

**INTRODUCTION TO THE
iRMX™ 86
OPERATING SYSTEM**

Order Number: 9803124-03

REV.	REVISION HISTORY	PRINT DATE
—01	ORIGINAL ISSUE	4/80
—02	Release 2.0 of the iRMX 86 Operating System	11/80
—03	Release 4.0 of the iRMX 86 Operating System	1/82

Additional copies of this manual or other Intel literature may be obtained from:

Literature Department
Intel Corporation
3065 Bowers Avenue
Santa Clara, CA 95051

The Information in this document is subject to change without notice.

Intel Corporation makes no warranty of any kind with regard to this material, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. Intel Corporation assumes no responsibility for any errors that may appear in this document. Intel Corporation makes no commitment to update nor to keep current the information contained in this document.

Intel Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in an Intel product. No other circuit patent licenses are implied.

Intel software products are copyrighted by and shall remain the property of Intel Corporation. Use, duplication or disclosure is subject to restrictions stated in Intel's software license, or as defined as ASPR 7-104.9(a)(9).

No part of this document may be copied or reproduced in any form or by any means without the prior written consent of Intel Corporation.

The following are trademarks of Intel Corporation and its affiliates and may be used only to identify Intel products:

BXP	Insite	iSBC	Multibus
CREDIT	Intel	iSBX	Multimodule
i	int _{el}	Library Manager	Plug-A-Bubble
ICE	Intelelevision	MCS	PROMPT
iCS	Intellec	Megachassis	RMX/80
i _m	iOSP	Micromainframe	System 2000
iMMX	iRMX	Micromap	UPI

PREFACE

If you are looking for a high-level introduction to the iRMX 86 Operating System, this manual will satisfy you. By reading this manual, you will acquire sufficient knowledge of the iRMX 86 Operating System to:

- See how the iRMX 86 Operating System can help you develop your application system in less time and at less expense.
- Begin reading the more detailed iRMX 86 manuals.

This manual, which is written for engineers and managers, is designed to be read completely in one or two sittings. It presents information starting with the most general and familiar terms that it then uses to define specific and new terms.

Throughout this manual, the expression "iAPX 86, 88-based microcomputer" is used to refer to any microcomputer that uses the Intel iAPX 86 or iAPX 88 microprocessor as its central processing unit.

CONTENTS

	Page
CHAPTER 1	
OVERVIEW OF THE iRMX 86 OPERATING SYSTEM	
Major Characteristics of the iRMX 86 System.....	1-1
Customers of the iRMX 86 Operating System.....	1-1
Commonly Used iRMX 86 Terminology.....	1-2
Purpose of the iRMX 86 Operating System.....	1-3
Organization of this Manual.....	1-3
CHAPTER 2	
CONSIDERATIONS RELATING TO REAL-TIME SOFTWARE	
Event Detection.....	2-1
Scheduling of Processing.....	2-1
Error Processing.....	2-1
Device Sensitivity.....	2-2
Service Selection.....	2-2
Mass Storage File Allocation Tradeoffs.....	2-2
Unneeded Features.....	2-2
Multiple Applications.....	2-2
Memory Requirements.....	2-2
Files and Multiple Users.....	2-3
Human Engineering.....	2-3
Application Development.....	2-3
Debugging.....	2-3
Chapter Perspective.....	2-4
CHAPTER 3	
BENEFITS OF THE iRMX 86 OPERATING SYSTEM	
Development Time.....	3-1
Cost of Implementation.....	3-2
Costs After Development.....	3-2
Chapter Perspective.....	3-2
CHAPTER 4	
FEATURES OF THE iRMX 86 OPERATING SYSTEM	
Architectural Features.....	4-2
Object-Oriented Architecture.....	4-2
Explanation of Object-Oriented Architecture.....	4-2
Advantages of Object-Oriented Architecture.....	4-4
Multitasking.....	4-4
Explanation of Multitasking.....	4-4
Advantages of Multitasking.....	4-4
Interrupt Processing.....	4-5
Explanation of Interrupt Processing.....	4-5
Advantages of Interrupt Processing.....	4-5

CONTENTS (continued)

