virtual DOS Machine Manager and routed to VEMM.

2. The VEMM makes the appropriate requests to VDH services to allocate or manipulate the EMS object.

### 6.1.1.1 Expanded Memory Manager Control Flow
During the initialization of the VDM the VDM Manager loads and initializes the EMS DOS stub device driver into the VDM address space.
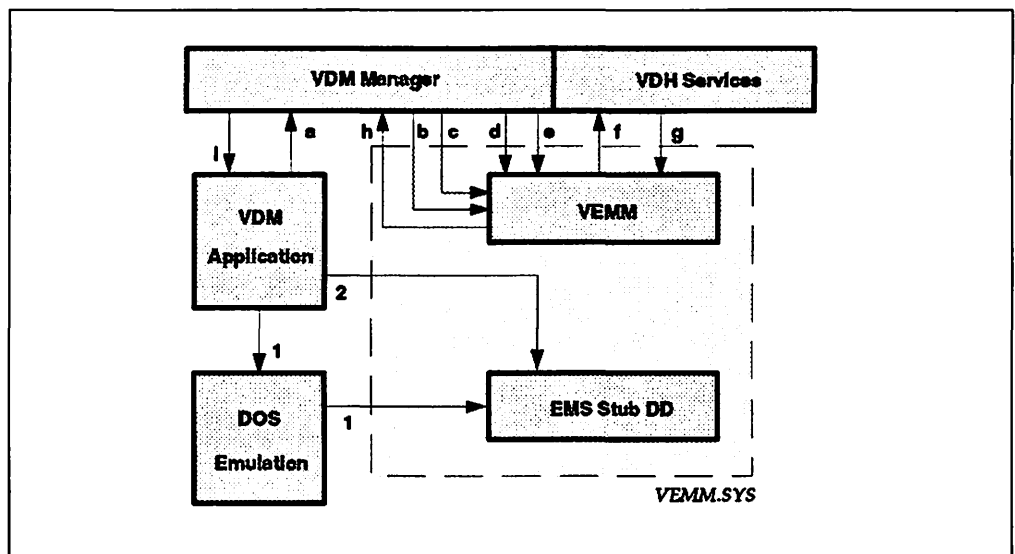


*Figure 27. Expanded Memory Manager Control Flow*

The VDM Application can use either of two methods to test for the existence of the VEMM:

1. Issue an open request (INT 21h Function 3DH) using the guaranteed device name of the VEMM driver. If the open function succeeds, either the driver is present or a file with the same name coincidentally exists on the default disk drive. To rule out the latter, the application can use IOCTL(INT 21h Function 44H) subfunctions 00h and 07h to ensure that VEMM is present. Don't forget to use INT 21H Function 3Eh to close the handle that was obtained from the open function, so that the handle can be reused for another file or device.

2. Look for a special signature in the EMS DOS stub device driver which signals the VDM Application that EMS is available. This search is done by using the address that is found in the INT 67h vector to inspect the device header of the presumed VEMM which is, in this case, the fooling EMS DOS stub device driver. Interrupt handlers and device drivers **must** use this method. If the VEMM seems to be present, the name field at a special offset of the device header contains a special string. This method is not only avoiding the relatively high overhead of an VDM DOS open function but is nearly foolproof. However, it is somewhat less well behaved because it involves inspection of memory that does not belong to the application.

The only task of the EMS DOS stub device driver is to tell the VDM DOS application that EMS is available. As soon as this is done the regular EMS business can start as explained in the following points:

a. The VDM Application issues a INT 67h service request.

b. The VDM Manager loads the VEMM.

c. The VDM Manager initialization, creation, termination calls for EMM-objects.

d. The VDM Manager traps the VDM application's INT 67h service request and routes it to VEMM.

e. The VDM callback for V86 call with far return.

f. The VEMM requests corresponding services from the VDH services:

   • Creation/termination handler registration

   • INT 67h pre-hooking

   • Linear address reservation

   • Memory management.

g. The result is returned.

This constellation also allows a VDM application to hook INT 67h.

Note that unlike most virtual device drivers, VEMM does not have a corresponding physical device driver. Instead, VEMM manages its memory using the OS/2 Version 2.0 operating system kernel's memory management services. EMS object allocation, reallocation, or deallocation is accomplished by requesting corresponding services from the operating system's memory manager. For example, when an application requests the allocation of an expanded memory object, VEMM asks the memory manager to allocate a memory object in linear memory outside any VDM.

### 6.1.1.2  Structure
The VEMM module consists of:

• An *Initialization Component* that determines the default size at boot time

• A *Creation Component* that initializes per-VDM structures when a VDM is created

• A *Router* that decodes application INT 67h (and routes the call to a service routine) and 30 service routines with associated sub-services.

Each VDM has a 255-entry EMS handle table used to keep track of the size and allocation of expanded memory objects, 16 register sets that indicate which parts of the expanded objects are mapped into the VDM address space, and save tables to save register set contents. Only one register set is active at a time. That active register set indicates the actual page table contents. Switching register sets or restoring a saved register set resets all aliases in the windows to those indicated by the new register set. Unmapped pages are set to "black hole" memory. The memory manager's page size for all these operations is 4KB. VEMM makes the adjustment for its 16KB page size.

### 6.1.1.3 Initialization

VEMM is typically installed at system initialization time, via a DEVICE= statement in CONFIG.SYS, as shown below:

```
DEVICE=C:\OS2\MDOS\VEMM.SYS 4096, 2048
```

To prevent VDMs from using all available memory, there is an overall limit on the amount of EMS memory, and a default limit for each VDM to prevent a VDM from requesting all available EMS memory. The defaults for these limits are specified in the DEVICE= statement for VEMM. The default limit for each VDM may be overridden using the DOS Settings feature.

Setting the overall limit to zero disables EMS in all VDMs, regardless of the per-VDM value. Setting the per-VDM value to zero disables EMS in all VDMs unless their entries on the Presentation Manager desktop specify a non-zero EMS size. Setting the EMS size to zero for an entry on the desktop disables EMS for that application only. Most users need never change the default value. DOS settings for frame position, and the size of extra mapping regions above and below 640KB may be configured by the user; see Chapter 11, "DOS Settings."

Upon installation, an initialization routine within VEMM supplies the entry point addresses of VEMM creation and close routines to the Virtual DOS Machine Manager.

Most VEMM setup is postponed until the first INT 67H service request is made. Only remappable conventional memory is set up before that time. This assures that other virtual device drivers have a chance to reserve ROM and device memory areas.

### 6.1.1.4 VDM Creation

Upon creation of a VDM, a VDH service is used to get the EMS size for that VDM. This will return a string if the DOS program's entry on the desktop has an associated EMS size. If not defined, the default size retrieved from CONFIG.SYS at system initialization is used. If EMS size is not zero, the following steps are then performed:

1. Two mappable windows are located and reserved.

2. Memory is mapped into the low window.

3. Interrupt 67h is hooked using a VDH service.

4. The V86 mode device driver stub is loaded.

5. An initial block of the handle table is allocated.

Upon VDM creation, VEMM allocates a block of memory in the area between 256KB and RMSIZE[3] and maps it into the VDM address space. VEMM requests VDH services to locate the largest free address range in the V86 mode address space above 640KB that is available for the mappable window. VEMM then reserves the largest range available that is at least 64KB and no more than 128KB in size, and is a multiple of 16KB. If an extended BIOS data area is present, the returned free range will be below this area so that BIOS cannot be inadvertently mapped away.

---

[3] The RMSIZE statement in CONFIG.SYS specifies the maximum size of a VDM's address space; values up to 640KB are allowed.

Waiting until creation time to reserve this memory allows virtual device drivers with actual hardware to claim their addresses first, since VEMM can place its memory at any available address. A consequence of this technique is that the space is reserved only for the VDM being created. It could be in a different location or be a different size for other VDMs.

VEMM performs mapping by requesting the operating system's memory manager to alias linear space inside a mappable window in the V86 mode address space to a memory region outside the V86 address space. The application can then access this part of the expanded memory object.

The VEMM virtual device driver prehooks interrupt vector 67h through a VDH service to catch software interrupts for EMS services. Prehooking means that VEMM gains control before the V86 mode interrupt vector is called. VEMM also provides a stub driver, the sole function of which is to indicate to DOS applications that EMS is available.

VEMM then arranges for the loading of a stub device driver in the VDM. This driver provides a header within the V86 mode address space which can be read by an application searching for the name of the real mode EMS driver. It also responds to a few simple requests made to its strategy routine, basically replying that it is present and ready. The stub driver does not actually process EMS service requests; these are handled by VEMM.

### 6.1.1.5 Routing
The router receives notification from the Virtual DOS Machine Manager when an application issues an INT 67h request. The router checks the request to ensure that it is valid, and then causes an exception which is routed to the Virtual DOS Machine Manager. The Virtual DOS Machine Manager then reflects the interrupt back into the VDM's interrupt vector table. This technique is necessary since interrupt vector hooks are only allowed after application code has been executed. The V86 mode interrupt vector for INT 67h causes another exception which is routed to the Virtual DOS Machine Manager which then calls VEMM.

The EMS *Alter Map and Call* service allows an application to have VEMM remap memory to place a routine within the V86 mode address space, call that routine and then remap memory to its previous state again after the routine issues a *far return*. This call can occur recursively; the application code that is called can in turn use the *Alter Map and Call* service.

VEMM does the initial remapping and stores the after-call remapping information on the client's stack. VDH services are used to call the application's routine and intercept the return. VEMM supplies the Virtual DOS Machine Manager with a V86 mode address to call and a VEMM handler which is invoked when the application routine does a *far return*. After the routine returns, VEMM restores the original mapping saved on the client's stack.

The *Remap and Jump* service is similar but does not require interception of an application routine's return or the saving of a mapping. Remapping is done and the CS:IP is changed to jump to the address provided by the application.

Information calls involve at most a quick search of structures. Structures are maintained to provide information about handles, allocations, and VEMM capabilities. Handle directory services are also provided. The number of pages VEMM reports as available is the minimum of the number of pages the VDM has left in VEMM pages and the amount the memory manager estimates is available.

### 6.1.1.6 Protection

A pseudo-random key is produced with the first protection call made by a VDM and also for the first protection call made after a key was returned. This key is given to the application which made the call that caused its generation. OSEnabled can be reset only by the owner of the key. The key owner can also return the key. OSEnabled indicates whether or not protected functions can be executed. The key will be generated by operations on the current time to ensure that the key changes, even for multiple calls between successive timer ticks.

## 6.1.2 EMS Object Mapping

Mappable windows are located by asking the Virtual DOS Machine Manager to provide free linear regions after other virtual device drivers have claimed the address ranges required for their hardware. The base window (region from 256KB to RMSIZE) is mapped to an expanded object at VDM creation time. This window is used as normal memory by DOS or DOS applications, but can also be remapped by applications that wish to do so.

Some applications assume mappable regions begin on 17KB boundaries, although this is not part of the EMS specification. OS/2 Version 2.0 follows this undocumented convention.

There are multiple techniques for saving and restoring mappings in LIM. Save tables and copies of parts of register sets copied to application memory can be used to save and restore mappings. All contain a pairing of physical segment and <handle, logical page> pairs. Saving of mappings is done by segment, handle, and logical page entries to the buffer in which the save is performed. For saves that save the entire mapping, the register index need not be stored. Mappings are restored by making a set of aliasing calls to the memory manager, and copying the new mapping into the active register set.

Illegal mappings are mapped to **black hole** pages. A black hole page is a page that does not cause faults, but which need not store values written to it. Black hole pages can be implemented as invalid addresses that float on the bus, ROM pages if there is no ROM caching, a wasted physical memory page, or a discardable page.

Structures returned to the client will use physical pages rather than segments since these are smaller for the client to store and are simpler to check for validity when restored. All save structures held by the V86 client use a checksum to detect tampering by the V86 client.

LIM allows an application to reallocate the special handle that contains conventional memory, thus allowing the expanded memory to be reused. This is supposed to be done only by an operating system program that knows the special handle is handle 0, but may conceivably be done by any application. Note that applications can deallocate the DOS memory area. If they do this and fail to restore it, COMMAND.COM is unable to reload and the VDM will terminate. This behavior is identical to a native DOS environment.

### 6.1.2.1 Register Sets

Application requests to map pages into a register set are handled by storing the new mappings in the register set data structure. A call is made to the memory manager to alias pages or set them to a black hole for unmapped pages.

The current register set is changed to a new register set by aliasing linear memory through memory manager calls according to the new registers and changing the current register set variable. Other calls allow saving and restoring register sets from an application-supplied array similar to Save/Restore above.

### 6.1.2.2 Allocating/Deallocating Objects

Upon receiving an application request to allocate, reallocate, or deallocate an EMS object, VEMM transforms the request into corresponding calls to the OS/2 Version 2.0 memory manager. Each EMS object is allocated as a separate memory object in a linear address range in the VDM's process address space, but outside the V86 mode address space. The memory manager returns the start address and size of each EMS object to VEMM, which then updates its EMS handle table accordingly.

Allocations are made by selecting a free EMS object handle; a free handle has a start address of zero. VEMM then requests the memory manager to allocate the required memory, and records the start address and size of the object as returned by the memory manager. The total allocation size for each VDM and the total allocations across all VDMs are maintained so that the maximum allocation size is not exceeded. If an allocation is of size zero, no actual allocation is made and a non-zero address and zero size are recorded in the handle entry.

When a deallocation request is made, the address in the handle is changed to 0 and the memory manager is called to free the allocation. Reallocation requests are serviced by passing on the request to a VDH service and recording the new size and start address. Since reallocations may lead to object movement, pages mapped from the object are unmapped before reallocation and remapped afterward.

When an application reallocates to zero, VEMM has the memory manager deallocate the memory object, and changes the handle table entry so it has zero pages with a meaningless non-zero address to indicate the handle is still in use. Objects of size zero are allowed in VEMM, but not in OS/2, so VEMM will free the memory but retain its own data for the object handle. When a non-zero reallocation is performed on the object, a new object is transparently allocated.

LIM allows an application to deallocate memory that is mapped into the current register set, alternate register sets, or save maps (all internal structures that save mappings). EMS is silent about what should happen if an application touches this mapped memory after deallocation. Since 8086 applications are generally allowed to search through the address space without harm, these deallocated pages should be remapped to a black hole.

Searching through all 255 SaveMaps and 15 non-current register sets is expensive even with optimizations. Exhaustive search slows deallocations and shrinking reallocations, and keeping track of the locations of mappings slows mapping operations. Therefore, upon deallocation or shrinking reallocation, only the current register set is checked for deallocated pages. Stored registers (255 SaveMaps and 15 RegSets for the VDM) will not be checked until they are reacti-

vated. When an invalid page is found during remapping, it is simply remapped to the black hole.

## 6.1.3 Per-VDM Data Allocation

Handle table entries, register sets, save tables, and handle names all require a good deal of space if used fully. Most of these data structures typically do not require their maximum possible size. For this reason, they are allocated dynamically by VEMM in order to reduce memory utilization.

Memory for a register set is allocated when clients allocate the register set. An array of 16 pointers will address buffers for allocated register sets; these pointers are null for free register sets that applications may yet allocate.

The handle table, handle name table, and save tables are all allocated with a directory structure. A directory is an array of pointers to allocation blocks; each allocation block contains enough space for multiple entries. This allows a specific entry to be found by specifying the block and entry offset within the block. Since each can have at most 255 entries, both the block and offset can fit in a single byte.

An allocated handle entry contains an index for its associated save table or name (if used). For unallocated objects, the index is zero. The smallest free handle will be allocated when a handle is needed, thus requiring less memory to be allocated for the handle table.

The name table is kept in packed form. When an entry is freed, the last entry is moved into the free spot, thus reducing space requirements.

Save table entries are larger and generally have short lifetimes. These will be allocated in the same way handles are allocated (that is, where the smallest available is allocated first). For all three of these tables, when new entries are needed and all blocks are full, a new block is allocated.

## 6.1.4 Problems with Expanded Memory

EMS requires a 64KB block of contiguous free memory in the address range 640KB to 1MB for its page frame. As we can see from Figure 26 on page 93 this memory range is shared with BIOS, hardware buffers and device drivers. If your application reports that no EMS memory is available, even if you have used the *DOS Settings* option *EMS_MEMORY_LIMIT* to set a non-zero value, it could be that a 64KB page frame location could not be found. See 6.2, "Expanded Memory (EMS) and Upper Memory (UMB)" on page 102 on how to resolve this contention.

If a program returns an error due to insufficient expanded memory, the following points should be addressed:

- Ensure that CONFIG.SYS and/or AUTOEXEC.BAT do not start unnecessary programs that use expanded memory.

- Change the DEVICE= statement for VEMM.SYS in CONFIG.SYS to provide more expanded memory to every VDM. Alternatively, use the *EMS Memory Size* DOS Settings to allocate more memory to a specific VDM. See Chapter 11, "DOS Settings."

VEMM for OS/2 was designed to install for EMS only when the DOS application makes its first EMS request. This was done for two reasons. First, it saves time

and memory. Second, it gives the DOS drivers or applications a chance to access adapters in the EMS page frame address space. OS/2 V2.0 will recognize adapters with the ROM signature header, but will not see adapter RAM/MMPIO areas.

VEMM determines that it has space available during DOS Session creation but a DOS program/driver could cause EMS to not install by accessing memory in the page frame **before** calling the EMS driver. Lotus 1-2-3 Version 2.3 was found to access the adapter space BEFORE calling EMS. On some machines, this caused no EMS to be present for Lotus 1-2-3.

To satisfy this situation, VEMM has than been changed (after GA) to not wait for the first EMS call. This should clear up the EMS problems related to programs accessing the adapter address space (X′C0000′-X′E0000′). However, it should be known that it is now possible for VEMM to claim areas that could actually be adapter address space. In general, the user should be aware of this situation. To resolve this we suggest to use the MEM_EXCLUDE property to force EMS not to use the desired address space.

## 6.2  Expanded Memory (EMS) and Upper Memory (UMB)

The following section applies to both VDM DOS Emulation and DOS VMB.

Expanded Memory Specification (EMS) is discussed in detail in Chapter 6, "Memory Extender Support" on page 93. One requirement of EMS is a *page frame* in real memory between 640KB and 1MB (hex addresses X′A0000′ to X′FFFFF′). Since IBM systems reserve addresses X′A0000′ to X′BFFFF′ for video, and X′E0000′ to X′FFFFF′ for BIOS, the EMS page frame is normally restricted to addresses between X′C0000′ and X′E0000′. This area can also be used for Upper Memory Blocks, where DOS device drivers and resident programs can be loaded. This frees up valuable space below 640KB for conventional DOS programs.

Unfortunately, memory between X′C0000′ and X′E0000′ is also needed for Option Adapter ROM and RAM. Indeed it can be difficult or even impossible to configure EMS on a system which has several intelligent adapters installed.

There is really no solution to this problem (sometimes known as "RAM Cram") under DOS. However OS/2 Version 2.0 provides an elegant alternative.

Normally a VDM inherits a memory map which mirrors the actual system hardware configuration; adapter ROM and RAM addresses set by the PS/2 Reference Diskette (or adapter switches on non-Micro Channel systems) are mapped into the VDM address space and are not available for EMS or UMBs.

But since the VDM occupies *virtual* memory this can easily be changed. The DOS Settings value *MEM_INCLUDE_REGIONS* parameter releases adapter addresses for use as EMS or UMBs. In most cases this can be set to the complete X′C0000′-X′DFFFF′ range.

If a VDM uses an adapter **directly** (usually via a DOS device driver), the adapter ROM or RAM address must not be specified in *MEM_INCLUDE_REGIONS*. Addresses of adapters used indirectly by the VDM (through OS/2 Version 2.0) may be included. For example, the full X′C0000′ to X′DFFFF′ range may be included on a SCSI-based PS/2, even though the SCSI adapter ROM may occupy

X'D8000' to X'DFFFF'. The DOS VDM does not directly access the SCSI adapter so it does not need SCSI ROM mapped into its address space. It can still access files on SCSI disks via the OS/2 Version 2.0 file system.

Another example could be a 3270 connection adapter. Depending on the setup, it could occupy 8KB of memory (for example, X'DE000' to X'E0000'). If you are using Extended Services and Communications Manager to establish a DFT connection to your /370 system, you could release this memory for use by DOS applications and specify this address frame in the Include Region. Of course, if you want to use a DOS-based emulator, such as *Personal Communications/3270 Version 2.0*, you can't include this area, as the DOS application and its device driver want to access this adapter directly.

---
**Note**

The *MEM_INCLUDE_REGIONS* parameter should be entered as shown above, using 5-digit hex addresses (not 4-digit segment addresses, as is often the case). Also, note that the range is inclusive - you must specify the second address as (for example) X'DFFFF', not X'E0000'. The parameter is not validity-checked when entered. If an invalid parameter is saved, the default (no include region) is used when the VDM is initialized; no error message is generated.

---

In summary, a typical DOS VDM may have a 64KB EMS page frame and 64KB of UMBs (or 128KB of UMBs) regardless of the hardware adapters installed. Such a configuration is not possible under PC DOS.

## 6.3 Extended Memory Support

The OS/2 Version 2.0 Multiple Virtual DOS Machines architecture provides support for the LIMA Extended Memory Specification Version 2.0 specification, in a similar manner to that provided for LIM EMS Version 4.0, using normal system memory and emulating XMS functions. The following discusses how MVDM support for the extended memory specification has been implemented.

The extended memory specification manages three different kinds of memory:

- High Memory Area (HMA)

- Upper Memory Blocks (UMBs) in the Upper Memory Area (UMA)

- Extended Memory Blocks (EMBs).

Each of these areas is discussed as they relate to the implementation of expanded memory support for VDMs in OS/2 Version 2.0. Figure 28 on page 104 below shows where these memory areas or blocks reside in memory.
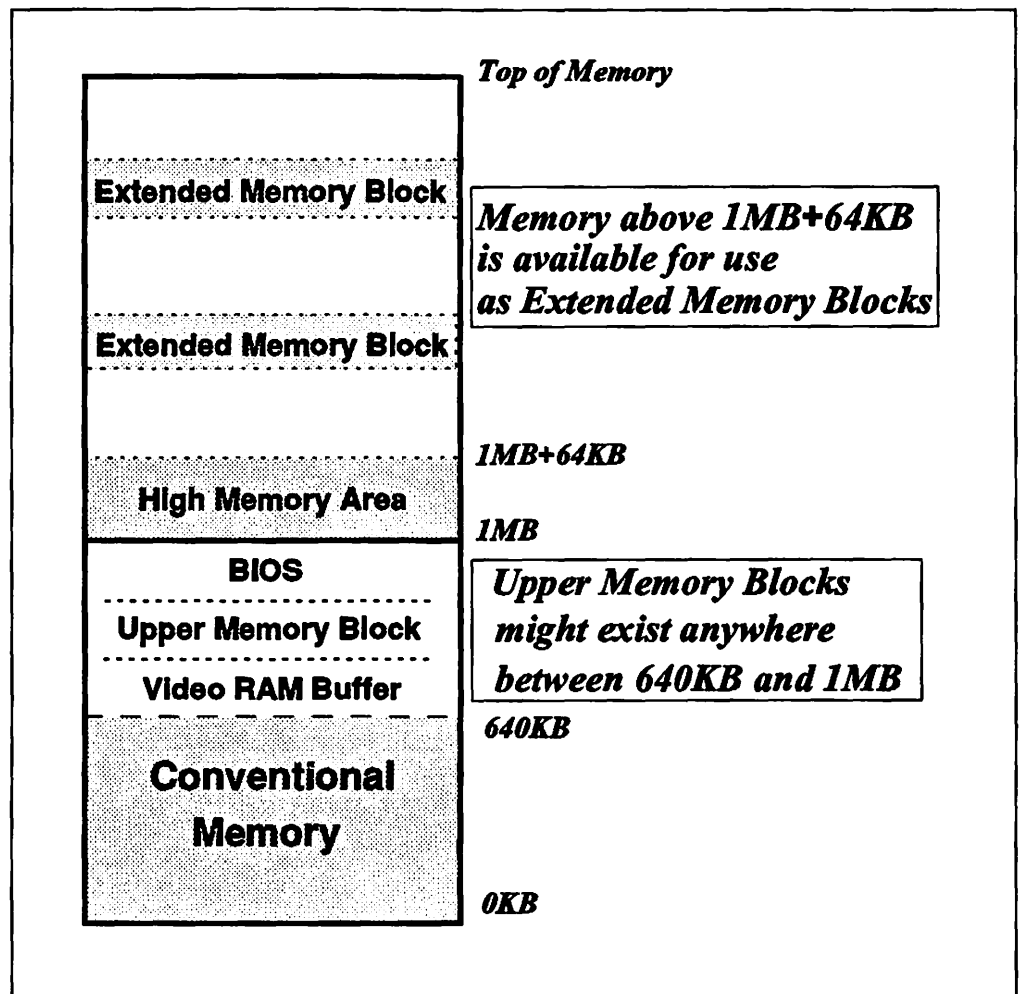
*Figure 28. Memory Map of Areas Supported by Extended Memory*

For more information regarding the Expanded Memory Specification, refer to Chapter 9, "DOS Protected Mode Interface" on page 181.

The OS/2 Version 2.0 LIMA XMS emulation provides the following functions:

- Implements all LIMA XMS Version 2.0 functions.

- Provides each VDM with a separate XMS emulation. Each VDM has its own High Memory Area, Upper Memory Blocks and Extended Memory Blocks; hence features such as interprocess communication work as if each VDM was running on a different real 80386. A VDM therefore cannot affect the availability of extended memory objects in other VDMs or access memory owned by other VDMs.

- Provides configurable limits for how much XMS memory is available across all VDMs as well as a limit per-VDM. The DOS Settings feature can override the per-VDM limit, subject to the constraint given by the overall limit, and can disable XMS altogether for a particular VDM if its installation conflicts with the program being run in the VDM.

- XMS can be removed and the operating system can run without loading XMS in any VDM session.

Applications which use extended memory may use the XMS support in the same manner as in a native DOS environment. In addition, portions of the DOS oper-

ating system, device drivers and TSR programs may be loaded into extended memory, thereby conserving memory within the DOS application address space for application use.

Note that older applications which access extended memory directly, rather than through an extended memory manager, may not be compatible with the XMS support under MVDM. For example, Microsoft Windows Version 2.x cannot make use of extended memory in a VDM.

## 6.3.1 Extended Memory Manager

XMS services are implemented under MVDM using a virtual device driver known as the **Virtual Extended Memory Manager (VXMM)** which is represented by the file VXMS.SYS (VXMS). VXMM provides a separate XMS emulation for each VDM by placing most VXMS control structures in a per-VDM data area outside the V86 mode address space. The amount of memory available to a VDM, the number of handles, and the existence of Upper Memory Blocks (UMBs) are all configurable parameters which may be altered on a per-VDM basis.

The VXMM hooks interrupt vector 2Fh in order to announce its presence to applications. It also provides a V86 stub device driver (XMM 3X device driver), which indicates to DOS applications that XMS is available, but more importantly acts as a V86 mode interface between the application and the VXMM proper.

VXMM depends heavily upon the memory manager. XMS object allocation reallocation, and deallocation are accomplished by requesting corresponding services from the memory manager. When an application requests the allocation of a block of extended memory, for example, VXMS asks the memory manager to allocate a memory object in linear memory outside any VDM. Reallocation and deallocation are handled similarly.

All EMS functions are executed by calling the XMS Control Function, the address of which can be obtained by a call to INT 2Fh. Arguments are passed in registers.
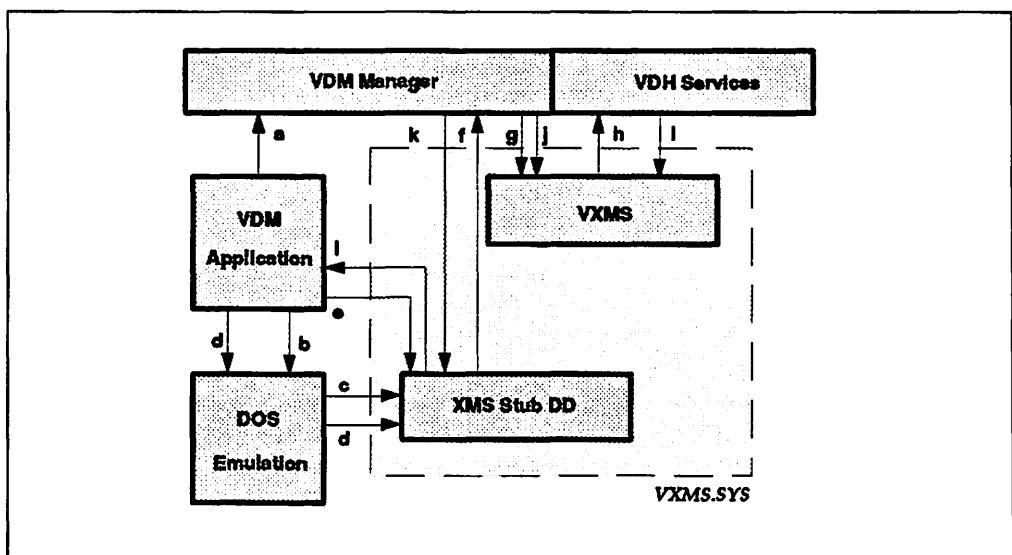


Figure 29. Extended Memory Manager Control Flow

During the initialization of the VDM the VDM Manager loads and initializes the XMM DOS stub device driver into the VDM address space. As soon as there is a XMS request the VDM Manager loads the the XMS virtual device driver VXMS.

a. The VDM Application issues a INT 15h service request. VXMS directly hooks INT 15h for function 87h and 88h. It does not provide any services through these calls but makes sure that no program tries to use extended memory directly. INT 15h function 88h will respond that no normal extended memory is available. Programs that want to use extended memory directly by using INT 15 and RAMdisks (electronic disks) using INT 15 won't work. The MS DOS RAMDRIVE for DOS 5.0 does work because it uses XMS instead of INT 15.

b. The VDM application issues INT 2Fh to determine if an XMS driver is installed.

c. The VDM application issues INT 2Fh to determine if an XMS driver is installed.

d. Next the VDM application issues a INT 2Fh to obtain the address of the XMS driver's control function. As soon as the VDM Application got the address of the XMS driver's control function it can use any of the functions and call the XMM DOS stub device driver directly.

e. The VDM application calls the XMS driver's control function to access all of the XMS functions.

f. The XMM DOS stub device driver calls breakpoint traps by the VXMM Control Function.

g. The VDM Manager initialization, creation, termination calls for XMS-Objects. The VDM Manager traps the VDM application's INT 15h service request and routes it to VXMS as well as XMS control function requests for XMS memory.

h. The VXMS requests corresponding services from the VDH services:

- Creation/termination handler registration
- INT 67h prehooking
- Linear address reservation
- Memory management
- Obtaining configuration information.

i. The result is returned.

Like VEMM and unlike most other virtual device drivers, VXMS.SYS does not have a corresponding physical device driver. Instead, it depends heavily upon the OS/2 Version 2.0 memory manager. XMS object allocation, reallocation and deallocation are accomplished by requesting corresponding services from the operating system's memory manager. For example, when an application requests the allocation of a block of extended memory, VXMM requests the memory manager to allocate a memory object in linear memory outside the V86 mode address space. Reallocation and deallocation are handled similarly.

### 6.3.1.1 Structure

The VXMS.SYS module consists of:

1. An *initialization component* that initializes global structures and reads the global configuration at boot time.

2. A *creation component* that initializes per-VDM structures, reads per-VDM configuration values, and installs the DOS device driver stub when a VDM is created.

3. A *router component* that receives control from the control function contained in the stub device driver, and dispatches the call to an appropriate service routine. In addition, the router function hooks interrupt vector 15h upon the first non-version-query call to VXMM, as required by the specifications, in order to:

   a. Preserve the state of the A20 line across block copies (service AH = 87h).

   b. Respond to service AH = 88h (Query Extended Memory) by reporting that there is no extended memory available.

4. A number of *service routines*, which perform the required XMS functions.

Applications request XMS services by calling a subroutine contained within the VXMM, known as the Control Function. The VXMS virtual device driver hooks interrupt vector 2Fh in order to announce its presence to applications.

### 6.3.1.2 Initialization

VXMS.SYS may be loaded at system initialization time by using a DEVICE = statement in CONFIG.SYS, as shown below:

```
DEVICE=C:\OS2\MDOS\VXMS.SYS 8192, 2048
```

This statement should be placed in CONFIG.SYS *after* the DEVICE = statement for VEMM.SYS, since VXMM queries VEMM to ensure that no conflicts occur in memory allocation.

The DEVICE = statement uses parameters to specify the total amount of available XMS memory, and the default limit for each VDM. In the above example, the overall limit is set to 8MB and the limit for each VDM is set to 2MB.

The virtual device driver VXMS.SYS can be configured as follows.

```
device = {path} vxms.sys {options}
```

The options are of the form "/keyword = value":

/XMMLIMIT=g,i    Sets the global (system-wide) maximum memory usage of the VXMS.SYS driver to g kilobytes, and a per-VDM maximum of i kilobytes. These values should be large enough to accommodate an automatic 64KB allocation in each VDM for the HMA. Values are restricted to the range 0..65535 (= 64Meg).

The values of g and i are rounded up to the nearest multiple of 4.

Specifying i = 0 suppresses XMS installation in all VDMs unless specifically overridden by a VDM-specific configuration string. (See below.)

Default: /XMMLIMIT = 4096,1024

| /HMAMIN=d | Sets the minimum request size (in kilobytes) for an HMA request to succeed. Values are restricted to the range 0..63. |
|---|---|
| | Default: /HMAMIN=0 |
| /NUMHANDLES=n | Sets the number of handles available in each VDM. Each handle occupies eight bytes. Values are restricted to the range 0..128. |
| | Default: / NUMHANDLES=32 |
| /UMB | Instructs XMM to create Upper Memory Blocks |
| | Default: off |
| /NOUMB | Instructs XMM not to create Upper Memory Blocks |
| | Default: /NOUMB |
| | All other keywords are ignored. Case is ignored. |

These options affect all VDMs, but can be overridden by a VDM's configuration strings. The same option names are available to VDMs (without the prefixing slash), except that XMMLIMIT only takes one numeric argument, corresponding to the value i above. The value g above cannot be changed once XMM is installed.

If a value of i=0 was specified on the DEVICE= line, to create a VDM with XMS installed, specify a configuration string "XMMLIMIT" with a non-zero value. Conversely, to have no XMS installed, specify a configuration string "XMMLIMIT" with a value of zero.

If UMBs are being used, it is crucial that VXMS.SYS be the last device driver loaded, for VXMS.SYS reserves all available addresses between 640KB and 1Meg for use as UMBs. Hence, any device drivers which reserve pages in that region (for example, VEMM) will not be able to install.

VXMS.SYS will fail to install if some other device driver has already reserved the region from 1MB to 1MB+64KB.

The overall limit comprises the only relationship between XMS memory objects in different VDMs, and is imposed to prevent XMS from acquiring all available memory. The default overall limit is 4MB, and the default limit for each VDM is 1MB. The default limit for each VDM can be overridden by installing an application on the desktop and choosing to specify the XMS size with the DOS Settings feature (see Chapter 11, "DOS Settings").

Setting the overall limit to zero disables XMS in all VDMs regardless of the per-VDM value. Setting the default limit for a particular VDM to zero disables XMS in all VDMs unless their start list entries specify a non-zero XMS size. Setting the XMS size to zero for an entry in the start list disables XMS for that application's VDM only. Novice users need never change the default values.

In addition to memory sizes, the number of handles and the presence or absence of Upper Memory Blocks are all configurable parameters which may be altered on a per-VDM basis using the DOS Settings feature.

Upon installation, an initialization routine within VXMS.SYS supplies the addresses of the VXMS.SYS VDM-creation and close routines to the Virtual DOS Machine Manager.

### 6.3.1.3 VDM Creation

Upon creation of a VDM, a VDH service is used to get the maximum XMS size for that VDM. This will return a string if the program's entry on the desktop has an associated VXMS size. If the per-VDM size is not defined, the default retrieved from CONFIG.SYS at initialization time will be used. If VXMS size is not 0, the following steps are then performed:

1. Upper Memory Blocks (UMBs) are found and reserved.

2. The High Memory Area (HMA) is reserved.

3. The real mode device driver stub is loaded.

4. The handle table and UMB list are initialized.

To find an available linear region to use for UMBs, VXMS requests VDH services to locate the largest free address range in the V86 mode address space above 640KB. VXMS reserves all the pages returned until the call fails.

VXMS requests the OS/2 Version 2.0 memory manager to allocate the 64KB region immediately above 1MB for use as the High Memory Area. The way in which this is accomplished depends upon a number of variables; see 6.3.2, "High Memory Area (HMA)" on page 110 for further details.

Waiting until creation time to reserve this memory allows virtual device drivers with actual hardware to claim their addresses first, since VXMS's UMBs can be placed at any available address. A consequence is that the space is reserved only for the VDM being created; it could be in a different location or be a different size for other VDMs.

VXMS then arranges for the loading of a stub device driver in the VDM. This driver serves three purposes:

* The device driver header can be read by an application searching for the name of the real mode VXMS driver. It responds to all device requests with "done" without actually doing anything.

* The device driver's initialization code attaches VXMS to interrupt vector 2Fh. Attaching to vector 2Fh must be delayed until after the virtual DOS environment has completed hooking all of its interrupts.

* The device driver contains the VXMS *control function*; calls to XMS services are not performed by calling a software interrupt, but rather by calling a V86 mode *far* subroutine called the control function. Moreover, XMS specifications require the control function to have a particular physical layout. Hence, the control function is placed in a DOS device driver so that it may have the layout required by the specifications and can transfer control to the virtual device driver code (the router function).

The stub device driver is used to transfer control to the router function. A DOS application invokes XMS functions by calling the control function as a *far* procedure, the address of which can be obtained by a different INT 2Fh call. In response to such a request, the INT 2Fh interrupt handler returns the address of the control function in the device driver stub. The Control Function then calls the protected mode VXMS entry point, and the router obtains control.

The interrupt hook cannot be performed by the VXMS creation function, since the virtual DOS environment does not establish its interrupt hooks until after all virtual device driver creation code has completed. DOS device driver initialization code is called after the interrupt vectors are set; therefore delaying the

hooking of vector 2Fh until DOS device driver initialization time succeeds in hooking the vector.

### 6.3.1.4 Routing

The router receives control from the control function within the stub device driver, as described above. After checking that the XMS service request is valid, the router calls the appropriate protected mode service routine, which in turn requests the OS/2 Version 2.0 memory manager to allocate and manipulate XMS objects.

Information calls involve at most a quick search of structures. The number of kilobytes VXMS reports as available is the minimum of the number of kilobytes the VDM has left before it hits its per-VDM XMS memory usage limit, the number of kilobytes all VDMs have left before hitting the system-wide memory usage limit, and the amount the memory manager estimates is available.

## 6.3.2 High Memory Area (HMA)

VXMS requests that the operating system's memory manager reserve the region of memory between 1MB and 1MB + 64KB, so that it may use that region for simulating the A20 address line wraparound. This region of memory is called the High Memory Area (HMA).

When the processor's A20 address line is disabled, the HMA is mapped to the first 64KB of conventional memory. When the A20 address line is enabled, the mapping depends on whether the HMA is in use. If the A20 address is not enabled, the HMA is mapped to black hole memory. Black hole memory can safely be accessed by a VDM, but values written to it cannot be retrieved (ROM or invalid physical addresses, for example). If the HMA is in use, VXMS requests the memory manager to alias a linear region inside the HMA to a memory object outside the V86 mode address space, which has been specially allocated for this purpose.

DOS Emulation code may reside in the HMA; this is specified by including the following statement in CONFIG.SYS:

```
DOS=HIGH
```

OS/2 Version 2.0 installation places this statement into CONFIG.SYS as a default, and the operating system is thus installed such that DOS Emulation runs in the HMA. The only drawback to using the HMA for DOS Emulation code is that applications are prevented from using the HMA. This is not usually a serious problem, since few programs require use of the HMA. It is recommended that DOS Emulation code is loaded in the HMA as this will free base memory for application use.

Note that if XMS size is less than 64KB for a VDM, the HMA is not emulated. All requests for the HMA will fail.

## 6.3.3 Upper Memory Blocks (UMBs)

VXMS attempts to reserve all unreserved pages in the region of memory between 640KB and 1MB; this region is often termed the **Upper Memory Area (UMA)**. The address ranges reserved in this manner will be used to simulate **Upper Memory Blocks (UMBs)**. Note that this allocation scheme requires that VXMS be the *last* device driver loaded; any device drivers loaded after VXMS will not be able to reserve any addresses in the UMA.

When a UMB is not in use, its corresponding range of addresses is mapped to a black hole. When it is in use, the range of addresses corresponding to the UMB being allocated is mapped to a memory object outside the V86 mode address space which is allocated for this purpose. This is similar to the technique used to map objects in the HMA.

VXMS uses a delayed UMB allocation scheme. Unlike conventional XMS implementations, no UMBs are allocated until the first UMB request. Upon receiving the first UMB request, VXMS queries the UMB region to determine which address ranges are available, and reserves those ranges. This technique supports memory mapped devices which lie in the same region from which UMBs are taken. Advanced users can use *Include/Exclude Regions* in the DOS Settings feature to tell VXMS which ranges are not to be used.

Note that by default, all UMBs are *owned* by the DOS Emulation kernel and are *not* available for application use. If an application wishes to use UMBs, the DOS = NOUMB statement must be included in CONFIG.SYS or the application can't get any UMB because they are already used by DOS. Alternatively, the ownership of UMBs for a single VDM may be enabled or disabled using the *DOS owns UMBs* setting; see Chapter 11, "DOS Settings."

VXMS allows coexistence with EMS services, in that it queries VEMM before reserving address ranges, so that VEMM may reserve the space it requires for its frame. As such, it is possible that an application using both EMS and XMS services will execute and function correctly.

### 6.3.3.1 DOS Device Drivers
DOS device drivers may be loaded into UMBs, thereby conserving memory within the 640KB DOS application space; this support is functionally compatible with that provided by DOS 5.0. Loading a DOS device driver into a UMB requires a number of additional statements in CONFIG.SYS; an example is given in Figure 30.

```
DEVICE = C:\OS2\VXMS.SYS
DEVICEHIGH = SIZE = 1A00 C:\OS2\MDOS\ANSI.SYS
DOS = UMB
```

*Figure 30. CONFIG.SYS - Loading Device Drivers into UMBs*

The first statement causes the VXMS virtual device driver VXMS.SYS to be loaded at initialization time. The second statement causes the ANSI.SYS device driver to be loaded into a UMB. The SIZE parameter ensures that the device driver is loaded into a UMB of the required size for its operation; if a UMB of this size cannot be allocated, the device driver is automatically loaded into low memory.

For DOS device drivers loaded into a specific VDM using *DOS Device Drivers* in the DOS Settings feature, the SIZE parameter is also supported. Specifying a SIZE parameter in DOS Settings will cause the device driver to be loaded into a UMB if possible; if UMBs are not present or if a sufficiently large UMB cannot be allocated, the device driver will be loaded into low memory.

Note that device drivers are always loaded into the largest available UMB; hence, in order to achieve efficiency in the utilization of UMBs, device drivers should be loaded in order of size, from largest to smallest. This is achieved by

placing the DEVICE= statements in CONFIG.SYS, or names of the device drivers in the *DOS Device Drivers* setting, in that order.

The third statement allocates ownership of UMBs to the DOS kernel, and prevents applications from accessing UMBs. This statement sets the default for all VDMs; if an application running in a VDM requires UMBs, the default may be overridden for that VDM using the appropriate DOS Settings function.

Note that some device drivers may not function correctly in a UMB if they rely on having all memory above them available for their use. If incorrect operation of a device driver is experienced in a UMB, the value of the SIZE parameter should be increased by modifying the DEVICEHIGH statement in CONFIG.SYS or by altering the appropriate DOS settings for the VDM.

### 6.3.3.2 TSR Programs

TSR programs may also be loaded into UMBs in order to conserve DOS application space. TSR programs such as APPEND, which are loaded by default when a VDM is started, are loaded into a UMB where possible, thereby saving approximately 6KB of memory. Loading a TSR program into a UMB is performed from the DOS command line or from a batch file using the LOADHIGH command, as shown in Figure 31.

```
LOADHIGH progname <program parameters>
```

*Figure 31. LOADHIGH Command - Loading TSRs into UMBs*

Note that parameters for TSR programs are supported by the LOADHIGH command.

TSR programs which rely on having all memory above their location available for their use may not execute correctly when loaded in a UMB. In such cases, the TSR must be loaded into low memory.

## 6.3.4 Extended Memory Blocks (EMBs)

Extended Memory Blocks (EMBs) reside in the region above 1MB, and are therefore not directly accessible from DOS applications running in V86 mode. The only way a DOS application can access EMBs is by using the VXMS service *Move Extended Memory Block*. VXMS then requests the memory manager to remap the EMB into low memory, from which it may then be accessed by the application. Each Extended Memory Block is allocated as a separate memory object with linear addresses outside the V86 mode address space.

Note that memory requests for UMBs and EMBs are made by applications in units of paragraphs and kilobytes, whereas the memory manager allocates in 4KB pages. VXMS rounds all allocation requests up to the nearest integral page multiple before passing the request on to the operating system's memory manager.

## 6.3.5 Allocating/Deallocating Memory

Application requests to allocate, reallocate, or deallocate Extended Memory Blocks are translated into corresponding call to the memory manager. A free handle table entry, indicated by a start address of zero, is selected and updated to contain the start address and size of the memory object. The total XMS memory size for the system and for each VDM is checked at this point to ensure that these limits are not exceeded.

Reallocation requests are serviced by passing the request to a VDH service and recording the new size and start address. When an application reallocates to zero, VXMS requests the memory manager to deallocate the memory object, and changes the handle table entry so it has zero pages with a sentinel non-zero address to indicate the handle is still in use. Objects of size zero are allowed in VXMS, but not in OS/2, so VXMS will deallocate but retain its own data for the handle. When a non-zero reallocation is performed on an object of size zero, a new object is transparently allocated by the memory manager.

All allocations (and reallocations) are rounded up to the nearest integral page multiple. Since there is no facility for telling the applications how much memory was actually reserved, the extra memory is wasted.

### 6.3.5.1 High Memory Area

There is only one HMA per VDM, so a single pointer suffices to manage the state of the HMA. If the pointer is zero, then the HMA is not in use. Otherwise, the pointer contains the linear address of the block of memory being used to simulate the HMA. Whether the HMA region is mapped to this block of memory is determined by the state of the simulated A20 line; see section 6.3.2, "High Memory Area (HMA)" on page 110.

When a request for the HMA is made, the pointer is tested against zero. If the pointer is non-zero, then the HMA is in use, and the request fails. Otherwise, the pointer is set to a newly allocated 64KB block of memory, and the HMA region is mapped to this block if the A20 address line is active.

When the HMA is released, the block of memory is freed, and the pointer is reset to 0. The HMA is then mapped to a black hole if the A20 address line is active. Once an HMA is freed, all information previously stored therein becomes invalid.

### 6.3.5.2 Upper Memory Blocks

For UMB allocation, a linked list of reserved address ranges is maintained. This list contains information about the start address of each reserved range, the base address of the physical memory block allocated and mapped into address range, and the length of the block. If the base address is zero, then the address range is not in use and is instead mapped to a black hole.

Allocations are made by searching through the list to find an address range for which the base address is zero and which is large enough to satisfy the request. If the address range exceeds the required size, it is split into two parts and a new object is allocated to hold the unused portion.

Deallocations are made by searching the list to find a structure whose starting address matches the one being deallocated. The physical memory into which the address range was mapped is freed, and the address range is instead remapped to a black hole. Finally, the newly freed object has its base address

set to zero to signify that it is not in use. It is then coalesced with any adjacent free blocks.

### 6.3.5.3 Extended Memory Blocks

Each VDM has a fixed table of up to 255 EMB handles, the exact number of which is under user control. Each entry of the table describes a single Extended Memory Block.

Each entry contains a field which records the number of active locks on the Extended Memory Block. Locking a handle prevents the corresponding Extended Memory Block form being reallocated or freed, and also prevents the base address from changing. As part of its function, the *lock* subfunction returns the 32-bit base address.

If an allocation is of size zero, no physical memory allocation is requested, but a sentinel non-zero address and zero size are recorded in the handle entry. The lock count for the newly created Extended Memory Block is reset to zero.

When a deallocation request is made for an Extended Memory Block with zero lock count, the address in the handle is changed to zero, and the memory manager is called to free the memory.

## 6.4 Problems with Extended Memory

If an application in a VDM encounters an error due to insufficient extended memory, the following points should be checked:

- Ensure the overall limit and the limit for the VDM are large enough to accommodate the amount of extended memory required by the application.

- Ensure that the DEVICE = statement for VMXS.SYS is in CONFIG.SYS.

- Ensure that the expanded memory driver VEMM.SYS, is not using all of the available memory. The amount of memory allocated to VEMM may be reduced by changing the parameters of the DEVICE = statement for VEMM to something less than that specified (or less than the default which is 4MB). If necessary, VEMM command may be disabled by removing or remarking out the DEVICE = statement in CONFIG.SYS.

- Ensure that CONFIG.SYS and/or AUTOEXEC.BAT do not start unnecessary programs that use extended memory.

If a program does not start and displays a message such as *High Memory Area (HMA) already in use*, the HMA may be freed by disabling the DOS = HIGH statement in CONFIG.SYS. If the statement is DOS = HIGH, UMB then the statement should be changed to DOS = UMB.

## 6.5 Summary

MVDM provides support for applications which use the LIM EMS Version 4.0 and LIMA XMS Version 2.0 memory extenders to access more than 640KB of memory. The memory requested is allocated from OS/2 Version 2.0 system memory, and is managed by the operating system kernel; special hardware is *not* required. Each VDM has its own copy of EMS or XMS memory objects, and the objects of one VDM are protected from access by another VDM.

Support for these memory extenders is provided by two virtual device drivers, VEMM.SYS and VXMS.SYS. Unlike most virtual device drivers, these drivers do not have corresponding physical device drivers, but access the operating system's memory manager to handle memory allocation requests from applications.

MVDM supports the loading of DOS device drivers and TSR programs into XMS Upper Memory Blocks, in order to reduce memory consumption below the 640KB line, thereby leaving more base memory for applications. Loading of these programs into UMBs is supported by the DEVICEHIGH statement in CONFIG.SYS and the LOADHIGH command included in AUTOEXEC.BAT or executed from the command line.

# Chapter 7. Installing and Migrating Applications

Installing DOS and Windows applications under OS/2 V2.0 is in most cases very similar to installing them in their native environments. However, since OS/2 V2.0 is a true multitasking operating system, we should ensure that the installation programs do not introduce incompatibilities with existing programs. The flexibility in tailoring the virtual DOS machine environment for these programs, also gives us an opportunity to easily tune DOS settings to suit our needs.

OS/2 V2.0 has a utility to help users place their application icons onto the desktop after they have been installed. The utility uses information stored in a migration database that is shipped with OS/2 V2.0.

Systems administrators can use another utility to create their own migration database to install their corporation's unique applications.

This chapter discusses the installation of DOS and Windows applications under OS/2 V2.0. It also shows how to use the application migration utilities shipped with OS/2 V2.0.

We describe the use of the migration program in this chapter, and show an example of how to use the PARSEDB utility to create a customized migration database.

## 7.1 Installing DOS Programs

Application installation methods vary widely in the DOS world. Some installations involve nothing more than copying the software from diskette to the hard disk. In more complex applications, the install procedure may check the workstation configuration (both hardware and software), implement copy protection and modify system files.

For most DOS applications, installation under OS/2 Version 2.0 is simply a matter of starting a DOS full-screen or windowed session and following the instructions supplied with the package as if the installation was taking place on a DOS system. However, some may not work correctly because of the special requirements of the installation program.

### 7.1.1 General Installation Procedure for DOS Programs

To install a new DOS program:

1. Read the installation instructions for the DOS program.

2. Select the *OS/2 System* icon.

3. Select the *Command Prompts* icon.

4. Select the *DOS Full Screen* icon.

5. Type the installation command as specified in the installation instructions on the command prompt.

   For example:

   a:install

6. Follow the instructions on the screen.

7. When installation is complete, close the Command Prompts folder.

To add an icon to the desktop, you can use either:

- The *Program* template in the *Templates* folder (refer to Chapter 10, "Running DOS Applications.")
- The *Migrate Applications* program (refer to 7.5, "Migrating Programs.").

## 7.1.2 Installation Programs with Special Requirements

Some DOS application installation programs will not run properly in the OS/2 V2.0 virtual DOS machine, or will not install the program correctly. Some of the possible problems are as follows:

1. The installation program attempts to verify the version of DOS that is running, and receives a response that it cannot understand. One example is Lotus 1-2-3 Release 3.1+.

   The OS/2 V2.0 virtual DOS machine environment can be customized to return a DOS version in response to the installation program query and thus bypass the problem. This is accomplished by changing the *DOS_Version* parameters in the DOS Settings facility, which is accessed by pressing the *DOS Settings* push button on the *Session* page of the *Settings* notebook.

   The DOS Settings facility and the available settings are described in detail in Chapter 11, "DOS Settings."

2. The copy protection or user registration scheme implemented by the application bypasses DOS and directly accesses the disk. The OS/2 V2.0 system will not allow the installation program to do this, since it may interfere with other applications, and will terminate the virtual DOS machine in which it is running.

It may therefore be necessary to perform the installation in a native DOS environment by rebooting the workstation with a DOS boot diskette. When the installation is complete, the workstation can be rebooted under OS/2 V2.0 and the application added to the Workplace Shell.

## 7.2 Planning Hard Disk Partitions

If the workstation is booted from a DOS diskette in order to perform the DOS application install, the installation is restricted to those logical drives that have been formatted as FAT. This is because logical HPFS drives cannot be accessed in a native DOS environment.

When the system is booted with DOS 5, HPFS drives are not assigned drive letters and are invisible to the user. If DOS 4.01 with CSD UR35284 is used to perform the boot, drive letters will be assigned to all drives, whether HPFS or FAT. However, you cannot access the files on the HPFS drives. With earlier versions of DOS, even FAT drives that lie beyond the first HPFS drive will not be assigned drive letters.

Some installation programs store directory information into control files that are used at run time. For example, WordPerfect** 5.1 records the path to its subdirectories. On a hard disk with both FAT and HPFS logical drives, this can cause the installation program run in native DOS to record drive assignments that are wrong when the application is started from a virtual DOS machine.

Consider the following example of a hard disk setup for dual boot or with Boot Manager:

Table 5. Drive Letter Assignment. *This table shows the way drive letter assignments may differ on a mixed FAT and HPFS hard disk when booted under DOS and OS/2 Version 2.0*

| Partition/Type | Drive letter under OS/2 | Drive letter under DOS |
|---|---|---|
| Primary Partition 1, Boot Manager | none | none |
| Primary Partition 2, FAT | C: | C: |
| Extended Partition, HPFS | D: | none |
| Extended Partition, FAT | E: | D: |

Note that FAT drive in the extended partition appears as drive E: to the OS/2 Version 2.0 virtual DOS machine, but appears as drive D: when booted under DOS. Consequently, if the DOS application is installed on that partition when the system was booted under DOS, the drive letter it records in its control files will be D:. When the system is rebooted under OS/2 V2.0 and the application is run from a virtual DOS machine, the application will be looking to drive D:, which under OS/2 V2.0 is assigned to the HPFS drive of the extended partition. This will cause the application to miss the information it is seeking.

The user may be able to change the control file information and correct the error through the application. However, if the system is booted from DOS and the application is started, it will again be looking for the wrong drive.

In order to avoid this confusion we recommend that HPFS logical drives be placed last. In the above example, the FAT and HPFS logical drives in the extended partition should be transposed. This will allow the drive letters for the FAT partitions to be the same regardless of whether the workstation is booted from DOS or OS/2 Version 2.0.

More details on hard disk management can be found in *Chapter 4 of the OS/2 2.0 Installation Guide.*

## 7.3 Installing Windows Programs

Windows programs installation are usually performed from the DOS command prompt, or the Windows Program Manager.

To install a new Windows program:

1. Read the the program installation instructions.

2. To install the program from a DOS command prompt:

   a. Select the *OS/2 System* icon.

   b. Select the *Command Prompts* folder.

   c. Select *DOS Full Screen*.

   d. Enter the installation command as specified in the installation instructions.

   For example:

   a:setup

e. Follow the instructions on the screen to complete the installation.

3. To install the program from the Program Manager:

    a. Select the *OS/2 System* icon.

    b. Select the *Command Prompts* folder.

    c. Select *WIN-OS/2 Full Screen*.

    d. Select *Run* from the *File* pull-down on the action bar.

    e. Enter the installation command as specified in the installation instructions.

       For example:

       a:setup

    f. Follow the instructions on the screen to complete the installation.

To add an icon to the desktop, you can use either:

• The *Program* template in the *Templates* folder (refer to Chapter 10, "Running DOS Applications.")
• The *Migrate Applications* program (refer to 7.5, "Migrating Programs.").

If you use the *Migrate Applications* option, the program icon will be placed in the *Windows Programs* folder or the *Additional Windows Programs* folder on the desktop.

The *Migrate Applications* program always sets up Windows programs to run in a WIN-OS/2 window session if possible. WIN-OS/2 window sessions cannot be used for programs that have to be run in real mode, such as Windows 2.0 programs. Therefore if you use the *Migrate Applications* utility on Windows 2.0 programs, open the *Settings Notebook* to the *Sessions* page and select the *WIN-OS/2 full screen* radio button.

## 7.4 AUTOEXEC.BAT and CONFIG.SYS

The installation program for a DOS or Windows application may alter the AUTOEXEC.BAT (usually to modify the PATH statement) and CONFIG.SYS (to modify the FILES and BUFFERS statements or add a DEVICE statement). Usually the copies edited are the ones found in the root directory of the OS/2 V2.0 boot drive. If the option is given the user should not allow the installation program to make the modifications before reviewing the changes. We recommend that you back up both of these files prior to running an installation. After installation, inspect the date and time stamps of the files to see if they have been modified.

The most common change made to the AUTOEXEC.BAT file is to the PATH statement, so that the program being installed can be started from any subdirectory. The function of the PATH statement can be provided in a virtual DOS machine by using the *Path and file name* and *Working directory* fields of the *Program* page of the *Settings* notebook.

Since the CONFIG.SYS is used for every virtual DOS machine, the device driver that an installation program adds will be loaded for all VDMs and consume system resources unnecessarily. We recommend that when the DOS application is added to the Workplace Shell the device driver statement be added via the *DOS_DEVICE* setting in the DOS Settings facility. This setting is accessed by

pressing the *DOS Settings* push button on the *Session* page of the *Settings* note-book.

## 7.5  Migrating Programs

OS/2 V2.0 provides a migration database (DATABASE.DAT) that contains param-eters and settings for commonly used DOS, Windows and OS/2 programs. This binary database file is used by the *Migrate Applications* program to place the program icons onto the desktop and customize their *Settings* notebooks to the recommended values.
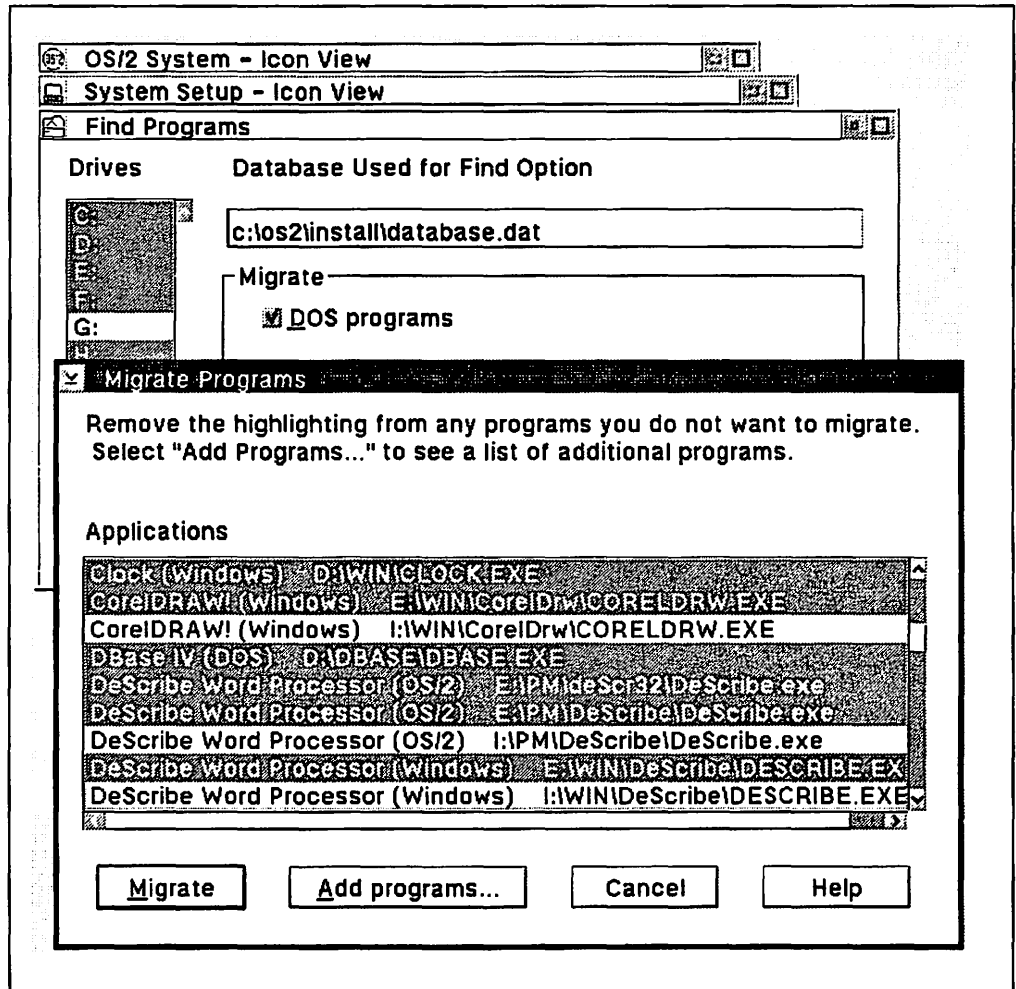
*Figure 32. The Migrate Applications Windows*

To use the *Migrate Applications* program, follow these steps:

1. Locate and select the *OS/2 System* icon on the desktop.

2. Select *System Setup*.

3. Select *Migrate Applications*.

   The *Find Programs* window appears. The *Database Used for Find Option* field displays the default database (\OS2\INSTALL\DATABASE.DAT). The *Migrate Applications* program compares programs on the hard disk with the list of programs in the database and places any that match in a DOS, OS/2, or Windows programs folder on the desktop.

4. From the *Drives* list, deselect the drives which should not be searched. The default is to search all drives.

5. Deselect the types of programs that should not be migrated in the *Migrate* type check boxes. The default is to migrate all the listed programs.

6. Select *Find....* The *Migrate Programs* window appears. Programs are listed in the *Applications* list box.

7. If your program is not on the list:

   a. Select the *Add Programs...* push button. The *Add Programs* window appears. Programs are listed in the Available Programs list.

   b. Highlight a program. The Working directory and Program title fields are filled in. Type a new title if required.

   c. Type the appropriate parameters for the selected program in the *Parameters field*.

   d. Select the types of programs to migrate in the *Program* type field. The *Migrate Applications* program creates the *Additional Programs* folders based on the types of programs specified.

   e. Select *Add*. The program moves to the *Selected Programs* list.

   f. Select *OK*. The *Migrate Programs* window appears.

8. Select *Migrate* to migrate all the selected programs. When migration is complete, the *Find Programs* window reappears.

9. Select *Exit*.

The *Migrate Applications* program creates a *DOS Programs* folder and a *Windows Programs* folder. The programs in these folders have the recommended pre-selected settings that work best for your programs' performance.

If the *Add Programs* option was used, an *Additional DOS Programs* folder and an *Additional Windows Programs* folder will also be created. The programs in these folders have default settings. If these programs do not run correctly, specify other settings. See Chapter 11, "DOS Settings" on the use of the settings.

## 7.6  Creating a Customized Migration Database

Some corporate users have an installed base of unique or custom-written DOS and Windows applications. These programs will not be listed in the migration database (DATABASE.DAT) that is supplied with OS/2 V2.0. The PARSEDB.EXE program is provided by OS/2 V2.0 to allow a system administrator to build a customized migration database that can be used to set up these unique applications on the Workplace Shell desktop.

## 7.6.1  PARSEDB

A customized migration database is created as follows:

1. Create the input text_database file

2. Run PARSEDB to create the binary database file.

To start PARSEDB, type the following statement from a command prompt:

```
PARSEDB [path] DBTAGS.DAT [path] text_database [path] binary_database
```

where:

- DBTAGS.DAT is the file name that contains the definitions for the tags used to define the DOS settings
- text_database is the name of the file that contains the program settings for a specific DOS, OS/2 or Windows program
- binary_database is the name of the new migration database file.

The text_database file is the main input file for PARSEDB that has to be created.

For example, type the following statement to create a new database named MYDATA.DAT:

```
PARSEDB E:\OS2\INSTALL\DBTAGS.DAT MYDATA.TXT MYDATA.DAT
```

Note that you **must specify a file name for the binary database file** to prevent the PARSEDB utility program from overwriting the default database file DATABASE.DAT.

When creating the text_database file, each program must have the following migration information:

**NAME**     Name of the file that runs the program.

**TITLE**     Program object name that appears below the icon.

**TYPE**     DOS, Windows or OS/2

**ASSOC_FILE**

File name associated with the file name specified in the Name field. Use this file name to uniquely identify the program.

**DEF_DIR**     Directory that the program is installed into.

ASSOC_FILE and DEF_DIR can have NULL values; NULL values must be included when defining the program if specific values for these fields cannot be provided.

When creating MYDATA.TXT, group the settings for a given program on consecutive lines. Use blank lines to mark the end of a program's settings. Begin nonblank lines with a token. The tag file DBTAGS.DAT defines valid token settings, limits, and default values for various DOS properties.

Here is the listing of DBTAGS.DAT:

```
//                       +--------------------+
//                       | NOTE TO TRANSLATOR |
//                       +--------------------+
// Do not translate keywords or data types and delete these lines when done.
// ===============================================================================
// dbtags.dat -- DOS setting "tags" used by PARSEDB and MIGRATE.  Each "tag"
// consists of an index, a keyword, and a data type.
//               +------------------------------------------------+
//               | DO NOT EDIT THIS FILE UNDER ANY CIRCUMSTANCES! |
//               +------------------------------------------------+
// ===============================================================================
// Allows BASIC-style comments.
// -------------------------------------------------------------------------------
1   REM                     NOP


// -------------------------------------------------------------------------------
// Required "fake" DOS settings.
// -------------------------------------------------------------------------------
2   NAME                    STR    // Filename used to execute application
```

```
3   TITLE                  STR    // Icon (desktop) title
4   TYPE                   BYTE   // Application type
                                  // Valid settings: DOS
                                  //                  Windows
                                  //                  OS/2
                                  //                  custom (for Microsoft
                                  //                  Windows apps which
                                  //                  must run full-screen)
5   ASSOC_FILE             STR    // Associated file (NULL if one isn't
                                  // known)
6   DEF_DIR                STR    // Default installation directory (NULL
                                  // if there isn't one)


// --------------------------------------------------------------------
// Other "fake" DOS settings.
// --------------------------------------------------------------------
7   FOLDER                 STR    // Name of folder to create and/or put
                                  // the application icon in
8   PARAMETERS             STR    // Application's command line parameters
9   WORK_DIR               STR    // Application's working directory


// --------------------------------------------------------------------
// "Fake" Windows settings
// --------------------------------------------------------------------
10  WIN_FILES              STR    // Files to be copied to the WinOS2
                                  // directory when the application is
                                  // migrated
11  COMMON_SESSION         BOOL   // Default: ON  -> the application is to
                                  //                 be run in the common
                                  //                 session
                                  //          OFF -> the application is to
                                  //                 be run "standalone"


// --------------------------------------------------------------------
// Real DOS settings.  NOTE: WIN_RUNMODE is not supported; all Windows apps are
// installed with SEAMLESS and WPS handles the mapping.
// --------------------------------------------------------------------
//  WIN_RUNMODE            INT    // Valid settings: 10 (REAL)
                                  //                 11 (STANDARD)
                                  //                 12 (AUTO)
                                  //                 13 (SEAMLESS)
13  COM_HOLD               BOOL   // Default: off
14  DOS_BREAK              BOOL   // Default: off
15  DOS_DEVICE             MLSTR  // Default: empty
16  DOS_FCBS               INT    // Limits: 0 to 255, default 16
17  DOS_FCBS_KEEP          INT    // Limits: 0 to 255, default 8
18  DOS_FILES              INT    // Limits: 20 to 255, default 20
19  DOS_HIGH               BOOL   // Default: off
20  DOS_LASTDRIVE          STR    // Limits: last physical drive to 'Z',
                                  //         default 'Z'
21  DOS_RMSIZE             INT    // Limits: 128 to 640, default 640
                                  // NOTE: increments of 16
22  DOS_SHELL              STR    // Default: "#:\OS2\MDOS\COMMAND.COM "
                                  //         "#:\OS2\MDOS\ /P" where #
                                  //         is the boot drive
23  DOS_STARTUP_DRIVE      STR    // Default: empty
24  DOS_UMB                BOOL   // Default: off
25  DOS_VERSION            MLSTR  // Default: DCJSS02.EXE,3,40,255
                                  //          DFIA0MOD.SYS,3,40,255
                                  //          DXMA0MOD.SYS,3,40,255
                                  //          IBMCACHE.COM,3,40,255
                                  //          IBMCACHE.SYS,3,40,255
                                  //          ISAM.EXE,3,40,255
                                  //          ISAM2.EXE,3,40,255
                                  //          ISQL.EXE,3,40,255
                                  //          NET3.COM,3,40,255
                                  //          EXCEL.EXE,10,10,4
                                  //          PSCPG.COM,3,40,255
                                  //          SAF.EXE,3,40,255
                                  //          WIN200.BIN,10,10,4
26  DPMI_DOS_API           STR    // Valid settings: AUTO (default)
                                  //                  ENABLED
                                  //                  DISABLED
27  DPMI_MEMORY_LIMIT      INT    // Limits: 0 to 512, default 3
28  EMS_FRAME_LOCATION     STR    // Valid settings: AUTO (default)
```

```
                                        //              NONE
                                        //              C000
                                        //              C400
                                        //              C800
                                        //              CC00
                                        //              D000
                                        //              D400
                                        //              D800
                                        //              DC00
                                        //              8000
                                        //              8400
                                        //              8800
                                        //              8C00
                                        //              9000
29  EMS_HIGH_OS_MAP_REGION    INT       // Limits: 0 to 96, default 32
                                        // NOTE: increments of 16
30  EMS_LOW_OS_MAP_REGION     INT       // Limits: 0 to 576, default 384
                                        // NOTE: increments of 16
31  EMS_MEMORY_LIMIT          INT       // Limits: 0 to 32768, default 2048
                                        // NOTE: increments of 16
32  HW_NOSOUND                BOOL      // Default: off
33  HW_ROM_TO_RAM             BOOL      // Default: off
34  HW_TIMER                  BOOL      // Default: off
35  IDLE_SECONDS              INT       // Limits: 0 to 60, default 0
36  IDLE_SENSITIVITY          INT       // Limits: 1 to 100, default 75
37  KBD_ALTHOME_BYPASS        BOOL      // Default: off
38  KBD_BUFFER_EXTEND         BOOL      // Default: on
39  KBD_CTRL_BYPASS           STR       // Valid settings: NONE (default)
                                        //                 ALT_ESC
                                        //                 CTRL_ESC
40  KBD_RATE_LOCK             BOOL      // Default: off
41  MEM_EXCLUDE_REGIONS       STR       // Default: empty
42  MEM_INCLUDE_REGIONS       STR       // Default: empty
43  MOUSE_EXCLUSIVE_ACCESS    BOOL      // Default: off
44  PRINT_TIMEOUT             INT       // Limits: 1 to 3600, default 15
45  VIDEO_8514A_XGA_IOTRAP    BOOL      // Default: on
46  VIDEO_FASTPASTE           BOOL      // Default: off
47  VIDEO_MODE_RESTRICTION    ENUM      // Valid settings: NONE (default)
                                        //                 CGA
                                        //                 MONO
48  VIDEO_ONDEMAND_MEMORY     BOOL      // Default: on
49  VIDEO_RETRACE_EMULATION   BOOL      // Default: on
50  VIDEO_ROM_EMULATION       BOOL      // Default: on
51  VIDEO_SWITCH_NOTIFICATION BOOL      // Default: off
52  VIDEO_WINDOW_REFRESH      INT       // Limits: 1 to 600, default 1
53  XMS_HANDLES               INT       // Limits: 0 to 128, default 32
54  XMS_MEMORY_LIMIT          INT       // Limits: 0 to 16384, default 2048
                                        // NOTE: increments of 4
55  XMS_MINIMUM_HMA           INT       // Limits: 0 to 63, default 0
56  DOS_BACKGROUND_EXECUTION  BOOL      // Default: on
57  DPMI_NETWORK_BUFF_SIZE    INT       // Limits: 1 to 64, default 8
```

The layout of each line in DBTAGS.DAT is as follows:

INDEX VALUE TYPE (optional comments)

where:

**INDEX**   Is a sequence number

**VALUE**   Is the name of the setting

**TYPE**    Is the type of the value.

TYPE is one of the following:

**NOP**   Comments; any line with this type is ignored

**STR**   A string value

**INT**   An integer value

**BOOL**    Boolean, with value of ON or OFF

**BYTE**    Program type, either DOS, OS/2, or Windows.

**MLSTR**   A multi-line string with component lines on individual lines in the text_database file.

Using these types, various settings for programs can be defined. Do not edit DBTAGS.DAT or create a new one; the tag file is available only as a reference when creating the MYDATA.TXT file.

PARSEDB checks the validity of all entries in MYDATA.TXT and compares them to the settings definitions in DBTAGS.DAT. If all entries are valid, PARSEDB creates a binary database named MYDATA.DAT.

Errors in the text file will cause PARSEDB to exit and display a message:

- A message that a file is corrupted indicates embedded ASCII NUL characters in the input text file.
- A message about an invalid setting indicates the use of a setting not found in DBTAGS.DAT. The message should include a line number and the name of the input file.
- A message that an entry has missing parameters indicates the absence of the minimum settings for the entry.

PARSEDB does not check for duplicate entries in the input text file, nor does it require settings to be in any particular order. It is also not case sensitive, that is, "Off" is treated the same as "OFF."

We recommend that a copy of the input text file (DATABASE.TXT) for the default migration database file (DATABASE.DAT) be made and used as the template for your own input file. A sample input text file is listed below.

```
REM ------------------------------------------------------------------------
REM Windows file browser
REM ------------------------------------------------------------------------
        NAME                       WNBROWSE.EXE
        TITLE                      File Browser
        TYPE                       Windows
        ASSOC_FILE                 WNBROWSE.HLP
        DEF_DIR                    \WNBROWSE
        MOUSE_EXCLUSIVE_ACCESS     OFF
        KBD_CTRL_BYPASS            CTRL_ESC
        COMMON_SESSION             ON
        VIDEO_SWITCH_NOTIFICATION  ON
        KBD_ALTHOME_BYPASS         ON
        DPMI_MEMORY_LIMIT          5


REM ------------------------------------------------------------------------
REM WordPerfect 5.1 by WordPerfect
REM ------------------------------------------------------------------------
        NAME                       WP.EXE
        TITLE                      WordPerfect
        TYPE                       DOS
        ASSOC_FILE                 WP.FIL
        DEF_DIR                    \WP51
        MOUSE_EXCLUSIVE_ACCESS     OFF
        IDLE_SENSITIVITY           88


REM ------------------------------------------------------------------------
REM PMCAMERA OS/2 screen capture utility
REM ------------------------------------------------------------------------
        NAME                       PMCAMERA.EXE
        TITLE                      PM Screen Capture
        TYPE                       OS/2
        ASSOC_FILE                 PMCAMERA.HLP
        DEF_DIR                    \PMCAMERA
```

*Figure 33. User Definitions for other Applications*

## 7.7 Summary

DOS and Windows application installation under OS/2 V2.0 is generally per-
formed from a DOS command prompt or from the WIN-OS/2 Program Manager.
In some cases, it may be necessary to boot from a DOS diskette to perform the
install. Modifications made to CONFIG.SYS and AUTOEXEC.BAT by installation
programs should be carefully reviewed.

If the OS/2 V2.0 system is to be set up for Boot Manager or dual boot to DOS,
the arrangement of the hard disk partitions needs to be planned.

The Migrate Applications program is used to migrate already installed DOS,
Windows, and OS/2 programs, and creates and places the program objects in a
folder on the desktop. If the DOS or Windows program is in the migrate data-
base \OS2\INSTALL\DATABASE.DAT, the Migrate Applications program automat-
ically selects the recommended DOS settings for the program.

The Migrate Applications program always sets up Windows programs to run in a
WIN-OS/2 window session, if possible. Some exceptions exist, for example,
CorelDraw!** and Arts & Letters**. Those programs use some very special
Windows programming techniques, which can cause some problems in a
"seamless" WIN-OS/2 VDM. This may not happen in every user scenario but it
was felt saver to install those applications as fullscreen SAVDMs.

The Migrate Applications program is used:

- During installation of the OS/2 Version 2.0 operating system if you have DOS, OS/2, or Windows programs already installed on your hard disk.
- If a DOS, OS/2, or Windows program is added to a working OS/2 Version 2.0 system.

A utility, PARSEDB, is supplied to help system administrators to add an organization's unique applications to the migration database, or to create their own.

# Chapter 8. Windows Applications

OS/2 Version 2.0 provides the capability for Windows applications to run under OS/2 Version 2.0, using its WIN-OS/2 component. With this support, applications written for Windows 3.0 and Windows 2.x can coexist and execute with OS/2 and DOS applications in the same machine under OS/2 Version 2.0.
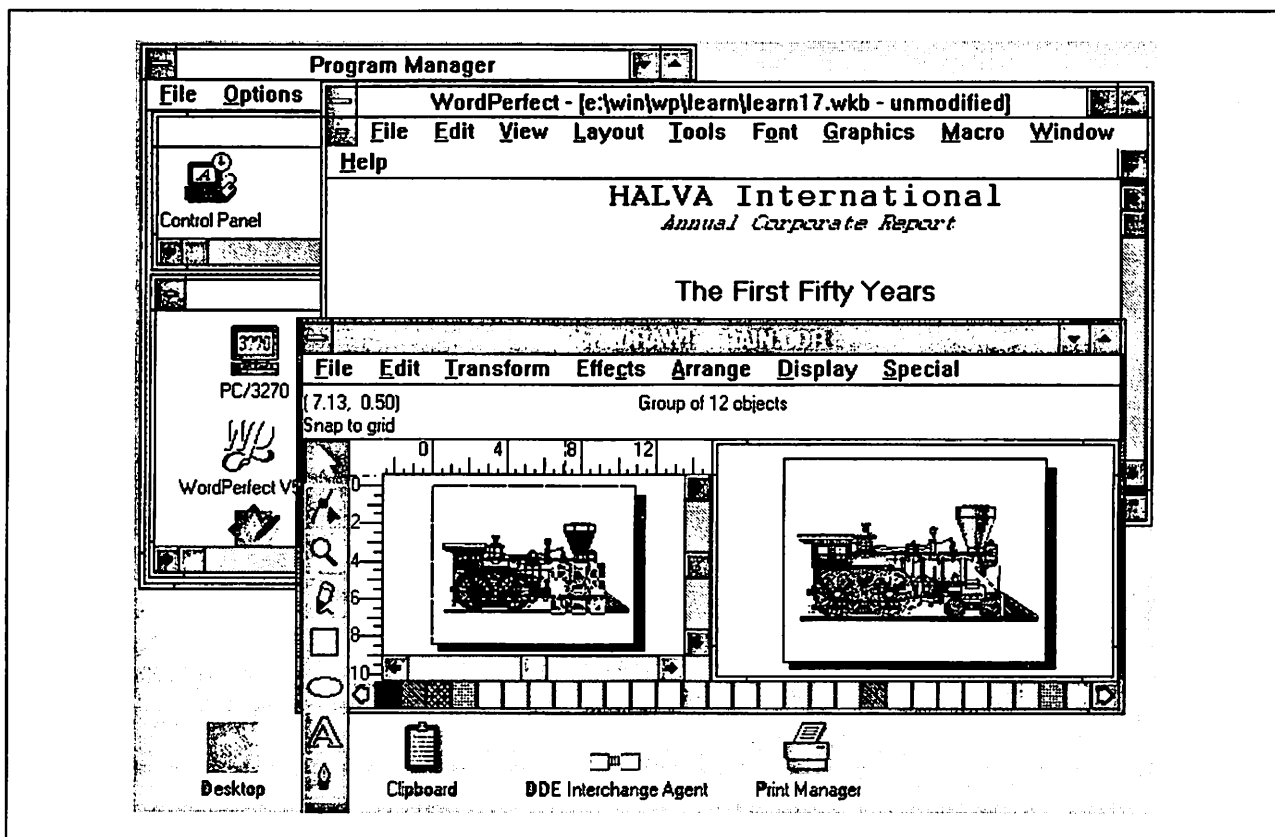


*Figure 34. Windows Applications Running under OS/2 Version 2.0*

Each Windows application executes as a protected mode process within a VDM. As such, Windows applications are subject to the same application protection facilities provided to other protected mode applications (both OS/2 and MVDM tasks) under OS/2 Version 2.0. Windows applications are protected from other Windows applications and from DOS and OS/2 applications executing in the system. This is in contrast to the native Windows 3.0 environment, where protection is limited to DOS applications (Windows applications share a common address space), and is only available when Windows is running in standard or 386 enhanced modes.

The execution of Windows applications in a protected environment allows these applications to take full advantage of the pre-emptive multitasking capabilities of OS/2 Version 2.0, with full pre-emptive multitasking between Windows applications, OS/2 applications and DOS applications. This is again in contrast to the native Windows 3.0 environment, where pre-emptive multitasking is available only for DOS applications and only when Windows 3.0 is running in enhanced mode, thereby impacting performance and preventing many applications written for previous versions of Windows from executing. OS/2 Version 2.0 has no such restriction.

## 8.1 Windows 3.0 Execution Modes

The native Windows 3.0 environment has three execution modes, though the options available to any user depend upon the machine's processor and the amount of installed memory. These modes are important, as they are relevant, to the discussion later of the way in which OS/2 runs Windows applications. The initial description of each mode comes from the *Microsoft Windows User's Guide*.

### 8.1.1 Real Mode

An operating mode that Windows runs in to provide maximum compatibility with versions of Windows applications prior to 3.0. Real mode is the only mode available for computers with less than 1MB of extended memory.

**Real mode** is equivalent to previous versions of Windows (2.x), and can address 640KB of conventional memory, plus LIM EMS Version 4.0 expanded memory. Extended memory can be used for a virtual disk or disk caching only.

Real mode requires an 8088 processor or above, and 640KB of installed memory. Real mode requires 384KB of free conventional memory after DOS and other memory resident software, including network drivers, is loaded.

Real mode is supported for Windows and its applications under OS/2 Version 2.0, in either of two ways:

- The WIN-OS/2 kernel provided by OS/2 Version 2.0 may be used to run Windows applications in real mode.

- The commercially available Windows 3.0 product may be run in real mode in a VDM, and real mode applications started from within this VDM by the Windows Program Manager.

Note that the commercially available Windows product *cannot* be run in standard or enhanced modes in a VDM due to Windows' memory management architecture; Windows assumes that it is a DPMI host and cannot act as a DPMI client. Many Windows applications run quite adequately in real mode; in fact, some applications written for Windows 2.x cannot run in any other mode.

### 8.1.2 Standard Mode

The normal operating mode for running Windows. This mode provides access to extended memory and also lets you switch among non-Windows applications.

**Standard mode** uses the 80286 processor's protected mode to provide direct access for Windows and Windows applications to up to 16MB of extended memory. Expanded memory for DOS applications is only supported with physical expanded memory cards (emulation of expanded memory using extended memory is not supported).

Standard mode requires an 80286 processor or above, and at least 1MB of installed memory, with a minimum 192KB of free extended memory. The XMS driver HIMEM.SYS must also be loaded. Windows applications must be written to comply with the memory management rules for Windows 3.0 in order to run in standard mode.

Standard mode is recommended by Microsoft when running only Windows applications (that is, no DOS applications) in certain configurations, even on an 80386 machine. In the Windows 3.0 manual, on page 429, it is suggested that users running only Windows 3.0 applications should run in standard mode, even on 80386 systems with 2-3MB of memory, as there is a performance improvement in doing so.

Standard mode is necessary for some Windows applications (for example, Microsoft Excel** Version 3.0). To accommodate such applications, OS/2 Version 2.0 must provide additional support. Basically, these applications need to access DPMI services for extended memory support, which is available under Windows 3.0 when running in standard or enhanced modes. See Chapter 9, "DOS Protected Mode Interface" for further information on DPMI support under OS/2 Version 2.0.

The other requirement is to supply Windows services to Windows applications. This service is provided in OS/2 Version 2.0 by modifying the Windows kernel and running it in standard mode in a VDM. As part of the joint development and cross-licensing agreement between IBM and Microsoft, IBM has access to the Windows source code. IBM has modified the source to provide a Windows kernel (WIN-OS/2) capable of running as a DPMI client within a VDM (the retail version of Windows 3.0 can *only* function as a DPMI host), and includes this kernel as part of the OS/2 Version 2.0 product.

OS/2 therefore supports Windows applications running in standard mode in a VDM. Use of the VDM design, which provides a self-contained DOS environment, means that the environment is identical, from the application's point of view, to running under Windows loaded in standard mode, on DOS. This design therefore provides the maximum compatibility with the DOS/Windows environment. In fact, it offers a wider range of compatibility, since Windows 2.x applications, which require real mode operation under Windows 3.0 in DOS, can be run concurrently with Windows 3.0 applications running in standard mode. This combination is not possible at the same time under DOS/Windows 3.0.

## 8.1.3  386 Enhanced Mode

A mode that Windows runs in to access the virtual memory capabilities of the Intel 80386 processor. This mode allows Windows to use more memory than is physically available and to provide multi-tasking for non-Windows applications.

**386 enhanced mode** uses the 80386 processor's protected mode to provide direct access for Windows and Windows applications to up to 16MB of extended memory. In addition, the virtual 8086 mode of the 80386 is used to provide multiple DOS environments for non-Windows applications. Most DOS applications can be run in a window. Virtual memory support is provided, for Windows applications only, using the demand paging feature of the 80386 processor.

386 enhanced mode requires an 80386 processor or above, at least 2MB of installed memory, with a minimum 1MB of free extended memory. The XMS driver HIMEM.SYS must also be loaded. Windows applications must be written to comply with the memory management rules for Windows 3.0 to run in 386 enhanced mode.

The 386 enhanced mode of Windows 3.0 provides a number of additional capabilities over standard mode:

- The capability for pre-emptive multitasking of DOS sessions

- Demand paging for efficient virtual memory.

Both of these capabilities are provided by OS/2 Version 2.0 itself for both DOS and Windows applications, independent of the Windows kernel. There is hence no need to provide such functions within the Windows kernel.

There are, however, a small number of Windows applications which require enhanced mode to run. Such applications require enhanced mode either because they rely on features only available in enhanced mode, such as Windows 3.0's permanent swap file capability (such as Caere Omnipage**), or have been coded using the WINMEM32.DLL, a set of routines that provide some 32-bit functions for Windows applications, such as Wolfram Research's Mathematica**.

It is believed that there are only, at maximum, three or four such applications on the market, which represents less than 0.3% of Windows 3.0 applications (assuming Microsoft's quoted figure of 1500 Windows applications). It is unlikely there will ever be many in the latter category of applications, since the WINMEM32.DLL is very difficult to use, and Microsoft itself warns in Appendix E of the *Windows Programmer's Reference*: "only experienced Windows application programmers with extensive experience writing assembly-level code should attempt to use these functions in an application."

This warning is necessary because even something as basic as memory management using these routines can be very complex, and requires the programmer to create assembly language interfaces between the 16- and 32-bit parts of a program (note that such "thunks" are provided by OS/2 Version 2.0 between 16-bit and 32-bit modules; see *OS/2 Version 2.0 - Volume 1: Control Program*). Charles Petzold, possibly the most widely respected authority on Windows programming, whose book on the subject is a standard reference work, concluded on this subject that "something is seriously wrong when memory access becomes difficult," and contrasted the current Windows approach with the ease of 32-bit memory management under OS/2 Version 2.0.

Applications which require enhanced mode will not be supported by OS/2 Version 2.0. Support of this mode requires Windows to run at the Ring 0 privilege level in the 80386 processor, which allows Windows or a Windows application to access *all* system memory and resources, including those belonging to the operating system itself. This could result in a serious breach of system integrity, and is therefore not permitted under OS/2 Version 2.0.

So, although there will almost certainly be a very small minority of Windows applications that will not run under OS/2 Version 2.0, the vast majority will run, and in a mode which allows access to their full function. Indeed, to the Windows application, the environment will appear exactly the same as if the application were running under DOS/Windows in standard mode.

## 8.2 Windows Applications under OS/2 Version 2.0

Under OS/2 Version 2.0, Windows applications are treated as special cases of DOS applications, which need a special environment in which to run. Therefore, the key to Windows application compatibility is to provide these applications with as similar an environment as possible to that experienced under DOS, while taking advantage of the inherent design superiority of OS/2.

## 8.2.1  Supported Components

The following components of Windows 3.0 are supported and available within the OS/2 V2.0 Windows kernel:

- Windows real mode kernel (WINOS2.COM and KERNEL.EXE)

- Modified Windows standard mode kernel (OS2K286.EXE)

- Modified DOS Extender (VDPX.SYS)

- Print Manager (Spool Function)

- Program Manager:

  - Permits the starting of multiple Windows applications in a VDM
  - Permits switching between Windows applications in the VDM

- Help Manager

- Video device drivers

- Keyboard, mouse and communications device drivers

- Task Manager

- Windows user and GDI DLLs

- Printer device drivers

- Clipboard support

- Setup, with only one function left in order to install Windows network device drivers (DLLs).

---

**Note**

This is really not necessary to do if you are already running any network requester code under OS/2 itself.  This would be transparent to the entire system and therefore every VDM and WIN-OS/2 session will have full access to the network, printers, files, etc.

---

- Control Panel, with functions limited to:

  - Printer Install
  - Color
  - Fonts
  - Sound
  - Mouse
  - International
  - KBD (Keyboard rate).

The Clock program is available in a Multiple Application VDM (MAVDM) (see 8.2.2, "Methods of Execution" on page 134).

The following Microsoft Windows 3.0 components are *not* included within OS/2 V2.0 (even though they would run just as any other Windows application):

- File Manager
- Systems Editor (SYSEDIT)
- Games
- Write
- Terminal
- Notepad
- Cardfile
- Calendar

- Calculator
- PIF Editor
- Paintbrush
- Recorder
- Wallpaper bitmaps

For each of these functions, equivalent capabilities are provided by OS/2 Version 2.0, or these functions are not required within the OS/2 Version 2.0 environment.

### 8.2.1.1 Multimedia Extensions (MME) for Windows

Multimedia Extensions for Windows can be installed under WIN-OS/2 and are fully supported.

Some multimedia applications may require more than the default DPMI memory. If that happens, the *WIN-OS/2-Setting DPMI_MEMORY_LIMIT* should be adjusted to the appropriate value.

## 8.2.2 Methods of Execution

Windows applications may be run under OS/2 Version 2.0 in five ways:

- The original licensed **Windows V3.0** or **Windows V2.x** may be started in a VDM, and will allow Windows V3.0 and Windows V2.x, and its applications, to be run in real mode.

- A **Single Application VDM (SAVDM)** may be started, for a single Windows application. The icon supplied with the Windows application will be defined within the Workplace Shell desktop.

- A **Multiple Application VDM (MAVDM)** may be started, which activates the Windows Program Manager, allowing the user to access a number of Windows applications.

- A **Separate "seamless" WIN-OS/2 VDM** may be used, which is basically a SAVDM running in its own window under the Workplace Shell. See 8.2.2.3, ""Seamless" WIN-OS/2 VDM" on page 137 for further information.

- A **common "seamless" WIN-OS/2 VDM** may be used, which is a special kind of MAVDM. WIN-OS/2 will start all "seamless" WIN-OS/2 VDMs in a single session, but in separate windows running under the Workplace Shell. See 8.2.2.4, "Common "Seamless" WIN-OS/2 VDM" on page 141 for further information.

- A **separate "seamless" WIN-OS/2 VDM** may be used, and the **WIN-OS/2 Program Manager** may be launched from there. This will allow to launch other Windows applications from there and therefore, this would actually be a MAVDM running in its own window under the Workplace Shell. Every Windows application launched from there would run in its own window under the Workplace Shell but share the same WIN-OS/2 kernel.

No license of Windows V3.0 or Windows V2.x is necessary to run Windows applications, as the Windows environment is an implementation of Windows V3.0 and Windows V2.x within OS/2 V2.0 (WIN-OS/2) itself. Multiple instances of any of the above methods may be started in the same system.

However, note that in the current release, "seamless" WIN-OS/2 VDMs and common "seamless" WIN-OS/2 VDMs may only be started on machines with VGA graphics. Seamless execution of Windows applications is not supported using other graphics modes.

The following applications are started (iconized) when the first VDM (either SAVDM or MAVDM) is started:

- Modified Windows Clipboard Viewer Program

- DDE Server/Agent Application

- Presentation Manager icon

- Task Manager (no icon)

- Windows Program Manager (not visible in a SAVDM)

- Clock (MAVDM only)

- Windows Control Panel (MAVDM only).

Each of these methods is described in the following sections.

### 8.2.2.1  Single Application VDM (SAVDM)

A SAVDM is the recommended way of running Windows applications under OS/2 Version 2.0 in a non-VGA system, such as a PS/2 with an XGA or 8514/A display adapter, because seamless execution is only supported with VGA graphics.  Using a SAVDM is also recommended if the user wishes to run Windows applications in *real mode* (seamless execution is supported only in Windows standard mode).

Since the Windows application runs in a self-contained Windows environment in its own VDM, it is fully protected from other applications, and the system is protected from it.  This means that if the application crashes for any reason, it only affects its own VDM, and thus only that one application.  Other Windows or DOS applications running in other VDMs are not affected, nor are OS/2 applications. This represents a significant improvement in reliability over DOS/Windows, in which a failure in one Windows application may bring down the entire Windows system or corrupt the data areas of other Windows programs, since all Windows applications and Windows itself share the same address space.
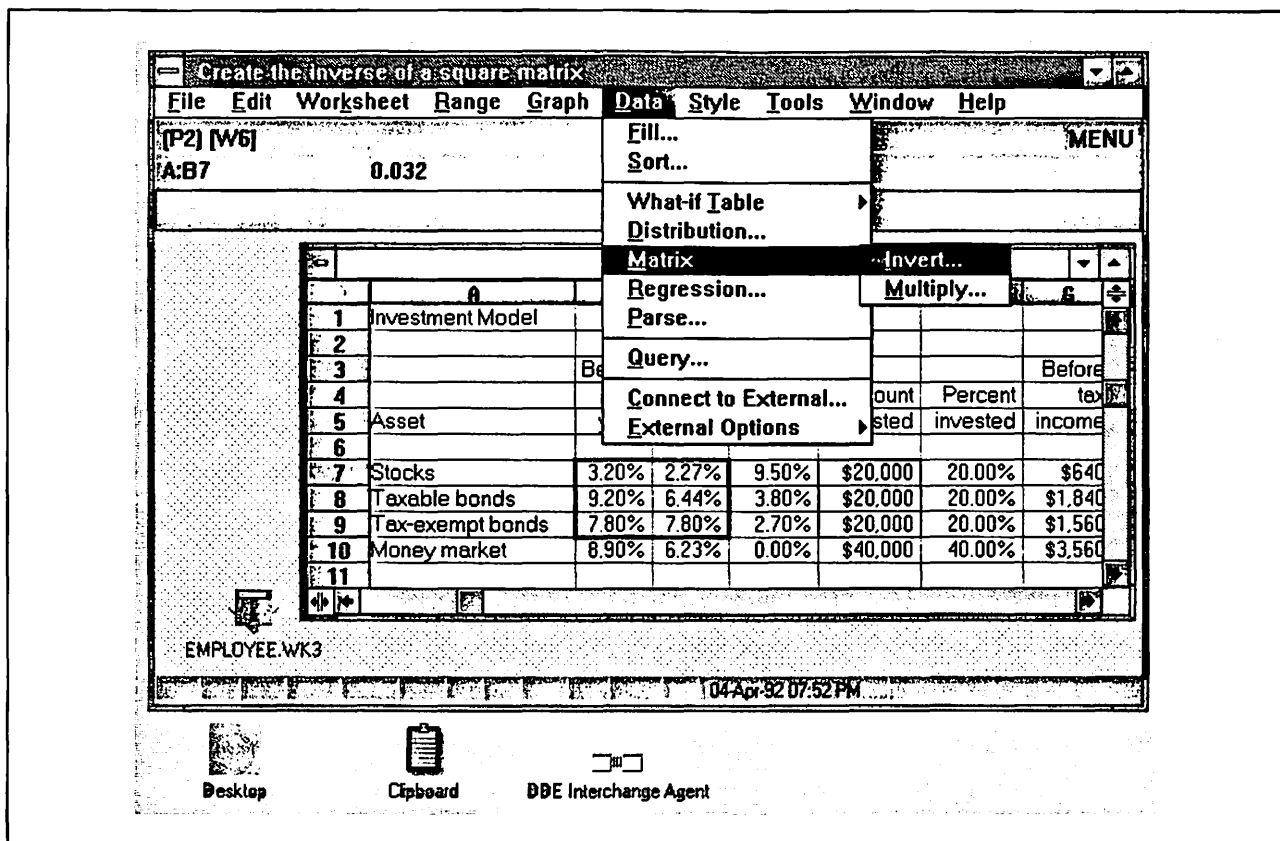
*Figure 35. Single Windows Application Running under OS/2 Version 2.0*

Also, by running in SAVDMs, Windows applications are timesliced more effectively than in a MAVDM or native DOS/Windows environment, since each application is under the control of OS/2's scheduler and its pre-emptive multitasking. In a MAVDM environment, all Windows applications are still subject to the cooperative multitasking under Windows itself.

***Ctrl-Esc*** is the key combination used to display the Windows "Window List."

***Alt-Esc*** is the key combination used to switch to the next session as defined in the Workplace Shell.

The SAVDM provides a simple approach to Presentation Manager integration. The application is loaded from the Workplace Shell in a very similar way to a DOS application, and the user may easily switch back to Presentation Manager, as well as share data via clipboard or DDE with other Windows or Presentation Manager applications. The icon displayed on the Workplace Shell desktop is the application's own Windows icon.

Each SAVDM will indicate the Windows execution mode based on the file type specified in the EXE header of the Windows application. *Real mode* will be indicated for Windows applications written for versions of Windows prior to 3.0. *Auto-Select* (real or standard mode) is selected by default for Windows 3.0 applications, based on processor type.

### 8.2.2.2 Multiple Application VDM (MAVDM)

The MAVDM is almost identical to running Windows 3.0 on a DOS-based machine. The MAVDM uses the Windows 3.0 Program Manager to start multiple Windows applications within the same VDM, running on a separate Windows desktop. It therefore provides maximum "look and feel" compatibility for the DOS/Windows user migrating to OS/2 Version 2.0.

Note that the use of a MAVDM or the common "seamless" WIN-OS/2 VDM is a requirement where Windows applications must communicate with one another via shared memory.

*Ctrl-Esc* is used within the VDM to display the Windows "Window List."

*Alt-Esc* is used to switch to the next session defined in the Workplace Shell.

For a MAVDM, the Workplace Shell icon will represent the MAVDM itself, rather than the applications executing within the VDM. Terminating an application within the VDM will not terminate the VDM itself. The user must select *Exit Windows* in the Windows Program Manager to terminate the VDM, or close the VDM from the Workplace Shell.

In the MAVDM and the common "seamless" WIN-OS/2 VDM environment, all Windows applications are still subject to the cooperative multitasking of Windows itself. Since several Windows applications are typically loaded in the same VDM, the MAVDM environment shares some of the pitfalls of DOS/Windows in terms of robustness. If one Windows application crashes within a MAVDM, it may cause all the applications within that VDM to fail. However, the effect is *only* within that VDM; other VDMs running DOS or Windows applications, and other processes executing under OS/2 Version 2.0, are not affected and continue execution. So even here there are benefits from running Windows applications under OS/2, for greater resilience from system crashes. Furthermore, the MAVDM environment provides additional error checking and handling over that provided by Windows 3.0 itself.

### 8.2.2.3 "Seamless" WIN-OS/2 VDM

One of the goals of OS/2 2.0 is to be the integrating platform of choice; that is, to provide a desktop environment from which all types of applications:

- 16-bit OS/2
- 32-bit OS/2
- DOS
- Windows

may be executed in a uniform manner. Although OS/2 V2.0 is able to support Windows applications effectively in SAVDMs, the additional ability to launch a Windows application from a Workplace Shell object, and execute it on the OS/2 desktop along with Presentation Manager and DOS applications, achieves the goal of creating an environment that is explicitly simple and uniform enough that the end user need not be concerned with the implicit differences between the types of applications that need to be executed in it. OS/2 V2.0 will concern itself with the differences and **"seamlessly"** accommodate the applications. This level of support extends not only to execution but to installation and configuration of the application as well.
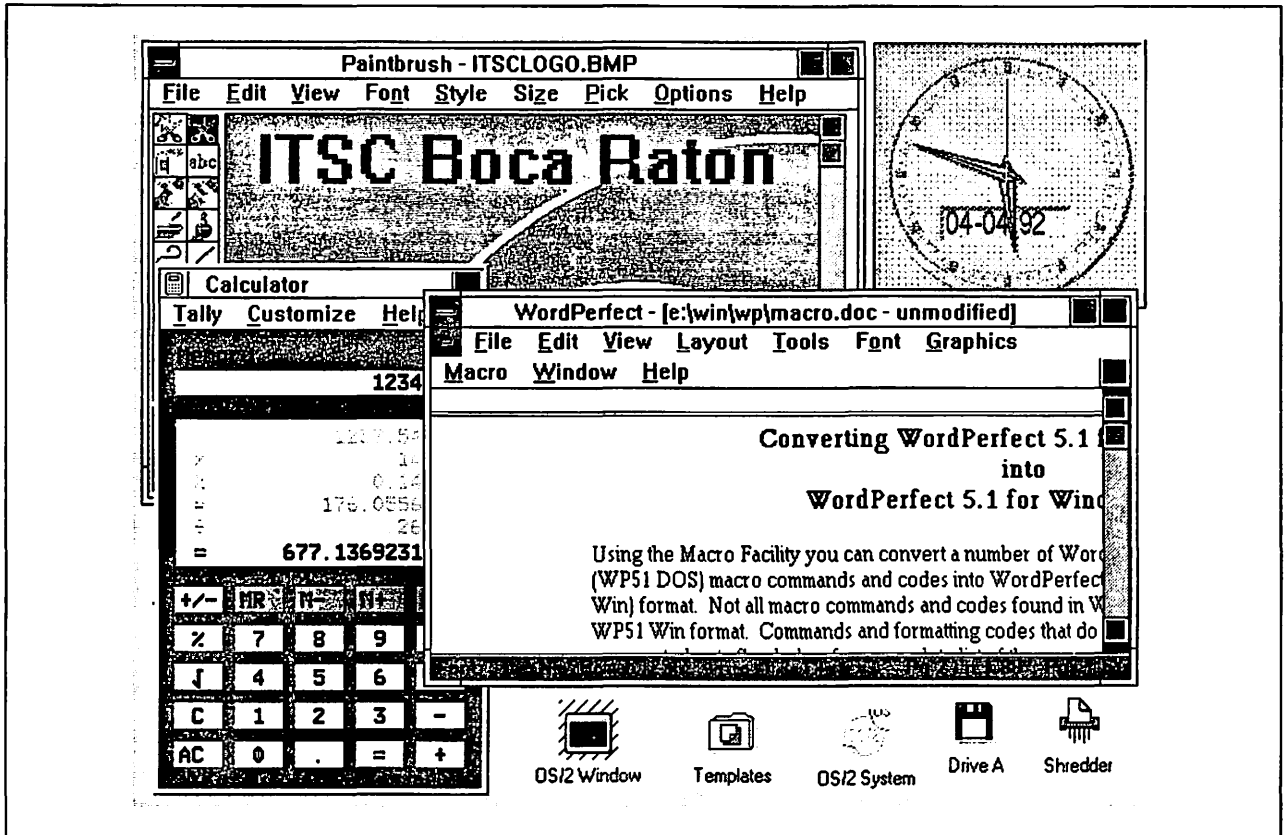
*Figure 36. Single Windows Application(s) Running "Seamless" on the OS/2 Version 2.0 Desktop*

The appearance of Windows applications on the Workplace Shell desktop requires that the Windows video device driver and the Presentation Manager video device driver communicate and coordinate their access of the video hardware. Each device driver effectively "owns" its area of the screen. Allowing the Windows display device driver to directly access the video hardware avoids the more cumbersome process of a complete task switch. However, this hardware access poses integrity problems in the areas of simultaneous access of hardware, rectangle invalidation handling, window management, and the exchange of window state information between Presentation Manager and seamless VDMs supported by separate video device drivers.

To address these problems, a high performance virtual device driver named (VWIN.SYS), capable of interprocess communication, was created. This VDD serializes the simultaneous accesses to the hardware, oversees the exchange of window state information between Presentation Manager and seamless VDMs, and establishes the addressability of Presentation Manager resources (either directly or indirectly) by the Windows display device driver.

When the system is initialized, the Presentation Manager display driver calls the VWIN.SYS driver, and passes a pointer to an array of structures that specify the protocol required to enable the Windows device driver to access Presentation Manager resources. To prevent a seamless Windows application from hanging the entire Workplace Shell desktop, the Windows video device driver and the Presentation Manager video device driver together implement and monitor a VDM "heartbeat" or counter. This counter is stored in the Presentation Manager display driver's data area and is made available to the Windows display driver.

The "heartbeat" counter information is made available to the Windows DD to indicate that hardware access is in progress by the Windows DD. The "heartbeat" counter is incremented by the Windows DD prior to the video hardware access. If a Windows application is locking up the Workplace Shell desktop, it is the responsibility of VWIN.SYS to identify the current owner of the semaphore, terminate the VDM and tell the Presentation Manager DD to clean up.

In the event that the Presentation Manager display device driver requests hardware resources and the time interval allotted for this access to occur is exceeded, then:

1. If it is the first request, the Presentation Manager display driver records the heartbeat value, process ID and thread ID of the process in control of the hardware, and raises an internal flag.

2. If it is the second request, and the heartbeat value, PID and TID have not changed, the Presentation Manager display driver calls the Windows display driver before clearing the flag, and passes it the PID and TID.

   - It is the responsibility of the Windows driver to use the PID and TID to identify the process that is monopolizing the hardware resources and inform the Presentation Manager driver.

   - If it is an active, seamless VDM, WIN-OS/2 will terminate the VDM and inform the Presentation Manager driver to clean up.

   - If the PID and TID are invalid, the Windows driver will inform the Presentation Manager driver to clean up.

   - If the PID and TID belong to a Presentation Manager application, the Windows driver will tell the Presentation Manager driver to attempt access again.

This algorithm is relatively simple but not totally fail-safe. It is quite possible to create a serialization mechanism that would safeguard the Workplace Shell desktop to a greater degree. However, when one considers the remoteness of the possibility of its failure (which requires a bogus PID or TID), the costs, in terms of a performance impact, would far outweigh the benefits incurred.

Some important points about "seamless" WIN-OS/2 VDMs:

- Note that the use of a MAVDM or a common "seamless" WIN-OS/2 VDM is a requirement where Windows applications must communicate with one another via shared memory.

- Only standard mode is supported in this mode of operation.

- Presentation Manager must provide extensive support for this implementation of WIN-OS/2. Basically, Presentation Manager supports a "black hole" and allows the WIN-OS/2 kernel to control it. A modified WIN-OS/2 and Presentation Manager display device driver is built into OS/2 Version 2.0 to support this mechanism.

- The "seamless" WIN-OS/2 VDM is only supported if OS/2 is configured for VGA mode, because only the Windows VGA display device driver is supported. This means that, on an 8514 or XGA equipped system, the Presentation Manager display device driver must be configured in VGA mode to be able to run Windows applications in a "seamless" WIN-OS/2 VDM. If the user selects a higher resolution display device driver such as XGA, Windows applications may only run in a SAVDM or MAVDM environment.

**Notes:**

1. Over time, more display device drivers will be enhanced to support this seamless mode of WIN-OS/2. Once available, installed and configured appropriately, WIN-OS/2 will provide seamless execution on other supported graphics modes.

2. Readers should check the *ReadMe* file in the *Information* folder for the latest information on this subject. Information in this folder explains how to reconfigure the system to have seamless WIN-OS/2 support as well as high resolution MAVDM and/or SAVDM sessions.

- A VDD (VWIN.SYS) provides the necessary services to the Windows kernel and Presentation Manager.

*Figure 37. Implementation of "Seamless" WIN-OS/2 VDM in OS/2 Version 2.0*

As shown in the figure above, PMVIOP.DLL contains a PMShield routine which is responsible for maintaining the entire Workplace Shell desktop, including the "black holes" which correspond to and are maintained by each "seamless" WIN-OS/2 VDM.

WinShield is the Windows VDM's counterpart of PMShield. The Workplace Shell desktop windowing state must be maintained in its entirety by this component. WinShield registers a service routine with VWIN.SYS, giving it the ability to post a message to PMShield whenever the Workplace Shell desktop state changes.

Upon creation of the first "seamless" WIN-OS/2 VDM, PMShield spawns three dedicated threads under the Workplace Shell to specifically service its "seamless" WIN-OS/2 VDM clients:

- Thread 1 is the *IPC Read Thread*, which normally suspends itself within VWIN.SYS, waiting for window-related events to occur in the "seamless" WIN-OS/2 VDM. The typical events sent by the WinShield are Create, Move, Size, Show Activate, etc. These events are duplicated by PMShield on the Workplace Shell desktop for the purpose of tracking the "black hole" windows.

- Thread 2 is the *Control Windows Procedure Thread* that the PMShield registers with PMWIN.DLL. This thread handles all relevant events that alter the state of the Workplace Shell desktop. Once in control, this thread will broadcast all Workplace Shell desktop initiated events asynchronously to all VDMs by calling VWIN.SYS. This causes a previously registered WinShield routine to be dispatched, giving it an opportunity to post an asynchronous message to itself.

- Thread 3 is the *Error Handling Thread*. All non-fatal errors on all "seamless" WIN-OS/2 VDM related operations will be reported through this mechanism where a Presentation Manager dialog box will pop up explaining to the user what went wrong.

On successful creation of a "seamless" WIN-OS/2 VDM, the Presentation Manager *Session Start* thread will notify VVGA.SYS to allow the started VDM to access the video hardware directly.

### 8.2.2.4 Common "Seamless" WIN-OS/2 VDM

There is no visible difference between Windows applications running in a "seamless" WIN-OS/2 VDM and those running in a common "seamless" WIN-OS/2 VDM. The technical differences between them are described in the following paragraphs. Everything else discussed in 8.2.2.3, ""Seamless" WIN-OS/2 VDM" is common to both.

To reduce the system resource usage in a low memory environment, users are given the option to start all "Seamless" WIN-OS/2 applications in the same VDM. This also helps to reduce startup time for Windows applications, and reduces the swap file space required. By default, Windows applications migrated from a DOS/Windows system at OS/2 installation time are migrated to a common "seamless" WIN-OS/2 VDM. The user has the option of prestarting one or more Windows applications in the common "seamless" WIN-OS/2 VDM by using the *Startup* folder in the Workplace Shell.

There is only one common "seamless" WIN-OS/2 VDM in the system. If the system is not currently configured to run "seamless" WIN-OS/2 VDMs, any Windows application which is defined for common "seamless" WIN-OS/2 VDM will be loaded and run in a fullscreen SAVDM.

By default, only the first Windows program launched from the Workplace Shell as a "seamless" WIN-OS/2 VDM will create a new VDM. Any subsequent Windows programs will share this environment, in exactly the same way as in a MAVDM full-screen session. This is known as the *common "seamless"* WIN-OS/2 environment.

However, the user may wish to protect these Windows programs from each other, and to maximize the timeslicing efficiency of Windows applications. This

can be done by checking the *Separate session* option on the *Session* page of the *Settings* notebook for any Windows program object under the Workplace Shell. That procedure would activate a "normal" "seamless" WIN-OS/2 VDM session.

The Workplace Shell Window List will contain an entry for the common "seamless" WIN-OS/2 VDM itself, in addition to an entry for each Windows program running in this VDM This provides also a mechanism for terminating this VDM from the Workplace Shell desktop, along with all the active Windows applications in it. As the user has a visual representation of the "contents" of the common "seamless" WIN-OS/2 VDM, the user knows which applications will be terminated if the *Close* option is chosen. If the common "seamless" WIN-OS/2 VDM hangs because one of the Windows programs is not behaving properly, the *Close* option on the entry for the common "seamless" WIN-OS/2 VDM will close down the entire VDM, including all Windows programs running in it. This behavior is similar to that of a MAVDM.