	Page
CHAPTER 4 (continued)	
Preemptive Priority-Based Scheduling.....	4-6
Explanation of Preemptive Priority-Based Scheduling.....	4-6
Advantage of Preemptive Priority-Based Scheduling.....	4-6
Multiprogramming.....	4-7
Explanation of Multiprogramming.....	4-7
Advantages of Multiprogramming.....	4-8
Intertask Coordination.....	4-8
Explanation of Intertask Coordination.....	4-8
Advantage of Intertask Coordination.....	4-12
Extendibility.....	4-12
Explanation of Extendibility.....	4-12
Advantage of Extendibility.....	4-12
Input/Output Features.....	4-13
Choice of I/O Systems.....	4-13
Explanation of Choice of I/O Systems.....	4-13
Advantages of Choice of I/O Systems.....	4-15
Device-Independent Input and Output.....	4-15
Explanation of Device-Independent Input and Output.....	4-16
Advantages of Device-Independent Input and Output.....	4-16
Hierarchical Naming of Mass Storage Files.....	4-16
Explanation of Hierarchical Naming.....	4-16
Advantages of Hierarchical Naming.....	4-18
File Access Control.....	4-19
Explanation of File Access Control.....	4-20
Advantages of File Access Control.....	4-20
Control Over File Fragmentation.....	4-20
Explanation of File Fragmentation.....	4-20
Advantages of Control over Fragmentation.....	4-21
Selection of Device Drivers.....	4-21
Explanation of Device Drivers.....	4-22
Advantages of Having a Selection.....	4-22
File Maintenance Programs.....	4-22
Explanation of File Maintenance Programs.....	4-22
Advantages of File Maintenance Programs.....	4-23
Customizing Features.....	4-24
Custom Interactive Commands.....	4-24
Explanation of Custom Interactive Commands.....	4-24
Advantage of Custom Commands.....	4-25
Application Loading.....	4-25
Explanation of Application Loading.....	4-25
Advantages of Application Loading.....	4-26
Start-Up System.....	4-26
Explanation of Start-up System.....	4-26
Advantages of Start-up System.....	4-26
Run-Time Binding.....	4-27
Explanation of Run-Time Binding.....	4-27
Advantages of Run-Time Binding.....	4-29
Error Handling.....	4-29
Explanation of Error Handling.....	4-29
Advantage of Error Handling.....	4-30

CONTENTS (continued)

	Page
CHAPTER 4 (continued)	
Dynamic Memory Allocation.....	4-30
Explanation of Dynamic Memory Allocation.....	4-30
Advantage of Dynamic Memory Allocation.....	4-31
Software Interface.....	4-31
Explanation of Software Interface.....	4-31
Advantages of Software Interface.....	4-31
Object-Oriented Debugger.....	4-32
Explanation of Object-Oriented Debugger.....	4-32
Advantage of Object-Oriented Debugger.....	4-32
On-Target Program Development.....	4-33
Explanation of On-Target Program Development.....	4-33
Advantages of On-Target Program Development.....	4-33
Bootstrap Loading.....	4-34
Explanation of Bootstrap Loader.....	4-34
Advantages of Bootstrap Loader.....	4-34
Configurability.....	4-34
Explanation of Configurability.....	4-35
Advantages of Configurability.....	4-38
Chapter Perspective.....	4-38
 CHAPTER 5	
A HYPOTHETICAL SYSTEM	
Interrupt Processing.....	5-3
Human Interface.....	5-4
Multitasking.....	5-4
Intertask Coordination.....	5-4
Multiprogramming.....	5-4
Run-Time Binding.....	5-5
Mass Storage Files.....	5-5
Device Independence.....	5-6
Chapter Perspective.....	5-6
 CHAPTER 6	
iRMX 86 LITERATURE	
Reading Tips.....	6-2
Introduction to the iRMX 86 Operating System.....	6-3
iRMX 86 Nucleus Reference Manual.....	6-3
iRMX 86 Basic I/O System Reference Manual.....	6-3
iRMX 86 Extended I/O System Reference Manual.....	6-4
iRMX 86 Human Interface Reference Manual.....	6-4
iRMX 86 Loader Reference Manual.....	6-4
iRMX 86 System Programmer's Reference Manual.....	6-5
iRMX 86 Installation Guide.....	6-5
iRMX 86 Configuration Guide.....	6-6
iRMX 86 Debugger Reference Manual.....	6-6
iRMX 86 Programming Techniques.....	6-6
Guide to Writing Device Drivers for the iRMX 86 and iRMX 88	
I/O Systems.....	6-6

CONTENTS (continued)