In the MAVDM and the common "seamless" WIN-OS/2 VDM environment, all those Windows applications are still subject to the cooperative multitasking under Windows itself. Since several Windows applications are loaded in the same VDM, the common "seamless" WIN-OS/2 VDM shares the same pitfalls as does the MAVDM. If one Windows application crashes within a common "seamless" WIN-OS/2 VDM, it may cause all the applications within that VDM to fail. However, as in a MAVDM, the effect is *only* within that VDM; other VDMs running DOS or Windows applications, and other processes executing under OS/2 Version 2.0, are not affected and continue execution. So even here there are additional benefits running Windows applications seamlessly under OS/2. Furthermore, as for the MAVDM environment, enhancements are made to provide additional error checking and handling for the common "seamless" WIN-OS/2 VDM.

A number of restrictions apply to the use of a common "seamless" WIN-OS/2 VDM. These are as follows:

- The DOS settings which will be in effect for the common "seamless" WIN-OS/2 VDM will be those which are defined by the *first* Windows program to start in this VDM. Changes to the settings for any subsequent Windows program in that VDM will *not* affect the actual settings of the common "seamless" WIN-OS/2 VDM. To make this obvious to the user, the *WIN-OS/2 Settings* button on the *Session* page of the *Settings* notebook is grayed for all Windows applications running in the common "seamless" WIN-OS/2 VDM once it is active. This implies that WIN-OS/2 settings cannot be viewed or changed once this VDM is started.

- The DPMI limit for the common "seamless" WIN-OS/2 VDM is higher than when defined for "seamless" WIN-OS/2 VDM, since multiple applications are likely to require more DPMI memory.

- Each Windows program running in the common "seamless" WIN-OS/2 VDM will have the same HAPP application handle), PID (process ID), and SGID (screen group ID). Any action taken on one of these values will cause that action against the entire VDM and not against only a specific instance inside the common "seamless" WIN-OS/2 VDM. For example, if a WinTerminateApp() call is issued, which uses the HAPP as input, then all applications running within the common "seamless" WIN-OS/2 VDM will be terminated. The user will be warned by a dialog that multiple Windows applications will be ended.

## 8.3 Installing WIN-OS/2 Support Under OS/2 Version 2.0

Windows application support is provided by default during the installation of OS/2 Version 2.0. If the user wishes not to install Windows support, the appropriate option must be chosen during OS/2 installation.

The OS/2 installation program detects the video resolution of the machine on which it is being installed. If Windows support is selected during "first time" installation, then the following configurations will be applied according to the detected video resolution:

**CGA, EGA**     Configured for full-screen (only) Windows support.

**VGA**     Configured for "seamless" WIN-OS/2 VDM support.

**8514/A, XGA**     During installation, a panel with the option to select seamless support (and downgrade to VGA graphics mode) is provided (see Figure 38). If full-screen (only) Windows support is selected, then the higher resolution is maintained.
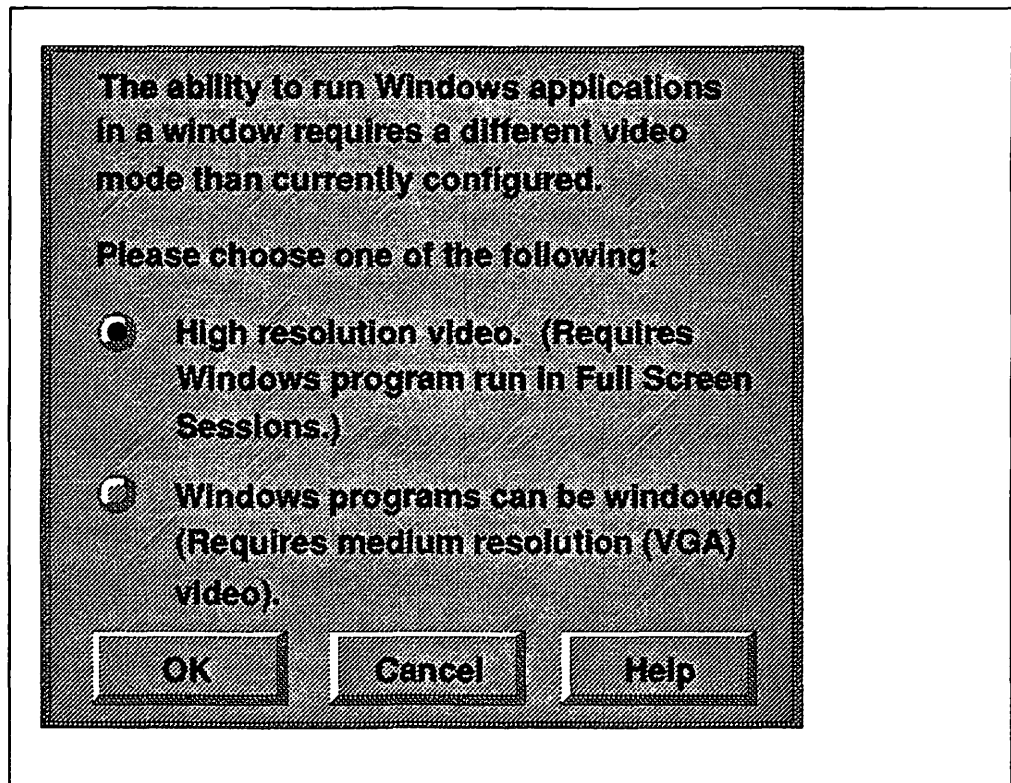


*Figure 38. Installing Windows Support under OS/2 Version 2.0*

---
**Note**

Readers should check the *ReadMe* file in the *Information* folder for the latest information on this subject. This folder contains information on how to reconfigure the system to have seamless WIN-OS/2 support as well as high resolution MAVDM and/or SAVDM sessions. Additional support information for SVGA display device drivers will be provided as well. *OS/2 Version 2.0 - Volume 1: Control Program* also discusses several installation and configuration aspects of OS/2 V2.0.

---

When Windows support is selected at installation time, all the files necessary to provide this support are installed in the following subdirectories:

- \OS2\MDOS\WINOS2

- \OS2\MDOS\WINOS2\SYSTEM

If the user decides to install Windows application support, DOS application support is automatically installed. OS/2 Version 2.0's CONFIG.SYS file is updated to include the above directories in the PATH statement.

Since Windows real mode requires 640KB of conventional memory and several MB of expanded memory (EMS), the EMS virtual device driver is also required. If the user did not select standard mode at installation time and wishes to add it at later time, the OS/2 Version 2.0 CONFIG.SYS must be modified by adding the following statements:

- DEVICE = C:\OS2\MDOS\VDPMI.SYS (DOS Protect Mode Interface)

- DEVICE = C:\OS2\MDOS\VDPX.SYS (DOS Extender Virtual Device Driver).

If these device drivers are not loaded, the Windows kernel will execute in real mode.

Windows can use expanded memory which conforms to the LIM EMS 4.0 specification when running in real mode. This memory is primarily used for storing background applications. In a DOS/Windows environment, an appropriate Expanded Memory Manager must be installed. Under OS/2 V2.0 this is not necessary, as the virtual device driver already provides that service. In standard mode, Windows may also use extended memory (XMS).

## 8.4 Migrating to OS/2 Version 2.0

Upon completion of the installation process, the user is given the opportunity to migrate installed Windows applications (defined to the Windows Program Manager) to the OS/2 Version 2.0 Workplace Shell. All Windows applications which are to be migrated must have the appropriate DOS and Windows settings defined in the **Certified Application Database (CAD)**, which is shipped as a standard component of OS/2 Version 2.0. See also Chapter 7, "Installing and Migrating Applications."

```
┌─ Note ──────────────────────────────────────────────────────────┐
│                                                                  │
│ Only the settings for those applications which have been certified via │
│ approved IBM testing channels will be held in the Certified Applications Data- │
│ base (CAD), and only those settings which differ from the default settings will │
│ be recorded.                                                     │
│                                                                  │
└──────────────────────────────────────────────────────────────────┘
```

If a referenced Windows application is found on any of the available disk volumes during OS/2 installation, the existing *.INI and *.GRP files will be read, the necessary changes applied to them, and the updated versions stored in the \OS2\MDOS\WINOS2 directory. This will effectively migrate the user's Windows desktop, including all Windows applications, into a MAVDM, SAVDM or seamless environment.

The CAD provides information enabling the installation procedure to automatically set the DOS settings for certified DOS and Windows applications. The user

is presented with a list of the certified applications found, which can then be migrated. The user may select any or all of these applications. The CAD is searched for each of the selected applications, and DOS and/or Windows settings information found in the database will be used to automatically assign settings to applications. Windows applications will be placed in a single "Windows Applications" folder. DOS applications are placed in a single "DOS Applications" folder.

The CAD is a binary database, generated from an ASCII database and a predefined tag file. Each field in the ASCII database starts with a descriptive tag that is associated with a value between 0-225 in the predefined tag file; the maximum number of tags is therefore 256. When the binary CAD generation tool encounters one of the descriptive tags, it generates an entry in the binary CAD with a 0-255 value specified in the predefined tag file. To add new or additional DOS properties, a short descriptive tag is created for the ASCII file and associated with an unused value between 0-255 in the predefined tag file. A length specification is also provided for the value in the tag file.

Each field in the binary CAD starts with a predefined tag value of 0-255 that identifies the field. This tag is followed by a *size* field, which in turn is followed by the actual value of the field.

Each application in the CAD has the following minimum information:

- The filename used to start the application
- The title of the application
- A pointer to the next application.

The filename that starts the application is used to identify the application on the hard drive. The next application pointer points directly to the next application entry in the CAD. This provides the ability to jump from one entry to the next without parsing all of the tags between entries in the CAD. The application title is displayed to the user if the application is found on the hard drive. The user will use this information to specify if the application is to be migrated.

The filename extensions held in the CAD will determine what files are searched for; that is all EXE, COM and BAT files.

When the applications have been migrated into the OS/2 Version 2.0 Workplace Shell, information for DOS applications is stored in the OS2.INI file. Windows settings for Windows applications are stored in the WIN.INI file, while their DOS-related settings are stored in the OS2.INI file.

## 8.5 Defining Windows Applications

As mentioned in the previous section, Windows applications may be automatically migrated to the Workplace Shell desktop at OS/2 installation time. However, for applications which are not defined in the Certified Application Database, or which are installed after OS/2 installation, a Workplace Shell object may be created from a template in the *Templates* folder. For such applications, the WIN-OS/2 application execution environment is defined to the Workplace Shell using the *Program* page of the program object's *Settings* notebook.
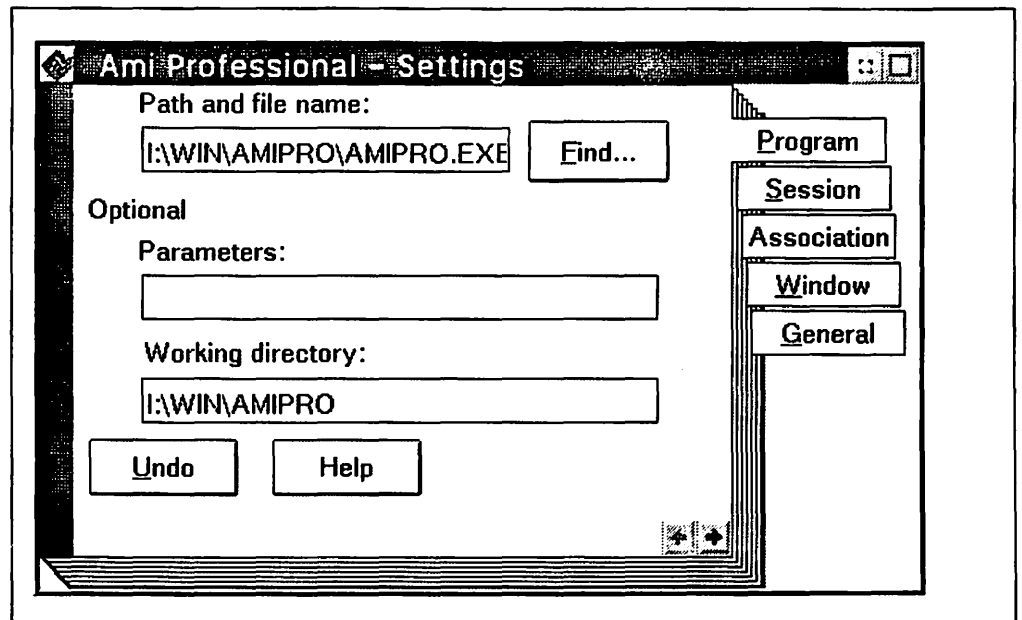
*Figure 39. Defining a Windows Application to OS/2 Version 2.0*

The *Session* page allows the user to change Windows settings via the *Windows Settings* dialog. This page defines whether the Windows kernel will execute in real, standard, or Auto-Select mode. *Auto-Select* mode is highlighted as the default. All DOS settings are selectable for Windows applications via the *Windows Setting* dialog; Windows settings are included in the same list.

## 8.5.1 Defining a Single Application VDM (SAVDM)

For a SAVDM, the Windows application name is entered into the *Path and Filename* field on the *Program* page of the *Settings* notebook. The Workplace Shell then determines the application type. Upon detecting that the application is a Windows application, the *Program Type* in the *Session* page of the notebook will be set to *Windows Full Screen*. When a user interactively creates a program object for a Windows application and the Workplace Shell determines it is a real mode application, *Windows Full Screen* will be marked as the default and the application will be started as a SAVDM.

Each SAVDM has its own icon on the Workplace Shell desktop, for the application within the SAVDM. This icon is the normal Windows icon for this application. The icon title text will be the text specified in the *Title* field in the *General* page of the *Settings* notebook.

## 8.5.2 Defining a Multiple Application VDM (MAVDM)

When defining a MAVDM, the user specifies WINOS2.COM for the *Path and Filename* field in the *Program* page of the *Settings* notebook. As explained above, the Workplace Shell detects that the application is a Windows application and sets the *Program Type* field in the *Session* page to *Windows Full Screen*.

The user may also define one or more Windows applications which will be activated when the VDM is started. This is achieved as follows:

1. The applications to be started are specified in the *Parameters* field of the *Program* dialog. Full path name and parameters should be specified.

   The syntax for the parameters field is:

```
/R|/S [{][!|^]App1 App-parms [,] .[!|^]App2 App-parms
```

- *IR* Windows real mode

- *IS* Windows standard mode

These parameter are active for the whole VDM and not on an application base.

- [ ] Optional Parameters

- *!* Start the Windows Application Minimized

- ^ Start the Windows Application Maximized.

No blank must be specified between '!' or '^' and the application name.

A MAVDM will be created if one of the following are present:

- {} Braces

- Comma separating the application names

- An application name is not passed as a parameter.

If neither the exclamation mark nor the caret is specified, the Windows application will start "normalized," approximately one third of the screen size.

Changes are effective immediately and are saved when the *Settings* notebook is closed or when the system is shut down. The *Default* button resets all settings to their previous values.

2. In the *Session* dialog *WIN-OS/2 full screen* must be selected.

The icon will be the *Windows full-screen* icon defined by WINOS2.COM. Individual icons for applications running in the MAVDM are not displayed on the Workplace Shell desktop.

## 8.5.3  Defining a "Seamless" WIN-OS/2 VDM

In order to obtain the highest integration of the Windows V3.0 and Windows V2.x application into the OS/2 V2.0 Workplace Shell, the following definitions must be provided in order to have the application execute in seamless mode:

1. Select the *Program* page of the program object's *Settings* notebook.

2. Enter the Windows application name in the *Path and Filename* field. A fully qualified path and filename is necessary.

3. Select the *Session* page of the program reference object's *Settings* notebook.

4. Click on the *WIN-OS/2 window* radio button.

5. All DOS settings are selectable for Windows applications via the *Windows Settings* page dialog; Windows settings are included in the same list.

The "*WIN-OS/2 Window*" radio button indicates that the associated Windows application program is to be initiated in a "seamless" Windows VDM. This is a single application VDM. However, the WIN-OS/2 Program Manager may be registered with the Workplace Shell as a "seamless" WIN-OS/2 program object. Once the Program Manager is up and running, the user may launch any other Windows program from it. Of course, this assumes that other Windows programs were installed through the Program Manager, and that they all run in the same WIN-OS/2 session.

A new parameter is added to the SYSTEM.INI file:

WOS2VDMApps = !clipwos2,!ddeagent

The WIN-OS/2 Program Manager reads this line for "seamless" WIN-OS/2 VDM, to determine which applications to pre-start.

Since "WIN-OS/2 Window" is the default Windows application setting, all Windows program objects that are created through the Windows Migration utility (for standard mode Windows applications) will default to that setting. If a user chooses to execute a Windows application by double clicking its program file's icon in the *Drives* folder, rather than its program object icon (if any), then the Workplace Shell will attempt to initiate it in a "seamless" Windows environment.

## 8.6 Starting Windows Applications

The following methods may be used to start Windows applications:

1. Select the application's program file from within the *Drives* folder. This method is not recommended, as it will not pick up any optimized DOS and/or Windows settings.

2. Enter the application name at an OS/2 command line prompt. This method is not recommended, for the same reason as stated above.

3. Install the application in a folder, in the Workplace Shell desktop as described in 8.5, "Defining Windows Applications" on page 145, and start it by double-clicking the mouse on its icon. This is the preferred method for starting a Windows application.

If the application is started from either the *Drives* Folder or an OS/2 command prompt, a SAVDM will be created. If the application is started from an icon, either a SAVDM, a MAVDM or a "seamless" WIN-OS/2 VDM will be created, depending on how the application was defined at installation.

### 8.6.1 SAVDM

A SAVDM is created for the execution of a single Windows application. The Workplace Shell actually starts WINOS2.COM as the application in the VDM, and the application to be started in the VDM is passed as a parameter to WINOS2. This process is transparent to the user; the definition of the SAVDM in the *Settings* notebook uses the Windows application name only.

If WINOS2 is to execute in real mode, the /r option will be automatically inserted into the parameter list for the VDM creation, based on the WIN-OS/2 settings. If standard mode was highlighted, /s is passed as a parameter to WINOS2. The default is to pass no Windows options, only the application name.

When the Windows application is terminated, WINOS2.COM terminates, which in turn causes the VDM to be terminated.

### 8.6.2 MAVDM

In the case of a MAVDM, the Windows Program Manager is loaded in the VDM, transparently to the user. The Program Manager's window is displayed maximized, and applications are then launched from the Windows Program Manager. In this case, the Workplace Shell's Window List will display the name of the Windows kernel (WINOS2.COM) executing in the VDM. It will not show each individual Windows application running within that MAVDM. This is different from a

SAVDM, where the Workplace Shell's Window List will display the name of the Windows application.

## 8.6.3 "Seamless" WIN-OS/2 VDM

OS/2 Version 2.0 does not allow a "seamless" WIN-OS/2 VDM to be started from a DOS or OS/2 command prompt, nor is there any programmed interface for starting a "seamless" WIN-OS/2 VDM application from a Presentation Manager application. In addition, a SAVDM or MAVDM cannot be switched to a "seamless" WIN-OS/2 VDM once the virtual DOS machine is running.

The "seamless" WIN-OS/2 VDM execution mode allows Windows applications to run from the Workplace Shell desktop in a manner that is virtually indistinguishable from other OS/2 and DOS applications. Double clicking a "seamless" WIN-OS/2 VDM application icon will cause that application to be run in a window on the Workplace Shell desktop. This single application "seamless" WIN-OS/2 VDM environment will be a separate Windows VDM, where the Program Manager is not visible.

In order to be perceived as a compatible part of the Workplace Shell environment, the "seamless" WIN-OS/2 VDM application's execution must be controllable in the same manner as the execution of other OS/2 and DOS applications. This has several implications:

- The name of each active "seamless" WIN-OS/2 VDM application will be included in the Workplace Shell Task List.

- Each "seamless" WIN-OS/2 VDM supports an appropriate context menu.

- The user is able to cycle through all open Workplace Shell windows in the standard *Alt-Esc* manner.

- The minimize/hide icons on Windows applications function consistently with other such icons on the Workplace Shell desktop.

- The window controls on a "seamless" WIN-OS/2 VDM window operate in the same fashion as analogous Workplace Shell controls. The display style of the window controls on a "seamless" WIN-OS/2 VDM window will be "Windows-style" however.

- When a "seamless" WIN-OS/2 VDM Windows application terminates, its Workplace Shell window is also terminated.

This approach allows the appearance of a Windows application executing in a "seamless" WIN-OS/2 VDM to conform as closely as possible to that seen when running in a native DOS/Windows environment, while its behavior is as close as possible to that of a normal Presentation Manager/Workplace Shell application.

### 8.6.3.1 If You can't get a "Seamless" WIN-OS/2 VDM to Work

Starting a Windows application requires a number of configuration options to be correctly completed prior to starting the application. Failure to do so may result in the application failing to start. This is typically due to one of three problems:

1. Failures that are due to the configuration of the overall OS/2 V2.0 system.

2. Failures that are due to the configuration of the overall Windows environment.

3. Failures that are due to the nature of a particular Windows application.

The first two classes result from not being "seamless capable"; that is, some part of the system's configuration is not set up properly for "seamless" WIN-OS/2 VDM operation. For example:

- An OS/2 V2.0 video device driver other than the "seamless" **VGA** driver is installed.

- VWIN.SYS is not installed, due to the following line being missing in CONFIG.SYS:

  DEVICE = C:\OS2\MDOS\VWIN.SYS

- The Windows video device driver referenced by the SDISPLAY.DRV= statement in SYSTEM.INI file does not point to the "seamless" Windows VGA device driver. The correct entry in the *SYSTEM.INI* is:

  SDISPLAY = SWINVGA.DRV

The third class arises because the "seamless" execution environment will be a SAVDM that runs in standard mode only. This means that real mode Windows applications will not be able to run in "seamless" WIN-OS/2 VDM ("seamless" VDMs). The Workplace Shell is able to gracefully handle the initiation of real mode Windows applications, because it can determine the mode of a Windows application from its program header. Such Windows applications will be automatically initiated in full-screen SAVDMs.

In addition, groups of applications which depend on the sharing of global Windows memory will not be able to run in a "seamless" WIN-OS/2 VDM, unless it is possible to manually initiate one application in the set and then have that application programmatically spawn the rest of the applications in the set. In theory, this situation should never occur because the casual sharing of Windows global memory is expressly against the Microsoft guidelines for Windows system programming. However, if there are applications that depend on such sharing, the user will have to explicitly know to run them in a full-screen MAVDM.

## 8.7 Windows Environment Settings

When Windows application support is selected during installation of OS/2 Version 2.0, a WIN.INI file is built. The options for devices selected for the OS/2 environment are included in this file.

Should the user migrate from a DOS/Windows 3.0 environment, the original WIN.INI created by Windows will be left unchanged. At installation time, the Windows installation process will examine the original AUTOEXEC.BAT file and search in all directories specified in the PATH statement in there for the original Windows WIN.COM file. If one exists, it will copy all Windows *.INI files and all the Windows application *.INI files from that same subdirectory into the \OS\MDOS\WINOS2 subdirectory. It will then modify these copies to adjust for the appropriate video, mouse, country and keyboard settings. This happens in accordance with the answers provided during the initial OS/2 setup.

```
┌─────────────────────────────────────────────────────────────┐
│  ┌───────────────────────────────────────────────────────┐  │
│  │                                                        │  │
│  │   Configure WIN-OS/2 Desktop                           │  │
│  │                                                        │  │
│  │   ┌─── WIN-OS/2 ──────────────────────────────────┐    │  │
│  │   │                                               │    │  │
│  │   │   ◉    Install standard WIN-OS/2 desktop       │    │  │
│  │   │                                               │    │  │
│  │   │   ◉    Copy WIN-OS/2 desktop from existing Windows** desktop │  │
│  │   │                                               │    │  │
│  │   │   ◉    Preserve WIN-OS/2 desktop currently installed │  │
│  │   │                                               │    │  │
│  │   └───────────────────────────────────────────────┘    │  │
│  │   ┌─── Additional Information ─────────────────────┐    │  │
│  │   │                                               │    │  │
│  │   │  Path to existing Windows system:             │    │  │
│  │   │                                               │    │  │
│  │   │  ┌─────────────────────────────────────────┐  │    │  │
│  │   │  │ |                                       │  │    │  │
│  │   │  └─────────────────────────────────────────┘  │    │  │
│  │   │                                               │    │  │
│  │   │   ☐   Update Windows desktop when WIN-OS/2 desktop is modified │  │
│  │   │                                               │    │  │
│  │   └───────────────────────────────────────────────┘    │  │
│  │                                                        │  │
│  │      ┌────────────┐          ┌────────────┐            │  │
│  │      │    OK      │          │   Help     │            │  │
│  │      └────────────┘          └────────────┘            │  │
│  │                                                        │  │
│  └───────────────────────────────────────────────────────┘  │
└─────────────────────────────────────────────────────────────┘
```
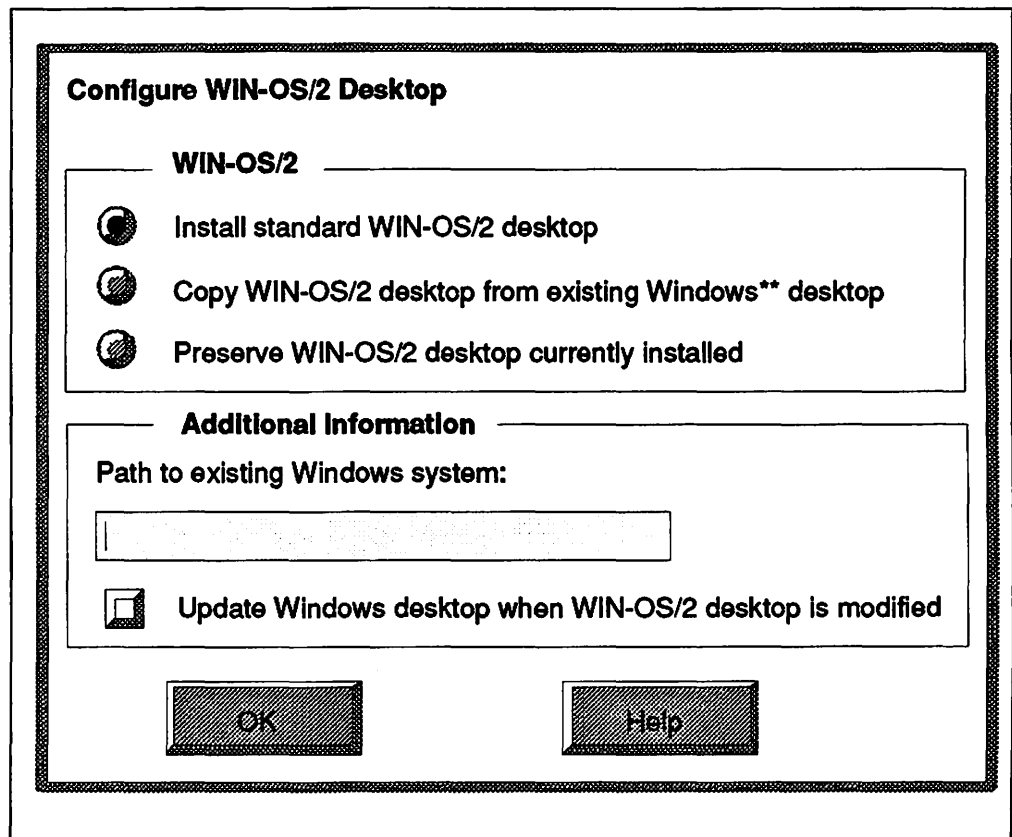
*Figure 40. Migrating the Windows Initialization Files*

The Windows group files (*.GRP) and other Windows application-specific *.INI files are also copied. The modified WIN.INI and PROGMAN.INI files will have their path statements modified to point to the new locations of these files.

Printer definitions for the Windows environment will also depend on the OS/2 setup, rather than on any previously defined printer device driver. Of course, all path statements in these files will be modified to point to the appropriate directories.

If a user installs OS/2 V2.0 over a previous version of OS/2 V2.0, the Windows install process will look for the existence of \OS2\MDOS\WINOS2\WINOS2.COM. If found, it will perform the same migration process from that base environment. If none of these files can be found, the Windows installation process will start from scratch, in the same way that a first-time Windows 3.0 installation would do it.

The following initialization files are created/modified:

• WIN.INI

• PROGMAN.INI

• CONTROL.INI

• SYSTEM.INI.

The initialization and group files are required to restore a corrupted Windows environment. Backups of these files should be taken prior to making any changes to this environment.

These files and their contents are described in the following sections.

## 8.7.1 WIN.INI

WIN.INI contains a number of sections which may be customized by the user, including which applications should be started or run when a Windows MAVDM is started. Each Windows application is recorded in a separate section indicating the drive and path to execute the application. The supported file extensions, for each application installed, are recorded in the *Extensions* section.

Users need to be careful when applying any changes to this file, especially when it comes to configuring of any device drivers. The user might install a device driver which either does not exist or is not supported under OS/2 V2.0, such as specialized video device drivers.

Most, but not all, Windows applications also have their private entries in the WIN.INI file. Usually, such entries consist merely of a pointer to the application's own working directory. However, some applications register all their installation-dependent configuration information, and may therefore be very dependent upon finding their data in this file. The migration process will take care of these entries and migrate them appropriately.

## 8.7.2 PROGMAN.INI

PROGMAN.INI contains the Windows Program Manager settings. This file contains the following sections:

- **Setting:** Describes the settings of the Program Manager, along with the user's preferences.

- **Groups:** Specifies the Program Groups that exist in Program Manager and the path name to their group files (*.GRP).

## 8.7.3 CONTROL.INI

CONTROL.INI contains the color and desktop settings for the Windows Control Panel. The following options are available:

- **Current:** Specifies the window color setting

- **Color Schemes:** Specifies the available color options

- **Custom Colors:** Specifies up to 16 customization colors

- **Patterns:** Specifies options for the desktop pattern.

## 8.7.4 SYSTEM.INI

SYSTEM.INI contains the global system information used by Windows when it starts. Changes are not effective until Windows is restarted.

The following sections are included:

- **Boot:** Lists the drivers and Windows modules. The OS/2 file contains new Boot section keywords which cover MAVDM and SAVDM default applications:

  **GOPM**     This program returns the user to the Workplace Shell

  **Clipboard**  The modified WIN-OS/2 Clipboard View program

  **DDEAGENT**
            The WIN-OS/2 DDE (Dynamic Data Exchange) program

  **Printman**  The modified WIN-OS/2 Print Manager program, in MAVDM only.

- **Boot.description:** Lists the names of devices the user can change using Windows Setup

- **Keyboard:** Contains information about the keyboard

- **NonWindowsApp:** This section should not contain any information, since non-Windows applications are started from the OS/2 desktop. In the case of a migrated Windows environment, this section might contain information, but under OS/2 V2.0 it will be ignored.

- **Standard:** Contains information required by Windows to run in standard mode

- **386Enh:** Contains information used by Windows to operate in 386 enhanced mode. This section is not used as OS/2 provides equivalent function.

## 8.7.5 DOS and WIN-OS/2 Settings

The following DOS settings are automatically defined for any Windows application under the Workplace Shell. If a user explicitly modifies these entries, following these settings is recommended:

| | |
|---|---|
| **WIN_RUNMODE** | AUTO |
| | OS/2 V2.0 will decide whether the Windows application will run in real or standard mode, unless the user specifically selects a mode. |
| **KBD_CTRL_BYPAS** | CTRL_ESC |
| | This keyboard sequence is required in a WIN-OS/2 MAVDM in order to bring up the Windows List. If not bypassed, the Ctrl+Esc sequence is trapped by the Workplace Shell. |
| **MOUSE_EXCLUSIVE_ACCESS** | ON |
| | Only necessary if the user wishes to run this SAVDM or MAVDM in a window under the Workplace Shell. Note this pertains to running the entire VDM in a Presentation Manager window, *not* to running in seamless mode. |
| **DPMI_MEMORY_LIMIT** | 2 (MB) |
| | This is the default for the Windows environment and can always be increased when needed. However, decreasing this value is *not* recommended. |
| **IDLE_SENSITIVITY** | 75 |
| | This is the default value. The performance of some Windows applications can be improved by re-adjusting this value to their specific needs. In particular, applications which make extensive use of the mouse may exhibit "jumpy" mouse movement when IDLE_SENSITIVITY is allowed to default; for such applications, IDLE_SENSITIVITY should be set to 100, which disables idle detection. |

## 8.8 Windows Device Drivers

Upon installation, the WIN.INI file is updated with the appropriate information for the following options. Installation will install the following Windows device drivers in the appropriate directories:

- Keyboard
- Mouse
- Video
- Printer
- Codepage/country.

If a device driver is supported in Windows but *not* supported by OS/2, the Windows version will not be supported.

---
**Note**

Any "illegal" combination of OS/2 and Windows display device drivers may cause the Windows environment to crash or not to come up at all.

---

On the other hand, the user can configure a useful dual screen configuration, which will actually run the Workplace Shell on one screen and Windows on the other simultaneously. OS/2 V2.0 may be run with the standard system display, such as VGA, XGA and so on, and in addition, another display adapter may be installed to run Windows applications, such as the IBM PS/2 Image Adapter/A, which is a Micro Channel card and supported on PS/2s. This requires the appropriate Windows display device driver to run exclusively on that adapter and the screen connected to it. Of course, the user must change several things (examples shown relate to the Image Adapter/A):

**CONFIG.SYS**  Add "DEVICE = C:\MYSUB\IADOSRFS.SYS"

This may be done via the DOS Settings for that particular Windows session object under the Workplace Shell.

**SYSTEM.INI**  Change "display.drv = IAA.DRV"

**IASETUP.EXE**  This is a DOS program which needs to be run once after installation. This utility will actually add some special entries to the WIN.INI file, which will tell the *IAA.DRV* display device driver what resolution and color setup to use on the Image Adapter.

These device drivers and programs can be found on the Image Adapter support diskette.

---
**Note**

Do not confuse the scenario above with OS/2's standard dual screen support, which is installed and configured automatically if a VGA *and* 8514 (or XGA) are found during initial installation. In that case, only one screen will be active at any given moment, while the display of the other will be frozen.

---

## 8.9 Print Support for Windows Applications

The installation procedure will update the new WIN.INI file to include the printer device driver details required by Windows for printers selected under OS/2. Installation selects a Windows printer device driver comparable with the OS/2 printer device driver for that printer. The Windows printer device driver will initially operate in its default mode. If the printer device driver must be configured in a mode other than the default mode, the printer should be configured from within the Windows Control Panel.

If there is no equivalent OS/2 printer device driver, the Windows device driver should be installed and configured via the Windows Control panel. The user should also use a printer port which is associated with the IBMNULL.DRV PM printer device driver on the OS/2 side. This will ensure the print data is passed straight through the port without OS/2's print subsystem modifying anything within the data stream. Even the usual "printer reset" should not occur.

### 8.9.1 Print Subsystem Architecture

There are some interesting and significant changes to the OS/2 print subsystem architecture which are used to support Windows applications, and which are worth noting:

- The print subsystem has been expanded to provide printing support for Windows applications running on WIN-OS/2.

- The OS/2 file system now intercepts ALL print jobs routed to an LPTx port, even those directed to WIN-OS/2 LPT1 to LPT3 and WIN-OS/2 LPT1.OS2 to LPT2.OS2 ports, and routes them to the OS/2 spooler. Jobs routed to a COM port are not intercepted by the file system and can proceed directly to the physical port via the serial port device driver.

Figure 41 on page 156 shows the WIN-OS/2 details of the print subsystem architecture in more detail. The interesting feature to note here is that a second spooler is present; that is, the WIN-OS/2 spooler. The WIN-OS/2 spooler is the OS/2 V2.0 implementation of the Windows spooler. Similarly, the WIN-OS/2 Print Manager and WIN-OS/2 Control Panel represent the OS/2 V2.0 implementation of the Windows Print Manager and Windows Control Panel. There are minor user changes apparent in the WIN-OS/2 Control Panel, but these provide extra function rather than take it away.
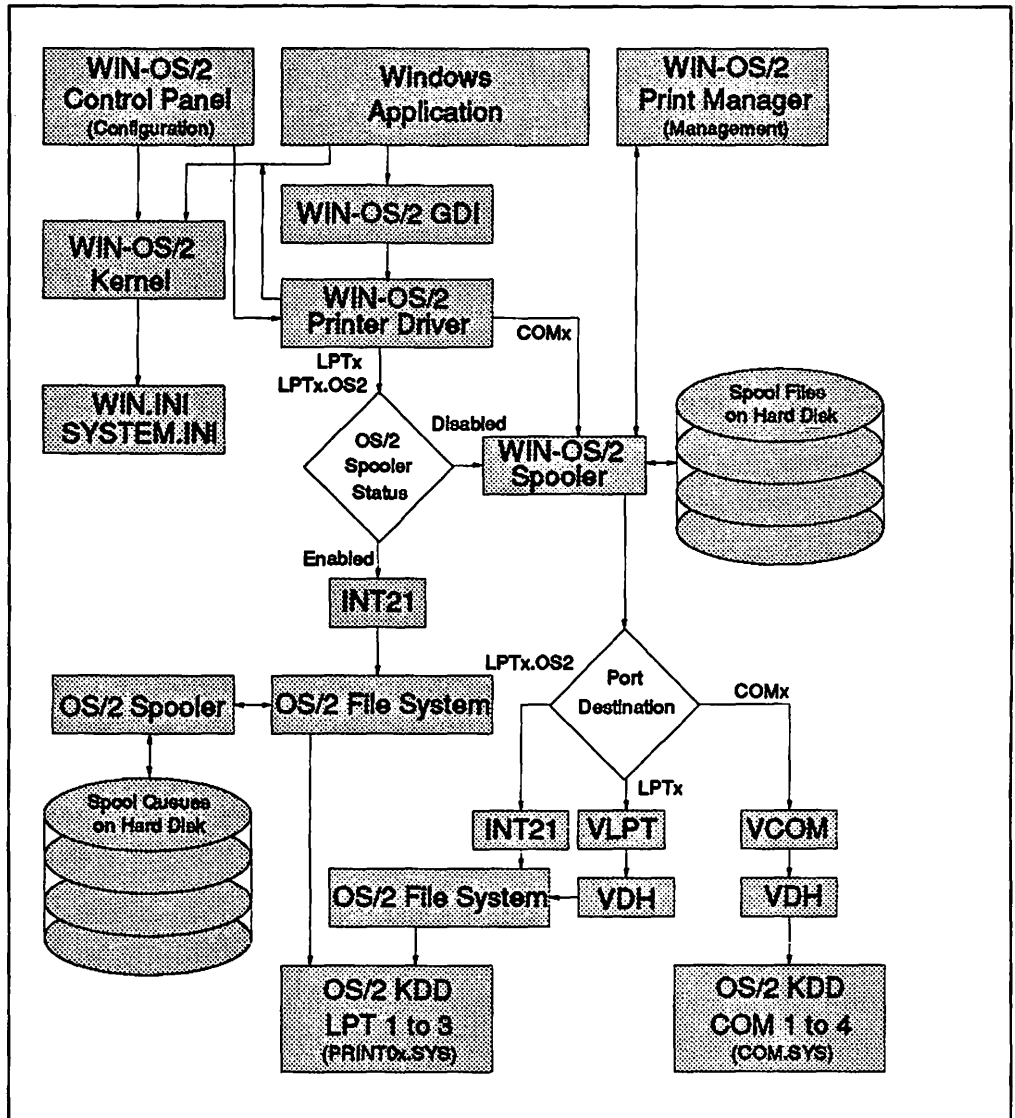
Figure 41. Detailed View of the WIN-OS/2 Data Connections

For WIN-OS/2 printing, raw print data is generated via the WIN-OS/2 printer driver and GDI (Graphical Device Interface). The WIN-OS/2 printer driver directs the data to the appropriate port but the data route then taken varies depending on whether or not the OS/2 spooler is enabled, as shown in Figure 42 on page 158.

If the OS/2 spooler is enabled, an INT21 is issued which provides a direct path for the print data to come into the OS/2 V2.0 file system. Jobs directed to LPTx or LPTx.OS2 ports are intercepted by the file system and are sent on to the SplQmxxx interface. The print data is then processed by the print subsystem as though it was raw data arriving from a PM queued application. Note that for this scenario, the print data is processed by the WIN-OS/2 printer driver and also part 2 of the equivalent OS/2 printer driver.

> **Note**
>
> It is important to ensure that the WIN-OS/2 and OS/2 printer drivers match to avoid conflict between them. If you use a WIN-OS/2 driver which has no OS/2 equivalent then use the IBMNULL driver.

Print jobs directed to COMx ports are not intercepted by the file system as for LPTx/LPTx.OS2 ports. Instead, they are passed directly to the serial kernel device driver.
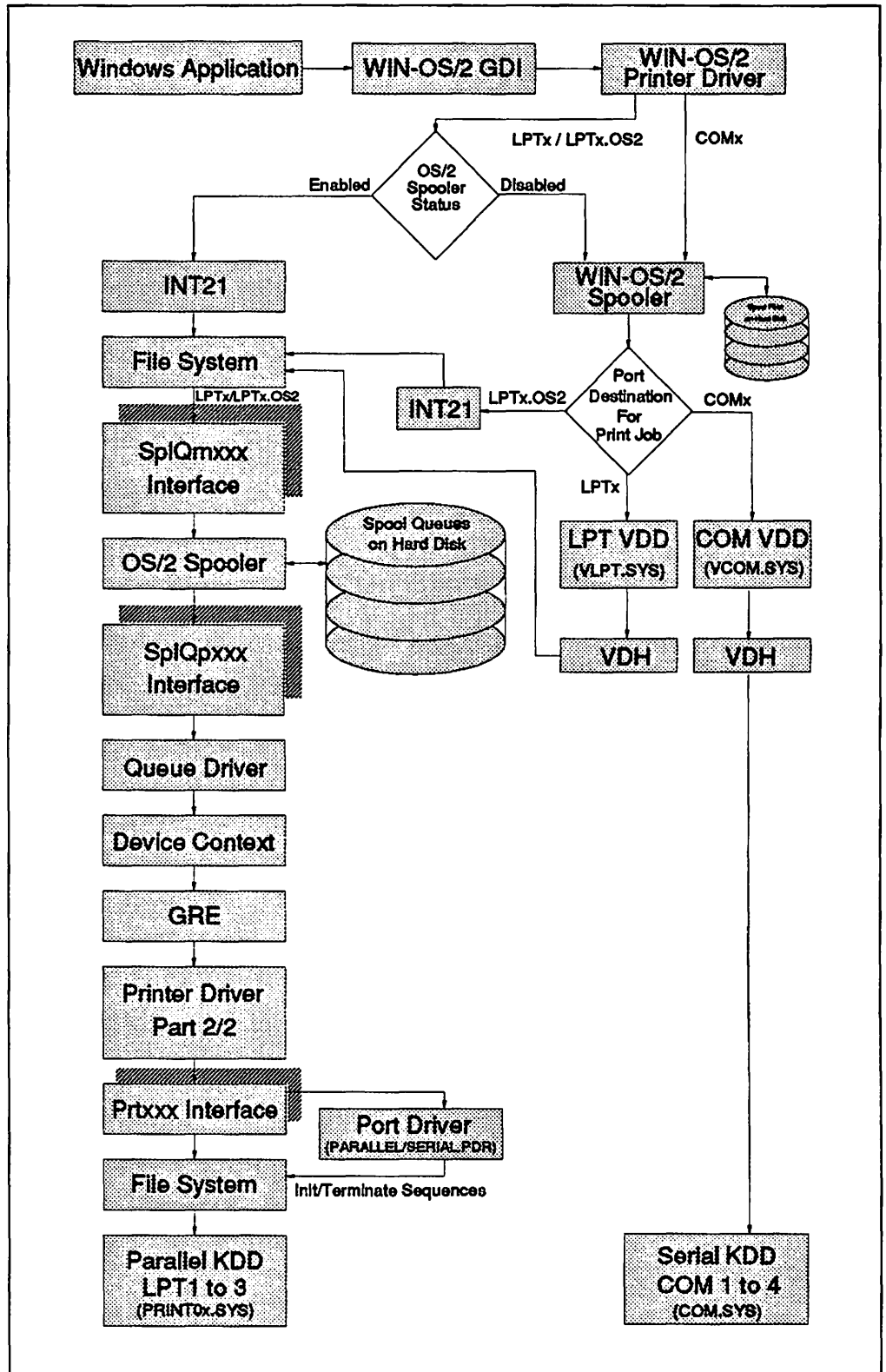
*Figure 42. Low Level View of the WIN-OS/2 Printing Data Flow*

If the OS/2 spooler is disabled, the print data bypasses many of the print sub-system components. In this scenario, the WIN-OS/2 spooler will be called upon to spool the print job, which is actually written to the root directory of the fixed disk. The queued spool files are distinguished by having the file extension .TMP.

If the WIN-OS/2 spooler is also disabled then the print job passes straight through.

With the OS/2 spooler disabled, there are three routes that print jobs can take, according to their port destination:

1. When it is the turn of a COMx job to be printed, the WIN-OS/2 spooler passes the print job to the COM VDD (Virtual Device Driver). The reason for this is that the job will ultimately be printed through the OS/2 serial KDD (Kernel Device Driver) which is COM.SYS. A VDD cannot call upon physical device drivers, such as COM.SYS, directly. Instead it must call on the services of the VDH (Virtual Device Helper) which is a programming interface that is able to do this on behalf of the VDD. Consequently, the VDD passes the print data to the serial KDD via the VDH.

> **Note**
>
> If you are printing to a COMx port, the WIN-OS/2 printer device driver needs to initialize that port and know about the handshaking protocol. To specify the correct settings, you will have to use the WIN-OS/2 Control Panel and click on the *Ports* icon.
>
> You should also make sure that you have installed the matching PM printer device driver under the Workplace Shell. If you don't have a matching version, use the IBMNULL printer device driver. This printer object needs to be assigned to the same COMx port and the *settings* must match the settings on the WIN-OS/2 side as well as the current printer setup.
>
> If you don't do that, the printing result will be unpredictable, especially for large and complex print jobs.

2. Jobs directed to LPTx are routed to the parallel physical device driver, PRINT0x.SYS.SYS, via INT21.

3. Jobs sent to LPTx.OS2 are routed directly to the parallel physical device driver from the WIN-OS/2 spooler.

> **Recommendation**
>
> It is strongly recommended that users operate the print subsystem with both spoolers enabled. Otherwise, print data from different jobs may become mixed up, and individual applications may have to wait until printing is completed before resuming execution.

For more details about the entire print subsystem, including DOS and WIN-OS/2 sessions, readers should refer to *OS/2 Version 2.0 - Volume 5: Print Subsystem.*

## 8.10 Font Support

The following discussion can also be found in *OS/2 Version 2.0 - Volume 5: Print Subsystem*, including more details about Presentation Manager, the Workplace Shell, and the print subsystem.

Fonts are used for output by the system to two devices:

1. Displays

2. Printers.

With the many types of displays and even more numerous types of printers one can imagine the complexity of tracking who can use which fonts and what do they look like on a 75 dot display as well as a 300 dot printer.

OS/2 Version 2.0 utilizes Adobe** Type Manager (ATM) for this specific purpose. There are two ATMs present in OS/2 Version 2.0, one for OS/2 PM applications and the other for WIN-OS/2 applications. These ATMs allow the system to provide a seamless look and consistent output while using *most* applications. Because there are two ATMs with some duplicate files, however, user installation and management is critical.

## 8.10.1 Adobe Type Manager Overview

ATM provides WYSIWYG text to all OS/2 V2.0 PM and WIN-OS/2 supported printers and displays. WYSIWYG (What You See Is What You Get) implies that what you see on the display is what you will see on the printed page. In reality, a 75 dot per inch display *can not* show you what a 300 dot per inch printer will produce. It is physically impossible. What it will give you is fully formed characters in virtually any size and a sense of the "balance" of the page with respect to line, word and letter spacing. With ATM you have the ability to install and use any of the hundreds of Type 1 Fonts compatible with the PostScript** Page Description Language. Thirteen Type 1 fonts are included in OS/2 V2.0 in four font groups (Times New**, Helvetica**, Courier and Symbol). This group provides a satisfactory basic working set, to which extra fonts can be added.

With ATM, users now have a wider choice of fonts, and can display and print them even on lower-cost devices. For example, PostScript-quality fonts can now be printed on non-PostScript devices such as the IBM 4019 and 4029, and the HP** LaserJet** series without the need to add the additional PostScript interpreter option. You still get excellent results, though slower performance, as the fonts will be rasterized in the PS/2 rather than the printer. Even an IBM 4201 or an IBM 5152 or other low-cost matrix printers can print Type 1 fonts but, of course, not with the same quality as laser printers. These same fonts can also be "installed" as downloadable fonts to be sent to PostScript printers for faster print performance.

The list of available fonts for *most* applications is a combination of the list in the printer device driver you have selected for output and the fonts you have installed in the ATMs. Tests with Lotus 1-2-3 for Windows, Aldus** PageMaker, and DeScribe** indicated that as soon as additional fonts were installed in the Font Palette of OS/2 PM and the ATM Control Panel of WIN-OS/2, all of these applications were able to include them in their font choice list, display them and print them. Some applications like CorelDraw!** 2.0 and Micrographx** Designer use their own internal outline format for fonts. They will use the same Type 1 fonts as ATM but the fonts must be installed seperately in each application.

Consult your application documentation to determine how each one handles fonts.

## 8.10.2  ATM File Formats

**Files: .FON**

These files hold the standard OS/2 V2.0 bitmap fonts and are copied there by the OS/2 V2.0 install procedure.

**Files: .PSF**

PostScript font files are the new Type 1 Font files in internal-to-PM format which is designed (for efficiency) for the core fonts *only* (Times New, Courier and Helvetica).

**Files: .AFM**

These files hold the Adobe Font Metrics and are used by OS/2 PM ATM. Information such as character widths and kerning pairs are in this file.

**Files: .PFB**

These files hold the Printer Font Binary and are used by both OS/2 PM ATM *and* WIN-OS/2 ATM. These are the files that can be downloaded into printer storage.

**Files: .PFM**

These files hold the Printer Font Metrics and are used by the PostScript device driver to download fonts and by WIN-OS/2 ATM.

**Files: .PFA**

These files hold the Printer Font ASCII and are embedded by the PostScript device driver into the PostScript print job if you installed them as downloadable fonts.

For non-ATM file formats, such as the native formats for PCL, PPDS and 420X printers.

OS/2 PM                    [OS/2 2.0 ATM Install]                    WIN-OS/2

         update                                              update

\CONFIG.SYS                                                    `\OS2\MDOS\WINOS2\SYSTEM.INI
\OS2\SYSTEM\OS2.INI

              install              install

                            \OS2\MDOS\WINOS2\ATM.INI
           \OS2\DLL\PMATM.DLL      \OS2\MDOS\WINOS2\ATMCNTRL.EXE
      IBM "Core Fonts" in \OS2\DLL  \OS2\MDOS\WINOS2\SYSTEM\ATMSYS.DRV
                            \OS2\MDOS\WINOS2\SYSTEM\ATM16.DLL

                            [Type 1 Font Install]

           update

         A^A                                              2.0  a
      Font Palette          Display Fonts              ATM Control Panel

    update    install                          install/create    update

           \OS2\DLL\font.AFM                    \PSFONTS\PFM\font.PFM
           \OS2\DLL\font.PFB                    \PSFONTS\font.PFB

\OS2\SYSTEM\OS2.INI

                            Non-PostScript
                            Printing
                                                  \OS2\MDOS\WINOS2\ATM.INI

      PostScript Driver
      Downloadable
      Font Install                            also if using PSCRIPT.DRV
                                              \OS2\MDOS\WINOS2\WIN.INI
           create

\OS2\DLL\PSCRIPT\font.PFA
\OS2\DLL\PSCRIPT\font.PFM         softfont#=font____.pfm,font____.pfb
                            PostScript
                            Printing

*Figure 43. File Structure of Adobe Type Manager*

## 8.11 ATM for WIN-OS/2

While ATM for OS/2 PM is integrated into the operating system, ATM for WIN-OS/2 is a separate program that is automatically installed into the WIN-OS/2 desktop for you. ATM for WIN-OS/2 is accessed by using the ATM Control Panel in the WIN-OS/2 Main group. If you plan to manage fonts frequently after your initial installation it is recommended that you create a program icon on your OS/2 PM desktop for the WIN-OS/2 ATM control panel. The next section will describe how to do this.

### 8.11.1 Installing ATM for WIN-OS/2

ATM for WIN-OS/2 is automatically installed when you install the OS/2 base code. However, the 13 "IBM Core Fonts" that are automatically installed for you in OS/2 PM are not installed under WIN-OS/2. In fact ATM for WIN-OS/2 is itself not active until you install at least one font. This is why you will see an "X" over the ATM icon when you start the WIN-OS/2 Full-Screen command prompt if you have not installed any fonts. Included on the device drivers diskettes are the necessary font files to install these "IBM Core Fonts" for WIN-OS/2. They are currently located on device driver diskette #5.

As mentioned previously, it is a good idea to have the WIN-OS/2 ATM icon on your OS2 PM desktop if you manage fonts frequently. There are two ways to do this:

1. You can use the Migrate Applications program in the System Setup folder. However, this action will place the icon in a folder called "Additional Windows Programs." You will then have to open that folder and drag-and-drop the ATM Control Panel icon onto your desktop.

2. You can also open the Templates folder and drag-and-drop the Program icon on your desktop and type in the prompts. ATM for WIN-OS/2 is located in \OS2\MDOS\WINOS2\ATMCNTRL.EXE. Use \OS2\MDOS\WINOS2 as your working directory. Under Session select WIN-OS/2 window if available, otherwise select WIN-OS/2 full screen.

Under certain conditions ATM may become disabled or you may want to remove it for performance reasons if you only use standard fonts. In either case ATM for WIN-OS/2 is activated based on entries in the SYSTEM.INI file located in the path \OS2\MDOS\WINOS2. If installed correctly you will see the following statements:

1. **system.drv = atmsys.drv**

2. **atm.system.drv = system.drv**

To disable ATM:

1. Delete the **atm.system.drv = system.drv** statement

2. Change **system.drv = atmsysdrv** to **system.drv = system.drv.**

### 8.11.2 Installing Additional Fonts for WIN-OS/2 ATM

Before installing *any* additional fonts for WIN-OS/2 make sure that *all* of your printers are installed and that they are configured for the correct output port. WIN-OS/2 maintains its font information in the ATM.INI for non-PostScript printers and in the WIN.INI for PostScript printers. For PostScript printers this means that PostScript printers installed *after* the font installation *will not* have those fonts in their list. Also if you change the output port of an installed PostScript printer

then your fonts will disappear unless the new port also had a PostScript printer assigned to it while the fonts were being installed.

To install an additional font:

1. Open the **ATM Control Panel**

2. Click on **Add...**

3. Double Click on the source of the new font(s)

4. Click on the **Font files** you wish to add

5. Click on **Add**

6. Click on **Exit**

You *must* restart Windows to access the new fonts. The system will prompt you to do this.

```
┌─── Important ───────────────────────────────────────────────────┐
│                                                                 │
│  When you install the fonts the system will prompt you with suggested paths │
│  for the files.  Although you may change the path, remember one thing:  OS2 │
│  PM ATM will be installing the .PFB file as well and will place it in a different │
│  directory.  You will be tempted to use the same directory for both OS/2 PM │
│  and WIN-OS/2.  We recommend that you use the default settings because │
│  when you delete a font with the OS/2 PM Font Palette it will also delete the │
│  .PFB font file!  This will make the font unusable, although still installed, in │
│  WIN-OS/2.  Allowing the system to keep your OS/2 PM and WIN-OS/2 fonts │
│  separate will save a lot of confusion in managing fonts.        │
│                                                                 │
└─────────────────────────────────────────────────────────────────┘
```

These files would have been added to the \PSFONTS directory after installing the Berthold Bodoni Antiqua fonts.

```
┌─── \PSFONTS. ───────────────────────────────────────────────────┐
│                                                                 │
│        5-01-90    8:00a    39648      0   bpbi____.pfb          │
│        5-01-90    8:00a    38664      0   bpb_____.pfb          │
│        5-01-90    8:00a    36930      0   bpi_____.pfb          │
│        5-01-90    8:00a    37381      0   bpli____.pfb          │
│        5-01-90    8:00a    35979      0   bpl_____.pfb          │
│        5-01-90    8:00a    38740      0   bpmi____.pfb          │
│        5-01-90    8:00a    38098      0   bpm_____.pfb          │
│        5-01-90    8:00a    35837      0   bprg____.pfb          │
│                                                                 │
└─────────────────────────────────────────────────────────────────┘
```

And these files would have been added to the \PSFONTS\PFM directory after installing the Berthold Bodoni Antiqua fonts.

```
┌─── \PSFONTS\PFM. ───────────────────────────────────────────────┐
│                                                                 │
│        4-10-92   10:42a    7016      0   BPBI____.PFM           │
│        4-10-92   10:42a    5830      0   BPB_____.PFM           │
│        4-10-92   10:43a    6019      0   BPI_____.PFM           │
│        4-10-92   10:43a    5934      0   BPLI____.PFM           │
│        4-10-92   10:42a    5700      0   BPL_____.PFM           │
│        4-10-92   10:43a    7869      0   BPMI____.PFM           │
│        4-10-92   10:43a    5571      0   BPM_____.PFM           │
│        4-10-92   10:43a    6004      0   BPRG____.PFM           │
│                                                                 │
└─────────────────────────────────────────────────────────────────┘
```

**Note:** The internal file format is different from the standard core fonts delivered with OS/2 V2.0. Those have been modified for performance reasons, where these fonts are installed as standard Type 1 fonts.