	Page
CHAPTER 4 (continued)	
iRMX 86 Terminal Handler Reference Manual.....	6-7
iRMX 86 Disk Verification Utility Reference Manual.....	6-7
Run-Time Support Manual for iAPX 86,88 Applications.....	6-7

FIGURES

1-1. The iRMX 86 Foundation For Application Systems.....	1-2
3-1. The iRMX 86 System Provides Economic Benefits.....	3-1
4-1. Features of the iRMX 86 Operating System.....	4-2
4-2. An Engineering Directory.....	4-17
4-3. A Marketing Directory.....	4-18
4-4. Hierarchical Naming of Files.....	4-19
4-5. Configuration of a Hypothetical System.....	4-35
4-6. The Dual Objective of iRMX 86 Configuration.....	4-36
5-1. The Hardware of the Dialysis Application System.....	5-2

TABLE

6-1. Correlation of Manuals and Features.....	6-8
---	-----

CHAPTER 1. OVERVIEW OF THE iRMX™ 86 OPERATING SYSTEM

The iRMX 86 Operating System is a software package designed for use with Intel's iSBC 86,88 Single Board Computers and with other iAPX 86,88-based computers. The Operating System is different from many other systems in that it is specifically designed to be incorporated in the products that you build.

The iRMX 86 Operating System consists of a collection of subsystems, each of which provides one or more features that can be used in your product. Based on the features that you need to build your product, you decide which subsystems you want. You then combine these subsystems to form a tailored operating system that precisely meets your needs.

MAJOR CHARACTERISTICS OF THE iRMX 86 SYSTEM

The iRMX 86 Operating System exhibits the following characteristics:

- It can simultaneously monitor and control unrelated events occurring outside the single board computer.
- It can communicate with a wide variety of input, output, and mass storage devices.
- It provides a powerful and flexible means for an operator to observe and modify the behavior of the system.
- It provides a base upon which to run a number of languages and other software tools.

These characteristics (especially when combined with features discussed in Chapter 4) make the iRMX 86 Operating System an excellent foundation for your software-based products (Figure 1-1).

CUSTOMERS OF THE iRMX 86 OPERATING SYSTEM

The iRMX 86 Operating System is designed for two types of customers. Original Equipment Manufacturers, OEMs, are companies that build products for resale. Volume End Users, VEUs, are companies that build products for use within their organization. Both types of customers can produce products more quickly and at less expense by using the iRMX 86 Operating System.

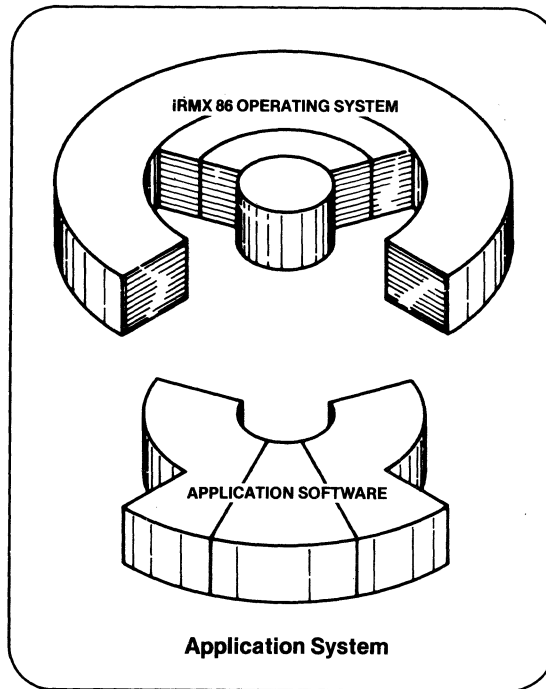


Figure 1-1. The iRMX™ 86 Foundation For Application Systems

COMMONLY USED iRMX 86 TERMINOLOGY

The following terms are used frequently in this book:

- Application

An application is the problem that you solve with your product.

- Application System

An application system is the product that satisfies the requirements of the application (Figure 1-1).

- Application Software

The application software is all the software you must add to the iRMX 86 Operating System in order to complete your application system (Figure 1-1).

- User

The user is the individual or organization who uses your application system.