---
**Important**

As information is added to the ATM.INI file, don't try to install or remove fonts just by copying or erasing files. Adding or deleting fonts must be done with the ATM Control Panel and/or the Print Manager.

For faster performance and better typeset quality don't forget to either install these fonts as downloadable, or download them directly, if you are using them with a PostScript printer, which does not have them already built-in.

---

## 8.11.3 Deleting Fonts for WIN-OS/2 ATM

The standard core fonts and the additional Type 1 fonts have to be deleted with the ATM Control Panel. To delete them:

1. Open the ATM Control Panel

2. Click on the font(s) to be removed

3. Click on **Remove**

4. Click on **Yes** in the confirmation box

5. Click on **Exit**

You *must* restart Windows to use the updated font list in the ATM.INI file. The system will prompt you to do this. The soft font entries in the WIN.INI file, however, *will not* be deleted. This means that you will still "see" the deleted fonts in the font list for your applications if you choose a printer(usually PostScript) that has these entries. Although the font name will appear, when you select it you will not get the expected screen font as it has been deleted from ATM. Instead you will get an installed font, usually Times or Helv, depending on the font you selected.

In order to remove the deleted fonts from the lists you must edit the \OS2\MDOS\WINOS2\WIN.INI file. The following example shows you a before and after version of the WIN.INI file if you wanted to remove all Berthold Bodoni Antiqua fonts from your application font lists.

```
┌─── WIN.INI after ATM delete but before manual edit. ──────────────────┐
│ [PostScript,LPT2.OS2]                                                  │
│ device=36                                                             │
│ feed1=1                                                              │
│ feed3=1                                                              │
│ feed5=1                                                              │
│ feed15=1                                                             │
│ softfonts=12                                                         │
│ softfont1=c:\psfonts\pfm\archi___.pfm,c:\psfonts\archi___.pfb        │
│ softfont2=c:\psfonts\pfm\balleeng.pfm,c:\psfonts\balleeng.pfb        │
│ softfont3=c:\psfonts\pfm\bpb_____.pfm,c:\psfonts\bpb_____.pfb        │
│ softfont4=c:\psfonts\pfm\bpbi____.pfm,c:\psfonts\bpbi____.pfb        │
│ softfont5=c:\psfonts\pfm\bpl_____.pfm,c:\psfonts\bpl_____.pfb        │
│ softfont6=c:\psfonts\pfm\bpli____.pfm,c:\psfonts\bpli____.pfb        │
│ softfont7=c:\psfonts\pfm\bprg____.pfm,c:\psfonts\bprg____.pfb        │
│ softfont8=c:\psfonts\pfm\bpm_____.pfm,c:\psfonts\bpm_____.pfb        │
│ softfont9=c:\psfonts\pfm\bpmi____.pfm,c:\psfonts\bpmi____.pfb        │
│ softfont10=c:\psfonts\pfm\bpi_____.pfm,c:\psfonts\bpi_____.pfb       │
│ softfont11=c:\psfonts\pfm\blackcha.pfm,c:\psfonts\blackcha.pfb       │
│ softfont12=c:\psfonts\pfm\saintfra.pfm,c:\psfonts\saintfra.pfb       │
└───────────────────────────────────────────────────────────────────────┘

┌─── WIN.INI after manual edit. ────────────────────────────────────────┐
│ [PostScript,LPT2.OS2]                                                  │
│ device=36                                                             │
│ feed1=1                                                              │
│ feed3=1                                                              │
│ feed5=1                                                              │
│ feed15=1                                                             │
│ softfonts=4                                                          │
│ softfont1=c:\psfonts\pfm\archi___.pfm,c:\psfonts\archi___.pfb        │
│ softfont2=c:\psfonts\pfm\balleeng.pfm,c:\psfonts\balleeng.pfb        │
│ softfont3=c:\psfonts\pfm\blackcha.pfm,c:\psfonts\blackcha.pfb        │
│ softfont4=c:\psfonts\pfm\saintfra.pfm,c:\psfonts\saintfra.pfb        │
└───────────────────────────────────────────────────────────────────────┘
```

**Note:** In addition to deleting the line entries you must also renumber the remaining lines and edit the total number in the **softfonts=** statement. If you have more than one printer installed you may have to edit other groups in the WIN.INI under other port entries.

## 8.12 Clipboard Support

OS/2 Version 2.0 provides clipboard support between Windows applications, in the same or separate VDMs, as well as support between Windows applications and OS/2 Presentation Manager[4] applications. The clipboard serves as a data-exchange feature acting as a common area to store data handles through which applications exchange formatted data. The same data may be represented in a number of different formats as specified by the application. Note that objects in the clipboard may be of any size and format.

---

4 OS/2 VIO applications may also use the Presentation Manager clipboard, by using the appropriate Win calls; for example, CTC's SPF/2** editor provides this function.

The combined OS/2 and Windows clipboard environment under OS/2 Version 2.0
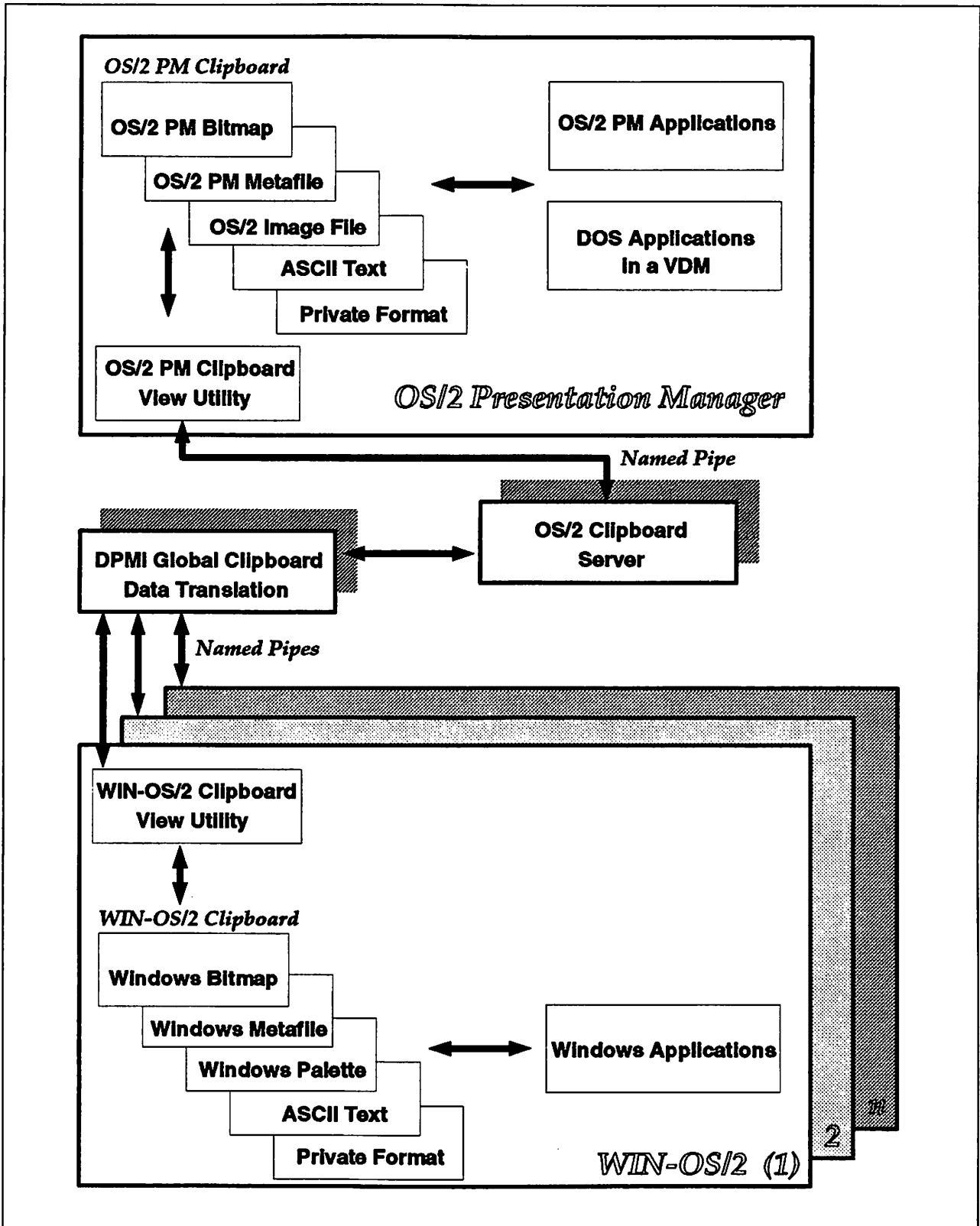is shown in Figure 44 on page 167 [5].



*Figure 44. OS/2 Version 2.0 Clipboard Environment*

Data is formatted in either a predefined or private format, before being copied to the clipboard. In most cases the data is copied to pre-allocated global memory and a function .call is used to copy the memory handle to the clipboard. Windows provides a number of predefined data formats:

**TEXT**      Null-terminated text

**OEMTEXT**   Null-terminated text using an OEM character set

**PICTURE**   Metafile picture structure

**BITMAP**    Device-dependent bitmap

**DIB BITMAP**
              Device-independent bitmap

**SYLK**      SYLK Standard data interchange format

**DIF**       DIF standard data interchange format.

**TIFF**      Tag Image File Format

**Any Private Format**
              Will be kept in the same format. This will be usable only by applications which *know* about this private format.

These formats will be recognized and exported/imported by the global clipboard server.

Data formats which are not supported by the clipboard server, cannot be transferred. Such formats can only be handled by the *local* clipboards. This means that such formats may only be used to exchange data between Presentation Manager applications, or between Windows applications running in the same VDM. In addition, some of the data formats will also be converted, because of several differences between Windows and Presentation Manager data structures. This is further discussed on page 171.

The *OwnerDraw* feature in the Windows clipboard is only supported within a MAVDM, as shared memory is required. OwnerDraw is a process whereby a Windows application takes control over the appearance of menu items and has responsibility for managing these menu items.

The native Microsoft Windows 3.0 clipboard provides support for both Windows applications and non-Windows applications. Non-Windows applications (that is, "native" DOS applications) run in either full-screen or "windowed" mode. This kind of environment is not supported in a MAVDM, because it is supported by the Workplace Shell directly. Therefore, clipboard support for native DOS applications is provided through Presentation Manager.

Should the user wish to capture the contents of a VDM running in full-screen mode, the following approach is adopted:

1. Switch to the Presentation Manager screen containing the VDM

2. Select the System menu on the VDM icon

3. Select *Copy All*.

---

[5] Note that not all data formats shown in this diagram will be supported by the global clipboard server. This is discussed in detail on the following pages.

This procedure will copy the VDM's video buffer to the Presentation Manager clipboard (either in ASCII format or as a Presentation Manager bitmap).

Selective copy is available in windowed mode. When a VDM is running in a window, the user may mark a specific rectangular area which will then be copied into the clipboard.

This level of clipboard data exchange is fully supported by the VDM itself. The Windows kernel is not involved at all.

## 8.12.1 WIN-OS/2 Clipboard Support

The WIN-OS/2 clipboard view utility will display the captured data in a number of formats, either predefined or private. *Auto* displays the data in the format it had when placed onto the clipboard.

The clipboard view program CLIPWOS2.EXE, installed in C:\OS2\MDOS\WINOS2, is available within each SAVDM and MAVDM by default. This is a modified version of the original Windows 3.0 clipboard program.

A Clipboard Server (Global Clipboard) runs as a protected mode background process under OS/2 Version 2.0, to service clipboard functions between VDMs. If the Clipboard Server is not executing, clipboard functions are limited to within a single VDM. The Clipboard Server (\OS2\MDOS\WINOS2\VDMSRVR.EXE) is automatically started with the first WIN-OS/2 VDM.

Should a user elect to exit from the Windows clipboard, a warning message will be displayed, advising that exiting will terminate public clipboard functions. The clipboard functions within each VDM are public by default, unless explicitly set to LOCAL, which restricts clipboard activity to that WIN-OS/2 session only.

The WIN-OS/2 clipboard viewer pull-down menus have been enhanced to include support for an *Options* menu, which contains the *Public Clipboard* option. Selecting this option causes changes to the local clipboard to be reflected in the public clipboard and vice versa. When deselected, the contents of the public and local clipboards will not affect each other.

The *File* pull-down menu now supports *Import/Export* functions; *Public* must be deselected from the *Options* pull-down menu before *Import/Export* can be selected.

**Export** will copy the current contents of the local clipboard to the Public clipboard.

**Import** will copy the contents of the public clipboard to the Local clipboard.

---
**Implementation Notes**

The Import/Export functions communicate via named pipes to the \pipe\CLPAgent to the clipboard program (CLIPWOS2.EXE) within each VDM. This could cause performance problems on systems which already' utilize named pipes heavily for other purposes or when the content of the clipboard data is too big, for example huge bitmaps. If you don't need to exchange clipboard data outside a MAVDM, you could keep it *local* and therefore get around any possible performance problems.

---

## 8.12.2  Using Cut and Paste

The following three scenarios describe the clipboard functions:

1. Cut and Paste from a Windows Application in a VDM to another application in a separate VDM; *Public* is deselected.

2. Cut and Paste between two Windows applications within the same VDM (MAVDM).

3. Cut and Paste between the OS/2 and Windows environments. Cut and Paste within the OS/2 environment remains essentially unchanged.

### 8.12.2.1  Scenario 1 - Cut/Paste Between Independent WIN-OS/2 Sessions

1. Cut the data into the local Windows VDM clipboard.

2. Select *Export* from the clipboard pull-down menu. The data is copied into the external clipboard.

3. Select the VDM containing the destination Windows application.

4. Select *Import* from the clipboard pull-down menu. The data is copied from the external clipboard into the local clipboard of the receiving VDM.

5. Paste the data into the destination Windows application.

**Note:**  Steps 2 and 4 above are not necessary if the clipboard is public in the source and destination VDM.

### 8.12.2.2  Scenario 2 - Cut/Paste Within A MAVDM

1. Cut the data into the Windows VDM clipboard

2. Paste the data from the clipboard into the destination application.

### 8.12.2.3  Scenario 3 - Cut/Paste Between OS/2 And WIN-OS/2

The OS/2 V2.0 clipboard is activated upon loading the operating system.  A new OS/2 utility named CLIPVIEW.EXE is located in the OS2\APPS\ directory, and has been provided to support the extended clipboard functions. CLIPVIEW.EXE must be started in order to view and transfer the contents of the OS/2 V2.0 clipboard.

With the exception of the *File* option of the Windows clipboard, the same pull-down menus are provided. The *Render* option is the same as the *Display* option in the Windows clipboard.  *Render* will display the contents of the clipboard in a number of different formats.  Because the contents of the clipboard are stored in separate areas in memory, it is possible to view both the ASCII (text) and graphics contents of the clipboard.

> **Note**
>
> An application may or may not clear the entire contents of the clipboard, prior to copying data to it.  For example, the system editor will always erase the entire Presentation Manager clipboard, and therefore any public Windows clipboard as well, before it copies its text data into it.  On the other hand, some of the Presentation Manager applications do not behave that way. They will only override the specific data areas which are copied into the clipboard and leave the other ones in the clipboard unchanged.

The global Windows VDM clipboard is visible to the Presentation Manager clipboard.  CLIPVIEW.EXE has been enhanced to perform the following two activities:

1. Update the Presentation Manager clipboard when changes are made to the global VDM Clipboard

2. Update the global Windows VDM clipboard when changes are made to the Presentation Manager clipboard.

The Presentation Manager clipboard server application is registered as "clipboard viewer" to receive notification of clipboard updates. This ensures that the following messages are forwarded to the clipboard server, so that when updates are made to the Presentation Manager clipboard, messages are sent to the Presentation Manager CLIPVIEW.EXE.

- **WM_DESTROYCLIPBOARD:** Signals that the contents of the clipboard are being destroyed

- **WM_DRAWCLIPBOARD:** Signals an application to notify the next application in the chain of a change to the clipboard

- **WM_HSCROLLCLIPBOARD:** Requests horizontal scrolling of the clipboard contents

- **WM_PAINTCLIPBOARD:** Requests painting of the contents of the clipboard

- **WM_RENDERALLFMTS:** Notifies the owner of the clipboard that it must render clipboard data in all possible formats

- **WM-RENDERFMT:** Notifies the clipboard owner that it must format the last data copied to the clipboard

- **WM_SIZECLIPBOARD:** Notifies the clipboard owner that the clipboard application's window size has changed

- **WM_VSCROLLCLIPBOARD:** Requests vertical scrolling of the clipboard contents.

---

**Note**

No changes have been made to the Presentation Manager API functions to accommodate this design. Presentation Manager applications will not notice any difference; there appears to be just one (Presentation Manager) clipboard as it always used to be. The same is true for Windows applications as well.

---

Data formats are translated from Presentation Manager to Windows formats and vice versa, as required. This translation is performed when data is placed in the global clipboard. The following data formats will be translated between Presentation Manager and Windows:

**Windows DIB bitmaps:**

The Windows device-independent bitmaps are translated to/from OS/2 Presentation Manager bitmaps.

**Windows bitmaps:** This translated pre-Windows 3.0 formatted bitmaps to OS/2 Presentation Manager bitmaps.

**Note:** This is a one way only translation.

**Windows Metafiles:** Windows metafiles are first internally converted to the Windows DIB format by the Windows clipboard viewer, before being forwarded to the global clipboard.

| PM Metafiles: | PM metafiles are first internally converted to the PM bitmap format by the PM clipboard viewer, before being forwarded to the global clipboard. |
| Text: | ASCII text will be translated in both directions. If the sending and receiving environment are using a different codepage, the appropriate codepage translation will take place. |

## 8.13 Dynamic Data Exchange

This section describes Dynamic Data Exchange (DDE) support between Windows applications in a full-screen VDM.

### 8.13.1 DDE Concepts

DDE is a message protocol for dynamic data exchange between Windows programs. Data may be shared among applications, the intention being to create an integrated Windows environment.

**Client, Server and Conversation:**

Two applications participating in dynamic data exchange are said to be engaged in a DDE **conversation**. The application which initiates the conversation is the client application. The application which responds to the client is the server application. An application may be engaged in several conversations at the same time, acting as a client in some conversations and as a server in others.

A DDE conversation takes place between two windows, one for each of the participating applications. The window may be the main window of the application, a secondary window associated with the application, or a hidden window. A hidden window is typically used to process DDE messages.

DDE identifies the units of data passed between the client and server with a three-level hierarchy of:

- Application name
- Topic
- Item.

Each DDE conversation is uniquely identified by the application name and topic. The *application name* is normally the name of the server application. The *topic* is a general classification of data, within which multiple data items may be exchanged during the conversation. The *item* is the actual information related to the conversation topic that is exchanged between the applications. Values for the data item can be passed from the server to the client, or from client to server. The format of the data item may be any one of the clipboard formats (see 8.12, "Clipboard Support" on page 166).

Once a DDE conversation has been initiated, the client may establish one or more permanent data links with a server. A data link is a communication mechanism by which the server notifies the client whenever the value of a given data item changes. The link is permanent in the sense that the notification process continues until the data link or DDE conversation is terminated.

The DDE link may be **warm** or **hot**. In a warm data link, the server notifies the client that a value of a given data item has changed, but the server does not actually send the data value to the client until the client requests it. In a hot data link, the server immediately sends the changed data value to the client.

Applications which support DDE typically provide a *Copy/Paste* command in the *Edit* menu to allow the user to establish a DDE link.

## 8.13.2 Windows Application to Windows Application

In a native Windows 3.0 environment, a Windows application (client) will broadcast a *DDE Initiate* message. Windows serially posts a message to every Windows application currently running and then awaits a reply. As described above, the *Initiate Conversation* message contains the DDE topic to which any Windows application can respond. The client application continues execution when all other applications have serviced the message. At this time, the client application communicates directly with the server applications, as opposed to the initial broadcast message.

OS/2 Version 2.0 provides two applications to support communications between VDMs, without altering the Windows code:

- A resident Windows application referred to as the DDE **ServerAgent (SA)**

- A DOS protected mode application referred to as the **DDEServer** (VDMSRVR.EXE).

### 8.13.2.1 ServerAgent

The Windows VDM's resident ServerAgent consists of two parts:

- A "ServerAgent" which sends and receives messages outside of the VDM

- One or more "agents" (each agent is a child window of the ServerAgent), which act as "clones" of applications running in other VDMs.

If either the DDEServer or the VDM's ServerAgent is not executing, DDE is not available outside of the VDM. The ServerAgent is automatically started when the Windows VDM is started. When started, DDE is in *public* mode. To keep it *local*, simply kill (close) the DDE Interchange Agent. To make it *public* again, simply start the Interchange Agent once more.

Should the user choose to kill the DDE Interchange Agent, an information message will be displayed indicating that DDE activity will be visible only to the Windows applications executing in the current VDM, discontinuing DDE communication with Presentation Manager applications and other Windows applications.

The ServerAgent is responsible for all routing of DDE messages, including broadcast messages beyond the confines of the VDM to the DDEServer. The ServerAgent communicates with the DDEServer (VDMSRVR.EXE) via named pipes.

Agent applications communicate with Windows applications in their VDM and the ServerAgent executing in their VDM. Only the ServerAgent uses named pipes. Agents send requests to the ServerAgent to be forwarded outside of the VDM.

## 8.13.2.2 DDEServer (VDMSRVR.EXE)

The DDEServer is responsible for routing requests from ServerAgents to the appropriate VDMs. The DDE process is schematically represented below:
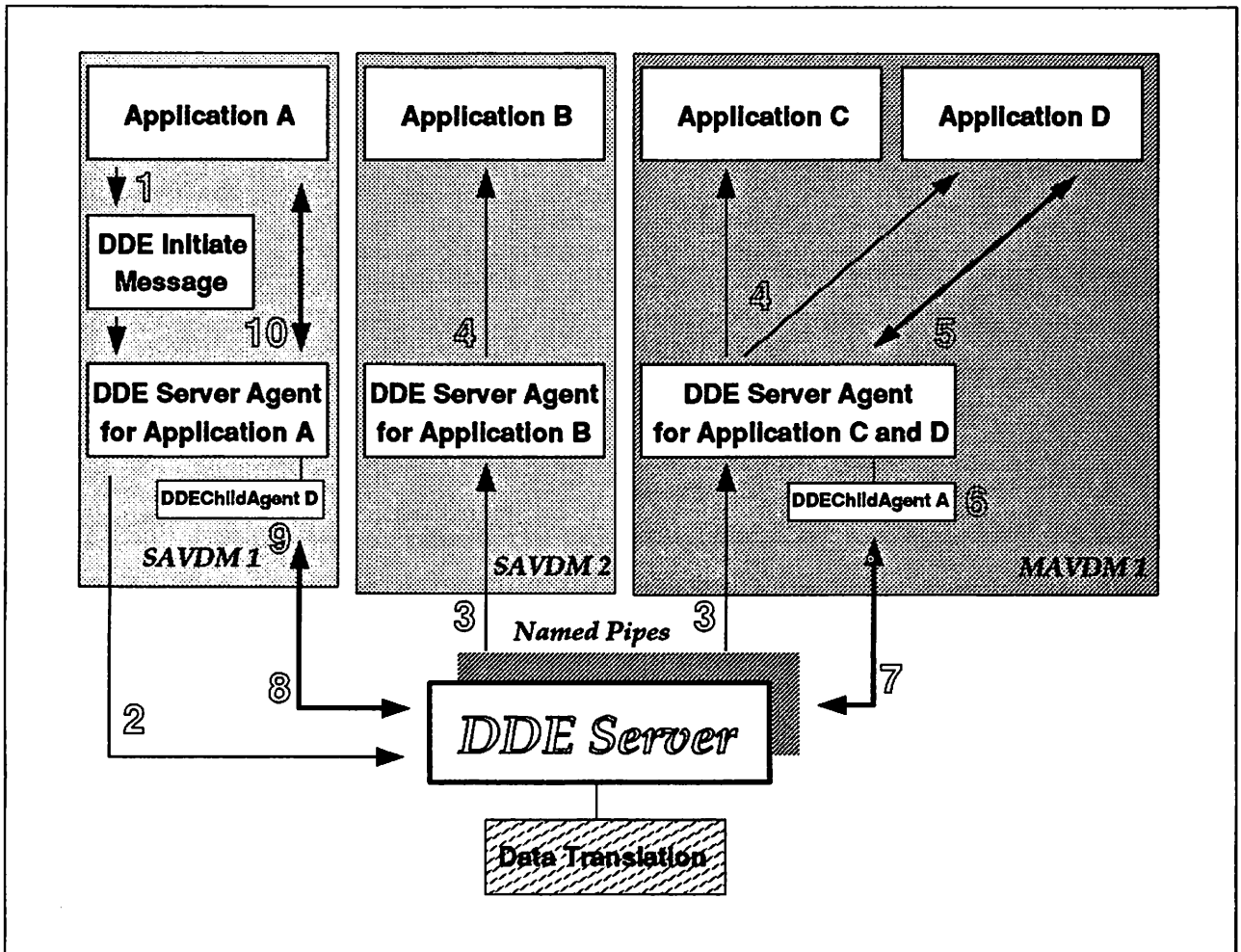


*Figure 45. DDE Process between Windows Environments*

The sequence of events for DDE communication between applications running in different WIN-OS/2 VDMs is as follows:

1. A DDE Initiate message is broadcast from Windows application A.

2. The message is forwarded by the ServerAgent to the DDEServer.

3. The DDEServer forwards this message to all other DDEAgents.

4. Each DDEAgent *broadcasts* this initiate message to all Windows applications in their VDM.

5. Windows application D responds affirmatively to this DDE initiate message.

6. The DDEAgent creates a child task ServerAgent A to act as an agent to Windows application A.

7. The DDEAgent forwards an *acknowledgement* to the DDEServer.

8. The DDEServer forwards this *acknowledgement* to the DDEAgent for the Windows application A.

9. This DDEAgent creates a child task ServerAgent D to act as an agent to Windows application D.

10. The DDEAgent forwards the response from Windows application D to the Windows application A.

From here on, the two Windows applications communicate through this established path.

Appl. A <-> DDEChildAgent D <-> DDEAgent A <-> DDEServer <-> DDEAgent D <-> DDEChildAgent A <-> Appl. D.

This mechanism isolates all named pipe transactions (Steps 2, 3, 7 and 8) to the DDEAgents and the DDEServer. It also gives the Windows application A the illusion of a point-to-point connection to Windows application D (because it will actually communicate with Windows application D's child agent in the same VDM).

The interprocess communication protocol used between the Windows applications and the DDEAgents is the original and unmodified DDE protocol.

If two Windows applications require significant amounts of DDE, these applications should be executed from within the same MAVDM. In such instances, the ServerAgent and DDEServer applications would not be required, thereby improving performance and usability. Once this is done, the user need only kill (close) the participating DDE Interchange Agent to ensure that all DDE messaging is kept *local*.

## 8.13.3  Windows Application to Presentation Manager Application

DDE support between Windows applications and Presentation Manager applications requires that the DDEServer be linked with the Presentation Manager DDE APIs. Both DDE messages and data formats are translated during the data exchange between Presentation Manager and any given VDM running a Windows application. This process consists of a protected mode DDEServer, a Windows DDE ServerAgent, as described above, and a Presentation Manager DDE ServerAgent. The Presentation Manager DDE ServerAgent is a mirror to the Windows DDE ServerAgent. The ServerAgent is responsible for routing all DDE messages beyond the confines of Presentation Manager to the DDEServer. The ServerAgent communicates with the DDEServer via named pipes.

The DDE process between Presentation Manager applications and Windows applications may be represented as follows:
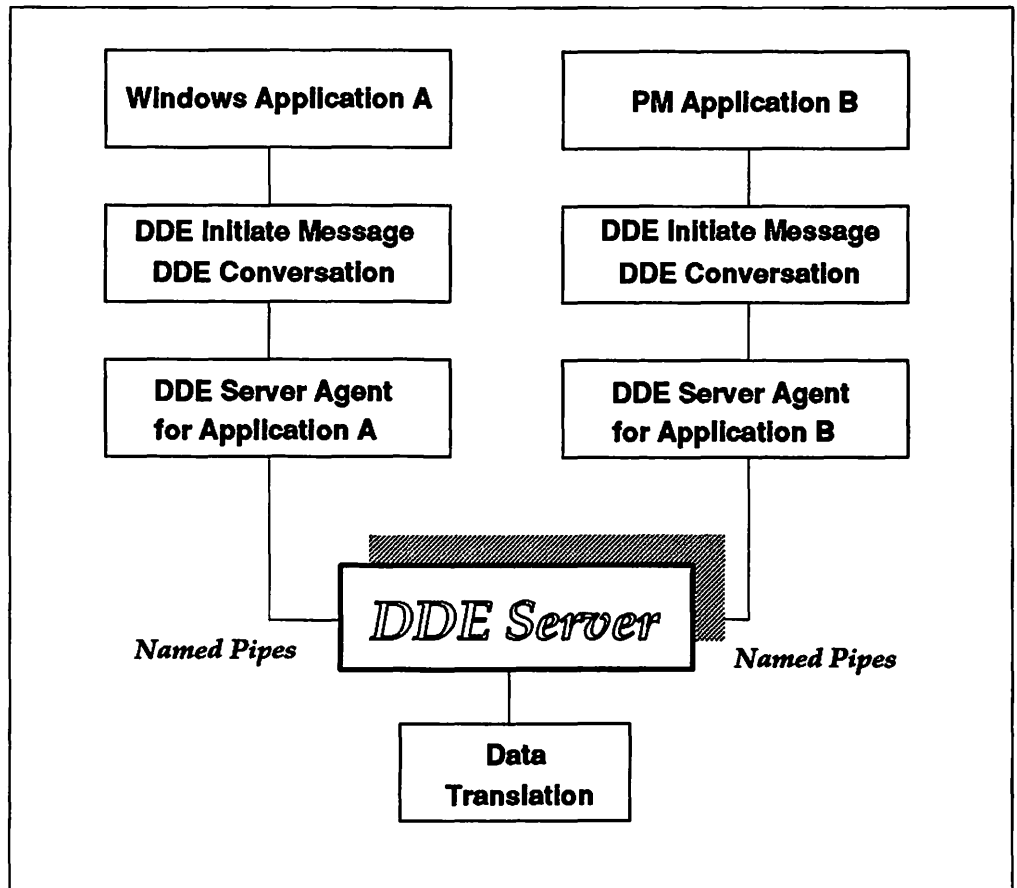
*Figure 46. DDE Process between Presentation Manager and Windows*

The following data formats are translated between the Presentation Manager environment and the Windows environment:

**Bitmaps:** Windows DIB format to/from OS/2 BITMAPINFO2 and Presentation Manager BITMAPINFO to/from Windows DIB format.

**Windows Device Dependent Bitmaps:**
Pre-Windows 3.0 format to Windows DIB format to/from Presentation Manager BITMAPINFO.

**Windows Metafiles:** Metafiles are converted to Window DIB format prior to being translated as above.

**PM Metafiles:** PM metafiles are first converted to Window DIB format prior to being translated as above, and are then forwarded to the global clipboard.

**Text:** Codepage translation is provided in both directions, if required.

Any data format which is not supported by the *global* DDEServer translation routines, can still be used on a *local* base, that means within the same VDM.

The Presentation Manager DDE ServerAgent will reside as a utility in a *Productivity* folder on the Workplace Shell. Where there is a demand to provide DDE support between Presentation Manager applications and Windows applications, the Presentation Manager DDE ServerAgent should be placed in the *Workplace Startup* folder. The DDE ServerAgent runs *only* as a minimized icon. To shut

down global DDE, the Presentation Manager DDE ServerAgent must be terminated through the Window List.

If a *seamless* Windows session is started, the DDE ServerAgent will automatically be started, so that this particular Windows application can automatically use DDE. Otherwise, it would appear to be isolated and this would confuse the novice user.

Existing DDE support between Presentation Manager applications remains essentially unchanged. Where DDE is only used between Presentation Manager applications, the DDEServer should be deactivated to improve performance.

## 8.14 Object Linking and Embedding

Object Linking and Embedding (OLE) focuses on document formats rather than on an application's ability to exchange data, as implemented using the DDE approach. OLE is concerned with sharing functionality as opposed to sharing data. The better the application of OLE, the less the concern with programs as opposed to their data.

All OLE transactions involve a client application and a server application. The server creates the embedded or linked document and is activated when any activity beyond display is required; the client packages and renders the object within its own document.

OLE objects are packaged within client documents either statically (embedded) or dynamically (linked). The entire contents of an embedded object, including references to the server application, are included in the client document.

OLE defines a format for compound documents which contain multiple forms of data. The data formats are understood and managed by multiple applications. The application uses various combinations of data to construct a **compound document**.

## 8.14.1 OLE Concepts

The concepts used in OLE are best described by contrasting them with the approach adopted by clipboard and DDE:

- When using the clipboard, an application obtains data from another application in a standard format, usually ASCII, a bitmap or a metafile. This data exists *only* as data; there is no link with the application that originally placed the data in the clipboard.

- When using DDE, an application also obtains data from another application in a standard format, such as ASCII, bitmap or metafile. However, the client can establish and maintain a link with the application which delivered the data. Should the data change in the server application, the client application's data is also updated.

OLE also enables an application to obtain data from another application; in this instance the data can be in two formats:

- One format is understood only by the application sending the data

- The *display* format (ASCII, bitmap or metafile) is understood by the receiving application, and is used to display the data on the screen.

When an object is linked, only the references to the actual data, including the name of the server application, are embedded within the client document. In both cases, the references allow the OLE libraries to execute the server and properly instruct it.

When initially embedding or linking objects, client and server applications typically exchange data using the Windows clipboard. When the server puts data on the clipboard, it uses various combinations of four types of data:

1. Native data is specific to the server and likely to be alien to the client.

2. Presentation data uses one of several display formats commonly used by Windows programs to render data.

3. OwnerLink data is the name and address identifying the object or application that owns the data.

4. ObjectLink data uses the same format as OwnerLink data but describes the application and object from which the data originated.

The order in which this data is put on the clipboard, or otherwise presented to the client, determines what type of object is intended.

For example, if OwnerLink data is presented first, then a linked object is intended. If Native data is first and OwnerLink is second, then the object can be embedded (though not necessarily rendered properly).

In fact, a presentation format is optional for an embedded object, partly because some objects are meant to be invisible. If no presentation data is available and understood by the client and no object handler is provided by the server, the object will not be properly rendered by the client.

The significance of this approach may be appreciated by way of an example. Voice annotation may be attached to a word processing application; the word processing application need not have any facility to support or manage voice. The word processor will store the data in two formats; the digitized sound and a display format (an icon). When the icon is selected in the document, a voice application is invoked and the word processing application passes the digitized sound to the voice application, which then plays the sound.

## 8.14.2 Linking versus Embedding

When an object is embedded, information from one document is inserted into a document in a different application. Embedding is similar to copying, but with one significant difference; to make changes to an embedded object, the user simply selects the object from within the destination document. The application in which the object was created is invoked, and the user may make the required changes. There is no need to switch among applications to view or change different kinds of information; it is all in one document.

When an embedded object is modified, the source document is not affected. For example, if a drawing is embedded into a report, changes made to the drawing within the report do not affect the original drawing which resides in its own file.

When an object is linked, many documents can share a single item of information. The object itself is not placed into the destination document; rather, a representation, or placeholder, for the object being linked is placed into the document. The object still exists in the source document, and the destination document merely contains a link to the object's location in the source document.

When changes are made to a linked object, the source document and *any other* documents linked to the object will reflect the changes. For example, if a drawing is linked to a report, any changes you make to the drawing appear in the source document and in any other reports linked with the drawing.

Access may be gained to the object from any document that contains a link to it, and changes may be made to the object from within any such document. The updated version automatically appears in all the documents that have a link to this object. Linking makes it easy to track information that appears in more than one place and which must be identical.

Only objects from saved documents may be linked. For example, if a drawing is created using Paintbrush, the drawing must be saved as a document before it may be linked from another document.

Not all applications can provide and accept objects. Some may only be the source of objects (server applications). Others (client applications) may only accept objects.

## 8.15 Summary

OS/2 Version 2.0 provides support for the execution of Windows applications within the MVDM architecture. This support allows the concurrent execution of multiple Windows applications, using both real mode and standard mode, with DPMI and Windows services provided as required by a Windows kernel.

Windows applications running under OS/2 Version 2.0 are provided with pre-emptive multitasking by the operating system. Full memory protection is also provided for the Windows applications; an errant application may not affect other applications executing in the system. A bug in an application will cause the termination of that application *only*.

Windows applications may be run under OS/2 Version 2.0 in three ways:

- Each application may run in its own VDM. This method of execution provides full protection for the application from other processes running in the system, and protects these other processes from errors in the Windows application.

- Applications may share a VDM, and may be started and controlled within this VDM using the Windows Program Manager. This method of execution provides protection for the Windows applications within the VDM from other processes, and protection for other processes from errors in the Windows VDM's applications. However, the applications within the VDM may affect one another since they share a common address space, just as if they were running natively under DOS/Windows.

- Windows applications may run "seamlessly" on the Workplace Shell desktop, along with windowed VDMs and Presentation Manager applications. This method of execution is similar to the case of a single application in a VDM, except that the Windows application shares the Workplace Shell desktop rather than running in its own full-screen session.

Any combination of these three methods may be used concurrently.

Windows applications may be defined on and started from the Workplace Shell desktop. Where a single application is defined for a VDM, or where the application will run seamlessly, the icon used to represent the application on the

desktop is the icon embedded within the Windows program which runs the application.

During installation of OS/2 Version 2.0 over an existing DOS/Windows 3.0 system, existing applications defined to the Windows Program Manager will be detected and migrated where possible to the OS/2 Version 2.0 Workplace Shell. The installation procedure uses application definition information contained in the Certified Application Database, which is shipped as part of the OS/2 Version 2.0 product.

OS/2 Version 2.0 allows communication between Windows applications running in the same or different VDMs, and between Windows applications and Presentation Manager applications. This communication is provided through clipboard, DDE and OLE support. Communication between Windows applications using shared memory is also supported, but only where Windows applications are executing in the same VDM.

In summary, OS/2 Version 2.0 provides an integration platform which allows Windows applications to coexist with one another and with DOS and OS/2 applications in a multitasking, fully protected environment, and which allows these applications to communicate with one another.

# Chapter 9. DOS Protected Mode Interface

Perhaps the most significant limitation of real mode operation, as used by DOS and similar operating systems, is the 1MB addressing limitation. This limitation can be overcome by executing applications in protected mode, but since the DOS operating system and most TSR applications must run in real mode, problems arise when applications attempt to access interrupts, TSRs or operating system facilities.

The **DOS Protected Mode Interface** (DPMI) is a protected mode programming interface for DOS applications, allowing such applications to run on an 80286 or 80386 processor in protected mode, while utilizing the real mode services of the operating system and device drivers. When an application wishes to access a DOS service, it makes a request to DPMI, which handles the appropriate address translations, switches the processor to real mode and makes the service request to DOS. The result of the request is then translated to the correct format for the protected mode application, the processor is switched back to protected mode, and control is returned to the application.

DPMI has been implemented in the Multiple Virtual DOS Machines component of OS/2 Version 2.0, and provides functions such as memory allocation and interrupt management for applications which use DPMI services. This support is provided through a component, implemented as a virtual device driver, known as the **DPMI API Layer**, in conjunction with the MVDM kernel.

This chapter provides a brief overview of DPMI, and describes its implementation under OS/2 Version 2.0.

## 9.1 DPMI Introduction

Most processor instructions that are available in real mode may also be executed from a protected mode task. Hence an application may overcome the limitations of real mode simply by executing in protected mode. However, direct access to physical hardware and interrupts is typically not permitted from a protected mode task running at Ring 3 privilege, and therefore DOS itself and many TSR programs must run in real mode. Protected mode specifications are such that communication between protected mode and real mode programs is difficult, making it difficult for an application to request services from DOS or a device driver.

For example, a TSR, with which an application communicates through a software interrupt or a shared buffer, cannot run in protected mode. The real mode address of the TSR, if used by the protected mode application, will not point to the same location in memory as would the same address if used in real mode, since the segment portion of the address is interpreted differently in the two modes. Address conversion is therefore required when passing between the two modes.

DPMI provides an interface between protected mode and real mode programs. DPMI consists of a set of protected mode functions which allow a DOS application to enter protected mode, allocate real mode memory, simulate real mode interrupts and function calls, intercept real mode interrupt vectors, etc. By using

these calls, an application running in protected mode can communicate with DOS or a TSR running in real mode.

DPMI facilitates the following:

- Allows DOS applications to run in protected mode

- Provides DOS applications with access to a large memory address space

- Provides DOS applications with mode switching and calls between real mode and protected mode

- Provides DOS applications running in protected mode with access to hardware facilities such as debug registers, in a way that maintains system integrity.

The term *real mode* is used to refer to code that runs in the low 1MB address space and uses segment:offset addressing. Under many implementations of DPMI, so-called real mode software is actually executed in virtual 8086 mode. Since virtual 8086 mode closely approximates real mode, V86 mode and real mode are interchangeable in the DPMI context.

One of the major benefits offered by DPMI is that of allowing DOS extenders to work effectively in a multitasking, protected mode environment. DOS extenders allow DOS applications to access extended memory while running in protected mode. These extenders switch between modes as required to:

- Execute application code in protected mode, in order to realize the enhanced addressing capabilities and protection facilities of protected mode .

- Access DOS services and TSRs in real mode, to perform functions which cannot be performed in protected mode.

The Microsoft Windows/286 DOS extender (running under DOS on an 80286 processor) was able to switch modes of its own accord. However, when running in virtual 8086 mode on an 80386 processor, a task cannot switch to protected mode; the required instruction is not legal for a V86 mode task. The architecture of DPMI, however, allows DOS extenders to request services using INT 31h DPMI calls; DPMI itself handles the mode switching and address conversion necessary to invoke the real mode services.

## 9.2 Virtual Control Program Interface

The forerunner to DPMI was the **Virtual Control Program Interface (VCPI)**, developed by Phar Lap Software** and Quarterdeck Office Systems**. VCPI allowed 80386-based protected mode DOS extenders to coexist with 80386-specific memory managers and expanded memory (EMS) emulators, such as QEMM-386 by Quarterdeck. Most current 80386-specific software products support, or are capable of using, the VCPI interface.

In VCPI, the DOS extender acts as a *client* and the EMS emulator acts as a *server*. The client invokes the VCPI server to:

- Switch between real mode and protected mode

- Allocate memory

- Program the interrupt controller(s)

- Inspect or set the 80386 debug registers.

If a DOS extender is loaded and a VCPI server is not present, the DOS extender may assume total control of the system and perform hardware-dependent manipulations directly. This can lead to system and data integrity problems.

---

**Note**

Do not confuse VCPI (Virtual Control Program Interface) with VPIC (Virtual Programmable Interrupt Controller).

---

While VCPI performs well for that which it was intended, it does not provide a platform for multitasking DOS extender applications. The deficiency lies in VCPI allowing client programs to run in Ring 0, the highest privilege level possible under the 80386 processor.

What was required was an interface capable of managing and controlling device initialization and providing centralized virtual memory management. Most important the interface had to shield one client from another.

## 9.3 The DPMI Specification

DPMI was devised by a committee of major software vendors. The first (and current) DPMI version is Version 1.0.

DPMI was defined to allow DOS programs to access extended memory while maintaining system protection. DPMI defines a specific subset of DOS and BIOS calls that can be made by DOS programs running in protected mode. These services are accessed via software interrupt 31h, using a defined series of functions which protected mode programs may use to allocate memory, modify descriptors and call real mode software.

Like VCPI, a DPMI *host* (or server) program provides mode switching and extended memory management services to *client* programs. Unlike a VCPI server, however, a DPMI host runs at a higher privilege level than its clients. A DPMI host supports demand-paged virtual memory and maintains complete control over the address space and hardware access of its clients.

Some DPMI implementations execute multiple protected mode programs in independent virtual machines. In such implementations, DPMI applications behave exactly like any other standard DOS programs. For example, they can run in the background or in a window, provided the environment supports these features. Programs that run in protected mode gain all the benefits of virtual memory and can utilize a 32-bit flat memory model if desired. OS/2 Version 2.0 provides a DPMI implementation of this nature.

DPMI services accessed via INT 31h are *only* available to protected mode programs. Programs running in real mode cannot use these services. The only exception to this rule is the service which allows an application to enter protected mode, which must be called by real mode programs before calling any other DPMI service.

Note that the majority of software vendors who released applications using the VCPI specification have since released versions which use DPMI instead, or have produced upgrades to their software to take advantage of DPMI.

## 9.3.1 DPMI Hosts and Clients

DPMI services are provided by a DPMI *host* program. Programs which use DPMI services are known as DPMI *clients*. Generally, DPMI clients fall into two categories:

1. Extended applications

2. Applications that use DPMI directly.

Most DPMI applications are likely to be extended applications. These applications are bound with a DOS extender, which is the actual DPMI client since it requests DPMI services on the application's behalf. The application calls DOS extender services, which are then translated by the DOS extender into DPMI service calls. The advantage of an extended application over one that calls DPMI services directly is that generally, an extender will support functions other than DPMI services. In fact, it is recommended that extenders look for extension services in the following order:

1. DPMI

2. VCPI/EMS

3. XMS

4. Top-down (INT 15h).

Extended memory may be allocated "top-down" by hooking the BIOS extended memory size system call (INT 15h, function 88h) and reporting less memory available than is actually present on the machine. This method may be used by DOS extenders to allocate a contiguous block of memory starting at the top of extended memory and growing downward. Since other applications querying the amount of memory available in the system will not be able to "see" this upper portion of memory, the memory is available solely to the DOS extender.

A DPMI client can provide a single set of functions to an application, and then translate these functions to one or more underlying services (for example, DPMI, EMS, and/or XMS) provided by the client. Where the corresponding host's services are lacking in a particular function, the extender must itself provide that function for the application. This is illustrated in Figure 47 on page 185.
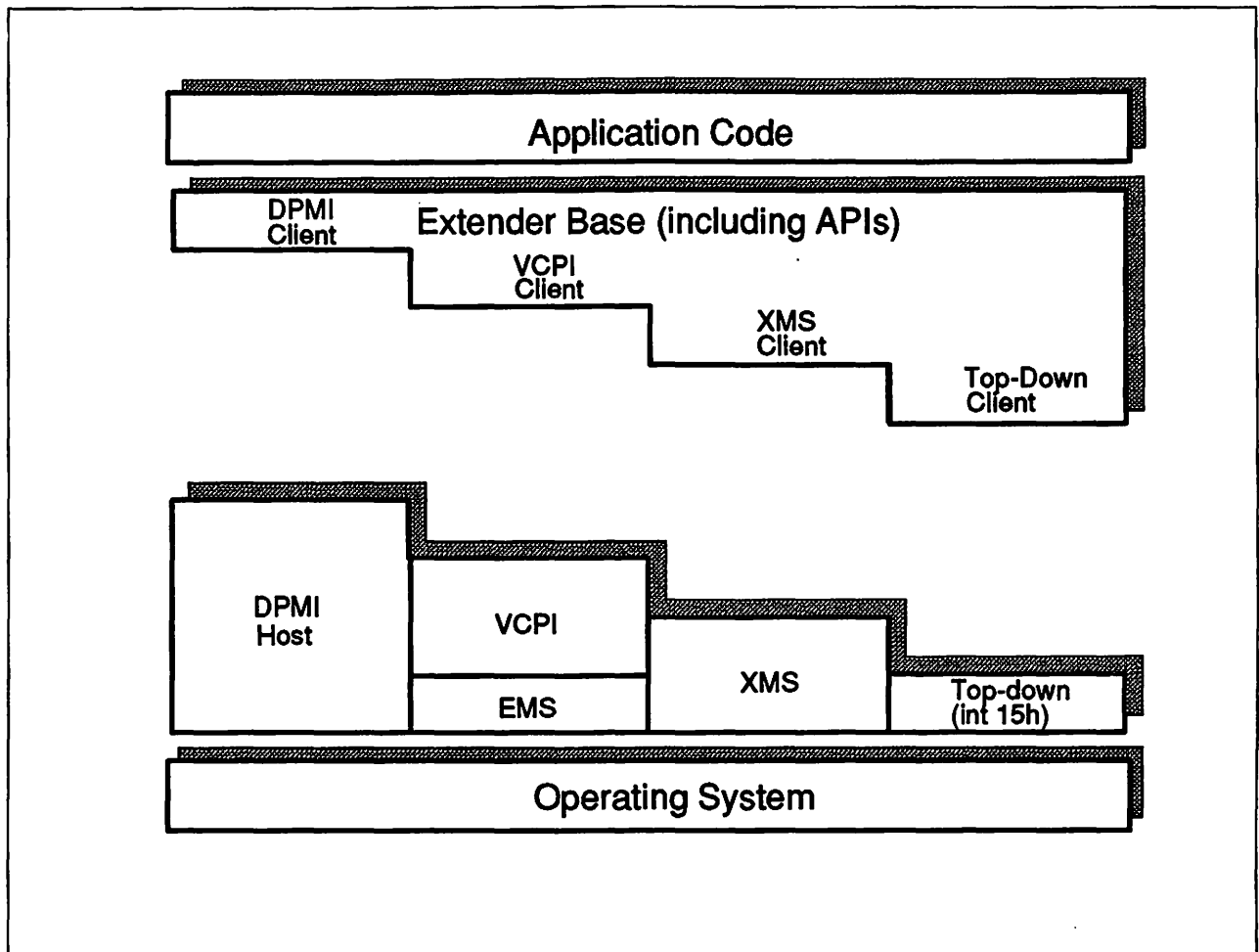
*Figure 47. Client/Server Structure for Operating System Extenders*

As shown in Figure 47, application code directly accesses a set of base extender functions. The extender then has separate modules for each type of extension service, and itself contains code to provide functions in which the underlying service layers are lacking.

Readers should refer to the *DPMI 1.0 Specification* published by the DPMI committee for information concerning the external interfaces available to DPMI applications. Copies of the specification may be obtained by contacting Intel Literature Sales, P.O. Box 58130, Santa Clara, CA 95052.

## 9.3.2 DPMI Services

The following is a brief outline of the DPMI services. For details regarding invocation of DPMI services from an application via INT 31h, please refer to the *DPMI 0.9 Specification*.

DPMI provides six main classes of services:

- Local Descriptor Table management
- Memory management
- Page management
- Interrupt management
- Translation

- Debug watchpoint.

Each of these services is briefly described in the following sections.

Note that DPMI services are normally never called by an application program itself, but are intended to be used by DOS extenders which request DPMI services on an application's behalf.

### 9.3.2.1 LDT Descriptor Management Services

The LDT descriptor management service provides interfaces for allocating, freeing, and creating protected mode descriptors in the current task's Local Descriptor Table (LDT). Access to the Global Descriptor Table is not provided, so that the DPMI server can protect itself from protected mode applications and isolate these applications from one another.

### 9.3.2.2 DOS Memory Management Services

The DOS memory management services provided an interface from protected mode applications to real mode INT 21h functions which are used to allocate, free and resize memory blocks. These services allow a protected mode applications to use memory below 640KB, to exchange data with DOS, ROM BIOS device drivers, TSRs and other real mode programs which are incapable of accessing data in extended memory.

### 9.3.2.3 Extended Memory Management Services

The extended memory management services are used to allocate, free and resize memory blocks above the 1MB boundary. If the DPMI server is an 80386 or 80486 control program and paging is enabled, the extended memory blocks are always allocated in units of 4KB.

### 9.3.2.4 Page Management Services

Under DPMI implementations which support virtual memory, applications may discard memory blocks or may not access them for long periods of time, in which case the memory block's contents may be swapped out to disk. In certain circumstances, such as interrupt handling code, this swapping must be disabled and the appropriate pages locked in physical memory. The page management services allow pages to be individually locked or unlocked.

### 9.3.2.5 Interrupt Management Services

These services allow protected mode applications to intercept real and protected mode interrupts and hook processor exceptions. Certain services allow a protected mode program to intercept hardware or software interrupts which occur in real mode or protected mode, or to install handlers for processor exceptions. Other interrupt services permit a process to enable or disable its own servicing or hardware interrupts without affecting the interrupt status of the entire system. DPMI accomplishes this by maintaining a virtual interrupt flag on a per-process basis.

### 9.3.2.6 Translation Services

The translation services permit control to be passed between operating modes. A protected mode program may transfer control to a real mode routine using a simulated far call or a simulated interrupt. Translation services also allow a protected mode program to declare a real mode callback, or entry point which can be invoked by the a real mode program.

### 9.3.2.7 Debug Watchpoint Services

The 80386 processor supports special registers that are used for debugging. Since the instructions to modify these registers can only be executed by code running at privilege level zero, protected mode debugging tools running in DPMI environments cannot modify the registers directly. These services provide mechanisms for setting and clearing debug watchpoints and detecting when a watchpoint has caused a fault.

## 9.4 DOS Extenders

Programs which use DPMI services are normally bound to DOS extenders, in order to run under any DOS environment. Most DOS extenders provide an interface to applications using an INT 21h multiplex. For functions which utilize DPMI services, the DOS extender then makes the appropriate INT 31h request.

Extenders that support DPMI will need to initialize differently when they are run under DPMI environments. They will need to enter protected mode using the DPMI real to protected mode entry point, install their own API handlers, and then load the DOS extended application program.

DOS extenders should check for the presence of DPMI before attempting to allocate memory or enter protected mode using any other API. When DPMI services are detected, extenders that provide interfaces that extend or are different from the basic DPMI interface will switch into protected mode and initialize any internal data structures. DPMI-compatible extenders that provide no API extensions should simply execute the protected mode application in real mode.

## 9.4.1 Loading DPMI Clients and Extended Applications

All DPMI applications begin execution in real mode. An application must run first as a real mode DOS program, but can then switch to protected mode by making a call to DPMI (or to a DOS extender). Once in protected mode, all INT 31h calls supported by DPMI may be issued by the application or its associated DOS extender functions.

A DOS extender and its application under DPMI are loaded and initialized as described below. The DOS extender:

1. Loads in real mode (or V86 mode on an 80386/80486 machine).

2. Checks for presence of a DPMI server.

3. Switches the CPU from real mode to protected mode, and loads registers with the appropriate selectors.

   If no DPMI server is present, the DOS extender checks for the existence of a VCPI server or XMS device driver before assuming total control of the CPU's execution mode, privileged control registers, and memory management hardware.

4. Uses DPMI services to build the protected mode environment to be used by the application.

5. Allocates extended memory segments to hold the application's code, data, and stacks.

6. Allocates selectors to be used by the application to execute in and/or address the memory segments.

7. Reads the application's code and data from disk into the segments.

   The DOS extender can mark pageable memory it uses below 640KB so as to reduce the demand for physical memory.

8. Installs its own handlers for any software interrupts (such as DOS INT 21h) that the application will execute.

Control is then passed to the application.

## 9.4.2 Processing in DOS Extenders

The way in which a DOS extender processes interrupts varies. Some INT 21h requests are passed directly to DOS. The DOS extender simply switches to real mode, calls DOS, and then switches back to protected mode when DOS returns after completing the function. File input and output, however, may demand that the DOS extender translate addresses, while other INT 21h functions such as DOS memory management must be replaced entirely by the DOS extender.

Unless the A20 address line has been explicitly enabled through the XMS interface, it cannot be assumed that memory from 1MB to 1MB+64KB-16 (the High Memory Area) is addressable once a program is running protected mode. If HMA is to be accessed, the A20 address line must be enabled through XMS *before* entering protected mode. XMS calls are not supported in protected mode.

This restriction is only important for software that wishes to access the HMA. Under all implementations of DPMI, the physical A20 address line will always be enabled while executing protected mode code. However, some 80386 specific DPMI implementations simulate 1MB address wrap for compatibility reasons. Under these DPMI implementations, the HMA will not be accessible unless the A20 address line is enabled through the XMS interface. This is the case under OS/2 Version 2.0.

## 9.4.3 Session Termination

When the DOS extender or its application issue the DOS terminate interrupt, DPMI traps the interrupt and releases all of the application's protected mode resources. The DPMI server passes the interrupt to real mode so as to permit DOS to clean up the program's real mode resources, including file and device handles and any memory blocks below 640KB.