OVERVIEW OF THE iRMX™ 86 OPERATING SYSTEM

PURPOSE OF THE iRMX 86 OPERATING SYSTEM

The iRMX 86 Operating System is your shortcut to the marketplace. By supplying you with features that can be used in a large number of application systems, the iRMX 86 Operating System allows you to focus your attention on the specialized application software. Since you spend less time and effort developing sophisticated system software, you can bring your application system to market faster and at a lower price.

ORGANIZATION OF THIS MANUAL

This manual is divided into six chapters. Some of the chapters are designed for managers, some for engineers, and others for both. The following paragraphs identify the audience and purpose of each chapter.

- Chapter 1 - Overview of the iRMX 86 Operating System

Chapter 1 provides managers and engineers with a very brief introduction to the iRMX 86 Operating System. It provides vocabulary needed in later chapters and contains a statement of purpose for each chapter in this manual.

- Chapter 2 - Considerations Relating to Real-Time Software

Chapter 2 introduces engineers to some of the obstacles that the iRMX 86 Operating System can eliminate. Managers who have had programming experience may want to read this short chapter.

- Chapter 3 - Benefits of the iRMX 86 Operating System

Chapter 3 provides managers with a discussion of the economic benefits of using the iRMX 86 Operating System. Interested engineers may also want to read this short chapter.

- Chapter 4 - Features of the iRMX 86 Operating System

Chapter 4 is a tutorial for engineers. It discusses the features of the iRMX 86 Operating System and, at the same time, it defines the vocabulary used in the other iRMX 86 manuals. Engineers who are already proficient at real-time, multitasking programming need only skim this chapter to ascertain the features of the iRMX 86 Operating System.

- Chapter 5 - A Hypothetical System

Chapter 5 is designed primarily for engineers. It describes a relatively simple application system. The purpose of this chapter is to illustrate the use of some of the features discussed in Chapter 4.

OVERVIEW OF THE iRMX™ 86 OPERATING SYSTEM

- Chapter 6 - iRMX 86 Literature

Chapter 6 contains a description of the other manuals associated with the iRMX 86 Operating System. This chapter is designed for engineers who need information more detailed than that provided by this introductory manual.

CHAPTER 2. CONSIDERATIONS RELATING TO REAL-TIME SOFTWARE

The kinds of difficulties encountered in real-time programming differ significantly from those found in other aspects of programming. This chapter briefly introduces some of the problems that face designers of real-time systems.

The purpose of this chapter is not to discourage you from building a real-time application system. Rather, its purpose is to show you the kinds of hurdles that the iRMX 86 Operating System can help you jump. Consequently, this chapter only poses questions -- it provides no answers. You can find the answers in the discussion of iRMX 86 features in Chapter 4 of this manual.

EVENT DETECTION

Real-time application systems monitor events in the real world. These events occur asynchronously, that is, at seemingly random intervals. When an event occurs, the system could be in the midst of processing information associated with a previous event. Even so, the system must be able to detect and record the occurrence of the second event.

SCHEDULING OF PROCESSING

Assuming that the system can detect and record the occurrence of an event, it still must decide in what order to process recorded events. For that matter, when the system is processing a relatively unimportant event and a critical event occurs, the system must be able to respond correctly. It must be able to postpone the processing of the less significant event until the more important one has been processed. Then, after the higher-priority processing, the system must resume where it left off.

ERROR PROCESSING

Suppose that during the processing of real-time events, an error is detected. How can the error be corrected, or how can its impact be limited, without adversely affecting the running of the system? The whole system, for instance, should not be shut down merely because an error is detected.

CONSIDERATIONS RELATING TO REAL-TIME SOFTWARE

DEVICE SENSITIVITY

Many real-time applications use one or more input or output devices. And sometimes, the devices associated with an application system must be changed. By allowing devices to be changed without requiring recompilation, the operating system can save much time and effort.

DEVICE SELECTION

What kinds of devices should an operating system support? Can it handle line printers? Disks? Bubble memories? Tapes?

MASS STORAGE FILE ALLOCATION TRADEOFFS

In any real-time system, performance is an important consideration. One decision that relates directly to performance must be made before formatting mass storage files. In some applications, large granularity (large amounts of information located contiguously) results in much faster retrieval. In other applications, large granularity does not improve performance, but does waste space on the device. The operating system must contend with the tradeoff between performance and optimal use of space on the device.

UNNEEDED FEATURES

Some OEM and VEU applications require features that other applications do not. An operating system should provide a means of selecting required features and rejecting unneeded features.

MULTIPLE APPLICATIONS

Sometimes there is a need to run more than one application on the same computer. Several applications might need to share some resources, such as hardware and perhaps some files, while reserving other resources for themselves.

MEMORY REQUIREMENTS

The memory requirements of some applications change according to the events that occur in the real world. If a system can share memory between applications, then the total amount of memory required for the system might be less than the sum of the maximum amounts required by each application.

CONSIDERATIONS RELATING TO REAL-TIME SOFTWARE

FILES AND MULTIPLE USERS

Some applications, such as key-to-disk and database-management systems, support more than one user. In such systems, two problems relate to mass storage files.

The first problem pertains to file naming. The users must be able to name files without concern for duplicate names. If they cannot, each user may be forced to guess at names that have not yet been assigned by other users.

The second problem deals with selective sharing of files. Multiuser systems often must be able to share and protect files. For instance, in a key-to-disk system, one operator may be entering data while another simultaneously verifies. This illustrates the need for sharing a file. Now suppose that the file contains confidential information. Once verified, the file must be protected against unauthorized reading and writing. This illustrates the need for restricting access. The system must provide for both sharing and restricted access.

HUMAN ENGINEERING

Applications must be controlled by people. Systems will often contain critical processes that operators must control with a minimum chance of error. An application system should provide an easily-understood set of interactive commands and messages by which operators may use the system.

APPLICATION DEVELOPMENT

Frequently the hardware on which an application system will be installed includes mass storage devices and file structures. If possible, the operating system should allow system development using this hardware. This means that you should be able to use language processors (assemblers, compilers, run-time support systems) linking utilities, editors, and file maintenance utilities. Minimum time should be required to install such development tools on the operating system.

DEBUGGING

Virtually all software, no matter how carefully checked out by manual inspection, contains some bugs. Usually, these bugs are detected by using the system until an error occurs. Once the error is found, an engineer begins tracing backwards from the error to the bug that caused it. When the bug is identified, the engineer modifies the software and eliminates the bug. This process, called debugging, is repeated until no more errors are found.

CONSIDERATIONS RELATING TO REAL-TIME SOFTWARE

This debugging process is not always straightforward in real-time systems. Often, bugs in real-time systems are dependent upon events in the real world (outside of the computer). In order to detect some real-time bugs, the system must continue to run even while it is being debugged.

CHAPTER PERSPECTIVE

If the foregoing considerations pertain to your application, then the iRMX 86 Operating System can save you an enormous amount of effort. To see how the iRMX 86 System resolves these and other similar problems, read Chapter 4.

CHAPTER 3. BENEFITS OF THE iRMX™ 86 OPERATING SYSTEM

You are reading this manual because you are planning to develop a real-time application system. As an OEM manager, you are interested in developing your application system quickly using the latest Very Large Scale Integration (VLSI) technology, while still holding down the cost of development. Furthermore, you want to minimize your costs after development. By serving as a foundation for your application software (Figure 3-1), the iRMX 86 Operating System can help you meet your objectives.

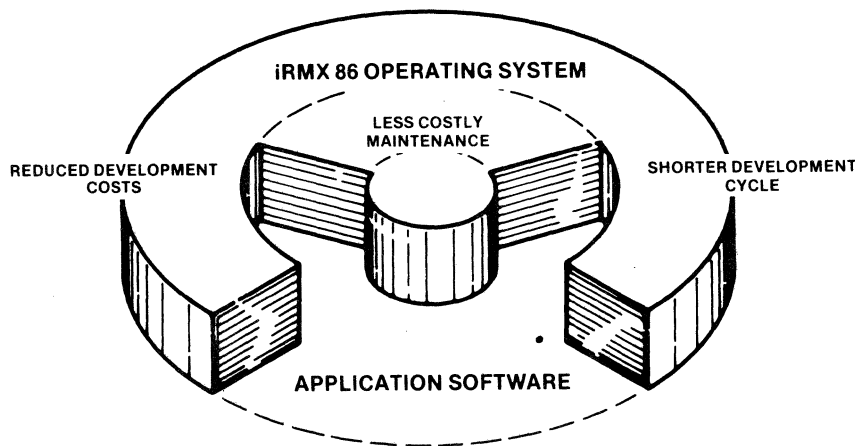


Figure 3-1. The iRMX™ 86 System Provides Economic Benefits

DEVELOPMENT TIME

The iRMX 86 Operating System helps you develop real-time application systems quickly. Acting as the foundation for your specialized application software, the iRMX 86 Operating System provides services that are required by many real-time applications. Since these services are supplied by the iRMX 86 Operating System, your application engineers spend no time writing software to manage multitasking, dynamic memory allocation, and other functions vital to many real-time applications. Rather, your engineers concentrate their efforts on the software that relates specifically to the application being solved. This greatly reduces the time needed to develop your application system.