## 9.5 DPMI Implementation in OS/2 Version 2.0

OS/2 Version 2.0 provides DPMI services to applications running in VDMs. The DPMI Version 0.9 specification is fully supported, and the architecture of the DPMI implementation is such that both the API functions and the underlying services are independently expandable to cope with future versions of DPMI.

DPMI support under OS/2 Version 2.0 is divided into three components:

- The DPMI API is implemented using the **DPMI API Layer**, a virtual device driver which services INT 31h requests from applications.

- The operating system kernel provides support for the DPMI VDD and the Virtual Programmable Interrupt Controller (VPIC).

- Protected mode hardware interrupts are routed via the VPIC.

DPMI is service request driven. An application first makes an INT 31H service request, which is handled by the DPMI VDD, calling the kernel for basic services such as allocating memory.

## 9.5.1 DPMI Services

DPMI services are requested using INT 31h requests, which are trapped by the DPMI virtual device driver, and either serviced by the VDD itself or routed to the operating system kernel.

The DPMI API Layer performs input parameter checking on all service requests, to validate requests and to enforce restrictions mandated by the DPMI specification.

### 9.5.1.1  LDT Descriptor Management

The 8086 Emulation component of MVDM arranges for allocation of a task's LDT upon initialization of the VDM. Under DPMI 0.9, all tasks in a VDM share the same LDT. Applications may modify descriptors only through DPMI service calls.

Three types of descriptors must be kept track of:

1. Per-task DPMI descriptors that the client may modify
2. V86 segment to selector mappings with descriptors that cannot be modified
3. Per-task DOS descriptors that the client cannot modify.

Memory used by a DPMI application is allocated by the OS/2 Version 2.0 operating system to the parent process of the VDM within which that application executes. Full memory protection is therefore provided for applications using DPMI services.

### 9.5.1.2  DOS Memory Management

DOS memory management services are implemented under OS/2 Version 2.0 in various ways, according to the nature of the service.

- *Allocate DOS memory and selector set*

  This service allocates DOS memory along with a set of descriptors to cover the allocation. For 32-bit clients, a single descriptor is set to cover the entire allocated region. For 16-bit clients, this descriptor is followed by descriptors to cover the rest of the region in 64KB segments. This allows 16-bit applications to refer either through a single large segment or through tiled selectors.

  A V86 mode DOS call is used to allocate memory from the DOS arena. Therefore, after initial setup, the DPMI API Layer switches back into V86 mode to issue the DOS call, and then traps the return from DOS in order to finish the service.

- *Free DOS memory and selector set*

  The allocated list is searched to make sure the region being freed is allocated. The allocation record is moved to the pending list with the request marked as free. A switch is then made to V86 mode and the INT 21h is simulated as above with the return trapped. When the return is trapped, return values are set up. If the call succeeded, the selectors that were allocated are freed. If a selector other than the allocated selector was passed in the free call, that selector set is freed as well.

- *Resize DOS memory and selector set*

  Resizing is done in the same way as the original allocation. The allocation record is moved to the pending list. The desired size is listed in the allocation record and new selectors are allocated if the size increases and new selectors are needed. Descriptors are allocated before reflection so the call can fail before allocating DOS memory in V86 mode if they are not available. The DOS call is then done as above.

  If the call failed, the new selectors are freed when the hook regains control. If it succeeded, new descriptors are set up if they were needed or descriptors are freed if the resize made some unnecessary. The return values for the client are set up and the allocation record is returned to the allocated list with its new size noted.

### 9.5.1.3 Extended Memory Management

Extended memory management services are also implemented in a number of ways, depending on the service.

- *Get memory information*

  This service uses memory management calls to load an application buffer with a variety of information about the memory. A VDH service is used to copy the data to user space, with appropriate exception handling.

- *Allocate*

  Memory is allocated using a memory management service. Record of the allocation noting the start address, allocated size, and sparse linear address size are kept in a hash table. The allocation records are kept in the DPMI API Layer per DPMI task area so that they can be cleaned up when the task terminates.

- *Free*

  This function is implemented quite simply; the DPMI API Layer per-task data area is checked for the allocation records, and the corresponding memory is freed via a call to the operating system kernel.

- *Resize*

  DPMI changes a memory object's size in one of two ways:

  — If the size of the object is to be decreased, pages at the end of the object are decommitted.
  — If the size of the object is to be increased, a new and larger object is allocated and a kernel worker is used to move the pages from the original object to the new one.

### 9.5.1.4 Page Locking

Page locking services are necessary on systems which deliver interrupts at interrupt time or which use DOS for paging. On a system that simulates interrupts and has its own file system (such as OS/2 Version 2.0) the calls are no-ops. They will simply return a success indication to the client.

### 9.5.1.5 Interrupt Hooking

8086 Emulation maintains a table of the current handler for each protect mode hook and exception. These services are implemented by calling 8086 Emulation to get the current value from this table or to set a new value in the table.

8086 Emulation offers a service to change the client's virtual interrupt state (the IF flag).

### 9.5.1.6 Translation (Protect/V86 Control Transfer)

These services provide cross-mode calls, state saving, and raw mode switching.

- *Real mode callback* (call protected mode from real mode)

  This service allocates a real mode callback breakpoint. When this break-point is called, the DPMI API Layer handler arranges a protect mode call.

- *Free real mode callback*

  If a real mode callback is still waiting to be completed, the callback record is marked to indicate it is no longer active. Freeing the callback record and the breakpoint are done when no outstanding calls are in progress.

- *State save/restore for each mode*

  This service returns a set of addresses, one for V86 mode and one for protect mode, which, if called by the client, save or restore the current register state for the other mode. This is necessary for applications which perform raw mode switching, to keep them from overwriting the task state for the alternate mode.

- Raw mode switching

  This service returns a set of addresses, one for V86 mode and one for protect mode, which, if called by the client, switch to the other mode. The breakpoints have the DPMI task identifier in the breakpoint data area. When the breakpoint is reached, if the ID is different from the current one, 8086 Emulation is called to report the switch. A VDH service is then used to do the requested mode switch.

Under OS/2 Version 2.0, an extension to the DPMI specification has been implemented, and is known as **DOS API** services. Protected mode applications issuing DOS or BIOS calls must pass buffers that can be accessed in V86 mode. The DOS API services relieve the application from having to do this work for DOS calls and some BIOS calls. This permits protected mode applications to use protect mode buffers (referenced by protected mode selectors) in DOS service requests. The translation services perform any necessary buffer copying.

Applications detect the presence of a DOS API translator by performing an INT 2Fh multiplex passing the name "MS-DOS" as an argument. The translator responds when it detects this name, indicating that translation will be performed. Applications that do not require translation may simply use the INT 31h simulate interrupt function to avoid translation.

### 9.5.1.7 Debug Registers

The Task Management component of OS/2 Version 2.0 manages watchpoints for OS/2 applications, the kernel debugger and VDMs. Interfaces for allocating, setting and freeing watchpoints and getting the Bx bits for allocated watchpoints are used by the DPMI API Layer to carry out these services. The DPMI API Layer keeps track of allocated watchpoints in the per DPMI task area so that it can clean up at termination and uses the tasking watchpoint services to manipulate watchpoints.

### 9.5.1.8 Other DPMI Services

A number of other services are provided under the DPMI specification. Their implementation under OS/2 Version 2.0 is described below.

- *Physical Address Mapping*

  In OS/2 Version 2.0, there is no way of knowing which addresses are used by device drivers. It is therefore not safe to allow direct access to devices which do not have VDDs. However, direct access from within VDMs is allowed.

  VDH services for reserving linear space, mapping, and page fault handling are all restricted to regions below 1MB+64KB. As such, a VDD with a linear address above 1 MB cannot virtualize hardware. All requests to this service will fail. The DPMI specification allows this so the operating system can protect devices.

- *Get Vendor Specific Entry Point*

  Vendors that add extensions to DPMI typically look for the name of their extension by hooking INT 31h. If the extension is requested by a DPMI client, the vendor-supplied routine issues an IRET instruction without jumping down the INT 31h protect mode chain. If the request is for a DPMI service not supplied by the vendor's routine , the routine continues down the INT 31h chain. Since the DPMI API Layer router is called at the end of the chain, any unrecognized service requests are signalled to the client by setting the carry flag to indicate that the call failed.

## 9.5.2 Kernel Support

As well as providing support for DPMI service requests issued by applications, the operating system must also provide support for the internal control functions of the DPMI host. This support is provided by various components of the operating system kernel.

### 9.5.2.1 8086 Emulation

The 8086 Emulation component of MVDM emulates the 8086 hardware environment, and therefore provides a number of services which are used by DPMI.

- *DPMI task entry, termination, mode tracking, control*

  When the application calls the protect mode entry to switch to protected mode, 8086 Emulation sets up tables for reflection of interrupts and exceptions. If the DPMI API Layer fails to complete the creation call, 8086 Emulation cleans up and returns the error to the application.

- *VDH service support*

  All support for VDH services to the DPMI API Layer is provided through 8086 Emulation.

- *Get/set support for protected mode handler interrupt and exception handlers.*

  Protected mode applications get and set vectors as in DOS. 8086 Emulation maintains tables of protected mode interrupt handlers and exception handlers.

- *Interrupt and exception reflection to protected mode*

  8086 Emulation virtualizes interrupts for VDMs. The reflection of "real mode" interrupts to protected mode for DPMI applications is therefore performed with the aid of 8086 Emulation.

- *Protected mode interrupt flag virtualization*

  8086 Emulation virtualizes the IF flag while in protected mode. In V86 mode, IOPL is usually 3 and applications directly change IF without trapping. IF flag virtualization is not done while in V86 mode because IOPL must be 3 to cut down on overhead. In protected mode, IOPL cannot be 3; otherwise no port protection is possible. Therefore, the IF flag is virtualized. This prevents VDMs from blocking real interrupts when running in protected mode.

  To determine if interrupts are allowed in a VDM that has a DPMI application running, the real IF bit in the CRF is checked. If interrupts are disabled here, then they are disabled. Otherwise, the virtual IF flag indicates whether interrupts are disabled.

- *HW interrupt support for the Virtual Programmable Interrupt Controller*

  8086 Emulation exports a VDH service to accept notification from the VPIC when it starts and stops hardware interrupt reflection. 8086 Emulation also tracks which hardware interrupts are hooked. The VPIC allocates and initializes the buffer at creation time in each VDM.

  Any VDD can use this structure to determine if a particular IRQ is hooked. The timer VDD, for example, can use this to avoid delivering timer ticks when the timer tick interrupts are not hooked in either protected mode or in V86 mode.

  When software interrupts are hooked, 8086 Emulation refers to this structure to determine if the interrupt is a hardware interrupt.

- *Services to read/write user space with exception handling*

- *Kernel and VDH service changes for exception handling when accessing user address space.*

  Services called when the client is in protected mode, and which manipulate the protected mode client address space, must be written to handle protected mode user space access exceptions. Services that cannot be called when the client is in protected mode must specify this in their headers.

  When a service can fault in protected mode, it must return a failure indication to the DPMI client. The client then cleans up and exits protected mode so that the exception can be reflected to the VDM (in V86 mode). This error also indicates whether a protected mode exception handler will be called.

### 9.5.2.2  Debug Watchpoint Management

Coordinates watchpoint use with OS/2 protected mode and kernel debugger.

### 9.5.2.3  Memory Management

Among the kernel services provided by the Virtual Memory Manager are:

- Allocate VDM LDT

- Free VDM LDT

- Allocate contiguous set of LDT descriptors

- Free descriptor

- Query maximum private linear address and ranges of physical memory

- Query maximum linear region and swap space available

- Other memory management services generally used by the kernel, such as services to allocate, free, and set sparse allocations, are also used.

Once descriptors are allocated they are changed directly by the DPMI API layer. Applications set descriptors only through requests to the DPMI API layer, which prevents settings that compromise protection.

## 9.5.3  Ring 0 Exceptions

All VDM linear addresses below 1MB + 64KB can be accessed by Ring 0 code (such as 8086 Emulation or DOS Emulation), without any exceptions being visible to the V86 mode application. This meant that there was no need to recover from faults at ring 0 when VDM applications ran only in V86 mode.

DPMI protected mode applications, however, do have addresses in their address space that can cause visible exceptions at ring 0. Most virtual device drivers are not affected because they never execute while the client is in protected mode. VDDs are affected only if both of the following conditions are true:

- The VDD runs while the client is in protected mode.

- The VDD accesses the client address space above 1MB + 64KB or using client selectors while the client is in protected mode. This can happen indirectly if a VDH service is called which manipulates the client's protected mode stack.

In such cases, the virtual device driver must include handlers for the exceptions.

## 9.5.4  DPMI API Layer Communication with the Kernel

The DPMI API Layer has a small, well defined interface with the kernel. At system initialization time, the DPMI API Layer is registered with the kernel through a VDH call and reports which version of DPMI it supports. If the VDD supports only DPMI Version 0.9, the kernel (which supports DPMI Version 1.0) adjusts the way in which it handles certain DPMI tasks.

Kernel services that may be useful to VDDs other than the DPMI API layer are exported as VDH services. Services that should have use restricted only to the DPMI API layer are made available through structures exchanged when the DPMI API Layer virtual device driver is registered.

## 9.5.5 Installation of DPMI

The OS/2 Version 2.0 installation procedure copies the DPMI API Layer virtual device driver DPM.SYS into the \OS2\MDOS directory on the user's system. If users decide to use selective install, they can choose any combination of EMM, XMS, or DPMI. When they select any of these memory options, the appropriate DEVICE= statement is added to CONFIG.SYS. The default memory statement in CONFIG.SYS is:

```
DEVICE=C:\OS2\MDOS\VEMM.SYS
```

If the user does not select DPMI support at installation time and wishes to add it at a later time, CONFIG.SYS must be modified by adding the statement:

```
DEVICE=C:\OS2\MDOS\VDPMI.SYS
```

## 9.5.6 DPMI and Microsoft Windows

DPMI 0.9 support is necessary for Windows 3.0 to run applications in protected mode (that is, in Windows standard mode). With DPMI implemented in Windows 3.0, Windows 3.0 applications (running in protected mode) are freed from the restrictive 640KB DOS address space.

Windows 3.0 is *not* a standard DPMI client and cannot run under DPMI in a VDM without completely subverting the operating system's memory protection and thereby potentially compromising system integrity.

Even with DPMI, Windows cannot run in 386 enhanced mode under OS/2 Version 2.0. The reason for this is that when Windows runs in enhanced mode it operates at Ring 0. Running Windows in 386 enhanced mode would therefore require bypassing the operating system's protection mechanisms, and would potentially compromise the integrity of the system.

## 9.6 Summary

Applications which experience memory constraints under DOS may overcome many of the inherent limitations of real mode by running protected mode. However, an application running in protected mode cannot easily access the facilities of real mode software such as the DOS operating system or TSR programs. DPMI provides an interface which allows an application to execute in protected mode, and to make DOS requests through DPMI. All mode switching and address conversion is handled by DPMI on the application's behalf.

DPMI resolves problems relating to device virtualization, intertask protection, and demand paging that occur when multiple protected mode DOS extender applications are run in a multitasking environment, in conjunction with memory managers and control programs.

DPMI is implemented under OS/2 Version 2.0 using a combination of virtual device driver services and kernel services to provide DPMI functions to client applications. The provision of these functions allows applications written to use DPMI services, such as applications which run under Microsoft Windows in standard mode, to run in a VDM under OS/2 Version 2.0.

# Chapter 10.  Running DOS Applications

OS/2 Version 2.0 allows the user to run multiple DOS applications concurrently, with each application running in its own virtual DOS machine, with pre-emptive multitasking and full memory protection.

This chapter describes the way in which a DOS application can be defined to the OS/2 Version 2.0 Workplace Shell, and the ways in which an application may be started.  It also discusses the way in which version-specific DOS applications may be run in virtual DOS machines under OS/2 Version 2.0.

## 10.1  Defining a DOS Application

A DOS application is typically defined to the Workplace Shell by creating a representative object for the application, and configuring the properties of that object using the *settings* view.  Configuring an object in this way allows the application to take advantage of the many customizable properties of the OS/2 Version 2.0 VDM  environment, and to tailor this environment to provide optimum performance and application compatibility.

## 10.1.1  Creating a Representative Object

To define a DOS application as an object under the Workplace Shell, do the following:

1. Open the *Templates* folder on the desktop, and copy the *Program* object by pointing to it with the mouse pointer, holding down mouse button 2 and dragging the icon to the desktop or to the folder in which the DOS application will reside.

2. The *settings* view for the new object will automatically open.  On the *Program* page of the *settings* notebook, complete the *Program title* and *Path and file name* fields for the application.  The *Parameters* and *Working directory* fields are optional, and depend on the application being installed.  Users should check the documentation supplied with the DOS application.
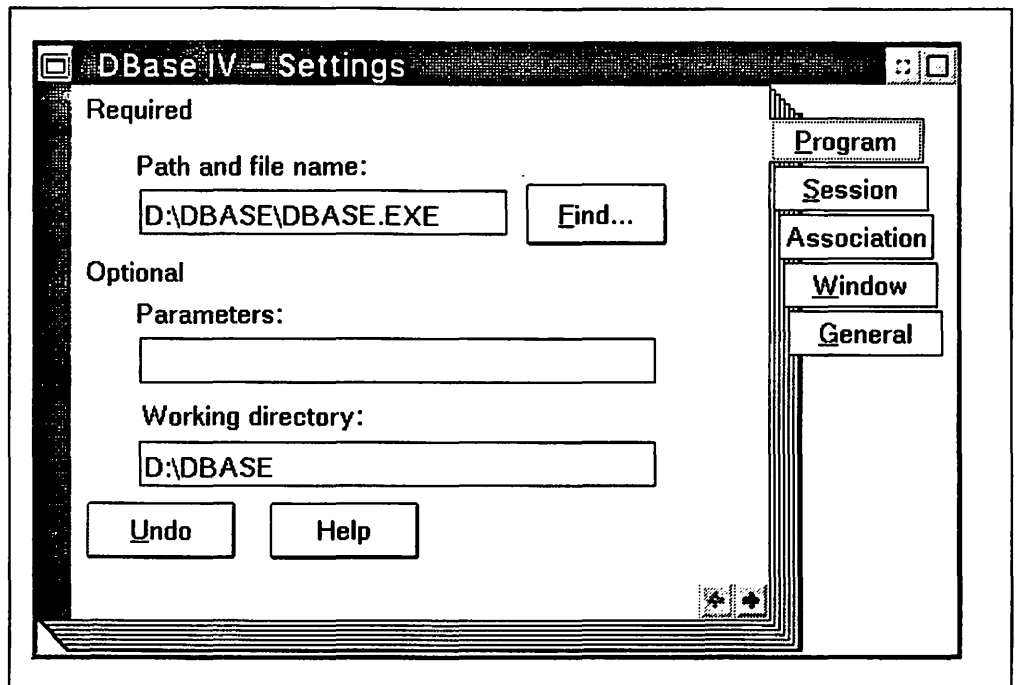
*Figure 48. The Program Page of the Settings Notebook*

3. On the *Session* page, select either *DOS Window* or *DOS Full Screen*, depending on how the DOS application will be run. In most cases, it is sufficient to select *DOS Window* since when maximized, the window will allow the full 25 rows by 80 columns to be displayed.

Another advantage of selecting *DOS Window* is that the user can use the copy and paste functions of the VDM to selectively transfer data between the DOS application, the clipboard, and any other application on the desktop that supports the clipboard. However, some graphics applications suffer performance degradation when run in windowed mode. For such applications, *DOS Full Screen* should be selected.

Note that once the VDM is started, a user can switch between *DOS Window* and *DOS Full Screen* modes by holding down the *Alt* key and pressing the *Home* key.

Do not confuse running a DOS application full screen with running it in a full screen window (a maximized window that fills the entire screen). There can be a significant performance difference between the two.

Note that clipboard function *Copy All* is also available for full screen virtual DOS machines. This allows the user to copy the entire DOS full screen to the clipboard (there is no way to mark only a portion of the screen). To perform this function, the user must press Ctrl+Esc to return to the desktop, click on the application icon with mouse button 2, and select *Copy All*.

4. On the *General* page, complete the *Title* field. The user can optionally choose to display a different icon from the default *DOS Window* or *DOS Full Screen* icons.
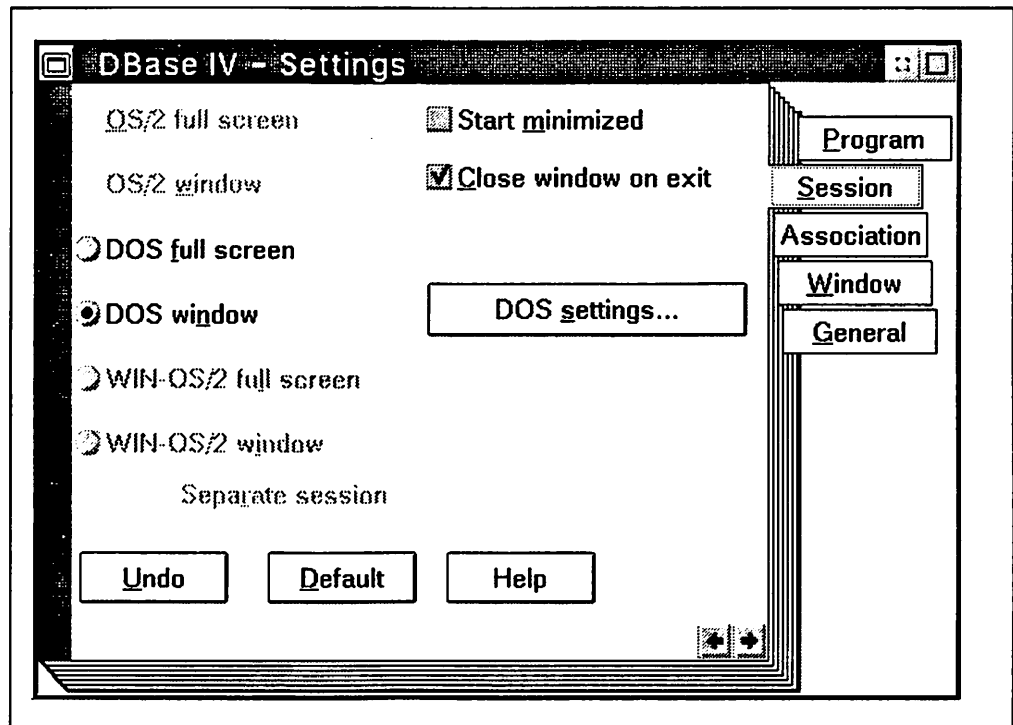
*Figure 49. The Session Page of the Settings Notebook*

5. If you select the *DOS settings* push button, the system brings up another
dialog. In this dialog, you can setup all DOS/VDM relevant characteristics,
unique to this particular program object.
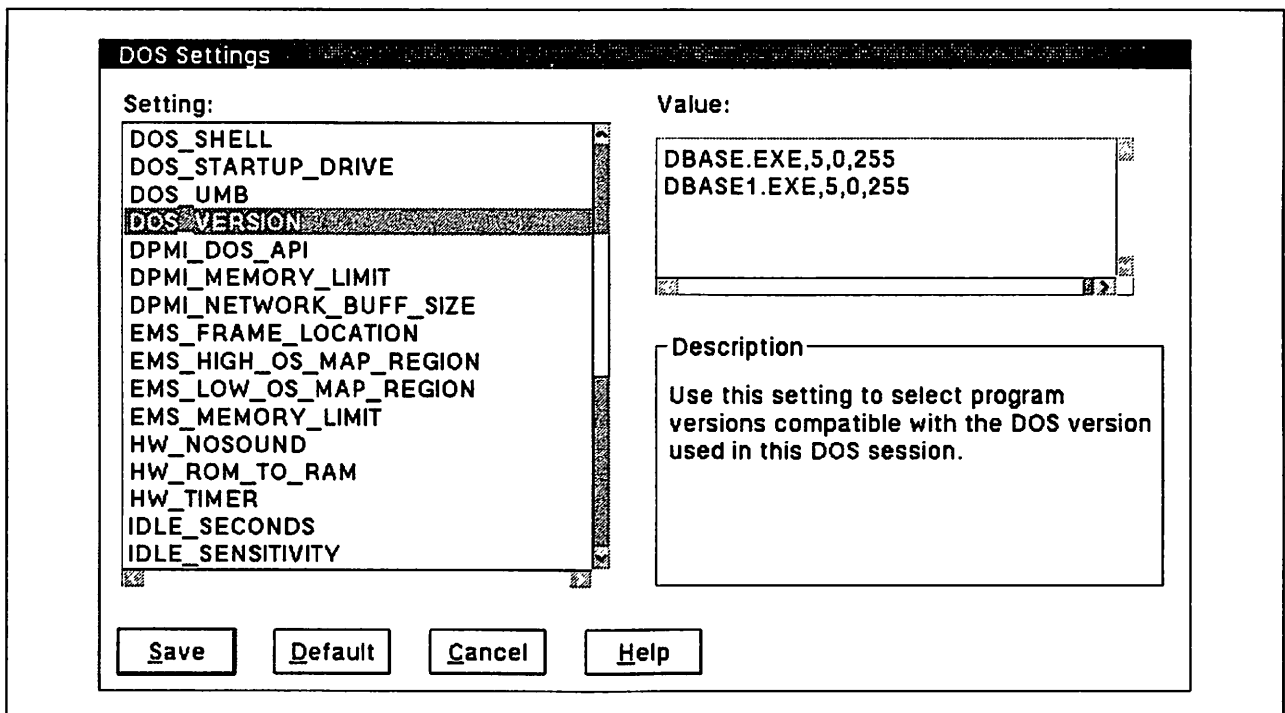


*Figure 50. The DOS Settings Dialog of the Settings Notebook*

When the *settings* notebook is closed, the application will appear on the desktop
or in the folder, with the specified application name beneath its icon.

## 10.1.2 Adding TSRs to the Workplace Shell

TSRs (Terminate-and-Stay-Resident) are DOS programs that stay resident in memory after terminating. This allows another DOS application to be loaded, while the TSR can still be accessed by a software or hardware interrupt, such as a hot-key sequence. An example is the dial-up terminal emulator FTTERM.

A TSR will not work if it is added to the Workplace Shell using the steps in the previous section. The virtual DOS machine closes when it detects the TSR terminating and gives it no chance to become resident.
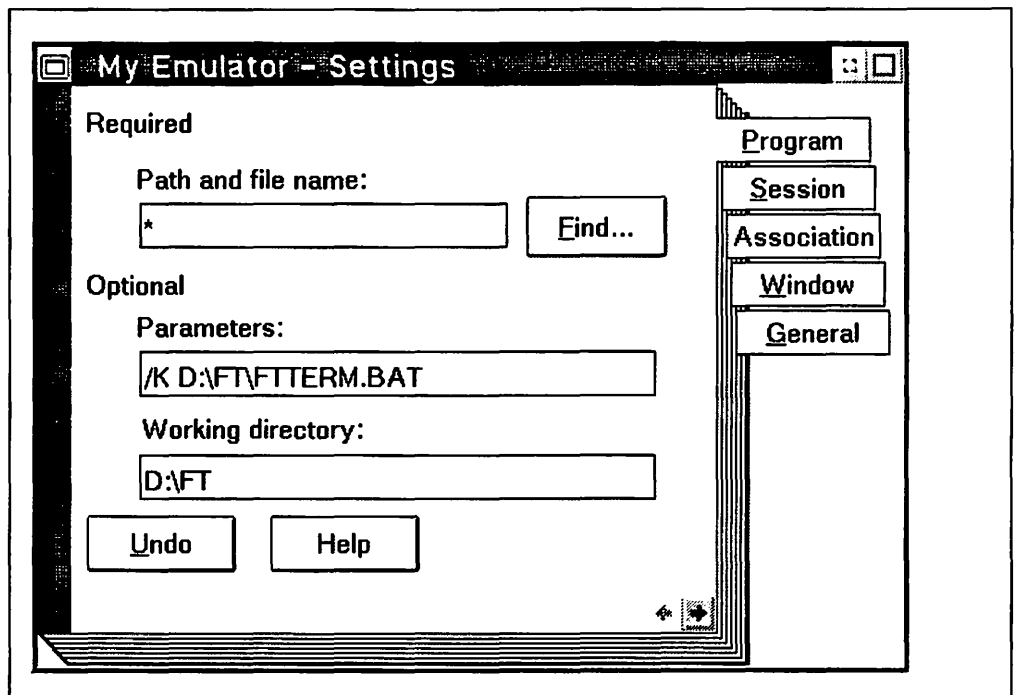


Figure 51. Setting Up a TSR Program

To add a TSR to the Workplace Shell, do the following:

1. Open the *Templates* folder on the desktop, and copy the *Program* object to the desktop or the required folder.

2. On the *Program* page of the *settings* notebook, complete the *Program title* field with an asterisk ("*").

3. Fill in the *Parameters* field with a "/k" followed by the path and program name of the TSR.

4. Complete the *Session* and *General* pages of the *settings* notebook as for other DOS applications.

## 10.1.3 Customizing the VDM Environment

The OS/2 Version 2.0 virtual DOS machine environment may be extensively customized to suit the requirements of a particular DOS application. Such properties as DOS device drivers, EMS/XMS memory configurations, and even the interface to hardware facilities can be specified individually for each VDM.

This customization is achieved using the DOS Settings facility, which is accessed by pressing the *DOS settings* pushbutton on the *Session* page of the *settings* notebook.

# ITSC Technical Bulletin Evaluation

**Technical Bulletin Title:** _____

**Technical Bulletin Form Number:** _____

This is an evaluation form to assess the quality of ITSC publications. Your feedback will help maintain the high quality of ITSC standards. Please fill out this questionnaire and send it to the address on the back of this page. No postage stamp is required if mailed in the U.S. Elsewhere, you may choose to have your IBM Marketing Representative forward your reply to the address listed on the reverse side of this form.

Date publication was ordered (MM/DD/YY)   ___/___/___

Date publication was received (MM/DD/YY)   ___/___/___

**Please rate on a scale of 1 to 5 the subjects below.**
**(1 = very good, 2 = good, 3 = average, 4 = poor, 5 = very poor)**

| | | | |
|---|---|---|---|
| Technical Bulletin organization | ___ | Grammar/punctuation/spelling | ___ |
| Accuracy of the information | ___ | Ease of reading and understanding | ___ |
| Relevance of the information | ___ | Ease of finding information | ___ |
| Completeness of the information | ___ | Lack of redundant information | ___ |
| Value of illustrations | ___ | Overall satisfaction | ___ |

**Please answer the following questions:**

a) Was the level of detail of the information adequate?     Yes___ No___

b) Did you find information duplicated that was available in other     Yes___ No___
IBM publications?

If yes, please name the publication:

_____

_____

c) Was the bulletin published in time for your needs?     Yes___ No___

d) Did this bulletin meet your needs?     Yes___ No___

If no, please explain:

_____

_____

**Comments/Suggestions:**

Thank you for your feedback.

Your name, company name, and address (optional):

_____

_____

_____

_____

ITSC Technical Bulletin Evaluation Form

The DOS Settings facility and the available settings are described in detail in Chapter 11, "DOS Settings."

## 10.1.4 Using the Migrating Applications Facility

Many common DOS applications can be set up on the Workplace Shell with their virtual DOS machine customized automatically by using the *Migrate Applications* facility of the *Systems Setup*. There is a file in the *\OS2\INSTALL* subdirectory called *DATABASE.DAT* that contains information on commonly available DOS, Windows V3.0 and OS/2 Version 1.3 applications. For DOS and Windows V3.0 applications this file includes the recommended *DOS settings* for the virtual DOS machine setup for that application.

After installing the DOS application, start the *Migrate Applications* facility and follow the dialog boxes to select the DOS application. If the migration is successful, an icon will be created for the application in a folder called *DOS Applications*.

Refer to Chapter 7, "Installing and Migrating Applications" on page 117 on using the *Migrate Applications* program.

## 10.2 Starting a DOS Application

A DOS application can be started in a virtual DOS machine by a user in one of several ways:

- By double-clicking mouse button 1 on the application's representative object on the desktop or in a folder

- By starting the application from an OS/2 or DOS command line

- By running an OS/2 batch file containing the *Start* command with the appropriate switches

Note that an OS/2 application may also start a DOS application by issuing a **DosExecPgm()** function call. The DOS application can be started as a dependent or independent child process of the OS/2 application.

There are certain limitations to starting DOS programs from an OS/2 V2.0 command prompt. For example, neither output redirection (using the ">" character) nor piping (using the "¦" character) works as it would from a DOS command prompt. When starting a DOS application from the OS/2 command prompt, OS/2 Version 2.0 calls the DOS command processor (COMMAND.COM), which then receives the application's name and parameters and starts it. The OS/2 V2.0 command processor does not start the application itself but transfers all control to the DOS COMMAND.COM. When we use redirection or piping from the OS/2 command prompt, it is only effective for the OS/2 session. Since OS/2 starts COMMAND.COM, and not the DOS application itself, OS/2 will only redirect the output of COMMAND.COM, not the application.

Thus with a DOS program XYZ, neither:

```
XYZ > DUMMY.FIL
```

nor:

```
XYZ ¦ more
```

would work from the OS/2 command prompt the way it does from a DOS command prompt.

One solution to the above limitation is to put the redirection or piping statement into a DOS batch file and call the batch file instead.

## 10.2.1 Starting From the Workplace Shell

In order for an application to be started from within the Workplace Shell, it must first be defined and configured as described in 10.1, "Defining a DOS Application" on page 197. Once this has been done, the application may be started simply by double-clicking mouse button 1 on the application's icon on the desktop or in a folder.

Note that when the application is started, the background of the icon will change to indicate that the application is in use. By default, if the user double-clicks mouse button 1 on the icon a second time, the operating system will *not* start a second instance of the application, but will simply bring the already started instance to the foreground.

If the user wishes to create a second instance of the application, a second representative object can be created by copying the original instance. Alternatively, the user can change the default behavior in the *Window* page of the *settings* notebook.

## 10.2.2 Starting From the Command Line

A DOS application can also be started from a DOS or OS/2 command line. Note, however, that starting the application in this way provides no opportunity to configure the VDM environment to support particular application requirements. Certain settings may be changed during application execution. However, such settings will be saved and will remain in effect until explicitly reset by the user.

The default settings also allocate resources for EMS, XMS and DPMI support, which may not be required by the DOS application. For these reasons, it is recommended that DOS applications which require non-default settings be configured and started from the Workplace Shell wherever possible.

When starting the application from a DOS command line, the application loads and executes within the VDM which displayed the command line. All DOS environment settings used by the application are those in effect for the VDM when the application was started. When the application terminates, control is returned to the command line.

When starting the application from an OS/2 command line, the operating system reads the program file from disk, and determines from the executable file header that the program is a DOS or Windows application rather than an OS/2 application. The operating system then automatically creates a VDM and loads the application into the VDM. When the application terminates, the VDM is also terminated and control returns to the OS/2 command line.

Note that in either case, execution is synchronous, and the command line is *not* available for use while the application is running. An application may be started asynchronously from an OS/2 command line using the START command. In this case, the operating system creates an asynchronous VDM and loads the application into this VDM. The OS/2 command line remains available even though the DOS application is still running.

## 10.3 Applications With Special Requirements

The virtual DOS machine environment normally runs a specialized version of DOS known as the DOS Emulation kernel, in which most DOS services are actually provided by components of the OS/2 operating system, transparently to the application and outside of the real mode address space in which the DOS application executes. This kernel is described in Chapter 4, "MVDM DOS Emulation."

For this reason, many DOS control structures are not accessible to DOS applications running in VDMs. Applications which access these control structures cannot be run in a "normal" VDM due to the lack of these structures. The DOS Emulation kernel does not support the use of block device drivers, and applications which require such device drivers are unable to use the DOS Emulation kernel.

In order to allow such applications to be run in VDMs under OS/2 Version 2.0, the Virtual Machine Boot facility is provided. This facility allows a "real" version of DOS to be loaded into a VDM, either from a DOS bootable diskette or from a diskette image stored on fixed disk. Since the real version of DOS is therefore running in the VDM, all features, characteristics and internal control structures of that version are available to an application running in the VDM.

An example of an application that needs to be run in this way is PC Support/400.

The Virtual Machine Boot facility is described in detail in Chapter 12, "Virtual Machine Boot."

## 10.4 Summary

Multiple DOS applications may be run concurrently under OS/2 Version 2.0, in virtual DOS machines with pre-emptive multitasking and full memory protection. By default, these applications access DOS and hardware services using an emulated version of DOS, which provides these services through the OS/2 operating system. Most of these DOS services are provided outside the 640KB real mode address space in which the DOS application executes, thereby allowing more memory (up to 630KB) for the application and its data.

DOS applications can usually be installed by starting a virtual DOS machine and running the install program from the command prompt. If the installation fails the user can boot the system from a DOS diskette to run the install, provided the hard disk has at least one FAT partition.

A DOS application may be defined as an object on the OS/2 Version 2.0 Workplace Shell desktop or in a folder, and started from the Workplace Shell. Applications defined in this way have their VDM environment configured to support particular application requirements and to allow the application to take full advantage of VDM features. Alternatively, the *Migrate Applications* facility can be used to place the application in the Workplace Shell and customize the *DOS settings*

A DOS application may also be started from the DOS or OS/2 command line. However, applications started from the DOS command line inherit the DOS environment settings of the VDM in which the command line is executing, and those started from the OS/2 command line inherit the default settings.

DOS Applications which require access to internal DOS control structures or block device drivers not supported by the DOS Emulation kernel may use the Virtual Machine Boot facility of OS/2 Version 2.0 to load a "real" version of DOS from a diskette or a diskette image stored on fixed disk. This capability allows such applications to run in a VDM.

.

# Chapter 11. DOS Settings

In order to provide the highest possible level of compatibility with DOS applications which make use of particular DOS properties or attributes, MVDM provides virtual DOS machines with many more customizable properties than comparable OS/2 sessions. MVDM provides a common mechanism which supports standard settings, and allows virtual device drivers to register custom settings. The CONFIG.SYS file contains a number of standard DOS settings; these are applied to all VDMs as they are created. Other settings may be specified for individual VDMs.

When running Windows applications in VDMs under OS/2 Version 2.0, certain DOS settings should be altered from their default values. The recommended values for these settings when running Windows applications are discussed in 8.7.5, "DOS and WIN-OS/2 Settings" on page 153.

DOS settings are used during creation and initialization of a VDM, and certain settings may also be altered dynamically during VDM execution. During initialization of the VDM, the VDM_CREATE hooks for all virtual device drivers defined for that VDM are called by the Virtual DOS Machine Manager. At this point, the virtual device drivers may call the **VDHQueryProperty()** helper service to get the values for required settings.

## 11.1 Registration

Information on DOS settings is stored in a "database" in the operating system kernel. This database is used to support all operations related to DOS settings. The following information is registered for each setting:

**Name**
: The name presented to the user. This may contain blanks, and related settings should have common prefixes so that they sort together in the list presented to the user (such as *Printer buffer size*, *Printer timeout*, *Printer automatic close*).

**Ordinal**
: For the "standard" settings, specific ordinals are used so that the kernel may obtain the value independently of the name string.

**Help File**
: The name of the help file containing help information on this setting.

**Help ID**
: The help ID of the main topic for this setting.

**Type**
: The setting type. The following types are supported:

  - Boolean
  - Integer
  - Enumeration (list of valid strings)
  - Single-line strings
  - Multi-line strings.

  This allows the user interface to display an appropriate control for each setting.

**Flags**
: These control aspects of the setting. In particular, the flags determine whether the setting can be changed while a VDM is running.

**Default Value** If the user does not supply a value, this default value is used.

**Validation information**

This information allows the user interface (and the kernel) to validate settings without involvement from the virtual device driver. For strings, this is the maximum string length. For integers, this is the minimum, maximum, and step values. For enumerations, this is the list of valid strings.

**Function**    This function is used for validating string settings, and for notifying the VDD when the user has changed a setting value for a running VDM.

## 11.1.1 Changing Settings Prior to Execution

The Workplace Shell enables a user to define objects which represent OS/2 and DOS applications, using the *Settings* notebook. For DOS applications, a *DOS Settings...* button is provided in the *Session* page of the notebook. Pressing this button causes the *DOS Settings* dialog box to be displayed. The user may then manipulate the DOS settings (which are initially set to their default values), and then save them.

The *DOS Settings* dialog box uses the **DosQueryDOSProperty()** function to get the list of settings and detailed information on each setting. It uses the **DosSetDOSProperty()** function to validate string settings.

## 11.1.2 Changing Settings During Execution

MVDM inserts a *DOS Settings...* menu item on the system menu for all VDMs. Selecting this menu item causes the DOS Settings dialog box to be displayed, which in turn allows the user to modify settings for the VDM. For full screen VDMs, the user must switch to the Presentation Manager Window List using the Ctrl+Esc key sequence, and display the context menu for the VDM session by clicking mouse button 2 on the VDM's entry in the Window List. The *DOS Settings...* option is displayed in the context menu.

Note that only those settings that have been registered as being modifiable at run time may be altered in this way; other settings are not presented in the dialog box.

## 11.1.3 Starting a VDM From Another Application

The **DosStartSession()** function under OS/2 Version 2.0 provides an *environment* pointer as one of its parameters. This pointer references a buffer which is used when creating a VDM. This buffer contains the buffer length, followed by one or more DOS settings. For each setting, the buffer specifies the type, name and value. The operating system parses the settings buffer as part of the **DosStartSession()** processing, in order to create initial values for these settings. Default values are assumed for any registered settings not specified in the **DosStartSession()** call.

For any settings which have not been registered, the information in the buffer is ignored. This allows the system to run without errors in the case where the virtual device driver that registered a setting is not loaded (for example, CONFIG.SYS was changed), and yet the Presentation Manager shell has saved a value for that setting.

## 11.2 Standard DOS Settings

```
┌──────────────────────────────────────────────────────────────────────┐
│ ┌────────────────────────────────────────────────────────────────┐   │
│ │ DOS Settings                                                   │   │
│ │                                                                │   │
│ │  Setting:                          Value:                      │   │
│ │  ┌──────────────────────────┐▲    ┌──────────────────────────┐▒ │   │
│ │  │ DOS_SHELL                │     │ DBASE.EXE,5,0,255        │  │   │
│ │  │ DOS_STARTUP_DRIVE        │     │ DBASE1.EXE,5,0,255       │  │   │
│ │  │ DOS_UMB                  │     │                          │  │   │
│ │  │ DOS_VERSION              │     │                          │▒ │   │
│ │  │ DPMI_DOS_API             │     └──────────────────────────┘  │   │
│ │  │ DPMI_MEMORY_LIMIT        │     ◄────────────────────────────► │   │
│ │  │ DPMI_NETWORK_BUFF_SIZE   │                                    │   │
│ │  │ EMS_FRAME_LOCATION       │    ┌─Description──────────────┐   │   │
│ │  │ EMS_HIGH_OS_MAP_REGION   │    │                          │   │   │
│ │  │ EMS_LOW_OS_MAP_REGION    │    │ Use this setting to select program │
│ │  │ EMS_MEMORY_LIMIT         │    │ versions compatible with the DOS version │
│ │  │ HW_NOSOUND               │    │ used in this DOS session. │   │
│ │  │ HW_ROM_TO_RAM            │    │                          │   │
│ │  │ HW_TIMER                 │    │                          │   │
│ │  │ IDLE_SECONDS             │    │                          │   │
│ │  │ IDLE_SENSITIVITY         │▼   │                          │   │
│ │  └──────────────────────────┘    └──────────────────────────┘   │   │
│ │                                                                │   │
│ │   ┌────────┐  ┌─────────┐  ┌────────┐  ┌──────┐               │   │
│ │   │ Save   │  │ Default │  │ Cancel │  │ Help │               │   │
│ │   └────────┘  └─────────┘  └────────┘  └──────┘               │   │
│ └────────────────────────────────────────────────────────────────┘   │
└──────────────────────────────────────────────────────────────────────┘
```
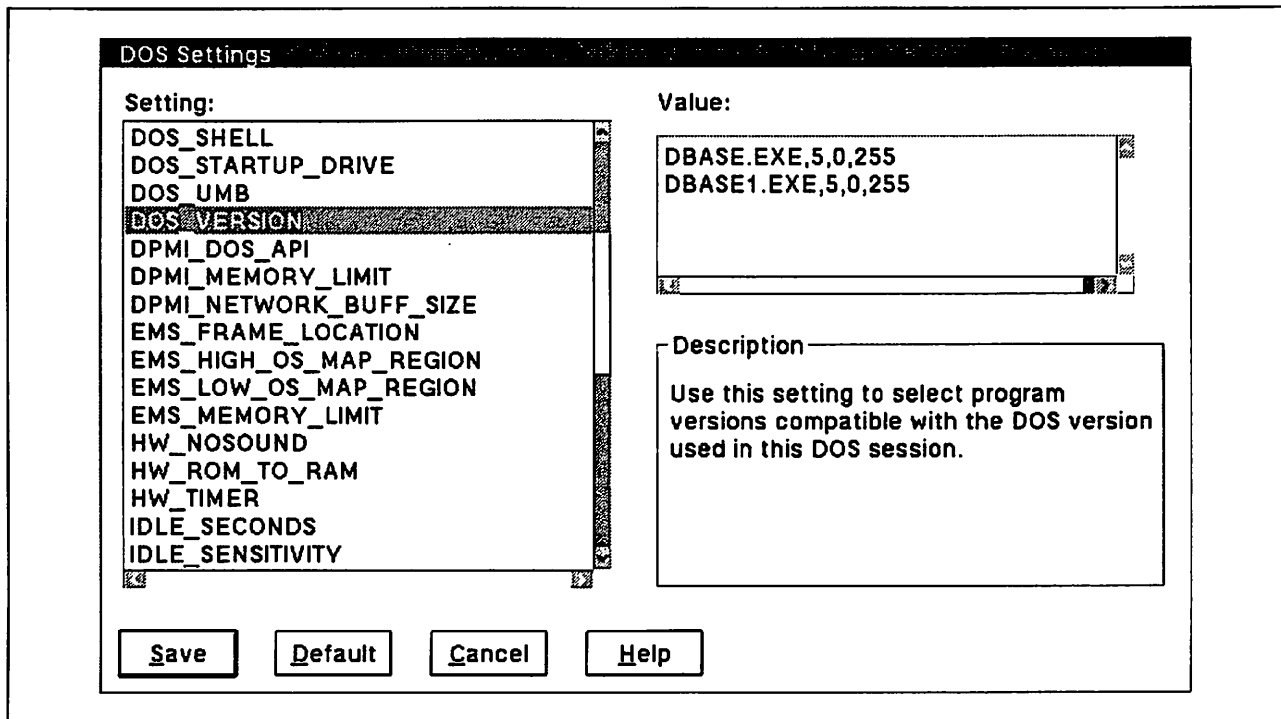
*Figure 52. The DOS Settings Dialog of the Settings Notebook*

The standard DOS settings which affect the operation of virtual device drivers supplied with OS/2 Version 2.0 are described on the following pages. The settings are grouped according to the general DOS system function to which they apply.

DOS settings may be changed in either of two ways:

- Settings which are described as settable "at VDM creation only," may only be changed prior to starting the VDM. If the DOS application is defined as an object under the Workplace Shell, this is done by selecting the *DOS Settings button* from the *Session* page in the application's *Settings* notebook.

- Settings which are described as settable at any time may be set in the manner described above, or they may be changed while an application is running in a VDM, using the DOS Settings.. option from the system menu.

Note that certain settings may be changed in both of the above ways.

### 11.2.1 Communications

The following settings control the communications environment (COM ports) used by a VDM.

#### 11.2.1.1 COM_HOLD

**Function:**  When set on, provides exclusive access to COM ports for the specified VDM, preventing other processes from using the port and preventing the operating system from releasing the port until the VDM terminates.

**Advantages:** For certain applications which use COM ports and which require multiple programs to access the COM port (for example, this setting prevents the COM port from being released when the first program terminates).

**Drawbacks:** If not required by the application running in a VDM, this setting may prevent other applications from accessing COM ports.

**Default:** Off.

**Settable:** At VDM creation only.

**Examples:** Certain bulletin board applications use one program to dial the BBS and another to exchange information; setting COM_HOLD on prevents the operating system from releasing the COM port when the first program terminates.

## 11.2.2 DOS Environment

The following settings affect the behavior of the DOS emulation environment within a virtual DOS machine.

### 11.2.2.1 DOS_BACKGROUND_EXECUTION

**Function:** When set off, suspends execution of the program when it is in the background.

**Advantages:** Many DOS applications are written on the assumption that they are single tasking and that all the resources of the workstation can be monopolized. It is not uncommon for a DOS program to continually poll for keyboard input (Examples are WordPerfect 5.1 and Lotus 1-2-3 R2.2). In a multitasking environment, this can impact system performance, especially when more than one such program is running. Turning the DOS application off when its virtual DOS machine is in the background reduces its demands on the system.

Also see 11.2.6.2, "IDLE_SENSITIVITY" on page 216 and 11.2.6.1, "IDLE_SECONDS" on page 216.

**Drawbacks:** Communications programs will fail if background execution is turned off, as will DDE for Windows applications.

Try changing the values of *IDLE_SECONDS* and *IDLE_SENSITIVITY* before turning *DOS_BACKGROUND_EXECUTION* off.

**Default:** On (Background execution is enabled).

**Settable:** At any time.

**Examples:** If more than two DOS programs are running and tuning with *IDLE_SENSITIVITY* and *IDLE_SECONDS* does not provide sufficient improvement, turn *DOS_BACKGROUND_EXECUTION* off for the least used application.

### 11.2.2.2 DOS_BREAK

**Function:** Enables or disables Ctrl+Break for the specified VDM. Also check for the *BREAK* statement in the CONFIG.SYS. Set *BREAK=ON* in the CONFIG.SYS to make Ctrl+Break and Ctrl+C working in addition to setting *DOS_BREAK* on.

**Advantages:** Enables a DOS application running in the VDM to be interrupted using the Ctrl + Break or Ctrl + C key sequences.

**Drawbacks:** This setting is useful only if an application must be quickly interrupted; the user may easily terminate a VDM by closing it from the Window List.

**Default:** Off (Ctrl + Break is disabled).

**Settable:** At VDM creation only.

**Examples:** If the user wishes to have the option to interrupt a DOS batch file running in a virtual DOS machine, this setting should be turned on.

## 11.2.2.3 DOS_DEVICE

**Function:** This setting can be used to add or modify information about DOS device drivers for the specified VDM, in addition to the information specified in CONFIG.SYS.

**Default:** When this setting is selected, a list is displayed which contains information about each DOS device driver specified in CONFIG.SYS. The information consists of the path and file name of each DOS device driver and its current parameters, if applicable. For example:

```
c:\os2\mdos\ansi.sys
```

The user may:

- Type the name of a DOS device driver to add it. Typing should begin on a new line.
- Delete all the information about a device driver to remove it.
- Type or delete to add, change, or delete a value.

**Settable:** At VDM creation only.

**Examples:** A program to support hardware such as a scanner may include a device driver that is needed only for itself. The device driver should be loaded with the *DOS_DEVICE* setting instead of in the CONFIG.SYS.

## 11.2.2.4 DOS_FCBS

**Function:** Specifies the maximum number of file control blocks (FCBs) which may be opened by applications running in the VDM. Note that this setting affects only those modules which use file-sharing.

**Advantages:** Reducing this setting may improve DOS application performance in a resource-constrained networking environment. When the maximum number of FCBs is opened by an application, the least recently used FCB is closed to allow additional files to be opened; see DOS_FCBS_KEEP below.

**Drawbacks:** Reducing this setting to an excessively low number may inhibit the performance of applications which use large numbers of files. Check application documentation for recommended FCB settings.

**Default:** 16.

**Settable:** At VDM creation only.

**Examples:** None.

### 11.2.2.5 DOS_FCBS_KEEP

**Function:** Specifies the number of FCBs that will be protected against automatic closure.

**Advantages:** If this setting is specified as "n," the first "n" files are protected against automatic closure as described in 11.2.2.4, "DOS_FCBS" on page 209. This may improve application performance.

**Default:** 8.

**Settable:** At VDM creation only.

**Examples:** None.

### 11.2.2.6 DOS_FILES

**Function:** Specifies the maximum number of file handles which may be opened in a VDM.

**Advantages:** Setting this value higher than the default may improve performance for applications which use a large number of files. Check application documentation for recommended settings.

**Drawbacks:** Setting the number of file handles higher than necessary reduces the available memory.

**Default:** 20.

**Settable:** At any time.

**Examples:** DBASE IV requires a *DOS_FILES* setting of at least 40.

### 11.2.2.7 DOS_HIGH

**Function:** Determines whether DOS is loaded outside the 640KB low memory address space.

**Advantages:** Loading DOS into high memory allows more available memory for application code and data within the 640KB address space.

**Drawbacks:** Applications which require access to DOS internal control structures require DOS to be loaded into low memory, and therefore cannot use this setting.

**Default:** Off (DOS is loaded into low memory).

**Settable:** At VDM creation only.

**Examples:** None.

### 11.2.2.8 DOS_LASTDRIVE

**Function:** Specifies the highest available logical drive letter for the specified VDM. This setting is similar to the *LASTDRIVE=* statement in a DOS CONFIG.SYS.

**Default:** Z.

**Settable:** At VDM creation only.

**Examples:** Each additional drive letter uses about 100 bytes. Setting the *LAST_DRIVE* to a lower letter such as J or K provides more conventional memory for an application.

### 11.2.2.9 DOS_RMSIZE

**Function:** Specifies the DOS memory size. This is the amount of memory which is available to DOS applications.

**Advantages:** The virtual video device driver uses this setting on certain video adapters to set even more than 640KB.

**Drawbacks:** This setting is of little use to most users as there is no point specifying less than 640KB.

**Default:** The default is 640KB.

**Settable:** At VDM creation only.

**Examples:** None.

### 11.2.2.10 DOS_SHELL

**Function:** To specify the DOS command processor, or to add parameters to affect the command processor. This setting points by default to COMMAND.COM. If a user has a different command processor, it should be specified here.

**Advantages:** The user may specify a command processor other than the default COMMAND.COM, if required by a specialized application, or may alter the environment space available for the VDM.

**Default:** C:\OS2\MDOS\COMMAND.COM C:\OS2\MDOS /P

**Settable:** At VDM creation only.

**Examples:** C:\OS2\MDOS\COMMAND.COM /E:1024 /P

### 11.2.2.11 DOS_STARTUP_DRIVE

**Function:** Specifies the location of the DOS kernel to be loaded into the VDM.

**Advantages:** Allows specific versions of DOS to be loaded into a VDM using the VMB facility, allowing the execution of version-dependent DOS applications.

**Drawbacks:** Performance may not be as good as the VDM kernel, which is optimized for the OS/2 V2.0 environment.

**Default:** The DOS Emulation kernel is loaded.

**Settable:** At VDM creation only.

**Examples:** See Chapter 12, "Virtual Machine Boot."

### 11.2.2.12 DOS_UMB

**Function:** Specifies whether DOS owns Upper Memory Blocks (UMBs) and manages the loading of device drivers and TSR programs.

**Advantages:** Setting DOS_UMB on allows use of the DEVICEHIGH= and LOADHIGH statements, to load device drivers and TSR programs into Upper Memory Blocks, thereby preserving space in low memory for use by applications.

**Drawbacks:** Certain applications which make use of UMBs need to access and manage the UMBs directly; such applications will not run when DOS_UMB is set on, because DOS owns the UMBs.

| **Default:** | Off (UMBs are owned by certain types of TSR programs and DOS device drivers if necessary). |
| **Settable** | At VDM creation only. |
| **Examples:** | None. |

### 11.2.2.13 DOS_VERSION

| **Function:** | Allows the operating system to report a "fake" DOS version number in response to a request from a program in the VDM, in order to support applications which check for a DOS version number. |
| **Advantages:** | Allows some programs that will not start unless they detect a prerequisite DOS version to run in DOS Emulation |
| **Default:** | 20 |
| **Settable:** | Before application initiation. |
| **Examples:** | Lotus 1-2-3 R3+ will run in DOS Emulation if it is "fooled" into thinking that it is running under DOS 3.3 by putting the following lines into the *DOS_Version* list box: |

- 123DOS.EXE,3,30,255
- 123.EXE,3,30,255
- LOTUS.EXE,3,30,255

## 11.2.3 DPMI

The following settings control the DPMI interface for a VDM.