BENEFITS OF THE iRMX™ 86 OPERATING SYSTEM

COST OF IMPLEMENTATION

The iRMX 86 Operating System helps reduce the cost of implementation in the following ways.

- By supplying the general services required by many real-time applications, the iRMX 86 System reduces your manpower requirements.
- Industry-standard languages are available for use with the iRMX 86 Operating System, and these languages are the same ones used on your Intellec Development System.
- The features of the Operating System simplify the process of development. These features, such as object-oriented architecture, device independence, and others, are discussed in Chapter 4.
- Support for VLSI devices is available now, which results in immediate improvements in speed and performance.

COSTS AFTER DEVELOPMENT

After your application system is developed, your major expense is maintenance -- the process of removing bugs, making changes, and adding features. The iRMX 86 Operating System helps minimize these costs.

First, a number of features of the iRMX 86 Operating System smooth the process of system design, reducing the probability of major design errors. These features, which include multitasking and multiprogramming, are described in Chapter 4.

Second, when errors do reveal the presence of bugs in your application software, the iRMX 86 System provides features to help find the bugs. These features include error handlers and an object-oriented, real-time debugger.

Third, the modularity provided by multiple jobs and tasks lets you make changes and additions without severely affecting the system's overall design.

CHAPTER PERSPECTIVE

The iRMX 86 Operating System is your economic ally. It helps you put your real-time application system in the hands of your users in less time and at less expense. It also allows you to use the latest improvements in VLSI technology while reducing your maintenance costs after your system is developed.

CHAPTER 4. FEATURES OF THE iRMX™ 86 OPERATING SYSTEM

This chapter provides you with moderately detailed descriptions of the following features (Figure 4-1) of the iRMX 86 Operating System:

ARCHITECTURAL FEATURES

- Object-Oriented Architecture
- Multitasking
- Interrupt Processing
- Preemptive Priority-Based Scheduling
- Multiprogramming
- Intertask Coordination
- Extendibility

INPUT/OUTPUT FEATURES

- Choice of I/O Systems
- Device-Independent Input and Output
- Hierarchical Naming of Mass Storage Files
- File Access Control
- Control over File Fragmentation
- Selection of Device Drivers
- File Maintenance Programs

CUSTOMIZING FEATURES

- Custom Interactive Commands
- Application Loading
- Start-up System
- Run-time Binding
- Error Handling
- Dynamic Memory Allocation
- Software Interface
- Object-Oriented Debugger
- Bootstrap Loading
- On-Target Development
- Configurability

Each section of this chapter deals with one of these features and, in case you are already familiar with some features, each section is organized for easy skimming. The first few sentences of each section provide a summary of the feature's value. The feature is then described in moderate detail and, near the end of each section, the advantages of the feature are discussed.

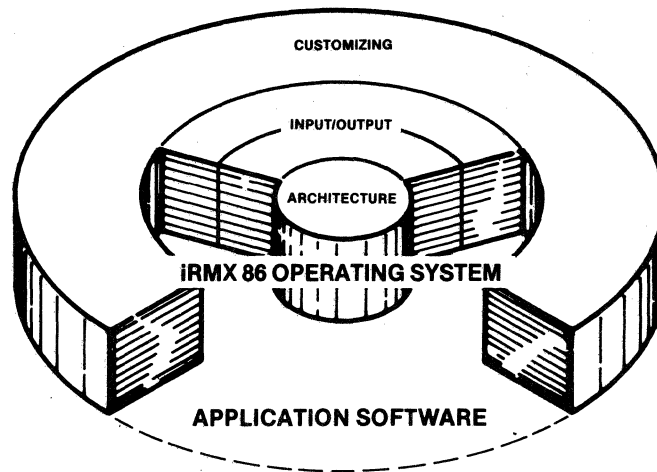


Figure 4-1. Features of the iRMX™ 86 Operating System

ARCHITECTURAL FEATURES

When Intel software engineers designed the iRMX 86 Operating System, they specified the basic processes and data structures of the system, including such characteristics as the partitioning of programs into "tasks," task scheduling, and task communication. These characteristics are referred to as the "architecture" of the system. The important architectural features of the Operating System are described here.

OBJECT-ORIENTED ARCHITECTURE

The iRMX 86 Operating System uses an object-oriented architecture because it makes the Operating System easy to learn and use.

Explanation of Object-Oriented Architecture

An operating system is a collection of software that is meant to be used by software engineers. Many operating systems are so complex that the majority of the engineers using them are unable to fully grasp their organization. In contrast, systems exhibiting object-oriented architectures are easier to understand. Their mechanisms are well defined, and they demonstrate a consistency that makes the operating system seem less awesome.

FEATURES OF THE iRMX™ 86 OPERATING SYSTEM

In other words, an object-oriented architecture is a means of humanizing an operating system. It uses a collection of building blocks that are manipulated by operators. Let's look at a typed architecture that you might be familiar with -- FORTRAN.

FORTRAN exhibits a typed architecture. Its building blocks are variables of several types. For instance, it has integers, real numbers, double-precision real numbers, etc. It also has operators (+, -, *, /, **, and others) that act on variables to produce understandable results. Let's turn back to operating systems and see how object-oriented architecture works in the iRMX 86 System.

The building blocks of the iRMX 86 Operating System are called objects and, as with FORTRAN variables, objects are of several types. There are tasks, jobs, mailboxes, semaphores, segments, and connections. There are also other types of objects, but we already have enough for an introduction.

Just as the variables in a FORTRAN program are acted upon by operators, the objects in an iRMX 86-based application system are acted upon by system calls. In other words, your application software uses system calls to manipulate the objects in your application system. For instance, the CREATE MAILBOX and DELETE MAILBOX system calls do precisely what their names suggest.

How does an object-oriented architecture make a system easier to learn and use? By taking advantage of useful classification. To illustrate this, let's return to FORTRAN. The variables of FORTRAN are classified into types because each type exhibits certain characteristics. For instance, all integer variables are somewhat similar, even though they can take on different values. Once you learn the characteristics of an integer variable, you feel comfortable with every integer variable. This similarity makes FORTRAN easy to master.

For the same reasons, the objects of the iRMX 86 Operating System are classified into types. Each object type (such as a semaphore) has a specific set of attributes. Once you become familiar with the attributes of a semaphore, you are familiar with all semaphores. There are no special cases.

This useful classification also applies to iRMX 86 system calls. Each type of iRMX 86 object has an associated set of system calls. These calls cannot be used to manipulate objects of another type without causing an error. (Our analogy breaks down at this point. FORTRAN operators almost always work on several types of variables.)

The beauty of the object-oriented architecture of the iRMX 86 Operating System can be summed up in one statement. Once you learn the attributes and the system calls associated with a type of object, you have complete knowledge of the behavior of the object type.

Advantages of Object-Oriented Architecture

The advantages of an object-oriented architecture depend upon your point of view. If you are an engineer, the advantage is that you can master the Operating System in a very short time. You can also focus your learning on the objects you plan to use. If you only need a few object types, you can ignore the others.

If you are a manager, you reap economic benefits. Because engineers can quickly become familiar with the iRMX 86 Operating System, you can trim large amounts of time out of your system's development cycle. Your system reaches your users far sooner and at far less cost than it could without object-oriented architecture.

MULTITASKING

The iRMX 86 Operating System uses multitasking to simplify the development of applications that process real-time events.

Explanation of Multitasking

The essence of real-time application systems is the ability to process numerous events occurring at seemingly random times. These events are asynchronous because they can occur at any time, and they are potentially concurrent because one event might occur while another is being processed.

Any single program that attempts to process multiple, concurrent, asynchronous events is bound to be complex. The program must perform several functions. It must process the events. It must remember which events have occurred and the order in which they occurred. It must remember which events have occurred but have not been processed. The complexity obviously grows greater as the system monitors more events.

Multitasking is a technique that unwinds this confusion. Rather than writing a single program to process N events, you can write N programs, each of which processes a single event. This technique eliminates the need to monitor the order in which events occur.

Each of these N programs forms an iRMX 86 task, one of the types of objects mentioned in "Object-Oriented Architecture." Tasks are the only active objects in the iRMX 86 Operating System, as only tasks can issue system calls.