### 11.2.3.1 DPMI_DOS_API

| **Function:** | Determines whether DOS API translation is enabled for the specified VDM. |
| **Default:** | AUTO (API translation is enabled if required). |
| **Settable** | At VDM creation only. |
| **Examples:** | None. |

### 11.2.3.2 DPMI_MEMORY_LIMIT

| **Function:** | Specifies the maximum amount of protected mode memory (in megabytes) available to DPMI applications running in the VDM. |
| **Advantages:** | For applications which require large amounts of DPMI memory, this setting may be used to increase the amount of available memory up to 512MB. |
| **Default:** | 2MB. |
| **Settable** | At VDM creation only. |
| **Examples:** | None. |

### 11.2.3.3 DPMI_NETWORK_BUFF_SIZE

| **Function:** | Specifies the size, in kilobytes (KB), of the network translation buffer for DPMI programs in this session. The range is from 1 to 64 KB. |
| **Default:** | 8KB. |

**Settable**    At VDM creation only.

**Examples:**    This setting allows you to configure the size of the translation buffer for Windows programs that transfer data over a network. If a network-specific Windows program does not run correctly under OS/2 V2.0, increase this setting, then restart the session.

# 11.2.4  EMS

The following settings control the behavior of EMS memory used by the VDM.

### 11.2.4.1  EMS_FRAME_LOCATION

**Function:**    This DOS setting allows you to change the location of the LIM EMS region. LIM EMS uses a 64KB address region known as an EMS page frame, through which programs can access expanded memory. (This allows programs to use more than 640KB of memory.)

**Advantages:**    If a user has problems when running a program that uses both a hardware device and LIM EMS expanded memory, the problem may be due to conflicting use of addresses by LIM EMS and the hardware device. If this occurs, the user should first use the *EMS_HIGH_OS_MAP_REGION* setting to set the extra address region used by EMS to 0. This may solve the problem. If the problem persists, the *EMS_FRAME_LOCATION* setting can be used to select a 64KB region that does not conflict with hardware.

The user can choose where to place the frame from a list of choices or can choose to have no EMS frame for programs which do not require a frame. The user can also reduce the *DOS Memory Size* setting and place the frame below 640KB.

**Drawbacks:**    The best solution, when problems due to hardware conflicts occur, is to use the *MEM_EXCLUDE_REGIONS* and *MEM_INCLUDE_REGIONS* settings to specify the addresses that the hardware uses rather than using this setting.

**Default:**    The default AUTO setting will lead to correct choices of LIM EMS addresses. Most users will never need to change this setting.

**Settable:**    At VDM creation time only.

**Examples:**    In some cases the default choice may conflict with addresses used by hardware on the machine. This can happen only for devices that are not supported by a virtual device driver.

### 11.2.4.2  EMS_HIGH_OS_MAP_REGION

**Function:**    In addition to the EMS page frame, some programs can use additional addresses to access expanded memory. This setting gives advanced users the capability to adjust the size of the additional EMS region.

See also 11.2.4.1, "EMS_FRAME_LOCATION."

**Advantages:**    An advanced user can use the *MEM_EXCLUDE_REGIONS* and *MEM_INCLUDE_REGIONS* settings to specify the addresses used by devices that do not have virtual device drivers, and can then set the size of the *EMS_HIGH_OS_MAP_REGION* appropriately for their program. This helps avoiding conflicts with addresses used by devices and programs.

| Default: | The value set is the size of the region in kilobytes. The default is 32KB. |
| Settable: | At VDM creation only. |
| Examples: | None. |

### 11.2.4.3 EMS_LOW_OS_MAP_REGION

| Function: | Some programs can use remappable conventional memory. Others do not use this feature. This setting allows advanced users to set the size of the remappable conventional memory available in a VDM. |
| Default: | The value set is the size of the region in kilobytes. The default is 384KB. |
| Settable: | At VDM creation only. |
| Examples: | None. |

### 11.2.4.4 EMS_MEMORY_LIMIT

| Function: | This setting controls the amount of EMS memory available to a VDM. |
| Advantages: | The user can set this to a higher value for running programs that require a large amount of EMS memory. Other programs do not use EMS at all. The size can be set to 0 in such cases, to disable EMS support for that VDM. Programs generally state whether they use EMS on the box or in their manuals. |
| Default: | The value set is the size of the region in kilobytes. The default size is 2MB. |
| Settable: | At VDM creation time only. |
| Examples: | If a spreadsheet runs out of memory, the amount of EMS memory can be increased and the VDM restarted. |

## 11.2.5 Hardware Environment

The following settings affect the virtual hardware environment provided by the virtual DOS machine.

### 11.2.5.1 HW_NOSOUND

| Function: | Enables or disables sound started by a DOS program. |
| Advantage: | Any sound from a program is heard unless sounds is disabled. An "x" in the check box indicates that the sound is to be heard. |
| Drawbacks: | No error sound will be heard if HW_NOSOUND is turned on. |
| Default: | OFF. |
| Settable: | At any time, including while a program is running in a VDM. |
| Examples: | Output from a music program may be disabled when the user wishes to hear another music program, or switch to another process to do something else. |

### 11.2.5.2 HW_ROM_TO_RAM

**Function:** Enabling HW_ROM_TO_RAM causes the operating system to copy read-only memory (ROM) and run the copy in 32-bit random access memory (RAM). With this setting enabled, BIOS operations run faster and system utilities may patch BIOS.

**Default:** OFF.

**Settable:** At VDM creation only.

**Examples:** This setting is useful if debugging the kernel. The change would allow normal breakpoints to be set in ROM and allow stepping over calls and loops.

**Warning:** If an application writes to a memory address used by the ROM while this setting is enabled, it may cause unpredictable results for that application and for every application run thereafter in the VDM.

### 11.2.5.3 HW_TIMER

**Function:** When enabled, allows an application to have direct access to the 8253 timer ports and prevents the operating system from trapping, or intercepting, the timer request and emulating a timer.

**Advantages:** Certain timing-critical applications will not run (or will run much slower) if accesses to timer ports are trapped and virtualized. In addition, the values they read do not accurately reflect the amount of time passed because they do not take trapping overhead into account. Enabling this setting allows certain timing-dependent code to run more effectively.

**Drawbacks:** Applications that change the divisor before this setting is enabled and then read the timer ports after the setting has been enabled may not function properly. If the setting is enabled first, the VDM will not detect changes to the divisor correctly, and the simulated interrupt frequency will be incorrect. Also, multiple applications using this setting may interfere with one another.

**Default:** Off. Most applications will operate normally with timer virtualization.

**Settable:** At any time. It is useful to alter this setting dynamically and watch for changes in application performance.

**Examples:** The ROMs on some machines implement very brief delays by polling the timer ports. These delays become unacceptably long unless direct timer port access is allowed.

## 11.2.6 Idle Detection

The following settings determine the way in which the operating system detects that an application in a VDM is currently idle. These settings should be used when an application exhibits poor performance, or where mouse movement in a DOS application is "jerky."

### 11.2.6.1 IDLE_SECONDS

**Function:**      When programs appear to be doing nothing but waiting for input, the operating system gives them less time to run. This is done to give preference to programs that are doing useful work. Some programs periodically appear to be waiting for input, but then change their behavior and continue after a time. This setting disables the "IDLE_SENSITIVITY" function for a period of time after useful work has been detected.

Also see 11.2.6.2, "IDLE_SENSITIVITY" below for more details on idle detection.

**Advantages:**   If a program appears to run slowly when there is an option for the user to provide input, this value should be increased.

**Drawbacks:**    Setting the value too high gives the DOS program more resources than it needs.

**Default:**      This value is in seconds. The default is no idle time allowed.

**Settable:**     The setting can be changed while the program is running to tune it to the proper value.

**Examples:**

- A game may pause, for instance, to wait for the user to make a choice, but then continues if the user does not react.

- When DOS 5 is run in a virtual machine boot session, (See Chapter 12, "Virtual Machine Boot") the DOS shell may fail to complete displaying the directory of the C: drive if *IDLE_SENSITIVITY* is set too low. *IDLE_SECONDS* should then be raised.

### 11.2.6.2 IDLE_SENSITIVITY

**Function:**      The idle sensitivity level sets a threshold for judging when applications will be considered idle. The value is the percentage of the maximum possible polling rate the application can perform. If an application polls at a rate higher than this value, it is considered "idle."

DOS programs often "poll" for input when they are waiting for a user response. For instance, a program may wait for a response by repeatedly checking to see if the user has hit a key. In a multitasking environment such as OS/2 Version 2.0, this wastes time when other programs could be running instead. The operating system detects idle programs by looking for a high rate of polling for input. When programs are judged to be waiting for input, they are given less time to run.

For example, if idle sensitivity is set to 75%, then an application repeatedly checking to see if input is available would have to do this checking at more than 75% of the maximum possible rate before it would be judged idle.

Idle detection is a "best guess" of what the program is doing. It could be that the program is polling at a very high rate, but is still doing useful work in between checking. It may be that the application checks at a fairly slow rate but still is doing nothing but waiting. The idle sensitivity threshold allows adjustment of the threshold for a particular application.

Also see 11.2.6.1, "IDLE_SECONDS."

**Advantages:** If an application receives input while running and seems to run slower than expected, the idle sensitivity should be set to a higher value. This lets the application poll at a higher rate without being judged idle. Setting the level to 100 turns idle detection off altogether. The application will be allowed to poll for input as often as it likes.

If an application is waiting for input and other applications do not appear to be running, the idle sensitivity should be adjusted downward. This lowers the threshold for judging the application idle.

**Default:** The default is 75%.

**Settable:** The setting can be changed while the program is running to tune it to the proper value.

**Examples:** Overall system performance can usually be improved when there are multiple DOS applications running if *IDLE_SENSITIVITY* is turned down.

Also see 11.2.2.1, "DOS_BACKGROUND_EXECUTION" on page 208.

## 11.2.7 Keyboard

The following settings affect the behavior of the keyboard and the interpretation of control key sequences issued within a VDM.

### 11.2.7.1 KBD_ALTHOME_BYPASS

**Function:** When enabled, prevents the Alt + Home key sequence from switching the VDM between full screen and windowed mode.

**Advantages:** Enabling this setting allows normal behavior for applications which themselves make use of the Alt + Home key sequence.

**Drawbacks:** When enabled, the user must use the Ctrl + Esc sequence to switch to Presentation Manager from a full screen VDM, then use the context menu of the class to switch the VDM to windowed mode.

**Default:** Off (Alt + Home will cause a switch between full screen and windowed mode).

**Settable:** At any time.

**Examples:** None.

### 11.2.7.2 KBD_BUFFER_EXTEND

**Function:** Increases a VDM's keyboard type-ahead buffer size.

**Advantages:** Provides greater keystroke buffering, consistent with the level available in VIO windows. Note that Ctrl-Break will flush the entire buffer, just as it does with the standard buffer.

**Drawbacks:** Applications which bypass the ROM BIOS input buffer and/or INT 16h may not benefit from this feature. There is also a small amount of additional memory overhead for every VDM.

**Default:** On. Most applications will benefit, and those that do not should not be adversely affected.

**Settable:**    At any time. This facilitates easy experimentation by the user in the (rare) event that a problem does arise.

**Examples:**    None.

### 11.2.7.3 KBD_CTRL_BYPASS

**Function:**    When enabled, inhibits one or more control key sequences, allowing an application in the VDM to use these sequences for its own purposes.

**Advantages:**    Enabling this setting allows normal behavior for applications which make use of control key sequences normally used by OS/2 Version 2.0.

**Drawbacks:**    Enabling this setting may prevent certain operations from being performed with OS/2 Version 2.0 and the Workplace Shell.

**Default:**    NONE (All control key sequences behave in the normal manner).

**Settable:**    At any time.

**Examples:**    None.

### 11.2.7.4 KBD_RATE_LOCK

**Function:**    Prevents a DOS application in a VDM from changing the system keyboard repeat rate.

**Advantages:**    Insulates machine from applications that modify the repeat rate in an uncontrolled or undesirable way.

**Drawbacks:**    Prevents the application's repeat rate from taking effect even when the application is the focus session.

**Default:**    Off. Most applications do not modify the repeat rate, and those that do are generally in accordance with the user's wishes.

**Settable:**    At any time.

**Examples:**    None.

## 11.2.8 Memory Extenders (EMS and XMS)

The following settings affect the behavior of the EMS and XMS memory extenders, if used in the VDM. For an explanation about the implementation of EMS and XMS support in VDMs, see Chapter 6, "Memory Extender Support."

### 11.2.8.1 MEM_EXCLUDE_REGIONS

**Function:**    This setting is used to specify address ranges which should be protected from use by EMS/XMS and direct access by applications. *This setting is intended for experienced users who understand the hardware.*

**Advantages:**    This setting restricts the use of EMS/XMS on certain ranges in the region between RMSIZE and 1MB. It also protects these ranges from being touched by user applications by portraying ROM there.

**Drawbacks:**    Some hardware adapters stop functioning if their addresses are touched in random fashion. If these ranges are defined excessively, they will adversely impact the function and performance of EMS and XMS services.

| Default: | By default, this setting is void. Each address is specified in hex and if there is no range specified, the length taken is a page (4KB). |
|---|---|
| Settable: | At VDM creation only. |
| Examples: | None. |

### 11.2.8.2 MEM_INCLUDE_REGIONS

| Function: | Specify regions which should be made available to EMS/XMS. This setting is used to specify some address ranges between RMSIZE and 1MB for use by EMS and XMS. |
|---|---|
| Advantages: | If there is a hardware adapter in this range which the user knows is not going to be used by a particular VDM session, then the address range used by this adapter should be made available to EMS and XMS. This will improve the performance of EMS and XMS services. *Only advanced users who know the addresses used by a card should use this setting.* |
| Default: | By default, this setting is void. |
| Settable: | At VDM creation only. |
| Examples: | See discussion in 6.2, "Expanded Memory (EMS) and Upper Memory (UMB)" on page 102. |

## 11.2.9 Mouse

The following settings affect the behavior of the mouse in a VDM.

### 11.2.9.1 MOUSE_EXCLUSIVE_ACCESS

| Function: | This setting allows VDMs to run applications which maintain their own mouse pointers. Some DOS applications manage their own mouse positions and movements; in many cases, the application's values for mouse sensitivity and/or double speed threshold are different from those of Presentation Manager. As a result, a Presentation Manager mouse pointer may be outside the VDM window while the application pointer is somewhere in the window not receiving any mouse events. This means having two asynchronous mouse pointers on the screen. |
|---|---|
| Advantages: | The user forces the physical mouse driver to send its events directly to the virtual mouse driver without going through Presentation Manager. Only one mouse pointer appears when the particular VDM window has the focus. |
| Default: | OFF. |
| Settable: | At any time. |
| | However, this only marks the VDM window and does not actually activate the setting. In order to activate it, the user must press a mouse button within the VDM window. The Presentation Manager pointer disappears, leaving only the application pointer. In order to regain the Presentation Manager pointer, the user must press any of the hot-keys (Alt, Ctrl+Esc, Shift+Esc). |
| Examples: | WordPerfect 5.1 has its own block-shaped mouse pointer, which will appear together with the system mouse pointer when the window has the focus. Turning *MOUSE_EXCLUSIVE_ACCESS* on |

allows the user to remove the system mouse pointer when in WordPerfect.

## 11.2.10 Printer

The following settings affect print functions within a VDM.

### 11.2.10.1 PRINT_TIMEOUT

**Function:** Use this setting to adjust the amount of time, in seconds, that the OS/2 V2.0 print subsystem waits before forcing a print job to the printer. In DOS, information sent by a program for printing goes directly to a printer. However, the OS/2 V2.0 print subsystem assembles print information in a spool file. After a specified period of time, during which the spool file does not grow larger, OS/2 V2.0 print subsystem sends the information to the printer as a single print job.

**Advantage:** There is no need to exit the DOS program before the print job is released by the OS/2 V2.0 print subsystem. This is useful for applications which do not explicitly close their print jobs.

**Default:** 15 seconds, configurable from 0 to 3600 seconds (0 seconds is no timeout).

**Settable:** At any time.

**Examples:** A timeout of 1 or 2 seconds is sufficient for small print jobs, such as copying the contents of the screen. However, when printing large files, formatting documents, or running calculations, the value must be set high enough to allow all print results to reach the spooler before the time limit expires. If not, results go in two or more spool files instead of one, and the resulting output may be unsatisfactory.

## 11.2.11 Video

The following settings control screen I/O operations within a VDM.

### 11.2.11.1 VIDEO_FASTPASTE

**Function:** Speeds up input from other sources than the keyboard.

**Advantages:** Improves the speed of paste operations from the clipboard to a DOS application.

**Drawbacks:** Does not work with all applications (in particular, some applications which monitor keyboard interrupts directly may experience errors).

**Default:** Off.

**Settable:** At any time. This facilitates easy experimentation by the user.

**Examples:** Pasting into the DOS command prompt, or any application using DOS Console I/O functions, will generally work. However, the Microsoft Editor (M) and its successor, Programmer's Workbench (PWB), can fail when using fast pasting because they rebuffer keystrokes in an internal buffer, which can overflow.

### 11.2.11.2 VIDEO_MODE_RESTRICTION

**Function:**   Extends the 640KB DOS address space by limiting video mode support.

**Advantages:**   For text-based or CGA graphics based applications, the video memory normally reserved just above 640KB for high-resolution graphics modes can be remapped to conventional memory, providing an additional 64KB (or 96KB, depending on graphics mode) for DOS applications, TSRs, and other programs. This is valuable for applications that do not take advantage of EMS or XMS memory extenders.

**Drawbacks:**   It is not possible to completely hide the fact that the video adapter is high-resolution graphics-capable; some applications may attempt to enable those modes and use the memory above 640KB as video memory, inadvertently corrupting application data. Care must therefore be taken when using this feature.

**Default:**   NONE. The complete list of settings is:

- None
- CGA modes only (adds 96KB)
- MONO modes only (adds 64KB).

**Settable:**   At VDM creation only.

**Examples:**   None.

### 11.2.11.3 VIDEO_ONDEMAND_MEMORY

**Function:**   Reduces swap space requirements for fullscreen VDMs.

**Advantages:**   Allows a full-screen VDM to run without pre-allocating a virtual video buffer for the worst-case video modes (high-resolution graphics modes). Using this setting does not prevent execution of graphics applications; it simply means that allocation of the buffer is delayed until it is needed. This can save a substantial amount of memory/swap space, which might be important under certain low-memory conditions. It also enables you to start a program quickly.

**Drawbacks:**   If allocation of a virtual video buffer for a full-screen VDM fails at the time the application changes video modes, the session must be frozen and switched back to the shell. Unless the user is able to free memory from another session, he may be unable to get the DOS application running again. This is a concern if the application contains unsaved data.

**Default:**   Off.

**Settable:**   At any time. This allows the user to save memory the next time the session is switched to full-screen.

**Examples:**   None.

### 11.2.11.4 VIDEO_RETRACE_EMULATION

**Function:** Simulates the video retrace status port to provide faster access.

**Advantages:** DOS applications that poll the video retrace status port often write to the screen only during the retrace interval, even though it is safe (on EGA and VGA adapters) to draw at any time without causing interference (also known as "snow"). This feature causes most applications to write to the screen more often, and compensates for the performance drag imposed by monitoring the port in the first place.

**Drawbacks:** Some applications may poll the port in such a way that overall performance is worse; this is sometimes true of applications that draw only during vertical (not horizontal) retrace. Unfortunately, while turning off trace emulation will restore performance, there is a risk that screen-switching will not be as reliable.

**Default:** On. Reliable screen-switching has higher priority over the minority of applications that will experience some drag in performance.

**Settable:** At any time. This allows the user to experiment with different settings in the event of a performance problem.

**Examples:** None.

### 11.2.11.5 VIDEO_ROM_EMULATION

**Function:** Emulates selected INT 10h ROM Video functions.

**Advantages:** Provides faster output for selected video functions than ROM services typically provide. This also has a dramatic effect on the performance of those functions in a window.

**Drawbacks:** Some ROMs may offer enhanced services that are not included in the emulation. Applications which rely upon these services may not execute correctly.

**Default:** On. Because the INT 10h ROM Video services are well-documented, incompatibilities are unlikely and the performance benefits of using the emulation are quite significant.

**Settable:** At any time. This allows the user to experiment in the event of a compatibility problem.

**Examples:** None.

### 11.2.11.6 VIDEO_SWITCH_NOTIFICATION

**Function:** Notifies a DOS application of a switch to/from full-screen mode.

**Advantages:** Allows applications that monitor this notification to redraw their screens as needed. This may be necessary for some video adapters that provide modes (and applications that use those modes) which are not fully supported by the OS/2 video driver or which are slightly incompatible. It is also valuable in situations where an OS/2 video driver has not allocated a virtual video buffer (see 11.2.11.8, "VIDEO_8514_XGA_IOTRAP" on page 223). Use this setting if you use the *VIDEO_ONDEMAND_MEMORY* DOS setting, because concurrent buffer allocation and screen switching can make a screen go black.

**Drawbacks:** When used indiscriminately, this feature may cause unnecessary and time-consuming screen redrawing. For standard MONO/CGA/EGA/VGA video modes, the OS/2 video driver should be able to restore application screens without assistance.

**Default:** Off. For standard hardware and standard video modes, this feature is not necessary.

**Settable:** At any time. This allows the user to experiment in the event of a compatibility problem.

**Examples:** Windows 2.x and 3.x understand this notification and will redraw themselves accordingly. For WIN-OS/2 sessions, set this setting on.

### 11.2.11.7 VIDEO_WINDOW_REFRESH

**Function:** Adjusts the window update frequency for a given VDM.

**Advantages:** For applications (particularly graphics) that write frequently to video memory, this value can be increased to reduce time spent updating the window and provide more processor time for the application.

> **Note:** This has no effect on updates based on other events such as keyboard input or synchronous scrolling operations or any video events other than refresh.

**Drawbacks:** A large refresh period can make an application unusable (or at least, very hard to use).

**Default:** 0.1 seconds. This has been found to yield the best overall performance.

**Settable:** At any time, in increments of 0.1 seconds. This allows for experimentation. The range is from 0.1 to 60.0 seconds.

**Examples:** This setting affects normal TTY-style output. Compare a DIR or TYPE operation before and after altering this setting.

### 11.2.11.8 VIDEO_8514_XGA_IOTRAP

**Function:** When set OFF, unrestricted access to 8514/A display adapter hardware. Note that this setting is *only* available for systems with 8514/A display adapters installed.

**Advantages:** Achieves higher performance for 8514/A applications and eliminates the overhead of the 1MB 8514/A virtual video buffer normally allocated for each VDM when set OFF.

**Drawbacks:** Screen-switching away from the application will result in immediate freezing of the application, and the system may not be able to reliably switch back; that is, the screen image may not be correct. This may be overcome by setting VIDEO_SWITCH_NOTIFICATION on, which notifies applications to redraw their own screen images. Note however, that not all applications will take advantage of the notification.

> **Note:** An application with this setting enabled may not be run in windowed mode, or copied to the clipboard, because there is no complete information about its state.

**Default:** Off.

**Settable:** No; may be set at VDM creation only.

**Examples:** When executing Windows 3.0 with the 8514/A display driver, certain operations such as painting dithered backgrounds will run significantly faster.

# 11.2.12 XMS

### 11.2.12.1 XMS_HANDLES

**Function:** Specifies the number of XMS extended memory block (EMB) handles. A handle is used with each XMS EMB. This number is required because XMS pre-allocates all the handle space to be compatible with XMS specifications. This setting should be used only if an application uses a large number of handles.

**Advantages:** This setting restricts the number of block handles, thereby reducing memory consumption.

**Drawbacks:** Specifying a large number of handles will increase memory consumption and adversely impact system performance.

**Default:** The default value of this setting is 32.

**Settable:** At VDM creation only.

**Examples:** None.

### 11.2.12.2 XMS_MEMORY_LIMIT

**Function:** Specifies the per VDM XMS memory limit. This setting should be used under the same guidelines as described above in 11.2.12.1, "XMS_HANDLES." The global limit is the overall maximum XMS memory consumption, and the per-VDM limit is the maximum allowed for each VDM. See also 6.3.1.2, "Initialization" on page 107 for defining global and per-VDM limit in the CONFIG.SYS.

**Drawbacks:** Specifying a large number may adversely affect system performance.

**Default:** The default value is 2MB per-VDM.

**Settable:** At VDM creation only.

**Examples:** 4096; this specifies a limit for each VDM.

### 11.2.12.3 XMS_MINIMUM_HMA

**Function:** Specifies the minimum HMA memory request allowed. This setting allows the user to fine tune the XMS. HMA is slightly less than 64KB in size. Only one request can be fulfilled from this area at a time.

**Advantages:** If a TSR takes a very small allocation, then it will waste this area for other applications. In such cases a limit can be specified.

**Default:** The default value is zero, which means all the requests will be allowed.

**Settable:** At VDM creation only.

**Examples:** 2048; this sets a limit of 2KB.

## 11.2.13 WIN-OS/2

The following setting is available when the selected virtual DOS machine type is *WIN-OS/2 full screen* or *WIN-OS/2 window*:

### 11.2.13.1 WIN_RUNMODE

**Function:**   OS/2 V2.0 can use two modes to run Windows programs:

- Real
- Standard

Real mode is the mode that Windows 2.0 programs run in. Windows 3.0 programs usually run in standard mode. For a detailed discussion, see Chapter 8, "Windows Applications." Use this setting to specify WIN-OS/2 mode for your session.

**Default:**   AUTO (the system selects standard mode as long as it has the OS/2 V2.0 Virtual Device Drivers required to support a standard mode WIN-OS/2 session in the OS/2 V2.0 operating system). AUTO enables the system to automatically choose between real and standard.

**Settable:**   At VDM creation time only.

**Examples:**   None.

## 11.3 Summary

The DOS Settings feature of MVDM provides the user with the ability to selectively configure and tune the virtual DOS machine environment to meet the requirements of particular applications. Since some DOS applications require certain features while other applications operate better without them, a VDM may be individually configured to provide the optimum execution environment for the application which will run within it.

DOS settings may be set by the user when adding an application to a group on the desktop, or in certain cases, during execution while an application is running within the virtual DOS machine. In the case where a VDM is created by another process using a **DosStartSession()** function call, a buffer may be provided containing the required DOS settings and their values.

# Chapter 12. Virtual Machine Boot

An important goal of OS/2 Version 2.0 is the ability to run past, current, and future DOS programs; indeed most DOS applications available today run unchanged in the MVDM environment.

However, it should be remembered that the "DOS" which runs in this case is highly optimized for (and specific to) an OS/2 Version 2.0 virtual 8086 machine. Because of this, there are fundamental internal differences between the DOS Emulation provided under OS/2 Version 2.0 and "real" DOS. Two problems may therefore arise:

1. Some programs may depend on specific DOS features such as internal control blocks, LAN Redirector hooks, or even undocumented functions, which are not present in MVDM's DOS Emulation.
2. Only DOS *character* device drivers can be loaded in MVDM's DOS Emulation. The user may own a *block* device (such as a special disk or tape drive) for which no OS/2 driver is available. (A block device is one accessed via a drive letter, such as E:).

**Virtual Machine Boot (VMB)** solves both these problems. It allows the user to boot "off-the-shelf" DOS and use block device drivers in an OS/2 Version 2.0 virtual DOS machine, ensuring greater compatibility for DOS applications.

Another possible use of VMB is to run DOS of a different National Language to that of OS/2 Version 2.0 itself. This may be useful in a multilingual environment.

## 12.1 VMB Environment

The 80386 processor and VDM component of OS/2 Version 2.0 together emulate a 8086 processor, keyboard, display, BIOS and other supporting hardware; in effect, a complete "virtual Personal Computer." It is therefore possible for "real" DOS to be loaded in a VDM session. Control is passed to the boot record (the first sector) of the DOS system diskette, which in turn loads and initializes the rest of the DOS kernel, just as it does when booting on a physical PC.

Indeed, the VDM environment is so similar to a real PC environment that VMB can actually support any 8086 operating system kernel, such as Digital Research's DR-DOS** and CP/M**, Microsoft MS-DOS**, or even a PS/2 reference diskette (but do not attempt to run diagnostics or change the configuration from a VDM; the results are unpredictable). However, since the purpose of VMB is to run current DOS applications, formal IBM support is announced for IBM PC DOS 3.x, 4.0, and 5.0 only.

Table 6 on page 228 shows the amount of *available* base memory for MVDM DOS Emulation, DOS in a VMB session, and native DOS. These figures show the amount of memory available after loading the operating system and mouse, EMS and XMS support.

| Table 6. Free Base Memory | | | | |
|---|---|---|---|---|
| **Setting** | **VDM DOS Emulation** | **DOS 5.0** | **DOS 4.0** | **DOS 3.3** |
| DOS low | 610 KB | 566 KB | 588 KB | 545 KB |
| DOS high | 633 KB | 612 KB | - | - |
| With mode restriction (CGA) | 728 KB | 707 KB | 653 KB | 670 KB |
| native DOS | - | 564 KB (low) 614 KB (high) | 545 KB | 562 KB |
| **Note:** Each configuration has HIMEM, EMS and Mouse drivers loaded. Values are approximate. | | | | |

A VDM using VMB is similar in function to any other virtual DOS machine. Multiple VDMs may be started and operated concurrently using Virtual Machine Boot. Each runs in its own virtual 8086 machine; access to hardware and other system resources is managed by MVDM and the underlying OS/2 Version 2.0 operating system.

## 12.2 Configuring Virtual Machine Boot

The DOS operating system loaded into a VDM by VMB may be:

1. An actual DOS system diskette
2. An image of a DOS system diskette saved to hard disk
3. A DOS partition on hard disk.

For any of the alternatives, we need to do the following:

1. Set up the start-up batch file AUTOEXEC.BAT

2. Set up the configuration file CONFIG.SYS

3. Provide OS/2 V2.0 with the real DOS to boot.

## 12.2.1 Preparing AUTOEXEC.BAT and CONFIG.SYS

The AUTOEXEC.BAT and CONFIG.SYS that will be used by the VMB at initialization are not the ones found in the root directory of the OS/2 V2.0 boot drive. The following table explains which AUTOEXEC.BAT and CONFIG.SYS will be used for the different DOS session types under OS/2 V2.0.

| Table 7. Location of AUTOEXEC.BAT and CONFIG.SYS | |
|---|---|
| **VDM Type** | **Location** |
| Virtual Machine Boot from diskette | drive A: |
| Virtual Machine Boot from diskette image | imbedded in diskette image file on the hard disk |
| Virtual Machine Boot from DOS boot partition | DOS boot partition |
| OS/2 V2.0 DOS emulator | root directory of OS/2 boot drive |

CONFIG.SYS requires special attention for the following reasons:

1. Write access to the hard disk is denied the Virtual Machine Boot session to preserve system integrity, since the real DOS is unaware of OS/2 V2.0 and the other applications running.

   The OS/2 device driver FSFILTER.SYS is provided to address the above problem.

2. HPFS partitions are not visible to the real DOS running.

   FSFILTER.SYS allows the DOS in the Virtual Machine Boot to access HPFS files.

3. OS/2 V2.0 provides its own mouse, EMS and XMS to each virtual DOS machine, so there is no need to load the equivalent drivers available for native DOS. Those provided with the real DOS should not be used.

   However, some DOS programs test for the presence of these drivers. OS/2 V2.0 provides the equivalent "stub" drivers to satisfy these programs that the services actually are available.

   The following types of device drivers should also be omitted from CONFIG.SYS:

   - Disk cache
   - Print spooler
   - RAM disk

   These facilities are already provided by OS/2 Version 2.0 and may be accessed by virtual DOS machines and VMB sessions; including them with DOS leads to unnecessary duplication, and may impact overall performance.

When the Virtual Machine Boot is started from a diskette image on the hard disk, the real DOS sees the diskette image as drive A:. The real drive A: cannot be accessed. OS/2 V2.0 provides a DOS program, FSACCESS.EXE, that can be used from the DOS command prompt or inserted in AUTOEXEC.BAT to overcome this problem.

We will cover each of these special requirements in detail in the following sections.

### 12.2.1.1  Drive Letter Allocation and Access

Drive letter allocation and access is one of the more complex areas of VMB, mainly due to the automatic drive letter allocation performed by DOS, and the limitations of earlier DOS versions. The following possible areas of confusion may arise for the user:

- If DOS is booted from an image file, it sees this image file as its A: drive. This prevents access to the *real* A: diskette drive. Attempts to write to the apparent A: drive will fail.

- Unlike the DOS Emulation kernel provided by OS/2 Version 2.0, the "real" DOS booted by VMB cannot see or access an HPFS partition on the hard disk.

- A DOS 3.x VDM cannot see a large (>32MB) FAT partition on the fixed disk, or FAT partitions beyond an HPFS partition on the disk.

- Even if the booted DOS can otherwise see the hard disk partition, it is only given read access. Attempts to write will fail with simulated errors such as "General failure writing drive C:". The user might mistake this for a genuine hardware fault.

- If the booted DOS loads a block device-driver, the allocated drive letter may be the same as that of a different device outside this VDM, thereby preventing access to that device from within the VDM.

The results could be somewhat disorienting for the user. To help resolve these issues, two utilities **FSFILTER** and **FSACCESS** are provided with OS/2 Version 2.0.

It is recommended that disk volumes should always be given a meaningful name, either when formatting or later using the LABEL command. This name will remain constant regardless of drive letter allocation, and will aid in identifying a particular volume, even if the assigned drive letter is different.

### 12.2.1.2 FSFILTER

FSFILTER.SYS is a device driver which manages DOS VDM access to OS/2 disks. FSFILTER.SYS should be copied from the \OS2\MDOS directory to the DOS diskette, and the following statement added to the CONFIG.SYS file of the bootable DOS diskette or image.

```
device=a:fsfilter.sys
```

This statement should precede any DEVICE= statements which load block device drivers.

Note that FSFILTER.SYS gives DOS full access to all OS/2 partitions, regardless of their file system type or partition size.

This is an important and somewhat surprising point. For example, DOS 3.3 (in a VDM) has no problem accessing a 300MB HPFS partition, once FSFILTER is loaded. I/O calls within the DOS virtual machine are passed transparently to OS/2 Version 2.0. DOS itself is unaware of the underlying file system. DOS can read, write and modify files on the hard disk, and for most configurations the drive letter mapping within the VMB session will match those of OS/2 Version 2.0.

The FSFILTER device driver occupies approximately 11KB of memory. It can be loaded into high memory under DOS 5.0 by specifying the command DEVICEHIGH = FSFILTER.SYS in CONFIG.SYS.

The users should also specify the path to COMMAND.COM in the SHELL= statement of CONFIG.SYS. For example, if DOS files have been copied to C:\DOS, the CONFIG.SYS file on a diskette intended for VMB should contain the following statement:

```
SHELL=c:\DOS\COMMAND.COM c:\dos /p
```

The first parameter specifies the command processor to be loaded. The second parameter specifies the reload path (that is, the COMSPEC path). This is preferable to a SET COMSPEC = command in AUTOEXEC.BAT.

Each block device driver loaded in DOS CONFIG.SYS is allocated the next free OS/2 letter *excluding LAN drives*. This can result in a drive letter clash. An example may illustrate the point. OS/2 drives are:

- **A:** Diskette drive 0
- **B:** Diskette drive 1
- **C:** Hard disk
- **D:** External diskette drive
- **E:** Remote LAN drive on a server

FSFILTER will ensure that a booted DOS sees these drives by the same letter. The booted DOS has the same access to the external diskette drive and LAN resources as does OS/2 itself. This is true whether the VMB session is started before or after user logon to the network, when remote drive letters are assigned.

However, a block device driver in a VMB session will also initialize as E:, so LAN drive access is lost. To remedy this, issue an "fsaccess f=e" command. The LAN drive is now accessible as F: within the DOS session.

Note that even when FSFILTER is loaded, the following restrictions still apply:

- A VMB session cannot see HPFS files or directories which have:
    - Long file names (9 or more characters)
    - Invalid FAT characters (for example, plus, comma, blank)
    - Multiple dot separators.

- HPFS file names containing lowercase letters are folded to uppercase.

- PC DOS commands which require low-level disk access will fail. These include:
    - CHKDSK
    - SYS
    - UNDELETE
    - FORMAT
    - UNFORMAT
    - MIRROR.

In such cases OS/2 Version 2.0 will simulate a disk error condition. DOS may interpret this as a hardware fault, or report that the command is not supported on a network or assigned drive.

### 12.2.1.3  FSACCESS

FSACCESS.EXE is a utility supplied with OS/2 Version 2.0 but intended to run in a VMB session. It cooperates with FSFILTER to manage drive letters within the VMB session. This serves three purposes:

1. Drives may be registered for filtering.

2. The drive letter for a device can be changed, giving consistency across sessions.

3. Letters can be removed in order to hide the OS/2 device from the VMB session.

The syntax of the FSACCESS command is:

```
┌──────────────────────────────────────────────────────────┐
│ FSACCESS ─────────────┬──────────────────────────────┬──  │
│                       │   ┌─ DOSletter ────────┐      │    │
│                       │  ┌┴┐                    │      │    │
│                       │  │,│                    │      │    │
│                       │  └─┘                    │      │    │
│                       ├─────── DOSletter - DOSletter ──┤    │
│                       │                                │    │
│                       └─────── DOSletter = OS2drive ───┘    │
└──────────────────────────────────────────────────────────┘
```
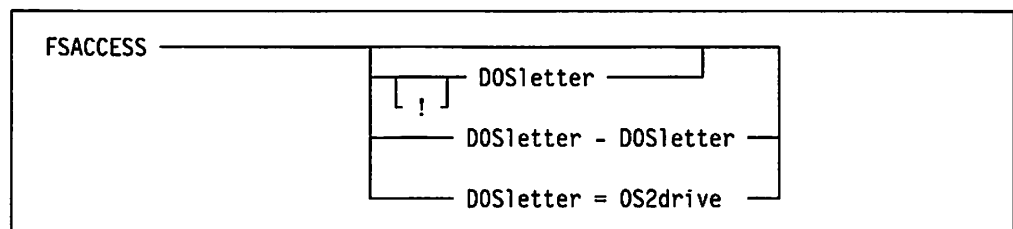
FSACCESS              Lists the current drive mapping. For example:

<div align="right">
Local C: is mapped to OS/2 C:<br>
Local D: is mapped to OS/2 D:<br>
Local E: is mapped to OS/2 K:
</div>

| | |
|---|---|
| FSACCESS F: | Registers DOS letter F: for filtering. References to F: will be sent to OS/2 Version 2.0. |
| FSACCESS !F: | De-registers DOS letter F: from filtering. |
| FSACCESS F:-H: | Registers DOS letters F: through H: for filtering. |
| FSACCESS M:=C: | Routes requests for DOS letter M: to OS/2 drive C: |

Parameters can be combined on a single command line, and the colon is optional.

When booting from an image file, it is often desirable to issue the command *FSACCESS A:* in order to access the A: diskette drive. This will remove access to the image file, so the booted DOS will be unable to reload its COMMAND.COM when necessary. It may be useful to copy all the DOS files to a subdirectory on hard disk, ensuring the PATH and COMSPEC point there.

An alternative is to access the diskette drive via a different letter. For example, a user may issue the command *FSACCESS G:=A*, then use G: to access the real A: drive. The image file remains as A:, avoiding PATH and COMSPEC problems.

## 12.2.2 Mouse, EMS and XMS Support

The booted DOS in a VMB session receives XMS (HIMEM), EMS, DPMI and mouse support services from its VDM environment (assuming the virtual DOS machine has default DOS settings). DOS should *not* therefore load its own HIMEM, EMS or mouse drivers; indeed they may cause errors in the VDM.

DOS programs call these services via appropriate API register parameters and a designated interrupt:

**Mouse**   INT 33h
**XMS**     INT 2Fh (multiplex)
**EMS**     INT 67h

These interrupts are trapped by the VDM environment, routed outside the virtual machine and handled by the OS/2 Version 2.0 operating system itself. This may present a problem for certain programs which first test for the presence of such services by issuing an OPEN command to the associated device driver, or which check that a valid interrupt handler is referenced by the Interrupt Vector Table. When a VMB session is started, these device driver names are not present, and the interrupt vectors point to null handlers. The application will therefore assume that the required services are not useable.

In order to avoid this problem, OS/2 Version 2.0 provides three alternative "stub" drivers:

- MOUSE.COM
- HIMEM.SYS
- EMM386.SYS

These stub drivers are very small (and use minimal memory when loaded) but satisfy programs which depend on drivers with such names being present. They respond to OPEN commands, and also set handler addresses in the Interrupt

Vector Table, thereby satisfying applications which check for the presence of the device drivers in either of these ways.

The user *must* load these OS/2 files rather than any similarly named files which may be shipped with DOS or applications, such as:

**DOS 4.0**    XMAEM.SYS, XMA2EMS.SYS

**DOS 5.0**    HIMEM.SYS, EMM386.EXE, MOUSE.COM

**DR DOS**    HIDOS.SYS, EMM386.SYS, EMMXMA.SYS

**Other**    MOUSE.SYS

There are two ways to achieve this. Assuming OS/2 Version 2.0 is installed on drive E:

1. Copy the above OS/2 files from E:\OS2\MDOS to the DOS diskette, and edit CONFIG.SYS and AUTOEXEC.BAT accordingly to load these files from the A: drive. VMDISK may then be run to create a bootable image if desired.

   ```
   device=a:himem.sys
   device=a:emm386.sys
   device=a:fsfilter.sys
   ```

   This method should be used if FSFILTER will be loaded into high memory using DOS 5.0:

   ```
   device=a:himem.sys
   device=a:emm386.sys
   devicehigh=a:fsfilter.sys
   ```

2. Edit CONFIG.SYS and AUTOEXEC.BAT to load these files directly from E:\OS2\MDOS. (FSFILTER.SYS must be loaded first if the OS/2 drive would otherwise be inaccessible to the booted DOS).

   ```
   device=a:fsfilter.sys
   device=e:\os2\mdos\himem.sys
   device=e:\os2\mdos\emm386.sys
   ```

   The second method has one notable advantage; if and when Corrective Service is applied to the OS/2 Version 2.0 system, and HIMEM, EMM386 or MOUSE are updated, it is not necessary to update your DOS diskettes and recreate image files. FSFILTER itself will have to be updated manually (unless the OS/2 Version 2.0 partition is directly accessible to DOS and FSFILTER is also loaded from here).

Note that EMS memory size and frame location are determined by DOS Settings, and *not* by parameters on the DEVICE= statement for EMM386.SYS. It is recommended that EMS and XMS support *should not* be configured unless required by the application running in the VMB session, since this can impact overall system performance.

We now look at the three different ways to prepare the real DOS to be booted in the VMB.

## 12.2.3 Booting from Diskette

The user may load a specific version of DOS or an equivalent 8086 operating system into a VMB session directly from a bootable diskette, by specifying A: at the value for *DOS_STARTUP_DRIVE* under DOS Settings. Note that this may affect the way in which applications in the VMB session may access the diskette

drives; see 12.2.1.1, "Drive Letter Allocation and Access" on page 229 for further discussion.

Here is an example using DOS 5:

1. From a system running DOS 5, format a diskette with the /s option.

2. Copy FSFILTER.SYS from the OS/2 V2.0 subdirectory \OS2\MDOS onto the diskette.

3. Create CONFIG.SYS and AUTOEXEC.BAT on the diskette.

   A sample CONFIG.SYS to use is as follows (OS/2 V2.0 is installed in E: drive in this example):

```
REM Load FSFILTER driver
DEVICE=A:FSFILTER.SYS
REM load the stub XMS and EMS memory drivers from OS2.
DEVICE=E:\OS2\MDOS\HIMEM.SYS
DEVICE=E:\OS2\MDOS\EMM386.SYS
```

   A sample AUTOEXEC.BAT to use is as follows:

```
@ECHO OFF
PROMPT $P$G
REM set the path to where the DOS files were copied
SET PATH=C:\DOS
SET COMSPEC=C:\DOS\COMMAND.COM
REM load the stub mouse driver from OS/2 V2.0
LH E:\OS2\MDOS\MOUSE
```

4. Create a DOS subdirectory on the hard disk and copy the real DOS files there.

5. Insert the DOS boot diskette in the A: drive.

6. Locate the *Command Prompts* folder. It is usually a folder in the *OS/2 System* icon on the Workplace Shell.

7. Open the *Command Prompts* folder.

8. Locate the *DOS from drive A:* icon and double click on it.

   The *DOS_STARTUP_DRIVE* setting of this icon is pre-set to the value *A:*.

The user cannot specify "B:" or an external diskette drive as the startup drive. There may be situations where it is desired to boot from a 5¼ inch diskette; typically the B: drive on PS/2 systems. One way to do this is by creating an image of the diskette, then booting this image (See 12.2.4, "Booting from Diskette Image" on page 235).

If a 5¼ inch diskette must be booted directly for some reason, this is possible if drive remapping is supported by the system (such as a PS/2 Model 57, 90 or 95). Normally A: is Drive 0 (3½ inch), and B: is Drive 1 (5¼ inch, if fitted). To change this, run "Set Startup Sequence" from the reference diskette, and ensure Drive 1 appears before Drive 0. Then the 5¼ inch drive will become the A: drive.

Some 5¼ inch drives (such as the IBM External 1.2MB drive and associated adapter) require a device driver, and are accessed as D: or higher. They cannot be specified as a startup drive, nor can they be readdressed as A:, but can be the source drive when creating a bootable image file.

## 12.2.4  Booting from Diskette Image

A user may also load a specific version of DOS or another 8086 operating system into a VMB session from a diskette image stored on the hard disk. This is achieved by specifying the fully qualified filename of the diskette image file as the value for *DOS_STARTUP_DRIVE* under DOS Settings.

Here is an example using the DOS 5  boot diskette created in the 12.2.3, "Booting from Diskette" on page 233 above:

1. Edit CONFIG.SYS on the diskette and add the following statement:

   `E:\OS2\MDOS\FSACCESS G: = A:`

2. Create the image of the boot diskette on the hard disk.

   Images may be created using the *VMDISK* utility supplied with OS/2 Version 2.0. The syntax of the VMDISK command is:

   *vmdisk <source drive> <image filename>*

   For example:

   `VMDISK a: c:\bootimg\dos50.vmb`

   The image file is a complete binary "dump" of the diskette, consisting of a short header record followed by the diskette's boot sector, FAT(s), and all data clusters. Its file size corresponds to the size of the source diskette, regardless of the amount of space actually *used* on the source diskette. No compression of the image is performed.

   The diskette must have a standard DOS format (FAT, 512 byte sectors). It is not possible to create, then boot, an image of a copy-protected diskette which has a non-DOS format. It *may* be possible to boot such a diskette directly in a VDM.

   The VMDISK utility can run under either DOS or OS/2, and supports all 3½ inch (720KB, 1.44MB and 2.88MB) and 5¼ inch (360KB and 1.2MB) source diskette formats.

   Note that VMDISK works one way only; it is *not* possible to create a diskette from a VMDISK image.

3. Proceed to add an icon to the OS/2 V2.0 Workplace Shell to launch VMB. Refer to 12.2.6, "Putting the Virtual Machine Boot Session in the Workplace Shell" on page 237 on customizing the Virtual Machine Boot, in particular the *DOS_STARTUP_DRIVE* setting.

## 12.2.5  Booting from a DOS Partition

If VMB will be used regularly, the most convenient method may be to do so from a DOS partition on hard disk, rather than via diskettes or diskette images. This may be achieved by specifying *C:* as the value for *DOS_STARTUP_DRIVE* under DOS Settings. Loading DOS from a DOS partition proceeds more quickly and offers the user a more "familiar" working environment. It is also easier to apply DOS Corrective Service to a disk partition than to diskettes or images.

Note that this method is different from that of a single hard disk partition with the Dual Boot feature available under previous versions of OS/2.

In order to load DOS from a DOS partition, the following requirements must be satisfied:

1. Boot Manager must be installed

2. DOS must be installed on a primary partition on the first hard disk in the machine

3. OS/2 Version 2.0 must be installed on an extended partition on the first fixed disk, or on another hard disk.

This will require re-partitioning on single-drive systems if the disk initially contains DOS alone, or earlier versions of OS/2.

Loading DOS from a DOS partition presents one significant problem. The DOS partition is itself bootable directly via Boot Manager, should the user so choose, and there may a requirement to boot this DOS partition directly on occasions. OS/2 Version 2.0 provides stub device drivers for functions such as EMS, XMS and mouse support in the VMB session, and these must be used in place of the normal DOS device drivers when DOS is booted in a VMB session. Since the same CONFIG.SYS and AUTOEXEC.BAT in the C: root directory is used, the question arises of which drivers should be specified for functions such as EMS and XMS support:

- If the partition is booted via VMB, the DOS drivers are inappropriate

- If the partition is booted directly via Boot Manager, the OS/2 stub drivers are inappropriate.

It might appear that the user would have to maintain multiple configuration files and rename or copy them depending upon whether the partition was being booted into a VMB session or directly from Boot Manager. This is clearly unsatisfactory. Fortunately there is a solution which avoids this. The key is to specify *both* sets of drivers, in the correct order, in CONFIG.SYS and AUTOEXEC.BAT.

The following example assumes:

- DOS 5.0 is installed on the C: Primary partition
- OS/2 Version 2.0 is installed on the E: Extended partition

CONFIG.SYS on the C: drive contains:

```
device=c:\dos\setver.exe
device=c:\dos\himem.sys
device=c:\dos\emm386.exe noems
device=e:\os2\mdos\himem.sys
device=e:\os2\mdos\emm386.sys
dos=high,umb
    ... etc ...
```

When this file is processed in an OS/2 VMB session, the DOS HIMEM load fails as it sees no available extended memory. EMM386.EXE cannot load as it sees protect-mode software already running. Then, the OS/2 HIMEM and EMM386 stub device drivers load as normal.

When this file is processed as part of a native DOS boot, the DOS HIMEM and EMM386 load as normal, but the OS/2 stub device drivers detect that they are not running under OS/2 and do not load themselves.

A similar technique works for mouse support in AUTOEXEC.BAT:

```
@echo off
prompt $p$g
set path=c:\dos
LH e:\os2\mdos\mouse
LH c:\dos\mouse
    ... etc ...
```

Note that here the OS/2 driver is listed first. When this file is processed in an OS/2 VMB session, the OS/2 stub loads first. The DOS mouse driver sees that a mouse driver is already present, and hence it does not install itself. When booting DOS natively, the OS/2 mouse stub device driver detects that it is not running under OS/2, and does not load itself. The DOS mouse driver then loads as normal.

## 12.2.6 Putting the Virtual Machine Boot Session in the Workplace Shell

We are now ready to put the VMB session as an object on the OS/2 Version 2.0 Workplace Shell desktop.

1. Locate the *Templates* folder. It is usually an icon on the Workplace Shell.

2. Open the *Templates* folder.

3. Locate the *Program* icon template.

4. Drag the *Program* icon template on to the desktop. This will cause the *Program Settings* notebook to appear.

5. Enter an asterisk "*" in the *Path and file name* field.

   See the example as shown in Figure 53.
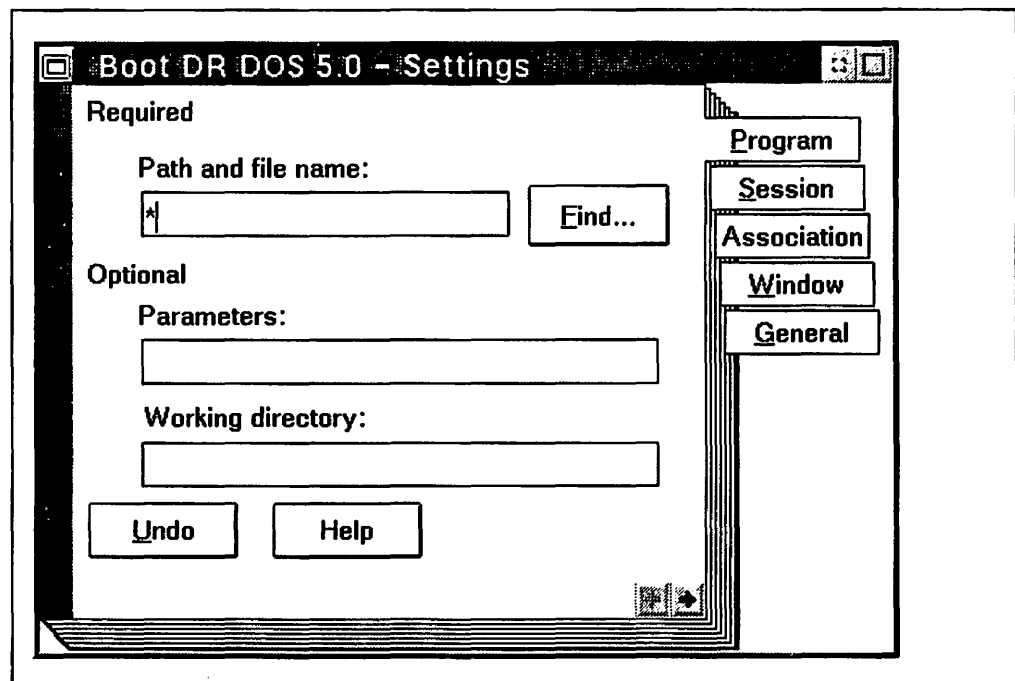


*Figure 53. The Program Page of the Settings Notebook for a VMB. All that is needed in the Path and file name field is an asterisk.*

6. Select the *Session* notebook tab.

The *Session* notebook allows the user to specify the session type and DOS Settings for the VMB session.

7. Select either *DOS full screen* or *DOS window* and then select the *DOS Settings...* button.

8. Select the *DOS_STARTUP_DRIVE* list item.

   The difference between a VMB session and a "normal" VDM is that the *DOS Settings* value of *DOS_STARTUP_DRIVE* is set. This setting determines the location of the DOS kernel to be booted. By default, MVDM's DOS Emulation is loaded. However, the user may specify a location from which to load DOS, in which case the version of DOS residing at that location is loaded.
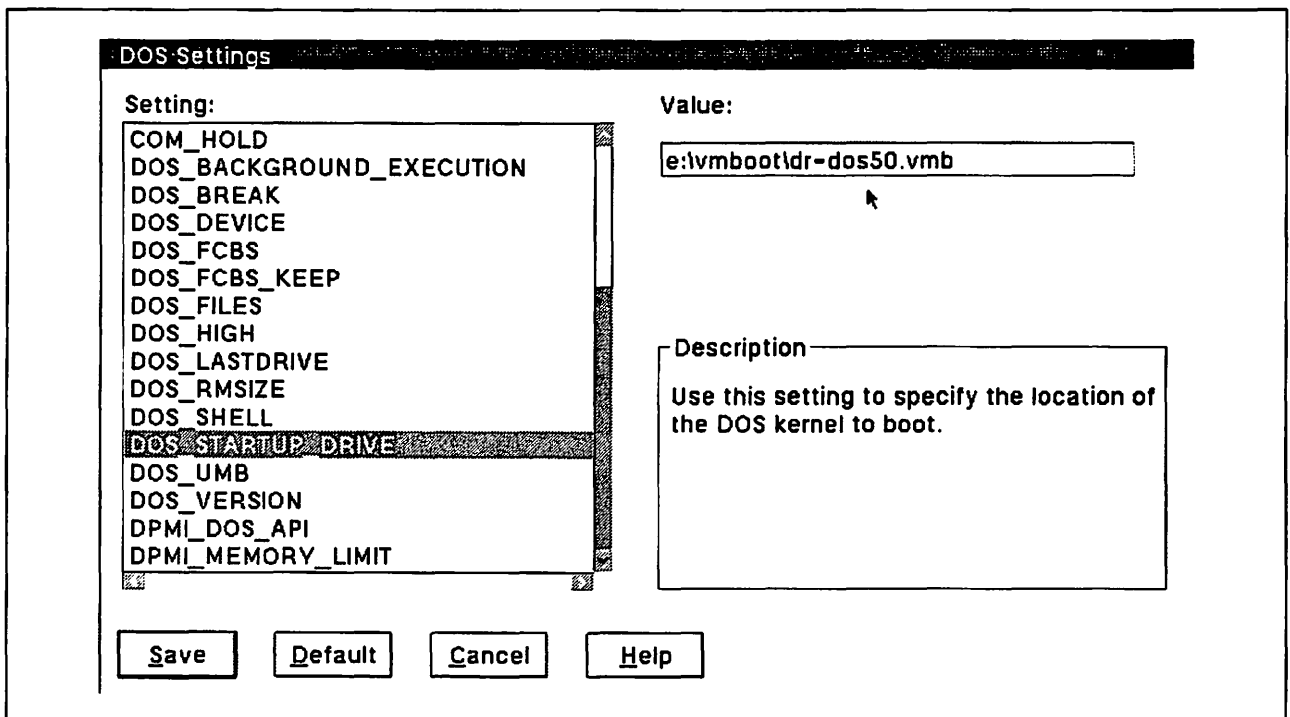


*Figure 54. DOS Settings - DOS_STARTUP_DRIVE.* This illustration shows the specification of a DOS diskette image named DR-DOS50.VMB, located on hard disk.

Example values for *DOS startup drive* are:

| **DOS Start up setting** | **Meaning** |
| --- | --- |
| **a:** | Boot using the diskette in drive A: |
| **c:\bootimg\dos50.vmb** | Boot using the specified DOS image file |
| **c:** | Boot using the primary partition of the C: drive |

The following restrictions apply:

- A second diskette drive (B:) or hard disk (D:) *cannot* be specified as the startup drive.

- To boot DOS from the C: partition, Boot Manager must be installed, and OS/2 Version 2.0 must reside in an extended partition on the first hard disk, or on another hard disk. See 12.2.5, "Booting from a DOS Partition" on page 235 .

9. Select the *Save* button to save the DOS settings and return.

10. Select the *General* notebook tab.

11. Change the *Title* field to your own name describing the new object.

12. Close the *Settings* notebook by double clicking on the system menu in the upper left corner of the window.

### 12.2.6.1 Other DOS Settings

DOS settings which control the VDM hardware environment are applicable to the VMB session and operate in the same way as for a DOS Emulation windowed or full-screen session. Those which modify the virtual DOS environment are ignored; the equivalent settings are instead determined by the CONFIG.SYS file of the booted DOS kernel. Ignored settings include:

- DOS_BREAK
- DOS_DEVICES
- DOS_UMB
- DOS_SHELL
- DOS_HIGH
- DOS_LASTDRIVE
- DOS_VERSION

The FCB limit is the lesser of either the booted DOS, or OS/2 Version 2.0 CONFIG.SYS value. The VMB session will by default have 640KB of real memory, mou se support, 2MB Expanded (EMS) memory, 2MB DPMI, and 2MB XMS memory.

### 12.2.6.2 Booting from an OS/2 V2.0 Program

By using *DosStartSession* it is possible to start a VMB session from an OS/2 V2.0 program. For more details see the following sample which shows how to boot from the disk drive A:. Of course, by changing *startd.Environment*, this sample can also be used to start a VMB from another hard disk partition or a boot image file from your hard disk.

```
/*
 *  BOOTA:  A simple program to start a DOS Boot session under OS/2 2.0.
 *          This program can be run from an OS/2 command prompt and it
 *          then starts to Boot DOS from the A: drive.
 *
 *  Last Modfied: 04/02/92
 *
 *  Author: Stacey Barnes
 *  Modified: Jeff Muir
 */

#define INCL_DOSSESMGR
#define INCL_DOSMISC
#include <os2.h>

/* messages used by BOOTA */
PSZ pBootAMsg = "BOOTA: Booting DOS from A: Drive.\r\n";
PSZ pBootSuccess = "Session started.\r\n";
PSZ pBootFailure = "Session could not be started.\r\n";

STARTDATA startd;              /* Session start information */
USHORT    SessionID, ProcessID;    /* Session and Process ID for new session*/

void main(void)
{
  USHORT        rc;

  /* Print header message */
  DosPutMessage(1,strlen(pBootAMsg),pBootAMsg);

  /* Init fields to Boot from A: drive */
  startd.Length              = sizeof(STARTDATA);
  startd.Related             = SSF_RELATED_INDEPENDENT;
  startd.FgBg                = SSF_FGBG_FORE;
  startd.TraceOpt            = SSF_TRACEOPT_NONE;
  startd.PgmTitle            = "Boot A: Drive";
  startd.PgmName             = NULL;
  startd.PgmInputs           = NULL;
  startd.TermQ               = NULL;
  startd.Environment         = "DOS_STARTUP_DRIVE=A:\0";
  startd.InheritOpt          = SSF_INHERTOPT_PARENT;
  startd.SessionType         = SSF_TYPE_VDM;

  /* Start the DOS Boot Session */
  rc = DosStartSession( &startd, &SessionID, &ProcessID );

  /* Print out either Success or Failure message */
  if(rc)
    DosPutMessage(1,strlen(pBootFailure),pBootFailure);
  else
    DosPutMessage(1,strlen(pBootSuccess),pBootSuccess);

  return;
}
```

*Figure 55. VMB from an OS/2 V2.0 Program.  This sample shows how to start a VMB from a DOS diskette by using an OS/2 V2.0 program.*

## 12.3  Managing the VMB Session

The user may interact with a VMB session in a similar way to any other virtual DOS machine.  The session may be scaled, minimized, maximized and switched between windowed and full-screen mode, and is subject to the same graphics mode limitations when windowed. However, a VMB session *cannot* be ended by typing EXIT at its command prompt. The session can *only* be closed from its system icon or the Window List.

Note that Ctrl-Alt-Del will reboot *the entire OS/2 Version 2.0 operating system*, not just the foreground virtual machine session.

## 12.4 VMB Limitations

VMB does not support:

- VCPI and other non-DPMI DOS extenders
- I/O to disk which bypasses the file system
- Feature adapter sharing without a virtual device driver
- Real-time or timing-critical DOS applications
- Some copy-protection schemes.

These limitations arise in the most part from the limitations of the MVDM environment, which are imposed in order to protect overall system integrity.

## 12.5 Summary

The DOS Emulation kernel which is normally used to support the execution of DOS applications in a VDM is exactly what its name implies; an emulation of the DOS operating system and associated services. While this suffices for most DOS applications, certain applications exist which take advantage of unique and/or undocumented features which exist only in specific versions of DOS.

To allow such applications to be successfully run under OS/2 Version 2.0, the VMB (Virtual Machine Boot) feature is provided. This feature allows a "real" DOS operating system, or indeed most other 8086 operating systems, to be booted in a virtual DOS machine. The operating system may be booted from either a diskette in drive A:, a diskette image on a hard disk, or a partition on a hard disk.

Applications which run using Virtual Machine Boot may take advantage of the EMS, XMS and mouse support provided to virtual DOS machines by OS/2 Version 2.0. This support is provided through stub device drivers supplied with OS/2 Version 2.0; these should be used in preference to the device drivers supplied with the operating system or applications.

# Appendix A. Running Personal Communications/3270 Version 2 for Windows

Personal Communications 3270 Version 2 for Windows offers a high-function 3270 emulator for the Windows environment. It supports a variety of host connections, including DFT, LAN via IEEE 802.2 protocol, LAN via NETBIOS and SDLC. It is possible to run this 3270 emulator in an OS/2 V2.0 "seamless" WIN-OS/2 VDM.
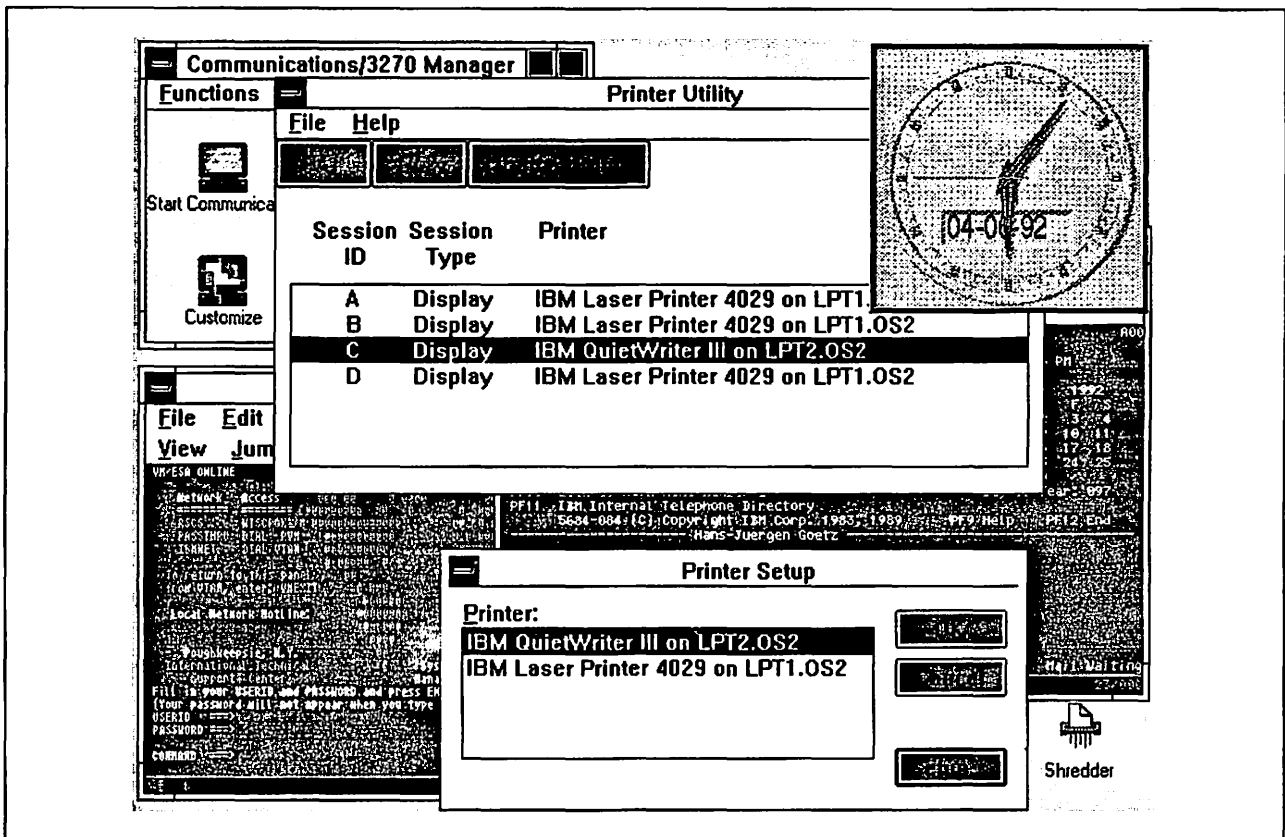


*Figure 56. Personal Communications/3270 for Windows running under OS/2 V2.0. In this picture it is using the Token-Ring LAN gateway and running as a "seamless" WIN-OS/2 application on the Workplace Shell desktop.*

We will describe below the procedure for installing and running Personal Communications/3270 for Windows for a host connection via a LAN using the IEEE 802.2 protocol. We will also describe how to install any Corrective Service Diskettes for this emulator package.

## A.1 Installing PC/3270 under WIN-OS/2

You must have installed OS/2 Version 2.0, including the WIN-OS/2 support in order for this to work.

1. From the OS/2 Desktop:

   - Open the OS/2 System Folder
   - Open the Command Prompts Folder
   - Select WIN-OS/2 full-screen

   This will start up the WIN-OS/2 environment. You will get the WIN-OS/2 Program Manager screen, just as you would if you had started Windows

natively. From here the installation of the product and the corrective service is just as it would be under Windows.

2. From the Program Manager Menu Bar:

   - Select File
   - Select Run
   - Insert PC/3270 Windows Diskette 1 in the A: drive
   - On the command line enter: **A:INSTALL**

   Now you will fill out the installation and configuration screens just as you would have installing PC/3270 directly under Windows.

   In this sample installation we will use the following throughout this part of the document:

   ```
   C:          is the drive where OS/2 Version 2.0 is installed
   C:\PC3270W  is the drive and subdirectory where PC/3270 is installed
   PC3270W     is the name of the configuration file we create
   ```

3. Select OK on the PC/3270 Installation logo screen.

4. On the Installation Start screen:

   - Select Create New Configuration file
   - Select OK

5. On the Customize Configuration screen:

   - Select Connection type.

     Our sample will use DFT, but you can select the one you want. We have tested DFT, LAN 802.2, SDLC and IIN Async at 9600bps.

     If you select other than DFT, you will need to configure your communications parameters before you can continue. You will need to refer to other documentation for this configuration information.

   - Select 2 for Number of Host sessions.
     Yours will probably vary, but start simple.

   - Select OK.

6. On the Installation End screen:

   - Enter a Configuration File name of **PC3270W**
   - Select Copy Necessary Files only (or if you want: All files)
   - Enter Drive and Directory of **C:\PC3270W\**
   - Select OK

     (These are the defaults)

7. On the Add PC/3720 to Program Manager screen:

   - Select WIN-OS/2 Main in the to Group section
   - Select OK

   There will be three more pop-up screens with information about the installation completion, just select OK on each of them to complete the installation.

---
**Note**

If you are configuring 802.2 LAN installations, you will probably get a PCS121 error at the completion of the install. This is because the install process is trying to update the CONFIG.SYS file and is having problems. Just continue with installing the corrective service diskette in the next section.

---

## A.1.1 Installing the Corrective Service Diskettes

Now install the PC/3270 Corrective Service Diskette(s).

1. From the Program Manager Menu Bar:

   - Select File
   - Select Run
   - Insert PC/3270 Corrective Service Diskette in the A: drive
   - On the command line enter: **A:INSTCSD**

   You will get a pop-up telling you that the CSD will replace files in the C:\PC3270W\ directory, select OK to continue the update.

   When the CSD installation is complete you will get a pop-up telling you that it is complete, select OK.

2. Close the WIN-OS/2 full-screen session.

3. On the Program Manager menu bar:

   - Select the Title Bar Icon (upper left corner)
   - Select Close
   - Select OK on the Exit WIN-OS/2 pop-up

Be sure to read the README.TXT file on the CSD diskette. It will have additional information about the corrective service that might apply to your installation.

## A.2 Creating a PC/3270 Batch File for OS/2 V2.0

You now need to check the WIN-OS/2 initialization file and create a batch file for PC/3270. This batch file will be used in the setup of the PC/3270 desktop object later.

## A.2.1 Checking the WIN-OS/2 Initialization File

The PC/3270 Windows installation should have updated the WIN.INI file. Check the C:\OS2\MDOS\WINOS2\WIN.INI file for the following:

```
⋮
[PCS3270]
DIR=C:\PC3270W\
⋮
```

## A.2.2 Creating the PC/3270-OS/2 Batch File

We will now create a new batch file that can be used to start any of the various PC/3270 configurations. Depending on the communications link you are using, you may need to execute a PC3270W.BAT file to invoke the WIN-OS/2 environment. The other types of links invoke WIN-OS/2 directly. This batch file will check for the presence of PC3270W.BAT and use it if it exists.

Create the file **C:\PC3270W\PC3270WO.BAT**

```
@ECHO OFF
IF EXIST PC3270W.BAT GOTO TSR
WINOS2.COM PCS3270.EXE
GOTO END
:TSR
PC3270W.BAT PCS3270.EXE
:END
```

## A.2.3 Communications Manager Mouse Support (CMMOUSE)

IBM CM Mouse Support (product 5799-PNJ, RPQ P85221) provides advanced mouse support for Personal Communications/3270 in the Windows environment. CM Mouse must be started in the same VDM (Virtual DOS Machine) as PC/3270. To achieve this, the batch files used to start Windows and PC/3270 must be modified as described in the following sections.

When PC/3270 is modified as described below, CM Mouse will automatically be started when you start PC/3270. CM Mouse may display a "waiting for PC/3270 to start..." message since both CM Mouse and PC/3270 will begin running at the same time. After PC/3270 starts and the host sessions are established, CM Mouse will automatically continue its initialization (the message "reading BDF files..." will be displayed). It is suggested that you enable the automatic startup feature of PC/3270 so that the host sessions are established automatically.

### A.2.3.1 Installing CM Mouse

Install CM Mouse from any OS/2 or DOS command line as described in the CM Mouse documentation. Note that you must have installed CSD 4002 or later for PC/3270.

The following sections assume that you have installed CM Mouse in the default C:\CMMOUSE directory. If you install CM Mouse in some other directory, change the directory names as necessary.

### A.2.3.2 Modifying the PC/3270 OS/2 V2.0 Batch File

The batch file created above should be modified as follows (the file is C:\PC3270W\PC3270WO.BAT). The changes from above are shown <u>with this emphasis</u> below:

```
@ECHO OFF
IF EXIST PC3270W.BAT GOTO TSR
WINOS2.COM C:\CMMOUSE\CMMOUSE.EXE,C:\PC3270W\PCS3270.EXE
GOTO END
:TSR
PC3270W.BAT C:\PC3270W\PCS3270.EXE
:END
```

### A.2.3.3 Modifying the PC/3270 Execution Batch File

Depending on the configuration of your system, there may or may not be a PC/3270 execution batch file on your system. If there is, it is named C:\PC3270W\PC3270WX.BAT. If this file does not exist on your system then skip this section.

Modify the line of this batch file which runs the WIN-OS/2 program (the modification is shown <u>with this emphasis</u>):

```
C:\OS2\MDOS\WINOS2\WIN C:\CMMOUSE\CMMOUSE.EXE,%1 %2 %3 %4 %5 %6 %7 %8 %9
```

Note that there must be *no blank spaces* on either side of the comma character. There may be other commands in this file; do not modify them.

## A.3 Setting up PC/3270 as a WIN-OS/2 Application

Now we have PC/3270 installed and the batch and configuration files ready to go. The next step is to create an object on the desktop and set the various attributes of that object.

### A.3.1 Create a New Object on the Desktop

To create an object for PC/3270, from the desktop:

- Open the Template Folder
- Select the Program folder with the right mouse button
- Select Create Another from the pop-up
- Select OS/2 Desktop from the list of folders
- Select Create on the bottom the window

The Program-Settings folder will now open for this new object so you can set the attributes in the next section.

### A.3.2 Setting the Attributes of the PC/3270 WIN-OS/2 Object

Now we have to set the attributes of this new object so that it will start PC/3270 as a Windows application.

The following are common to all types of connections. The LAN 802.2 and 3174 Peer connections will require some additional steps covered at the end (they need some unique device drivers).

You are now on the Program Settings for this Object. This is where you need to set up all of the various attributes that will go with this object. You move around by selecting the proper *tab* on this *notebook*.

1. Select the Program tab (should be selected):

   - Enter a Path and File name of: **C:\PC3270W\PCS3270.EXE**
   - Enter a Working Directory of: **C:\PC3270W**

     The Icon should now show the PC/3270 Icon. If not then the path and/or file name is entered incorrectly.

2. Select the General tab:

   - Enter a Title of: **PC/3270 for WIN-OS/2** (or something else you want)

3. Select the Window tab:

   - Select Minimize window to desktop for the Minimized Button Behavior (this will minimize the PC/3270 Icon on the desktop instead of the minimized window viewer folder).

4. Select the Session Tab:

   - Select WIN-OS/2 window.
   - Select Separate session (this will allow PC/3270 to load required Terminate-Stay-Resident (TSR) programs even if it is not the first WIN-OS/2 session started).
   - Select WIN-OS/2 settings.

5. From the WIN-OS/2 settings screen:

   - Select and set COM_HOLD = ON (for async only).
   - Select and set DOS_HIGH = ON (allows DOS to be loaded above 640KB).

- Select and set DOS_UMB = ON (allows TSR programs to be loaded in upper memory blocks).
- Select and set IDLE_SENSITIVITY = 100 (disables the idle detection so PC/3270 will get the maximum amount of processor time).
- Select and set KBD_ALTHOME_BYPASS = ON (so PA2 will work).
- Select and set DOS_SHELL to:

  C:\OS2\MDOS\COMMAND.COM C:\OS2\MDOS /P   /C  C:\PC3270W\PC3270WO.BAT

  **Note:**  This is the batch file we created in the previous step.

- Select SAVE when complete.

6. Close the Settings window:

- Select the Title Bar Icon (small PC/3270 Icon in upper left hand corner of Settings screen), or press F10.
- Select Close to close and save these object changes.

## A.4  Additional Setup for LAN Connections

The LAN connections require some additional device drivers in order to communicate with the adapter.

```
┌─ Note ──────────────────────────────────────────────────────────────
│
│ When PC/3270 is using a LAN Adapter, then that adapter cannot be used by
│ any other program on this workstation.  At this time there is no virtual IEEE
│ 802.2 device driver available to allow adapter sharing, which means that
│ PC/3270 will have exclusive use of this adapter when it is running.
│
└──────────────────────────────────────────────────────────────────────
```

We will set up PC/3270 to use a Token-Ring adapter. You could set it up to use Ethernet, PC Network or 3174 Peer (LAN over Coax) using the same technique.

## A.4.1  Installing LAN Support Program and RESETOKN.SYS

You must install the PC LAN Support program so that you will have the proper device drivers.  You should use the COPY command to copy the device drivers from the LSP 1.2x diskette in drive A:.

```
MD C:\LSP
COPY A:\DXMA0MOD.SYS C:\LSP
COPY A:\DXMC0MOD.SYS C:\LSP
```

Additionally you should get the RESETOKN.SYS device driver and copy it into the C:\LSP directory.

```
COPY A:\RESETOKN.SYS C:\LSP
```

The RESETOKN.SYS device driver will reset the Token-Ring adapter when it is invoked, it is not required, but suggested.  This will allow you to stop and restart PC/3270 in a Token-Ring environment.

RESETOKN.SYS can be retrieved from:

- CompuServe by issuing GO IBMOS2 and downloading RESTKN.ZIP from SECTION 17, IBMFILES.
- IBM National Support Center Bulletin Board System by downloading RESTKN.ZIP.
- Internal IBM users can GET the RESTKN PACKAGE from OS2TOOLS.

## A.4.2 Updating the PC/3270 Object for LAN Device Drivers

What we will be doing is updating the WIN-OS/2 session attributes to add some DEVICE_DRIVER statements.

From the OS/2 Desktop:

- Select the PC/3270 Icon with the right mouse button.
- Select the arrow just to the right of Open.
- Select Settings.

    You are now on the Program Settings for the PC/3270 object, just like we were before when we did the majority of the setup above.

- Select the Sessions tab.
- Select WIN-OS/2 settings.
- From the WIN-OS/2 settings screen:
  - Select and Set DOS_DEVICE and enter the following in the Value window:

        ```
        C:\LSP\RESETOKN.SYS
        C:\LSP\DXMA0MOD.SYS 001
        C:\LSP\DXMC0MOD.SYS 400000010135
        C:\PC3270W\PCS802.SYS V=N
        ```

    ┌─ **Note** ─────────────────────────────────────────────

    "400000010135" is the locally administered address (LAA) for the LAN, it may be optional and different for your installation.

    └────────────────────────────────────────────────────────

  - Select SAVE when complete.
- Close the Settings window.
  - Select the Title Bar Icon (small PC/3270 Icon in upper left hand corner of Settings screen), or press F10.
  - Select Close to close and save these object changes.

## A.5 Operating PC/3270 for Windows under OS/2 V2.0

You should now have the PC/3270 for Windows Icon on the desktop and are ready to start PC/3270. Just open the object and wait for the sessions to start.

For some of the configurations you will get the Communications/3270 Manager screen, and you will have to Start Communications. If you want, you can go into Profile and set Start Automatically so that it will automatically start the sessions subsequently.

You will see that you get the A, B, etc., sessions as well as a Communications/3270 Manager session. All of the Icons look the same, but they have different titles. If you minimize them, they will go to the Minimized Window Viewer folder.

## A.5.1 A Couple of Warnings and Suggestions

- If you have an XGA or 8514/A display adapter, be sure that you had selected VGA mode for WIN-OS/2 during OS/2 installation. This is mandatory for PC/3270 for Windows to run in a "seamless" WIN-OS/2 VDM.

- Remember that the adapter is in use EXCLUSIVELY by PC/3270. This is true for all of the adapters (DFT, SDLC, LAN).

- If you included the RESETOKN.SYS mentioned earlier then you can shutdown and restart PC/3270 Token-Ring connections. This package also comes with a RESETOKN.EXE file that can be used to close the adapter so other applications can use it, if desired. You would have to invoke this after shutting down PC/3270 or it will become very upset!

- If you get message PCS232 - PCS802.SYS Module not found, you probably set up the DOS_DEVICE statements incorrectly, or forgot to install the LSP code.

- If you get message PCS234 - The Current Configuration File Does Not Include Valid TSR Information; when you click on the CM/3370 Manager Start Communication Icon, then you either forgot to update the DOS_SHELL option, or have a bad PC3270WO.BAT. The problem is that the PC3270W.BAT does exist, but was not executed (loads the TSRs).

- If you get message PCS212 - PC/3270 is installed incorrectly, you probably installed PC/3270 under Windows 3.0 running in enhanced mode. You will need to reinstall PC/3270.

- If you open the new PC/3270 object and it closes after a few seconds, then you probably have the DOS_SHELL or the PC3270 Batch file PC3270WO.BAT set up incorrectly.

- If you open the PC/3270 object a second time, and it just sits there or hangs the machine, you might not have set Separate Session set on or you did not include the RESETOKN.SYS driver.

- Sometimes OS/2 does not know when a Windows application closes. Therefore when you do a shutdown, the desktop thinks that the application is still running. When you start OS/2 the next time, it will automatically start the application again.

  There is a way around this:

  - Bring up the OS/2 Window List (Ctrl-Esc).
  - Select the line that says WIN-OS/2 and has PC/3270 listed under it.
  - Click the right mouse button.
  - Select Close, this will close all the applications, and the WIN-OS/2 environment.

  A side effect is that all Windows Applications that OS/2 thinks are still open will be officially closed. The applications will no longer have hash marks over their icons. You can now Shutdown gracefully.

  You can circumvent this effect by running Personal Communications/3270 for Windows a *separate WIN-OS/2* session.

## A.5.2 Limitations

The limitations of running Personal Communications/3270 for Windows under OS/2 V2.0 should be noted:

1. If Personal Communications/3270 for Windows is started after OS/2 V2.0 LAN requester is running, it will cause the LAN requester to fail.

2. File transfer can only be performed from the virtual DOS machine in which Personal Communications/3270 for Windows is running.

3. The adapter card used by Personal Communications/3270 for Windows for communication with the host cannot be accessed by another program. Since the 802.2 device driver is not (yet) virtualized, Personal Communications/3270 for Windows has direct access to the Token-Ring card.

No other LAN services can be made available via another program using the same card.

> **Note**
>
> A possible solution for this problem could be to install a second Token-Ring adapter. However, this will not help because the LAN Support Program will try to initialize both Token-Ring adapters, if installed. This can cause trouble for OS/2 device drivers which are using the second adapter at the same time.

4. You need to run the program RESETOKN.EXE before starting this virtual DOS machine and after closing this virtual DOS machine. This will reset the adapter and make it available to another process. If you don't use RESETOKN, the Personal Communications/3270 for Windows virtual DOS machine cannot be stopped and restarted, as the IEEE 802.2 device driver is not (yet) virtualized.

# Appendix B. Running DOS PC Support/400 in OS/2 V2.0

This appendix covers the instructions for the installation of DOS PC Support/400under OS/2 V2.0, and the restrictions for this environment.

In DOS PC Support/400, the Shared Folders device driver is a block device driver. Since the virtual DOS machine of OS/2 V2.0 does not support block device drivers, DOS PC Support/400 must be run in a Virtual Machine Boot session. We will describe below the setup of DOS PC Support/400 in a DOS 5.0 Virtual Machine Boot. The process creates a DOS 5.0 boot diskette, which is then copied as a diskette image to the hard disk. This enables the DOS PC Support/400 Virtual Machine Boot to be started from the hard disk rather than a diskette.

## B.1 Installation Preparation

The following are required before starting the installation:

1. Basic DOS PC Support/400 Installation diskette(s) V2R1 or later

2. DOS 5.0 bootable diskette, formatted with the system (/S) option

3. If using a LAN, the LAN Support Program diskette 1.22 or later

4. *DOS PC Support/400 Installation and Administration Guide* (SC41-0006).

## B.2 Installation

If DOS PC Support/400 is to use a LAN, we must first install the LAN device drivers on the DOS 5.0 diskette with the LAN Support Program diskette. This puts the DXMx0MOD.SYS drivers in the CONFIG.SYS of the DOS 5.0 diskette.

1. Run the DOS PC Support/400 installation program.

   a. During the installation, you will be asked which *"Drive your personal computer starts from?"*. This must be answered as "A" drive.

   b. When you are asked to *"Insert diskette from which you will start the personal computer in drive A."*, insert the DOS 5.0 diskette.

   Further instructions on how to use the Install program are in the guide.

2. When the DOS PC Support/400 installation program has stopped, add the following line to the **top** of the CONFIG.SYS file on the DOS 5.0 diskette:

   ```
   DEVICE=C:\OS2\MDOS\FSFILTER.SYS
   ```

   NOTE: if you have installed OS/2 V2.0 on a drive other than C:, use that drive letter instead.

   FSFILTER.SYS is a special DOS device driver that allows a Virtual Machine Boot session to access (read/write) all OS/2 V2.0 drives (both FAT and HPFS). Without this driver, the Virtual Machine Boot session can only READ from OS/2 V2.0 FAT drives.

3. Create a Virtual Machine Boot diskette image file from your DOS 5.0 diskette with the following steps:

   a. Put your DOS 5.0 diskette in drive A:

b. At a OS/2 or DOS command prompt, enter:

   VMDISK A: C:\PCSDOS50.DSK

   This will create a file that contains an image of the DOS 5.0 diskette.

4. Create a new VDM object on the Workplace Shell desktop.

   a. Locate the *Templates* folder. It is usually an icon on the Workplace Shell.

   b. Open the *Templates* folder.

   c. Locate the *Program* icon template.

   d. Drag the *Program* icon template on to the desktop. This will cause the *Program Settings* notebook to appear.

   e. Enter an asterisk "*" in the *Path and file name* field.

   f. Select the *Session* notebook tab.

   g. Select either *DOS full screen* or *DOS window* and then select the *DOS Settings...* button.

   h. Select the *DOS_STARTUP_DRIVE* list item and then enter:

   C:\PCSDOS50.DSK

   in the upper right entry field.

   i. Select the *Save* button to save the DOS settings and return.

   j. Select the *General* notebook tab.

   k. Change the *Title* field to *DOS PC Support/400* or your own name describing the new object.

   l. Close the *Settings* notebook by double clicking on the system menu in the upper left corner of the window.

   Now there will be a *DOS PC Support/400* icon on your desktop. Double click on it to bring up DOS PC Support/400.

---

## B.3 Restrictions

The following general restrictions apply to this environment:

1. Only the Basic DOS (Not Extended DOS) version of DOS PC Support/400 is supported.

2. Only V2R1 or greater versions of DOS PC Support/400 are supported.

3. Only a single DOS PC Support/400 virtual DOS machine(VDM) is supported.

4. If OS/2 Version 2.0 Extended Services is used, the OS/2 version of PC Support must be run instead of the Basic DOS version, as the device drivers used by DOS PC Support/400 requires exclusive control of any adapter used for DOS communications.

5. When a Virtual Machine Boot session is started from a diskette image file, the "A:" drive within the VDM will refer to the diskette image file. If you would like to access the physical drive "A:," OS/2 V2.0 supplies a program called FSACCESS.EXE to do this. See the online command reference for more information.

6. The default hot-key of Alt-ESC is reserved for OS/2 V2.0 and must be changed in order to have hot-key support. This can be done by using the Configure WorkStation Function (CFGWSF.EXE) program to create/change a keyboard profile.

7. For LAN connections, only the LAN Support Program Version 1.22 or later is supported.

8. If you decide to close the DOS PC Support/400 Virtual Machine Boot session while the LAN router is active, you must wait two minutes before starting up the VDM again. This time will allow the card to reset itself on the LAN. The DOS PC Support/400 VDM will appear to hang if it is restarted too soon.

9. For an SDLC router, if you decide to close the Virtual Machine Boot session while the router is active, you must stop the router or power off the modem first. Failing to do so could cause a system trap to occur.

10. The ASYNC router is **NOT** supported at this time.

# Appendix C. Running Lotus 1-2-3 in a VDM

Lotus 1-2-3 is one of the most popular DOS programs available. Many customers use one or another of the several versions on the market. Frequently, they encounter problems of insufficient RAM to process their large spreadsheets.

This appendix discusses how to set up and run Lotus 1-2-3 Release 2.3 (which can use EMS memory), and Lotus 1-2-3 Release 3.1+ (which uses DPMI memory) in a virtual DOS machine in OS/2 V2.0 to handle large spreadsheets.

## C.1 Lotus 1-2-3 Release 2.3

In order to configure EMS support for the virtual DOS machine, we must ensure that a contiguous 64KB block of RAM is available in the address range 640KB to 1MB to be used as the EMS Page Frame (Refer to Chapter 6, "Memory Extender Support" on page 93). Do the following:

1. Boot the system with the reference diskette and in *Set Configuration* take a look at the memory map.

2. Print or make a note of the memory addresses of the different hardware device drivers. For example, 3270 Connection may have an address of *0D6000H* (D6000 hexadecimal).

   If a 64KB contiguous block cannot be found the *DOS Settings* for the virtual DOS machine will have to be used to make a block available.

3. Reboot under OS/2 V2.0.

4. Open the *Templates* folder and drag a *Program* icon to the desktop.

   The *Settings* notebook should open.

5. Enter the following in the *Path and file name* field (change the path according to your installation):

   D:\123r23\123.exe

6. The *Working Directory* should be the same as the path in the *Path and file name* field.

7. Select the *Session* tab.

8. Set session type as DOS Full Screen (Window will work, but slower)

9. Open *DOS Settings*.

10. Select *DOS_UMB* and set it to OFF (default is ON)

11. Select *MEMORY_INCLUDE_REGIONS* and enter the addresses that you do not need for this virtual DOS machine. For example, the 3270 Connection card is not used by Lotus 1-2-3 Release 2.3, so the device driver address for it (D6000) can be entered. Refer to the online help for the syntax.

12. Select *EMS_MEMORY_LIMIT* and set it to accommodate the largest expected spreadsheet.

13. Select *SAVE* to save the settings.

14. Select the *General* tab and change the *Title* to "Lotus 1-2-3 Release 2.3."

15. Close the *Settings* notebook.

The Lotus 1-2-3 Release 2.3 icon should now be available for use.

## C.2 Lotus 1-2-3 Release 3.1+

The Lotus 1-2-3 Release 3.1+ install program checks to make sure it is running in true DOS. The OS/2 V2.0 virtual DOS machine *DOS Settings* allow you to create a DOS session that returns a simulated DOS value to the Lotus INSTALL.EXE and therefore fool it into thinking it has the real DOS.

1. Open the *OS/2 System* folder.

2. Find and open the *Command Prompts* folder.

3. Drag a copy of the DOS command prompt to your desktop.

4. Open the *Settings* notebook.

5. Select the *Sessions* tab.

6. Select *DOS Settings*.

7. Select *DOS_VERSION* and enter:

   INSTALL.EXE,3,30,255

   in the list box.

8. Save the settings and close the *Settings* notebook.

9. Open the new DOS command prompt session and run A:INSTALL from the prompt.

10. After installation completes, discard the icon in the shredder.

Lotus 1-2-3 Release 3.1+ is usually started in DOS with the 123.EXE program. However, 123.EXE is a FAMILY API EXE file; it can be used to start both the DOS as well as the OS/2 version. Consequently, if we try to add a program icon to the desktop to start 123.EXE, OS/2 V2.0 will detect that it can be started as an OS/2 program and gray out the option to run it in a DOS session on the *Session* page of the *Settings* notebook. You also cannot use the *Migrate Applications* utility to add 123.EXE to the desktop, as it is detected as an OS/2 program.

This problem is overcome by starting 123.EXE from a DOS batch file. We then enter the DOS batch file name in the *Path and file name* field of the *Program* page of the *Settings* notebook.

Add the Lotus 1-2-3 Release 3.1+ icon to the desktop as follows:

1. Create a 123R31.BAT file with any Editor.

   The batch file should contain the following (Adjust *123MEMSIZE* to accommodate the largest spreadsheet):

   ```
   SET DOS16M = 2
   SET 123MEMSIZE=2048
   123.EXE
   ```

2. Place this file in the Lotus 1-2-3 Release 3.1+ subdirectory.

3. Open the *Templates* folder and drag a *Program* icon to the desktop.

   The *Settings* notebook should open.

4. Enter the following in the *Path and file name* field (change the path according to your installation):

   D:\123R3\123R31.BAT

5. The *Working Directory* should be the same as the path in the *Path and file name* field.

6. Select the *Session* tab.

7. Set session type as *DOS full screen* (Window will work, but slower).

8. Select *DOS_VERSION* and enter:

   123DOS.EXE,3,30,255
   123.EXE,3,30,255
   LOTUS.EXE,3,30,255

   in the list box.

9. Save the settings and close the *Settings* notebook.

10. Select DPMI_MEMORY_LIMIT. Adjust the value to be about 2MB larger than 123MEMSIZE defined in the DOS batch file 123R31.BAT.

11. Select *SAVE* to save the settings.

12. Select the *General* tab and change the *Title* to "Lotus 1-2-3 Release 3.1+."

13. Close the *Settings* notebook.

The Lotus 1-2-3 Release 3.1+ icon should now be available for use.

# Appendix D. Memory Extender Architectures

This appendix provides an overview of the LIM EMS Version 4.0 and LIMAXMS memory extender architectures, for those readers who require an understanding of their implementation and who are not already familiar with the design of these memory extenders.

## D.1 Expanded Memory Specification (EMS)

The expanded memory specification (EMS) was initially developed by two companies, Lotus and Intel. Microsoft later joined the consortium, and the specification has since become known as LIM EMS.

A number of versions of the EMS specification have been produced. LIM EMS Version 3.0 required a 64KB window anywhere in the area between 640KB and 1MB, and provided up to 8MB of expanded memory. As more hardware adapters with their own ROM were installed, it was often difficult to find a free 64KB contiguous memory area for the mappable window.

A revised version of the EMS specification has been produced, known as LIM EMS Version 4.0. This version allows DOS applications to allocate and access up to 32MB of expanded memory in up to 255 expanded memory objects. Regions of these objects can be mapped into the 8086 address space (below 1MB) allowing DOS applications to access large address spaces at the cost of having to explicitly remap the memory that is to be accessed. Alternate page tables for quick switches among mappings, function calls with remapping, and numerous ways to save and update mappings or move or exchange memory contents are provided.

## D.1.1 EMS Overview

The *EMS Specification* is a document that describes 30 functions and many sub-functions, which are called by DOS applications using software interrupt 67h. EMS creates memory objects in expanded memory and then provides mappings such that addressing below 1MB accesses parts of these expanded memory objects. At any given time, the 8086 application can directly access only 1MB of memory, but additional expanded memory can quickly be mapped into the addressable 1MB range. In effect, parts of the 8086 address space become moving "windows" into larger virtual memory objects.

The Intel 8086/8088 processors need special EMS memory adapters and are not part of the following discussion. Special EMS memory adapters are also required for 80286 machines. While certain software-based EMS emulation packages are available, which utilize the normal extended memory area above 1MB for that purpose, those emulations are relatively slow and unstable compared to "real" EMS hardware adapters. However, neither of the two types of EMS solutions was supported under previous versions of OS/2.

# D.1.2 EMS Functions

The following is a brief summary of LIM EMS Version 4.0 functions. Note that it is a summary of the EMS specification itself, and *not* of its implementation under OS/2 Version 2.0.

### D.1.2.1 Allocating/Reallocating/Deallocating Expanded Memory

An allocation request can be made for a number of expanded memory pages and, if successful, a handle is returned. This handle is then used by the application to reallocate or deallocate memory.

### D.1.2.2 Mapping Expanded Memory

Logical pages in an object can be mapped into physical address ranges addressable by the 8086 processor. A mapping indicates the relation between EMS physical pages and <EMS Handle, EMS Logical Page> pairs that the application requires. One example would be to map an expanded memory video buffer (EMS logical pages) to the mappable window (EMS physical pages) to create a video image in expanded memory. An EMS service request can then be used to move the image from expanded memory to screen memory.

A single logical page can be mapped to multiple physical pages. This is used by programs such as Lotus 1-2-3. When a single logical page is mapped to multiple physical pages, a write to any of these physical pages writes to the same expanded memory. An application can write to one address and then read the results from another address. This feature can be used to provide independent mappings to a shared structure for multiple modules. To implement this aliasing, multiple page registers must all point to the same memory. This leads to a requirement for the memory manager to support aliasing of page table entries.

A physical page can be unmapped. Reads/writes to unmapped memory do not kill the application, but an application cannot depend on reading what it has previously written. LIM EMS specifies that a program must unmap mappable windows before allowing another program to run, in order to protect the memory mapped by the first program.

The specification does not stipulate what happens to programs that touch unmapped memory, only that the physical pages are "inaccessible for reading and writing". One possible implementation is to map unmapped pages to nonexistent physical pages (the Microsoft Windows/386 product does this). Another alternative is to map them to physical ROM. This can lead to problems, however, on some machines that cache ROM. The safest alternative is to use a single physical page of memory.

### D.1.2.3 Information Calls

An application may obtain information about the EMS resources available, current mappings, and handle usage. In a multiprogramming environment or where TSRs are loaded, however, this information may be out of date by the time it is used. For instance, an application may determine the amount of LIM memory available, but before getting the opportunity to request an allocation, a TSR may request EMS memory. The application would find less memory available than expected.

### D.1.2.4 Saving/Restoring Mappings

Several calls are available for an application to obtain data that can later be returned to EMS to restore a mapping. For some calls, this information is saved internally by the EMS driver. For others, it is returned to the application.

One type of save operation automatically saves the registers pointing to the first 64KB of the mappable window to an internal EMS buffer associated wtih an EMS handle supplied by the user. Typically, this is used by a TSR to save and restore the current mappable window by saving to an EMS buffer associated with a handle owned by the TSR. Other save operations return either complete or partial information to the application, which the application can later provide to restore memory mappings. Still other calls allow EMS the option to store register information on the application's stack. There are five different ways to save and restore registers in addition to techniques for setting the registers.

### D.1.2.5 Alternate Register Sets

This is an optional feature of EMS. Mapping can be done to any of a number of register sets. The application can then switch the active register set. The effect is similar to switching page tables under OS/2 Version 2.0. Alternate Register Sets can be protected by the first application to claim a protection key. Only the application with the key will be allowed to claim a register set or switch the active one unless no one claims the key or the process with the key permits others to use the alternate sets.

This feature is typically used by DOS extenders such as Microsoft Windows. Switching memory during a task switch can be accomplished by turning on permission for changing register sets using the permission key, switching the current register set, and turning alternate set permission off. Even when alternate register sets are not supported, save and restore operations for register set 0 are simulated with data passed to and from the application.

### D.1.2.6 DMA Register Sets

This is an optional feature of EMS. These register sets allow association of a DMA channel with a register set. All DMA on that channel is remapped through the associated DMA register set allowing EMS remapping during DMA. When this feature is not supported, remapping of register sets may be delayed until DMA completes.

### D.1.2.7 Program Execution

This function allows an application to execute a procedure or subroutine which lies in an expanded memory area not currently mapped into the 8086 address space. EMS will perform a remap to bring the procedure or routine into the 8086 address space, and pass control to the specified entry point. This may be accomplished in either of two ways:

- JUMP passes control to the specified entry point but makes no provision for return.

- CALL passes control to the specified entry point and after the application routine returns, sets up an address mapping that will be in effect when control returns to the calling routine. The return address is that of the instruction following the INT 67h service request.

This function allows applications to store code in expanded memory.

### D.1.2.8 Data Movement

Copy and exchange services provide data movement between any combination of conventional or expanded memory. The start of a region of expanded memory is indicated by handle, logical page and offset. The memory being affected need not be currently mapped into the 8086 address space. Overlapping copies succeed without corrupting data, and a return code indicates overlap to the application. Exchange operations may not overlap.

This function allows applications to conveniently move portions of expanded objects around in expanded memory, or move them to or from conventional memory, without having to first remap the objects into the 8086 address space.

### D.1.2.9 EMM Protection

Limited protection is available. The first application that requests a key can turn enable or disable access to alternate and DMA register sets. There is no protection for memory objects. Any application can determine all handles in use and perform any operations on them. Within a single EMS implementation, a badly behaved application can wreak havoc on any application using EMS.

For example, one Windows application may write over the memory of any other. This is consistent with the general lack of protection in the DOS environment, where applications have free access to the machine's physical memory.

### D.1.2.10 OS Support

On power up, an EMM implementation which supplies mappable conventional memory allocates all mappable conventional memory to handle 0 and maps it in. This is typically all memory above the memory on the system board up to 640KB. This occurs before the operating system starts up and allows programs like Windows to remap conventional memory. Programs that remap conventional memory are required to reset the mapping before returning to the operating system. EMS does not enforce this, however.

## D.2 Extended Memory Specification (XMS)

The LIMA Extended Memory Specification (XMS) Version 2.0 provides a standard interface for the use of extended memory on Intel 80286, 80386, and 80486 computers. XMS functions allow moving code and data objects into extended memory, and from extended memory to base memory.

## D.2.1 XMS Overview

LIMA XMS is a specification for an extended memory programming interface on the Intel 80286, 80386, and 80486 processors. The XMS specification is a short document offering 18 functions which are accessed through a **control function** supplied by the XMS driver. All XMS functions are executed by calling the control function, the address of which is obtained by a call to INT 2Fh. Arguments are passed in registers.

It does not specify hardware or processor speeds and does not depend on any particular operating system. (The technique for determining if XMS is present is based on the DOS interrupt vector 02Fh, but can be easily provided in any OS that supports XMS.)

XMS manages three different kinds of memory:

1. **High Memory Area (HMA)** is the first 64KB of extended memory. By activating the A20 address line, a real mode application can access memory in this region as if it were conventional memory. The HMA is exactly 65520 bytes (64KB - 16 bytes) long.

2. **Extended Memory Blocks (EMBs)** are blocks of extended memory which lie beyond the HMA. They are not accessible from real mode and serve only for data storage. Memory can be moved between extended and conventional memory by a memory move function provided by the XMS driver. Without leaving V86 mode, code cannot be executed from EMBs and they serve only for data storage. The specification offers up to 64 megabytes of extended memory, divided into as many as 255 blocks.

3. **Upper Memory Blocks (UMBs)** are regions of memory between 640KB and 1MB which may be used like conventional memory. The size and number of UMBs is dependent upon the hardware configuration. XMS provides a standard means of obtaining and using them. Once a UMB is allocated, its memory is always available, and since the memory lies in conventional memory, code may be executed in it at any time.

The major characteristics of these three types of expanded memory are summarized in Table 8.

| Table 8. Types of Expanded Memory. Memory types utilized by VDMs. | | | |
|---|---|---|---|
| Attribute | HMA | EMB | UMB |
| Is extended memory | X | X | |
| Accessible from real mode | X * | | X |
| Data may be stored | X | X | X |
| Code may be executed from real mode | X | | X |
| Interrupt handler may be stored | | | X |
| Size may be changed after initial allocation | | X | |
| Minimum size | 64KB | 1KB | 16 bytes |
| Maximum size for one | 64KB | 64MB ** | 384KB ** |
| Maximum size total | 64KB | 64MB ** | 384KB ** |
| Number available | 1 | 255 | 24KB |
| **Note:** | | | |
| 1. * Accessible only if the A20 address line is active | | | |
| 2. ** These are values implied by the specifications. Values in practice will be considerably lower. More reasonable values are 1MB for EMBs and 0KB to 32KB for UMBs. | | | |

The three different types of expanded memory are mapped into physical memory in different ways by the XMS driver, as shown in Figure 57 on page 266.
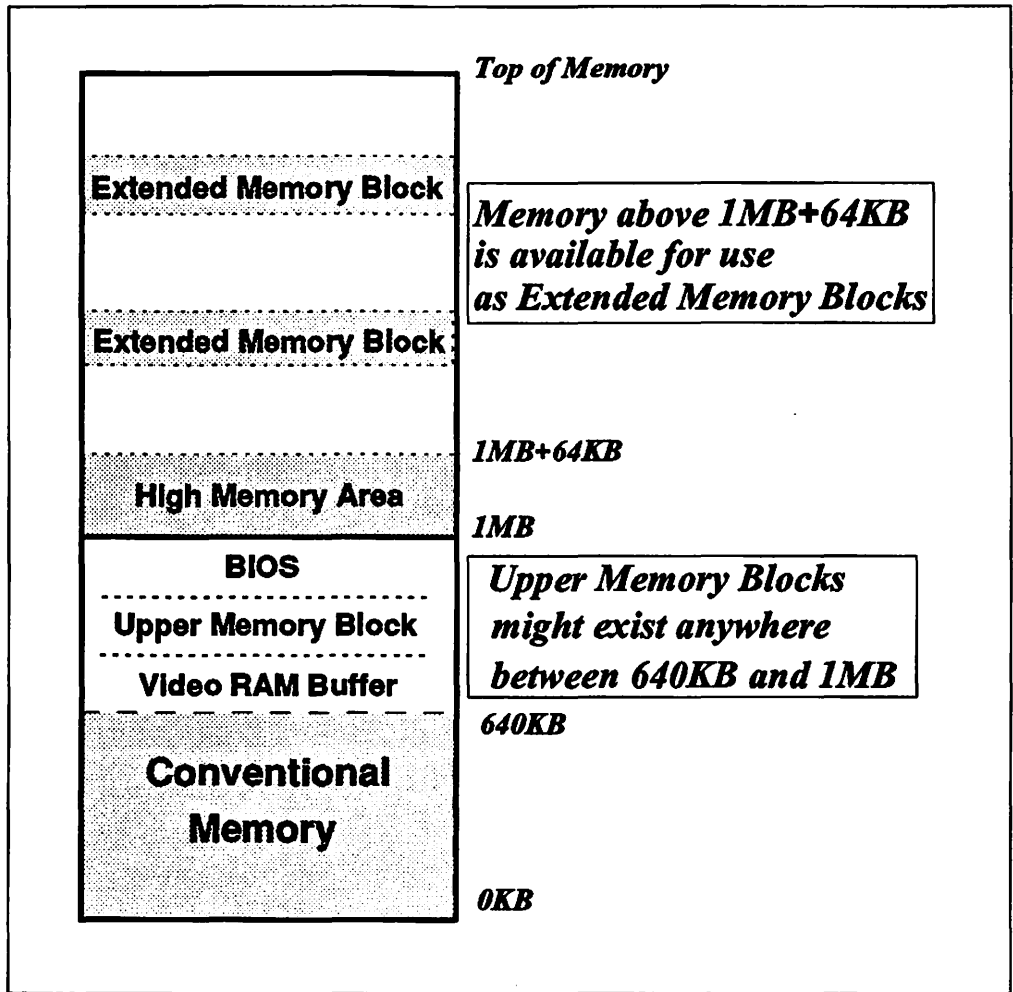
Figure 57. Memory Map of Extended Memory (HMA, UMA, and EMBs)

## D.2.2 XMS Functions

The following is a brief summary of LIMA XMA functions. This is a summary of the specification itself, and not of its implementation in OS/2 Version 2.0.

### D.2.2.1 Determining XMS Presence

Calling interrupt 2Fh with AH = 43h and AL = 00h will return AL = 80h if an XMS driver is installed. Calling interrupt 2Fh with AH = 43h and AL = 10h will return the *far* entry point address of the XMS control function in ES:BX. The control function *must* be called as a *far* procedure.

### D.2.2.2 Requesting/Releasing HMA

There is only one 64KB HMA and it cannot be divided. An application which requests the HMA is given the entire HMA, even if it uses only part of it. When an application has successfully requested the HMA, it is guaranteed sole access to it until it is released. As part of the request, the application indicates how much of the HMA it expects to use. If this value does not exceed a user-specified threshhold, the request is denied. This test is performed so that the HMA is given only to applications which make substantial use of the HMA.

### D.2.2.3 A20 Address Line Control

Two pairs of functions are used to control the status of the A20 address line. The application may control the A20 address line either globally or locally. Global control is used by programs which have control of the HMA. Local control is used by programs which need to access extended memory directly. Global settings are kept in a simple on/off flag, whereas local control uses a counter. Hence, the number of "local disable" calls must equal the number of "local enable" calls before the A20 line is actually disabled, whereas a single "global disable" call suffices to disable the A20 address line, regardless of how many "global enable" calls have been made.

### D.2.2.4 Allocating/Reallocating/Deallocating Extended Memory Blocks

An allocation request can be made for a particular-sized EMB (in kilobyte units) and, if successful, an EMB handle is returned. This handle is used to reallocate, lock, unlock, or deallocate memory. It is also used to move memory between the EMB and conventional memory or other EMBs. An EMB may be locked and while locked, it may not be reallocated or deallocated, nor may its base address change.

### D.2.2.5 Allocating/Deallocating Upper Memory Blocks

An allocation request can be made for a particular-sized UMB (in paragraph units) and, if successful, the segment number of the UMB is returned, as well as the actual size of the UMB. This segment number is also used to deallocate the UMB. UMBs may not be resized.

### D.2.2.6 Information Calls

The application can obtain information about the XMS memory resources available and handle usage. In a multiprogramming environment or where TSRs are loaded, this information may be out of date before being used. For instance, an application may determine the amount of XMS memory available, but before getting the opportunity to request an allocation, a TSR may request XMS memory. The application would find less memory available than expected.

### D.2.2.7 Data Movement

A move or copy function provides data movement between any combination of conventional or extended memory. The memory being affected need not be locked. The start of a region of extended memory is indicated by handle and offset. Overlapping copies will succeed provided the source address is below the destination address. Moreover, blocks being moved must be of even length; word alignment is not required, however. This function is the only method of accessing an extended memory block without leaving real mode.

# Appendix E. Multiple Virtual DOS Machines Lab Sessions

## E.1 Lab Exercises

These lab sessions provide practical demonstrations of OS/2 Version 2.0's Multiple Virtual DOS Machine capabilities. The individual topics that will be covered in this lab are:

- VDM Configuration
- The Virtual DOS Machine Manager
- Using the Clipboard
- VDM Use of the Speaker
- VDM Interprocess Communications

## E.2 Requirements

The following are required to do the labs:

- OS/2 Version 2.0
- DOS Version 5.0
- CLIPVIEW.EXE program, in the productivity folder
- The following programs in your C:\ITSCLABS directory:
  - ENVIRON.EXE program
  - GRAPHIC.EXE program
  - INT19.EXE program
  - QENV.BAT program
  - QCONFIG.EXE program
  - SOUND.EXE program
  - PMCHART.EXE program
  - PaintBrush program
  - WinGif program

## E.2.1 Lab Session 1: VDM Configuration

### E.2.1.1 Objective

In this exercise, students will create a new group folder and configure a VDM within that folder according to specified parameters.

First you will create a new group named *Test* on the desktop. Copy the *OS/2 Full Screen* item from the *Command Prompts* folder to the newly created folder using the **drag** mechanism.

Next, you will change the "OS/2 Full Screen" item in the folder "Test" to the following parameters:

- Change the title to "My Window"
- Match the parameters and the program type to start a DOS window.
- Deselect device driver ANSI.SYS
- 512KB DOS memory size
- Select an environment size of 200 bytes
- 1024KB expanded memory.
- 2048KB extended memory.

Finally, you will perform some checks to verify the changes.

- Use the QCONFIG.EXE program to check the memory size manipulation.

- Verify that the environment size is approximately 200 bytes.

### E.2.1.2 Steps

Create a new folder:

1. Peel off a folder from the Templates folder.

2. Rename the new folder.

3. Select an OS/2 full-screen object by single-clicking with the **left** mouse button on "OS/2 full-screen" in the Command Prompts folder.

4. Press and hold Ctrl.

5. With the mouse pointer on OS/2 full-screen press and hold the **right** mouse button. **After** pressing the mouse button release the Ctrl key. When you move the mouse, you "drag" the selected application around the desktop. As long as the mouse pointer is within an area, where you might drop the application, the mouse pointer appears as an icon. Otherwise, the mouse pointer appears as a "No Go" sign.

6. Move the mouse pointer to the client area of your new folder and release the right mouse button.

7. Bring up the "Context Menu," by clicking on it with mouse button 2.

8. Open the "Settings."

9. Change the Program Title.

10. Select "DOS Windowed" as the Program type.

11. Press push button "DOS Settings."

12. Select "DOS memory size (KB)" and change it to 512KB.

13. Append to the "DOS shell" property the following without the quotes: "/E:200."

14. Change the "EMS memory limit (KB)" to 1024KB.

15. Change the "XMS memory limit (KB)" to 2048KB.

16. Press push button "Save" to save your new DOS Settings.

17. To close that window and save your changes you must double-click on the system icon.

Now, as your "My Window" object has been updated, proceed as mentioned in the "Expected Results"

## E.2.1.3 Expected Results
After successfully completing the exercise, check the result by double-clicking on "My Window" in the folder *Test*. A VDM should start with a DOS command prompt, which should look like the following example, if you have a PROMPT statement included in the AUTOEXEC.BAT file like the one shown on *Require-ments* section of this lab.

←[37;40m[DOS] C:\>

If the DOS prompt looks like this, the ANSI.SYS is not active. In this case, you did well. Otherwise, try once more, because the ANSI.SYS driver is still active.

Now start the program QCONFIG.EXE using the following syntax:

QCONFIG /P

from within the DOS command prompt. Don't worry about the appearance of the current command prompt. The output of QCONFIG.EXE should look like this:

```
Operating System: OS/2 Standard Edition Version 2.00 CSD Level
Date & Time     : 01/14/1992  17:40:20
Product Number  : 8573-121
Model ID        : F8           Sub-Model ID    : 0B
BIOS Revision   : 00           BIOS Date       : 01/18/89
Machine Type    : IBM PS/2 Model P70
Processor       : Intel 80386          Processor Speed : 20 Mhz
Estimated Speed : 17.6 Mhz
CoProcessor     : None
Bus Type        : Micro Channel 32-Bit Bus
Keyboard Type   : Enhanced
Mouse Type      : PS/2 Mouse
Equipment       : 1 Parallel Port(s)
                : 1 Serial Port(s)
                : 1 Diskette Drive(s)
                : 1 Fixed Disk(s)
                : Pointing Device
Primary Video   : VGA
Diskette Drive 1: 3.50"  - 1.44M - 80 Track - Type 4
Fixed Disk 1    :  115 MB  =  117760 KB  =  120586240 bytes
Local - Drive C: Size   5096K =    4.9M   Avail   2636K =    2.5M
Local - Drive D: Size  23472K =   22.9M   Avail   6126K =    5.9M
Local - Drive E: Size  45956K =   44.8M   Avail  16986K =   16.5M
Local - Drive F: Size   8152K =    7.9M   Avail   7636K =    7.4M
--- MORE ---
```

```
Mouse Type      : PS/2 Mouse
Equipment       : 1 Parallel Port(s)
                : 1 Serial Port(s)
                : 1 Diskette Drive(s)
                : 1 Fixed Disk(s)
                : Pointing Device
Primary Video   : VGA
Diskette Drive 1: 3.50"  - 1.44M - 80 Track - Type 4
Fixed Disk 1    : 115 MB  = 117760 KB  =  120586240 bytes
Local - Drive C: Size    5096K =    4.9M   Avail  2636K =   2.5M
Local - Drive D: Size   23472K =   22.9M   Avail  6126K =   5.9M
Local - Drive E: Size   45956K =   44.8M   Avail 16986K =  16.5M
Local - Drive F: Size    8152K =    7.9M   Avail  7636K =   7.4M
Total Memory    : 8064 KB = 7.8 MB
Conventional    :  513 KB       Unallocated   :   476 KB
Extended Memory : 7424 KB       Unallocated   :     0 KB
Expanded Memory : 1280 KB       Unallocated   :  1024 KB
Total Slots     : 3       System (DISK)  : 1       User Slots     : 2
Expansion Slot 1:  * No Adapter Present
Expansion Slot 2: ID E1FF - IBM 3270 Connection Version B
Expansion Slot 3: ID DF9F - IBM Integrated Fixed Disk Controller
--- QCONFIG Ver 1.47 (Jan  3 1992)  by Jeff Muir  IBM Copyright 1991 (c)---

C:\>
```

1. If a base memory size of **512KB**, and an expanded (EMS) memory size of **1024KB** is reported, then you've done well.

2. Next, check the environment size with the ENVIRON.EXE and QENV.BAT files. Execute QENV.BAT shown in Figure 58 on page 273 and the following will occur:

   - The environment space is filled.
   - The ENVIRON.EXE file is executed.
   - The environment settings are shown.
   - The dummy environment settings are removed.

### E.2.1.4 Optional

There is a way to check the environment size without the use of special programs like ENVIRON.EXE and QENV.BAT. It is not simple because we cannot display the environment size by using a command.

The SET command shows all the current settings in the environment of COMMAND.COM. The environment size defaults to approximately 150 bytes.

1. To make sure that the environment buffer is full, issue some environment settings like:

   ```
   SET a=12345678901234567890
   SET b=12345678901234567890
   SET c=12345678901234567890
       :   :  =      :                :
       :   :  =      :                :
       :   :  =      :                :
   ```

2. Execute the SET commands until you encounter an error message "Out of environment space."

3. Now, save the environment settings in a file. The syntax for this action is as follows:

   ```
   SET  >  MYENV.TXT
   ```

4. Check the size of the new file (MYENV.TXT) with the DIR command:

DIR MYENV.TXT

5. From the file size displayed by the DIR command, subtract the number of lines in MYENV.TXT.

This is necessary because each line in the file is terminated by CrLF (0D0A) characters and the environment strings are terminated by a single character (hex 0) only.

If you can use this method and calculate a size at 200, you did very, very well.

```
@rem QENV.BAT
@rem Purpose :     Query DOS environment size
@rem Author  :     Bernd Westphal
@rem W10390 VDM Lab - VDM Configuration
@echo off
cls
echo Filling free environment space ....
echo.
echo Ignore any messages like "Out of environment space".
echo.
set Dummy1=Dummy.Text.to.fill.the.environment.space
set Dummy2=%Dummy1%
set Dummy3=%Dummy1%
set Dummy4=%Dummy1%
set Dummy5=%Dummy1%
cls
environ.exe
set Dummy1=
set Dummy2=
set Dummy3=
set Dummy4=
set Dummy5=
cls
echo The dummy environment settings have been removed.
```

*Figure 58. QENV.BAT Batch File*

```
/*******************************************************\
 *  Program name: ENVIRON.C                           *
 *  Created      : 5. May 1990                        *
 *  Revised      :                                    *
 *  Author       : Bernd Westphal                     *
 *  Purpose      : Get DOS environment size for VDM lab *
 *  Compile      : cl environ.c                       *
 *  Input param  : none                               *
 \*******************************************************/

#include <stdio.h>
#include <string.h>
#include <ctype.h>

void main(argc, argv, envp)
   int    argc;
   char *argv[];
   char *envp[];
{
   int    charcount = 0;                    /* # of char  */

   printf("Current environment settings:\n\n");
   printf("----------------------------\n");
   while (*envp)
   {
      printf("%s\n", *envp);
      charcount += strlen (*envp) + 1;     /* add 1 for the string terminator */
      *envp++;
   }
   printf("----------------------------\n");
   printf("\nTotal environment size is %d bytes.\n\n", charcount);
   printf("Press Enter to continue ...\n");
   getchar();
}
```

*Figure 59. C Source Code for ENVIRON.EXE*

## E.2.2  Lab Session 2: Reboot Virtual DOS Machine

### E.2.2.1  Objectives
When running DOS 3.3, 4.0, etc., if an INT 19h is called, the result is a reboot of the entire system. The objective of this lab is to show that the execution of an INT 19h in an OS/2 VDM is handled by the Virtual DOS Machine Manager (VDMM). What happens in OS/2 Version 2.0 when a program running in a VDM issues an INT 19h?

### E.2.2.2  Steps
In this exercise, you are required to start a DOS application program that issues an interrupt INT 19h.

Perform the following steps:

1. Open a DOS Windowed session.

2. Execute INT19.EXE.

3. When prompted, select ENTER to issue the INT 19h.

Keep in mind what happened so far. Then proceed with the following steps:

1. Open an OS/2 Windowed session.

2. Execute INT19.EXE.

3. When prompted, select ENTER to issue the INT 19h.

Did the results meet your expectations?

### E.2.2.3  Expected Results
After you have successfully completed the exercise, please note that the interrupt INT 19h did not reboot the system. Instead, the interrupt was routed to the Virtual DOS Machine Manager (VDMM) by the General Protection Handler. The VDMM terminates the VDM when receiving the INT 19h.

If you start a DOS application program from an OS/2 command prompt, control is passed to the Virtual DOS Machine Manager which then starts the VDM. Execution of INT 19h does **NOT** terminate the OS/2 session. Instead, INT 19h terminates the VDM session only. Control is returned to the OS/2 session.

The INT19.EXE is a compiled BASIC program. The source code is shown below.

```
'********************************************************
'*  Program name: INT19.BAS                        *
'*  Created     : 05/05/90                          *
'*  Revised     :                                    *
'*  Author      : Bernd Westphal                     *
'*  Purpose     : Execute INT 19h in an OS/2          *
'*                VDM environment. Only the current   *
'*                should be terminated.               *
'*  Compiler    : IBM BASIC Compiler/2 V1.00          *
'*  Compile     : BASCOM INT19 /O                     *
'*  Link        : LINK INT19;                         *
'*  Input param : none                                *
'********************************************************

' Variable definition for Interrupt call
TYPE RegType
    ax    AS INTEGER
    bx    AS INTEGER
    cx    AS INTEGER
    dx    AS INTEGER
    bp    AS INTEGER
    si    AS INTEGER
    di    AS INTEGER
    flags AS INTEGER
END TYPE

DECLARE SUB Interrupt (intnum AS INTEGER, inreg AS RegType, outreg AS RegType)

DIM InRegs AS RegType
DIM OutRegs AS RegType

    ' *** Program code ***
    CLS
    COLOR 15
    PRINT "OS/2 Virtual DOS Machine + INT 19h"
    PRINT STRING$(80, 196);
    PRINT
    PRINT "Execution of INT 19h under DOS on a 8086 processor"
    PRINT "will reboot the system."
    PRINT
    PRINT "To prevent a system reboot running under OS/2 Version 2.0,"
    PRINT "the Virtual DOS Machine Manager terminates the current"
    PRINT "VDM if an INT 19h occurs."
    PRINT
    PRINT "Press Enter to execute the INT 19h interrupt"
    PRINT "or press Esc to terminate."
    PRINT
    PRINT "Your choice: ";
GetChr: kb$ = INPUT$(1)
    SELECT CASE kb$
        CASE CHR$(27)
            CLS
            END

        CASE CHR$(13)
            PRINT "OK, restarting ..."
            CALL Int86(&H19, InRegs, OutRegs)

        CASE ELSE
            GOTO GetChr
    END SELECT
```

*Figure 60. INT19.BAS Source Code.  This program runs in compiled BASIC only, because the Int86 function call is not available in interpreted BASIC.*

# E.2.3 Lab Session 3 - The Clipboard Viewer

### E.2.3.1 Objective

In this exercise, you are using the clipboard support for the VDM environment. Partial and complete copying of text and graphical screen contents is the main task of this exercise as follows:

1. Fill a VDM windowed session with text data (DIR /W) and copy the screen contents to the clipboard. Use the Clipboard Viewer to check your results.

2. Create some graphics in VDM windowed session (GRAPHIC.EXE) and copy a part of the graphics to the clipboard. Again check the Clipboard Viewer.

3. Copy text (more than one line!) from the clipboard into EDLIN.

### E.2.3.2 Steps

First step:

- Start Clipboard Viewer.

- Create a VDM windowed session.

- Put some text in the VDM, for example, use the DIR /W command.

- Switch to the Presentation Manager screen group and click on the VDM's icon.

- Select "Copy all" to copy the screen contents to the clipboard.

- Check the Clipboard Viewer window which now should contain the copied text.

- Check if you can paste the content of the clipboard into the PM sample application PM Chart.

Second step:

- Start a VDM windowed session.

- In the session, execute the "GRAPHIC.EXE" program.

- Click on the system icon and select "Mark"

- Select an area within the window with the mouse pointer.

- Click on the system icon and select "Copy"

- Check the Clipboard Viewer, which now should contain the copied data.

- Check if you can paste the content of the clipboard into the PM sample application PM Chart.

Third step:

- Start the famous EDLIN editor in a VDM windowed session by entering EDLIN at the DOS command prompt.

- Enter "i1" to put EDLIN into a mode where you can enter text.

- Select "Paste" from the system or icon pull-down menu. The text is copied from the clipboard character-by-character into the input area of the editor.

### E.2.3.3 Expected Results

After you have successfully completed each exercise, the Clipboard Viewer should have presented the text or graphical material you copied from the DOS VDM.

```
'*********************************************************
'*   Program name: GRAPHIC.BAS                          *
'*   Created      : 05/14/90                             *
'*   Revised      :                                      *
'*   Author       : Bernd Westphal                       *
'*   Purpose      : Draw some graphics                   *
'*                  for VDM clipboard lab session        *
'*   Compiler     : IBM BASIC Compiler/2 V1.00           *
'*   Compile      : BASCOM GRAPHIC /O                     *
'*   Link         : LINK GRAPHIC;                        *
'*   Input param  : none                                 *
'*********************************************************

     SCREEN 2                        ' select 640 x 200 graphics mode
     CLS                             ' clear the screen
     FOR X=1 TO 640 STEP 10
        LINE (320,199)-(X,0)         ' draw some lines
     NEXT
     FOR X=1 TO 640 STEP 10
        LINE (320,0)-(X,199)         ' draw some lines
     NEXT
     LOCATE 12,31                    ' position the cursor
     PRINT SPACE$(21)                ' print 21 blanks
     LOCATE 13,31                    ' position the cursor
     PRINT " IBM ITSC Boca Raton "   ' print some text
     LOCATE 14,31                    ' position the cursor
     PRINT SPACE$(21)                ' print 21 blanks
     KB$ = INPUT$(1)                 ' check for keystroke
     SYSTEM                          ' return to DOS
```

*Figure 61. GRAPHIC.BAS Source Code*

# E.2.4 Lab Session 4: VDM Use of the Speaker

### E.2.4.1 Objective
In this exercise, you will be asked to run multiple sessions of a DOS application that accesses the speaker system. The provided SOUND.EXE program plays music. Note the behavior of the system's sessions in view of the fact that the speaker is a piece of hardware which is not virtualized.

### E.2.4.2 Steps
1. Open a VDM session.

2. Start the program in a VDM session by issuing the following command:

   SOUND

3. Repeat steps 1 and 2.

### E.2.4.3 Expected Results
After starting the first session of the SOUND.EXE program, the speaker will produce some more or less beautiful noise. Starting a new VDM session with SOUND.EXE will prevent the first session's SOUND.EXE from executing because it is switched to the background (the speaker system is not virtualized!).

The second session cannot access the speaker because it is blocked until it is granted access to the speaker.

You cannot terminate the program by pressing "Enter" because the "PLAY" instruction (refer to the program listing in Figure 62 on page 280) could not be completed.

If the first session is switched back to the foreground, it resumes execution. Pressing "Enter" ends session 1 and now session 2 has access to the speaker and begins to play the music.

```
'*****************************************************
'*  Program name: SOUND.BAS                          *
'*  Created    : 05/14/90                            *
'*  Revised    :                                     *
'*  Author     : Bernd Westphal                      *
'*  Purpose    : Access the speaker system in a      *
'*               VDM environment                     *
'*               Only 1 VDM has access to the speaker *
'*  Compiler   : IBM BASIC Compiler/2                *
'*  Compile    : BASCOM SOUND /O;                    *
'*  Link       : Link SOUND;                         *
'*  Input param : none                               *
'*****************************************************

     CLS                          ' clear the screen
     PLAY ON                      ' trap background music events
     ON PLAY(3) GOSUB PlayMusic   ' If there are less than 3 notes
                                  ' in the buffer gosub line 1000
     PRINT "Press ENTER to end."  ' display info, how to end program
     '
     PLAY "MB"                    ' background option for PLAY
     GOSUB PlayMusic              ' start the music
     '
     kb$ = ""                     ' keyboard input buffer
     WHILE kb$ = ""               ' start of loop
        LOCATE 3, 1               ' position the cursor
        COLOR c                   ' change color and print some text,
                                  ' to show, that music executes
                                  ' independent
        PRINT "Playing your favorite music ..."
        c = c + 1                 ' next color
        IF c > 15 then c = 1      ' no blinking mode
        kb$ = INKEY$              ' get a character if present
     WEND                         ' end of loop
     COLOR 7                      ' white on black
     SYSTEM                       ' return to DOS

PlayMusic:
   PLAY "t180 o2 p2 p8 L8 GGG L2 E- p24 p8 L8 FFF L2 D"
   RETURN
```

Figure 62. SOUND.BAS Source Code

# E.2.5 Lab Session 5: VDM Interprocess Communications

### E.2.5.1 Objective
The objective of this lab is to show that an OS/2 session can exchange data with a DOS session in the same system.

In this exercise, you are required to start an OS/2 application program that first creates a number of "named pipes." The OS/2 application then waits for a DOS BASIC program to connect to the pipe. This connection is performed by one thread. Afterward, the main OS/2 program sends data to change the screen colors of various (connected) DOS BASIC programs.

### E.2.5.2 Steps
Perform the following steps:

1. Start the "PIPEOS2.EXE" program within an OS/2 windowed session.

   Make certain that you use a numeric parameter to specify the number of pipes for PIPEOS2.EXE to create.
2. When prompted, start a DOS windowed session.
3. Run the BASIC program called "PIPEDOS.BAS."

   Type:

   BASICA PIPEDOS

   at the command line prompt.
4. Return to the OS/2 Windowed session.
5. Enter the letter that corresponds to the color you want the DOS window to display.

   To end the DOS BASIC program, send a "Q" from the OS/2 session.

   To end the OS/2 session, enter null.

### E.2.5.3 Expected Results
The DOS session should have been able to connect to a single or many named pipe(s) that were created and maintained by the OS/2 session. Afterward, data was passed to the DOS session(s) in the form of single characters that altered the color of the DOS screen.

After you have successfully completed the exercise, please note that this is only one way of communicating between DOS sessions and OS/2 sessions.

The source code is included here with this exercise.

## E.2.5.4 Source Code PIPEDOS.BAS

```
01' *****************************************************/
02' *****************************************************/
03' ***                                           ***/
04' ***   Program name: PIPEDOS.BAS                ***/
05' ***                                           ***/
06' ***   Created      : 05/18/90                  ***/
07' ***                                           ***/
08' ***   Revised      :                           ***/
09' ***                                           ***/
10' ***   Author       : Tim Sennitt                ***/
11' ***                                           ***/
12' ***   Purpose      : To demonstrate the use of a named ***/
13' ***                  pipe to communicate to an OS/2     ***/
14' ***                  session.                   ***/
15' ***   Compile      : none                       ***/
16' ***                                           ***/
17' ***   Input param : none                       ***/
18' ***                                           ***/
22' *****************************************************/
23' *****************************************************/
30 CLS:KEY OFF
40 COLOR 7,0
50 OPEN "r",1,"\PIPE\TIMSP",1
60 FIELD 1,1 AS A$
70 GET 1
80 IF A$="B" OR A$="b" THEN BKGRND =  9:GOTO 90    ' Blue
81 IF A$="C" OR A$="c" THEN BKGRND =  3:GOTO 90    ' Cyan
82 IF A$="G" OR A$="g" THEN BKGRND = 10:GOTO 90    ' Green
83 IF A$="P" OR A$="p" THEN BKGRND =  5:GOTO 90    ' Purple
84 IF A$="R" OR A$="r" THEN BKGRND = 12:GOTO 90    ' Red
85 IF A$="W" OR A$="w" THEN BKGRND =  7:GOTO 90    ' White
86 IF A$="Y" OR A$="y" THEN BKGRND =  6:GOTO 90    ' Yellow
87 IF A$="Q" OR A$="q" THEN SYSTEM                 ' exit system
90 COLOR 0,BKGRND:CLS:GOTO 70
```

## E.2.5.5 Source Code PIPEOS2.C

```
/****************************************************/
/****************************************************/
/***                                            ***/
/*** Program name: PIPEOS2.C                     ***/
/***                                            ***/
/*** Created     : 16th May 1990                ***/
/***                                            ***/
/*** Revised     : 26th February 1992           ***/
/***                                            ***/
/*** Author      : Tim Sennitt, Dorle Hecker     ***/
/***                                            ***/
/*** Purpose     : To demonstrate the use of an OS/2 ***/
/***               created named pipe connecting to  ***/
/***               many DOS sessions            ***/
/***                                            ***/
/*** Compile     : icc /O+ pipeos2.c            ***/
/*** or          : cl386 pipeos2.c              ***/
/***                                            ***/
/*** Input param : A number between 1 and 255   ***/
/***               (number of pipe instances)   ***/
/*** .                                          ***/
/****************************************************/

/****************************************************/
/*** DEFINES                                    ***/
/****************************************************/
#define INCL_DOS
#define INCL_DOSNMPIPES
/****************************************************/
/*** INCLUDES and VARIABLES                     ***/
/****************************************************/
#include <os2.h>
#include <stdlib.h>
#include <string.h>

#ifdef __IBMC__
   void    _System NewThread(ULONG threadArg);
#else
   void    NewThread(ULONG threadArg);
#endif
   TID     threadID[512];
   HPIPE   piphand[255];
   ULONG   threadArg, threadFlags, stack_size;
   ULONG   outbuffer, inbuffer, timeout, BytesWrit;
   USHORT  rc, loopsize, i;
   char    prep_string[11];
/****************************************************/
/*** MAIN PROGRAM                               ***/
/****************************************************/
main(argc, argv, envp)
int argc;
char *argv[];
char *envp[];
{
   BOOL fEnd_Correct=FALSE;
   threadFlags = 0;        /* start thread immediatly  */
   stack_size = 1024;      /* give stack size in bytes */
   threadArg  = 1;

   if ( argc != 2 )] (loopsize = atoi(argv[1])) == 0 )
     { printf("You have not specified the correct bacon size !\n");
       printf("The syntax is PIPEOS2 NNNN (where NNNN is a \
number between 1 and255)\n");
       exit(0);
     } /*end-if*/
   for (i = 1; i < loopsize+1; i++)
     {
       rc = DosCreateThread(&threadID[i], NewThread, i,
                            threadFlags, stack_size);
       if (rc != 0)
         { printf("DosCreateThread error = %d\n",rc);
           exit (1);
         } /*end-if*/
       printf("Pipe-Thread number %d created\n",i);
       printf("Please start the DOS client\n");
     } /*end-for*/

   printf("Now lets send some data to it......\n");

   /****************************************************/
   /* At this point we will loop getting keyboard input      */
```

```
   /* and writing this to our named pipe (until we enter null)   */
   /****************************************************/
   printf("ENTER\n [B]lue, [C]yan, [G]reen, [P]urple, \
[R]ed, [W]hite, [Y]ellow or [Q]uit\n");
   gets(prep_string);
   while (prep_string[0] != 0)
     {
       if (prep_string[0] == 'q' )] prep_string[0] == 'Q')
         { for (i = 1; i < loopsize+1; i++)
             { rc = DosWrite(piphand[i],
                             (PVOID)prep_string,
                             strlen(prep_string),
                             &BytesWrit);
               if (rc !=0) printf("Return code from DosWrite=%d\n",rc);
             } /* end-for */
           prep_string[0]=0; fEnd_Correct=TRUE;
         }
       else
         { for (i = 1; i < loopsize+1; i++)
             {
               rc = DosWrite(piphand[i],
                             (PVOID)prep_string,
                             strlen(prep_string),
                             &BytesWrit);
               if (rc !=0) printf("Return code from DosWrite=%d\n",rc);
             } /* end-for */
           printf("ENTER\n [B]lue, [C]yan, [G]reen, [P]urple, \
[R]ed, [W]hite, [Y]ellow or [Q]uit\n");
           gets(prep_string);

         } /* endif */
     } /* endwhile */

   if (!fEnd_Correct)
     { prep_string[0]='q';
       rc = DosWrite(piphand[i],
                     (PVOID)prep_string,
                     strlen(prep_string),
                     &BytesWrit);
       if (rc !=0) printf("Return code from DosWrite=%d\n",rc);
     }
   exit(0);
}

/****************************************************/
/* This is our thread process which creates N named pipes then */
/* waits for the DOS sessions to connect to them.   */
/****************************************************/
void NewThread(ULONG threadArg)
{

outbuffer = 25;
inbuffer  = 25;
timeout   = 200;

   rc = DosCreateNPipe("\\PIPE\\TIMSP\0",     /* create pipe   */
                       &piphand[threadArg],
                       0x4001,
                       0x0008,
                       outbuffer,
                       inbuffer,
                       timeout);
   if (rc != 0)
     { DosBeep(300,200); DosBeep(100,200);
       exit(1);
     }
   DosBeep(300,200); DosBeep(600,200);

   /****************************************************/
   /* now we wait for our DOS session to connect to us      */
   /****************************************************/
   rc = DosConnectNPipe(piphand[threadArg]);
   if (rc != 0)
     { DosBeep(100,200);
       exit(1);
     }
   DosBeep(600,200);
   printf("DOS Session number %d connected\n",threadArg);
}
```

## E.2.6 Lab Session 6: VDM Boot

### E.2.6.1 Objective

In this exercise, the student will create a new **DOS Full Screen** item in the command prompts folder. This new DOS session will be configured so as to boot a "shrink wrap" version of DOS 5.0 instead of utilizing OS/2 2.0s emulated version of DOS.

In order to boot an 8086 kernel into a VDM, that kernel's boot record must be obtained from either a bootable diskette, an image file of that diskette, or a DOS hard disk partition.

The student will be required to configure a VDM which can, in turn, boot DOS from any of these sources.

### E.2.6.2 Steps

1. Ask your instructor for a bootable DOS 5.0 diskette.

2. Create an image of this diskette, using the VMDISKS utility.

3. Copy a DOS session object onto the desktop.

4. Change the "DOS_STARTUP Drive" setting for it.

5. Try to boot this image.

6. Create another DOS object, using the C:\ITSCLABS\IBMDOS33.VMB image.

7. Boot this one as well.

8. Study the CONFIG.SYS of these DOS diskettes.

9. Check things like HPFS, XMS, EMS, mouse, etc.

### E.2.6.3 Expected Results

The student should learn:

• That different versions of DOS can be booted from a VDM.

• That these versions can all run parallel.

• That these VMBOOT sessions have access to all OS/2 resources.

# E.2.7 Lab Session 7: Windows Clipboard

### E.2.7.1 Objective

In this exercise, the student will get a feeling for the different clipboard environments and data formats.

### E.2.7.2 Steps

1. Repeat the steps from lab session 3.

2. Start a Windows session.

3. Check the content of the Windows clipboard view utility.

4. Check the content of the PM clipboard view utility.

5. Issue the following commands:

   ```
   COPY C:\ITSCLABS\PBR*.* C:\OS2\MDOS\WINOS2
   COPY C:\ITSCLABS\WING*.* C:\OS2\MDOS\WINOS2
   ```

6. Register the Windows programs *PaintBrush* and *WinGif* under the Windows Program Manager.

7. Start these two programs.

8. You can also install the same program directly under the Workplace Shell.

9. Try to start them as SAVDM, versus MAVDM, and check out the differences.

10. Try to modify the pasted data and copy it back into the clipboard.

11. Try to do the same with metafiles.

12. Try to do the same with text files.

13. Try to pass data from Windows to PM.

14. Start a second Windows Session and repeat the same steps.

15. Switch off the "public clipboards" and check if you can transfer data.

16. Check out the "import" and "export" features of the clipboard viewers.

17. Also check out the system editor, the picture view utility and some other programs, installed in your productivity folder.

### E.2.7.3 Expected Results

The student should gain a better understanding of:

- The clipboard architecture.

- The different data formats.

- Their implications.

- What can and what cannot be transferred.

- The local versus the public clipboard.

- The different capabilities and functions among different PM and Windows programs.

- Starting Windows applications from Windows versus Workplace Shell.

# Glossary

**aliased page.** Under the 80386 paged memory implementation, a physical page in memory which is referenced by two or more sets of page directory/page table entries, thereby enabling different memory addresses to reference the same physical memory location. This technique is used to implement the 64KB wraparound feature for virtual 8086 mode, and for shared memory implementation.

**ANSI.** American National Standards Institute; U.S.-based organization which defines standards for computing devices, protocols, programming languages, etc.

**API.** Application Programming Interface; term used to describe the set of functions by which an application may gain access to operating system services.

**A20 address line.** The 21st address line of 80x86 CPUs; the first 20 address lines are numbered 0 to 19. Enabling the A20 address line allows access to the HMA. Note, however, that many applications assume that the A20 address line is permanently disabled, so all programs which use the A20 address line should disable it when they terminate.

**BIOS.** Basic Input/Output System; code which controls the interface between a system and its attached devices, at the hardware level.

**bit.** A binary digit, which may be either zero or one. Bits are represented within a computing device by the presence or absence of an electrical or magnetic pulse at a particular point, indicating a one or a zero respectively.

**block device driver.** A device driver for a block device, which, like a disk drive, reads and writes blocks of data identified by some form of block address. Block devices are identified by a drive letter that DOS assigns (D:, E:, and so on).

**byte.** A logical data unit composed of eight binary digits (bits).

**CD-ROM.** Compact Disk Read-Only Memory; technology where data is stored on an optical disk for reading by a computer system equipped with an appropriate reading device. CD-ROM storage media may not be updated by the computer system, although certain implementations allow the media to be erased and re-written.

**DDE.** See Dynamic Data Exchange.

**debugging.** The process of removing "bugs" or errors from application code.

**device driver.** Code which handles the translation of generic device commands into specific commands for the required physical device and vice versa, allowing operating system interaction with physical devices attached to the system.

**DLL.** Dynamic link library; application module containing routines and/or resources, which is dynamically linked with its parent application at load time or runtime rather than during the linkage editing process. The use of DLLs enables decoupling of application routines and resources from the parent program, enhancing code independence, facilitating maintenance and reducing resident memory consumption.

**DMA.** Direct memory addressing; technique by which transfers to and from system memory are made by an independent control chip rather than by the system's main processor, thereby resulting in improved overall performance.

**DOS.** Disk operating system; generally used in reference to IBM DOS, the single-tasking 16-bit real-mode operating system designed for Intel 8086 processors, and developed by Microsoft Corporation as MS DOS in the early 1980s. IBM subsequently licensed MS DOS for use on IBM Personal Computer and Personal System/2 machines, and has since undertaken joint development of later versions of the operating system in conjunction with Microsoft.

**DOSEM.** See DOS Emulation.

**DOS Emulation.** Subcomponent of the Multiple Virtual DOS Machines component of OS/2 Version 2.0, which provides emulation of DOS software services to applications running in virtual DOS machines. DOS Emulation either provides an equivalent service or invokes the appropriate protected mode service within the OS/2 Version 2.0 kernel. Also known as DOSEM.

**DOS partition.** Fixed disk partition which is formatted for the DOS operating system, typically using the FAT file system.

**DOS Protected Mode Interface.** Architecture whereby real mode applications may gain access to protected mode services in 80286, 80386 or 80486 systems, by acting as "clients" and making requests to a protected mode "server" task. DPMI is typically used to enable DOS applications and DOS extenders to function correctly in a multitasking, protected mode environment.

**DOS Settings.** Function provided by the Multiple Virtual DOS Machines component of OS/2 Version 2.0 which allows a user to customize the behavior of a

virtual DOS machine to suit the application running in that virtual DOS machine. Settings may be configured once by the user and saved, or applications may provide their own configuration information which is used by the virtual DOS machine upon startup.

**DPMI.** See DOS Protected Mode Interface.

**dynamic data exchange.** Interprocess communication protocol used by applications to define dynamic links. Information updated in one application is automatically reflected in other applications linked to the first application via DDE. DDE is supported under OS/2 Version 2.0 between Windows applications, between Presentation Manager applications, and between a Windows application and a Presentation Manager application.

**EMM.** See Expanded Memory Manager.

**EMS.** Expanded Memory Specification; term used to describe the standard developed by Lotus, Intel and Microsoft for access to expanded memory by real mode 80x86 applications.

**EMS handle.** EMS implementations must provide for between 64 and 255 virtual memory objects that can be allocated by EMS applications. Each memory object is referred to by an associated EMS handle. A handle to an object is returned when an allocation request is granted. Allocation is in page sized units. Handles can be named, allowing sharing between processes.

**EMS logical page.** Each memory object that is allocated is divided into logical pages. To refer to a particular piece of allocated memory, an application uses a handle to point to the object and a logical page number (starting from 0) to indicate the offset into the object. Logical pages are always relative to a handle.

**EMS page.** Memory is allocated by EMS in page sized units; the standard page size is 16KB. Optionally, an implementation can also offer "raw" pages at whatever size is convenient, although 16KB pages must always be supported. The raw page size can be set to 16KB to provide only a single page size.

**EMS physical page.** Just as EMS object handles use logical pages indexed from 0, EMS Physical Segments are also numbered from 0. A particular address that can have expanded memory mapped into it can be referred to either by the address of a physical segment or by a physical page number. These are two alternate ways to refer to the same location in the 8086 addressable space. By convention, physical page 0 is the beginning of the mappable window and succeeding physical pages occupy contiguous pages in the window. Mappable physical pages in conventional memory (<640KB) follow the mappable window in this numbering scheme. Note that in EMS terminology a "physical page" does *not* mean physical

memory; it is a way to refer to a particular address below 1MB.

**EMS physical segment.** EMS works by remapping addresses in the 1MB 8086 addressable space to expanded memory. An EMS physical segment is a 16KB block of addresses in the 1MB 8086 address space, that can be mapped to part of an expanded memory allocation. When a physical segment is mapped to an EMS logical page, accessing the physical segment accesses the logical page. Logical pages can *only* be accessed through physical segments.

**EMS register set.** EMS memory mapping is accomplished by a set of registers, one for each mappable physical page. A register specifies the current associated logical page. Registers can be unmapped. EMS optionally offers multiple *alternate register sets* and *DMA register sets*. A register set is, in essence, a page table for mappable windows. Alternate register sets then, are multiple page tables, only one of which is active at any given time.

**enhanced mode.** See 386 enhanced mode.

**expanded memory.** Memory in 80x86 processors, typically on special hardware adapters, which is accessed by real mode 8086 applications using the LIM EMS specification. Up to 32MB of expanded memory are supported by EMS Version 4.0.

**Expanded Memory Manager.** Virtual device driver which provides emulation of LIM EMS Version 4.0 services to DOS applications running in virtual DOS machines. Unlike most virtual device drivers, the Expanded Memory Manager does not use a physical device driver, but interacts directly with the operating system's memory manager to map EMS memory requests into the system's linear address space. Also known as EMM.

**extended memory.** Memory in 80286, 80386, and 80486-based machines which is located above the 1MB address boundary and accessed using the LIMA XMS specification.

**extended memory block (EMB).** Under the XMS specification, a block of extended memory located at or above 1MB + 64KB, which can only be used for data storage.

**Extended Memory Manager.** Virtual device driver which provides emulation of LIMA XMS services to DOS applications running in virtual DOS machines. Unlike most virtual device drivers, the Extended Memory Manager does not use a physical device driver, but interacts directly with the operating system's memory manager to map XMS memory requests into the system's linear address space. Also known as XMM.

**FAT.** File Allocation Table; term used to describe the file system implemented by DOS and also supported by OS/2. This file system uses a file allocation table to contain the physical sector addresses of all files on the disk. The FAT file system is supported by OS/2 Version 2.0, along with the newer HPFS and other installable file systems.

**flat memory model.** Conceptual view of real memory implemented by OS/2 Version 2.0, where the operating system regards memory as a single linear address range of up to 4GB.

**FSACCESS.** Utility provided with the VMB component of OS/2 Version 2.0, which allows the user to redirect logical drive letters in a VMB session, in order to avoid clashes and ensure compatibility with drive letters used by other processes.

**FSFILTER.** Utility provided with the VMB component of OS/2 Version 2.0, which allows the operating system loaded within the VMB session to recognize and access HPFS drives and large partitions which would otherwise not be accessible by that operating system. FSFILTER traps and redirects all disk I/O through OS/2 Version 2.0's own device drivers.

**GB.** Gigabyte; 1024 megabytes, or 1024 x 1024 x 1024 bytes.

**high memory area.** The first 64KB of extended memory above 1MB. The High Memory Area (HMA) is unique in that code can be executed in it while in real mode, using the A20 address line wraparound. The HMA officially starts at FFFF:0010h and ends at FFFF:FFFFh, making it slightly less than 64KB in length.

**HIMEM.SYS.** The Extended Memory Manager in general use for DOS.

**HMA.** See high memory area.

**HPFS.** High performance file system; file system first implemented under OS/2 Version 1.2, offering enhanced performance over the original FAT file system implemented in DOS and prior versions of OS/2. HPFS is an optional installation item under OS/2 Version 2.0; the FAT system may also be used to retain compatibility with DOS.

**Interrupt.** An electrical signal generated by a device or adapter within the system, to inform the operating system that an event, such as the completion of an I/O operation, has occurred. The operating system then processes the interrupt by passing control to a particular piece of code which handles the appropriate action in response to the event indicated.

**I/O.** Input/Output; term used to collectively describe the techniques and devices through which a computer system interfaces with storage devices, external systems and the user.

**IOPL.** Input Output Privilege Level; term used in Intel 80x86 processor terminology to refer to tasks executing at privilege level 2, which have the authority to directly access physical I/O devices.

**KB.** Kilobyte; 1024 bytes.

**LIM.** Lotus-Intel-Microsoft; term used in reference to the consortium which developed the Expanded Memory Specification (EMS), which provides a standard interface for the use of expanded memory by real mode 80x86 applications.

**LIMA.** Lotus-Intel-Microsoft-AST; term used in reference to the consortium which developed the Extended Memory Specification (XMS), which provides a standard interface for the use of extended memory by real mode 80x86 applications.

**mappable window.** Under EMS, the mappable window is composed of EMS physical segments. Called the page frame in normal EMS terminology, this is the range of real mode addresses used by most applications to reference expanded memory. The window must be at least 64KB and located between 640KB and 1MB.

**MAVDM.** See Multiple Application VDM.

**MB.** Megabyte; 1024 kilobytes, or 1024 x 1024 bytes.

**Boot Manager.** Feature of OS/2 Version 2.0 which allows multiple partitions to exist on fixed disks in the same machine, with a separate operating system on each partition. At boot time, the user may select the desired operating system with which to start the machine.

**Multiple Application VDM.** Method of running Windows applications under OS/2 Version 2.0, whereby the Windows Program Manager is started in a virtual DOS machine, and Windows applications are started using the Program Manager. Errors in any Windows application may potentially result in termination of the entire VDM This is method of running Windows applications is not recommended unless applications require communication using shared memory.

**Multiple Virtual DOS Machines.** Feature of OS/2 Version 2.0 which enables multiple DOS applications to execute concurrently in fullscreen or windowed mode under OS/2 Version 2.0, in conjunction with other 16-bit or 32-bit applications, with full preemptive multitasking and memory protection between tasks. See also virtual DOS machine.

**MVDM.** See Multiple Virtual DOS Machines.

**NPX.** Numeric Processor Extension; term used in reference to the exception condition generated by the 80386 processor when an application issues a

numeric coprocessor instruction in a machine with no coprocessor installed. Note that OS/2 Version 2.0 will trap the NPX exception and emulate the numeric coprocessor function within the operating system, returning the result to the application exactly as if a physical coprocessor were installed.

**NULL.** A binary zero. In "C" programming terms, NULL is typically used to refer to a pointer which is set to the binary zero value.

**Object Linking and Embedding.** Protocol for linking multiple data formats (such as text, image, voice etc.) to create a compound document, which may be viewed and manipulated as a coherent whole.

**OLE.** See Object Linking and Embedding.

**OS mappable window.** In addition to the normal mappable window, additional memory below 1MB can also be remapped. This is typically all memory from 256KB to 640KB. This is used by task management programs (such as Microsoft Windows) to map a different section of expanded memory to this region for each application it runs. An application can, however, remap this memory. No protection is provided. This memory is automatically allocated to a special handle and is mapped when EMS starts, before the operating system touches the memory.

**page.** Granular unit for memory management using the 80386 and 80486 processors. A page is a 4KB contiguous unit of memory, which the processor manipulates as a single entity for the purpose of memory manipulation and swapping.

**physical device driver.** Protected mode device driver used by the OS/2 Version 2.0 operating system and protected mode processes to access hardware devices. DOS applications running in virtual DOS machines do not directly access physical device drivers; virtual device drivers are utilized by these applications, and the virtual device driver in turn communicates with the physical device driver.

**PIC.** Programmable Interrupt Controller; component of the 80386 processor complex which handles interrupts generated by devices within the system.

**POST.** Power-On Self-Test; code typically stored on ROM (although the IBM PS/2 Model 90 and 95 allow POST code to be stored on fixed disk) which is invoked when a machine is powered on, in order to test the hardware.

**privilege level.** In the context of the Intel 80386 processor architecture, the level of authority at which a task executes. There are four available privilege levels; under OS/2 Version 2.0, Level 0 is used for operating system kernel code; Level 1 is not used; Level 2 is used for applications which directly address I/O devices (such as communications applications); Level 3 is used for general application code. In many

publications, these protection levels are referred to as "rings," since the protection scheme of the 80386 is often depicted diagrammatically as a series of concentric circles.

**protected mode.** Mode of operation for the Intel 80286 and 80386/80486 processors, whereby the address space is expanded to 16MB (80286) or 4GB (80386/80486), and memory references are translated via segment selector and offset, enabling full memory protection between processes executing in the system. With the 80386/80486, paging is available in protected mode.

**RAM.** Random Access Memory; term used to describe memory which may be dynamically read and written by a processor or other device during system operations. RAM is typically used to store program instructions and data which not being operated upon by the processor at the current moment in time, but which are required for the logical unit of work currently being carried out.

**real mode.** (1) Default mode of operation for the Intel 80286 and 80386/80486 processors, and the only mode of operation for the 8086 processor. In real mode, the processor acts as a 16-bit device, its physical memory address space is limited to 1MB, and memory references translate directly to physical addresses. With the 80386 and 80486, paging is not supported in real mode. (2) Execution mode for Windows 3.0, which provides applications with up to 640KB of conventional memory (384KB after loading DOS, Windows and other memory-resident software) and which supports expanded memory using LIM EMS Version 4.0 specifications.

**ROM.** Read-Only Memory; term used to describe memory which may be read, but not written to, during system operations. ROM is typically used to store basic hardware initialization instructions, BIOS or self-testing code, which is required to be available prior to accessing the disk subsystem.

**SAVDM.** See Single Application VDM.

**Seamless Windows.** Method of running Windows applications under OS/2 Version 2.0, whereby the application runs on the Workplace Shell in a similar manner to a normal Presentation Manager application. The user is not normally aware that the application is a Windows application. This is the preferred method of running Windows applications, and is similar to the Single Application VDM method.

**segment.** Unit of memory addressable by the Intel 80x86 processors. With the 8086 and 80286 processors, a segment may be from 16 bytes to 64KB in size. With the 80386 and 80486 processors, a segment may be up to 4GB in size.

**segment selector.** Field which specifies the base address of a memory segment when using the seg-

mented memory model. The selector is 16 bits in length on an 80286 processor, and 32 bits in length on an 80386 or 80486 processor.

**service layer.** Executable code which performs the operating system function requested by an application using an API.

**signal.** An event occurring within the operating system which will effect the execution of the current process; for example, the user may hit the Ctrl + Break key combination, which would result in a signal being generated, instructing the operating system to terminate the current process. Signals typically override the dispatching algorithms of the operating system.

**Single Application VDM.** Method of running Windows applications under OS/2 Version 2.0, whereby a virtual DOS machine is created for each Windows application. The application runs within this VDM, and is protected from any other application running in the system. This is the recommended way to run Windows applications if not running seamlessly on the Workplace Shell desktop.

**standard mode.** Execution mode for Windows 3.0, available when running on 80286, 80386 or 80486 systems. Standard mode makes use of these processors' protected mode to provide access to up to 16MB of extended memory. Windows applications must conform to memory management rules in order to use standard mode.

**stub device driver.** Virtual device drivers include a stub so that the DOS application can detect and address the device driver. This stub device driver executes in V86 mode within each VDM, while the main portion of the virtual device driver executes in protected mode.

**stub virtual DOS kernel.** Portion of the MVDM component's DOS kernel which is loaded into each VDM. DOS services are either provided directly by the stub virtual DOS kernel, or are routed to the OS/2 protected mode kernel.

**task state segment.** Area of memory containing information relating to the current machine state, including CPU register contents, and the current software state of a task, such as file descriptors and priority information.

**TSR.** Terminate and Stay Resident; term used to describe DOS applications which modify interrupt vectors to insert their own interrupt processing code. This technique is used frequently by low-level utility programs, device drivers and other system code such as network drivers. TSR programs use memory within the DOS address space, thus leaving less memory available for application use.

**TSS.** See task state segment.

**UMB.** See upper memory block.

**upper memory block.** Under XMS, a block of memory available on some 80x86-based machines, and located between DOS's 640KB limit and the 1MB address space boundary. The number, size, and location of these blocks may vary widely depending on the types of hardware adapter cards installed in the machine.

**VDDM.** See Virtual Device Driver Manager.

**VDM.** See Virtual DOS Machine.

**virtual device driver.** Form of device driver used by DOS applications executing in a virtual DOS machine, in order to access devices which must be shared with other processes in the system, such as the screen or mouse. A virtual device driver typically maps DOS device commands to the physical device driver in the protected mode environment under OS/2 Version 2.0.

**Virtual Device Driver Manager.** Subcomponent of the Multiple Virtual DOS Machines component of OS/2 Version 2.0, which loads, initializes, and communicates with virtual device drivers. Also known as the VDDM.

**Virtual Device Helper.** Set of operating system services used by virtual device drivers to request, release and manipulate system resources.

**virtual DOS machine.** A protected mode process under OS/2 Version 2.0 which emulates a DOS operating system environment, such that DOS applications executing within the virtual machine operate exactly as if they were running under DOS. DOS virtual machines support both text and graphics applications. Virtual DOS machines make use of the virtual 8086 mode of the 80386 and 80486 processors.

**Virtual DOS Machine Manager.** Subcomponent of the Multiple Virtual DOS Machines component of OS/2 Version 2.0, which provides the ability to start and interact with DOS applications running in virtual DOS machines. The virtual DOS machine manager operates in conjunction with the Virtual Device Driver Manager.

**VEMM.** See Virtual Expanded Memory Manager.

**Virtual Expanded Memory Manager.** Virtual device driver which provides emulation of LIM EMS Version 4.0 services to DOS applications running in virtual DOS machines. Unlike most virtual device drivers, the Virtual Expanded Memory Manager does not use a physical device driver, but interacts directly with the operating system's memory manager to map EMS memory requests into the system's linear address space. Also known as VEMM.

**virtual machine.** See virtual DOS machine.

**VMB.** Component of OS/2 Version 2.0 which operates in conjunction with MVDM component to allow a real copy of DOS or another real mode operating system to be loaded into a virtual DOS machine. VMB allows the execution under OS/2 Version 2.0 of applications which exploit specific features of an operating system or version, and which cannot be supported using DOS Emulation.

**VMB session.** A virtual DOS machine in which an operating system has been loaded using the VMB component of OS/2 Version 2.0.

**virtual machine monitor.** Routine supplied by operating systems which utilize the virtual 8086 mode of the 80386 processor, to provide emulation of interrupts and exceptions resulting when an 8086 application issues an instruction which cannot be executed in virtual 8086 mode. The 8086 Emulation component of MVDM is a virtual machine monitor.

**Virtual Programmable Interrupt Controller.** Virtual device driver used in the Multiple Virtual DOS Machines component of OS/2 Version 2.0 to emulate DOS interrupt services, allowing DOS applications running in virtual DOS machines to issue and receive interrupts.

**virtual 8086 mode.** Mode of operation of the Intel 80386 and 80486 processors, which allows the processor to execute multiple concurrent tasks with each regarding the processor as its own distinct 8086 processor. This mode of operation provides full preemptive multitasking and full memory protection between the virtual 8086 tasks. Also known as V86 mode.

**VMDISK.** Utility provided with the VMB component of OS/2 Version 2.0, which allows the user to create diskette images which may then be used to load specific versions of DOS or other real mode operating systems into a virtual DOS machine using VMB.

**watchdog timer.** Hardware timer provided by the 80386 processor to ensure that 8086 applications running in virtual 8086 mode do not disable interrupts for an unnecessarily long period. Such lengthy disabling may cause unrecoverable system errors. The watchdog timer generates periodic non-maskable interrupts which allow an operating system to detect an errant 8086 application and terminate it.

**windows.** In this document, generally refers to Microsoft Windows 3.0. Exceptions are noted in the text.

**WIN-OS/2 kernel.** Portion of MVDM which services Windows function calls from applications running within a VDM. The WIN-OS/2 kernel provides support for Windows applications in VDMs under OS/2 Version 2.0.

**Workplace Shell.** Standard user interface component of OS/2 Version 2.0 that provides an object-oriented interface for the end user. The implementation of the Workplace Shell is based upon the system object model.

**Workplace Shell object.** An object created by the Workplace Shell, typically at the request of the user or an application. A Workplace Shell object is very similar in concept to an application object, in that it possesses data and methods that operate upon that data. See also application object.

**XMM.** See Extended Memory Manager.

**386 enhanced mode.** Execution mode for Windows 3.0, available when running on 80386 or 80486 systems only. 386 enhanced mode makes use of the virtual memory capabilities of the the processor, and allows use of certain 32-bit functions. Applications must conform to memory management rules in order to use 386 enhanced mode.

**80386.** Intel 80386 microprocessor; the 32-bit processor upon which the OS/2 Version 2.0 operating system is based.

**80486.** Intel 80486 microprocessor; a 32-bit processor which implements a superset of the 80386 processor instruction set.

**8086 Emulation.** Subcomponent of the Multiple Virtual DOS Machines component of OS/2 Version 2.0, which provides emulation of Intel 8086 processor services by utilizing the virtual 8086 mode of the 80386 processor.

# Index

## Special Characters
.TMP   158
"seamless" WIN-OS/2 VDM   137
  Clipboard Viewer   137
  common "seamless" WIN-OS/2 VDM   141
  PM DDE   137
  Standard Mode   137
  VDD   137
  VGA   137
  VWIN.SYS   137
"seamless" WIN-OS/2 VDM Support
  DOS Settings for Common "seamless" WIN-OS/2
  VDM   142
  Programming Interface for Common "seamless"
  WIN-OS/2 VDM   142
  Restrictions   142

## Numerics
64KB   33
8086 Emulation
  definition   6, 21
  I/O port trapping   33
  interrupt handling   32
  use of virtual 8086 mode   30
  VDM creation   20
8086 Emulation (V86)
  initialization   20

## A
A20
  wraparound   33
Adapter memory   102
Address line A20   33
Adobe Type Manager   160
Alias   24, 33
Application compatibility   2, 10, 22, 205
Application restrictions   5, 14, 105
Application support   2, 10
ASYNC router   255
ATM   160
ATM Control Panel   165
atm.system.drv   163
ATMCNTRL.EXE   163
atmsys.drv   163
ATTRIB   50
AUTOEXEC.BAT   46, 120

## B
BACKUP   50
BIOS   18, 23, 59

bitmap font   161

## C
Certified Application Database   144
Client/Server   26
clipboard   169, 198
  DOS Full Screen   198
  full screen   198
Clipboard Viewer   137
CLIPWOS2.EXE   169
COM.SYS   159
Commands   47
Common "seamless" WIN-OS/2 VDM   141
COMSPEC   230
COMx port   159
CONFIG.SYS   45, 97, 107, 110, 111, 120, 236
CONTROL.INI   152
Copy and Paste   198
copy protection   117
Creating a DOS Object   197
Creating a VDM   19, 38
Cut & Paste
  DOS Full Screen   198

## D
data flow   155
DATABASE.DAT   121
DATABASE.TXT   126
DBTAGS.DAT   123
DDE   172
DDEServer   173, 174
DEBUG   48
DELDIR   49
deleting font   165
Descriptor privilege Level   31
Device drivers   5, 7, 53, 203
  block   203
DEVICE statement   55, 56, 97, 107, 111
Device-independent pointer services   86
DEVICEHIGH statement   111
DIR   50
Disabling interrupts   32
Display Management   25
DOS 5.0 compatibility   22
DOS application capabilities   4, 10, 13, 205
DOS Compatibility Box (OS/2 V1.x)   4
DOS device drivers   111, 209
DOS Emulation
  creation   20
  definition   6, 22
  initialization   20
  standard devices   44
  system initialization   38

graphical data interface  156
graphics applications  198

# H

Hardware Interrupt Manager  32
Hardware interrupts  60
High Memory Area (HMA)  9, 110, 113, 224
   Upper Memory Area (UMA)  9
Hooking interrupt vectors  43
HPFS  229

# I

I/O port trapping  33
   line service  33
IBMNULL  157
IBMNULL.DRV  155, 159
Initializing a TSS  19
Initializing a VDM  20
Initializing the 8086 Emulation (V86)  20
Initializing the DOS Emulation  20
Installing Applications  117
Installing DOS Programs  117
Installing WIN-OS/2 Support Under OS/2 Version
  2.0  143
   Certified Application Database  143
   Selective Install  143
Installing Windows Programs  119
INT 21h services  29, 37, 42
INT 67h  94
Intel 80386 processor  1
Interprocess Communication  26
   Named Pipes  26
Interrupt handling  2, 23, 31, 55, 60
IPC  26

# K

kernel device driver
   COM.SYS  159
   PRINT0x.SYS  159
Keyboard Management  25

# L

large partitions  229
LIM EMS Version 4.0
   See EMS Version 4.0
LIMA XMS Version 2.0
   See XMS Version 2.0
LOADHIGH command  112
Logical Video Buffer  24
Lotus 1-2-3  257
   installation  257
   insufficient memory  257
Lotus 1-2-3 Release 3.1 +  258
   installation  258

Lotus 123 Release 3.1 +  118
LPTDD.SYS  66

# M

MAVDM  137
MEM  47
MEM_INCLUDE_REGIONS  102
Memory extenders  14
   LIM EMS Version 4.0  9, 94
   LIMA XMS Version 2.0  103
   XMS Version 2.0  9
Memory management  17, 38, 87
Memory protection  3, 11, 61
Memory utilization  4, 22, 23, 37, 211
Microsoft Windows support
   definition  11
   enhanced mode  11
   memory protection  11
   pre-emptive multitasking  11
   real mode  11
   standard mode  11
Migrating Applications  117
Migration
   Certified Application Database  144
   Windows systems to OS/2 Version 2.0  144
migration database  121, 122
   Customized  122
Mouse Management  26
MSCDEX  81
Multimedia Extensions  134
Multiplane graphics mode  84
Multiple Application VDM  137, 148
Multitasking  3, 11

# N

named pipes  26
network  133
new fonts  163

# O

Object Linking and Embedding  177
ObjectLink  178
outline font  161
OwnerLink  178

# P

Page alias  24, 33
Page fault exception  24, 88
PARSEDB  126
   Errors  126
PARSEDB.EXE  122
PC Support/400  203
PCSDOS50.DSK  254
Personal Communications/3270 for Windows for
  Windows  243

# Readers' Comments

**OS/2 Version 2.0**
**Volume 2: DOS and Windows Environment**

**Publication No. GG24-3731-00**

Use this form to tell us what you think about this manual. If you have found errors in it, or if you want to express your opinion about it (such as organization, subject matter, appearance) or make suggestions for improvement, this is the form to use.

To request additional publications, or to ask questions or make comments about the functions of IBM products or systems, you should talk to your IBM representative or to your IBM authorized remarketer. This form is provided for comments about the information in this manual and the way it is presented.

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you.

Be sure to print your name and address below if you would like a reply.

_____     _____
Name                                Address

_____     _____
Company or Organization

_____     _____
Phone No.

IBM®

Fold and Tape                    **Please do not staple**                    Fold and Tape

NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

# BUSINESS REPLY MAIL

FIRST CLASS MAIL    PERMIT NO. 40    ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

IBM International Technical Support Center
Department 91J, Building 235-2
Internal Zip 4423
901 NORTHWEST 51ST STREET
BOCA RATON  FL  33431-1328

Fold and Tape                    **Please do not staple**                    Fold and Tape

# Readers' Comments

**OS/2 Version 2.0**
**Volume 2: DOS and Windows Environment**

**Publication No. GG24-3731-00**

Use this form to tell us what you think about this manual. If you have found errors in it, or if you want to express your opinion about it (such as organization, subject matter, appearance) or make suggestions for improvement, this is the form to use.

To request additional publications, or to ask questions or make comments about the functions of IBM products or systems, you should talk to your IBM representative or to your IBM authorized remarketer. This form is provided for comments about the information in this manual and the way it is presented.

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you.

Be sure to print your name and address below if you would like a reply.

Name _____    Address _____

Company or Organization _____    _____

Phone No. _____    _____

Fold and Tape | **Please do not staple** | Fold and Tape

NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

# BUSINESS REPLY MAIL

FIRST CLASS MAIL    PERMIT NO. 40    ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

IBM International Technical Support Center
Department 91J, Building 235-2
Internal Zip 4423
901 NORTHWEST 51ST STREET
BOCA RATON  FL  33431-1328

Fold and Tape | **Please do not staple** | Fold and Tape

# Readers' Comments

**OS/2 Version 2.0**
**Volume 2: DOS and Windows Environment**

**Publication No. GG24-3731-00**

Use this form to tell us what you think about this manual. If you have found errors in it, or if you want to express your opinion about it (such as organization, subject matter, appearance) or make suggestions for improvement, this is the form to use.

To request additional publications, or to ask questions or make comments about the functions of IBM products or systems, you should talk to your IBM representative or to your IBM authorized remarketer. This form is provided for comments about the information in this manual and the way it is presented.

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you.

Be sure to print your name and address below if you would like a reply.

.

.

.

---
Name

---
Address

---
Company or Organization

---

---
Phone No.

---
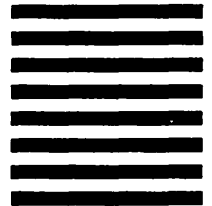
IBM®

Fold and Tape                    **Please do not staple**                    Fold and Tape

NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

# BUSINESS REPLY MAIL

FIRST CLASS MAIL    PERMIT NO. 40    ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

IBM International Technical Support Center
Department 91J, Building 235-2
Internal Zip 4423
901 NORTHWEST 51ST STREET
BOCA RATON  FL  33431-1328

Fold and Tape                    **Please do not staple**                    Fold and Tape

GG24-3731-00

GG24-3731-00

PRINTED IN THE U.S.A.

IBM

GG24-3731-00