Advantages of Multitasking

Multitasking simplifies the process of building an application system. This allows you to build your system faster and at less expense. Furthermore, because of the one-to-one relationship between events and tasks, your system's code is less complex and easier to maintain.

INTERRUPT PROCESSING

The iRMX 86 Operating System is an interrupt processor. When an interrupt occurs, the iRMX 86 Operating System schedules a task to process the interrupt. This method of event detection improves the performance of your application system.

Explanation of Interrupt Processing

There are two ways that computer systems can schedule processing associated with detecting and controlling events in the real world -- polling and interrupt processing. Polling is implemented by having the software periodically check to see if certain events have occurred. An example of polling from a human perspective can be created using a class of students and a teacher. If, rather than spotting raised hands, the instructor specifically asks each student in the class if the student has any questions, then the instructor is polling the students.

Polling has a major shortcoming. A significant amount of the processor's time is spent testing to see if events have occurred. If events have not occurred, the processor's time has been wasted.

The second method of controlling processing is interrupt processing. When an event occurs the processor is literally interrupted. Rather than executing the next sequential instruction, the processor begins to execute a task associated specifically with the detected event.

The classroom example used earlier to portray a polling situation can also be used to illustrate interrupt processing. If a student has a question, he raises his hand and speaks the instructor's name. The instructor, interpreting this as an interrupt, finishes his sentence and deals immediately with the student's question. Once the instructor has answered the student's question, he returns to what he was doing before he was interrupted.

Advantages of Interrupt Processing

Interrupt processing of external events provides your application system with three benefits.

- Better Performance

Interrupt processing allows your system to spend all of its time running the tasks that process events, rather than executing a polling loop to see if events have occurred.

- More Flexibility

Because of the direct correlation between interrupts and tasks, your system can easily be modified to process different events. All you need to do is write the tasks to process the new interrupts.

FEATURES OF THE iRMX™ 86 OPERATING SYSTEM

- Economic Benefits

Because interrupt processing allows your system to respond to events by means of modularly coded tasks, your system's code is more structured and easier to understand. Modular code is less costly to develop and maintain, and it can be developed more quickly than monolithic code.

PREEMPTIVE PRIORITY-BASED SCHEDULING

The iRMX 86 Operating System uses preemptive, priority-based scheduling to decide which task runs at any instant. This technique ensures that if a more important task becomes ready while a less important task is running, the more important task begins execution immediately.

Explanation of Preemptive Priority-Based Scheduling

In multitasking systems, there are two common techniques for deciding which task is to be run at any given moment. Time slicing, where tasks are run in rotation, is the technique used in time-sharing systems. The second technique, priority-based scheduling, uses assigned priorities to decide which task is to be run.

Within priority-based scheduling, there are two approaches. Non-preemptive scheduling allows a task to run until it relinquishes the processor. Even if while running it causes a higher-priority task to become ready for execution (for instance, by deallocating a peripheral device), the original task continues to run until it explicitly surrenders the processor.

The second approach to priority-based scheduling is preemptive. In systems using preemptive scheduling, the system always executes the highest priority task that is ready to run. In other words, if the running task or an interrupt causes a higher-priority task to become ready, the operating system switches the processor to the higher-priority task.

Advantage of Preemptive Priority-Based Scheduling

Preemptive, priority-based scheduling goes hand-in-hand with the interrupt processing discussed earlier. The priorities of tasks can be tied to the relative importance of the events that they process. This enables the processing of more-important events to preempt the processing of less-important events.

MULTIPROGRAMMING

Multiprogramming provides your system with the ability to run more than one application on one iAPX 86,88-based microcomputer. This helps reduce hardware costs.

Explanation of Multiprogramming

Multiprogramming is a technique used to run several applications on a single application system. By using this technique, the hardware is used more fully. More processing is squeezed out of each hardware dollar.

In order to take full advantage of multiprogramming, you must provide each application with a separate environment; that is, separate memory, files and objects. The reason for the isolation is to prevent independently developed applications from causing problems for each other.

For instance, suppose that two unrelated applications share a temporary file on a disk. If Application 1 writes information to the file and Application 2 writes over the file, Application 1 has problems. The only way to avoid this kind of problem with shared files is to create some form of mutual exclusion. But if the two applications must interact even to the point of excluding each other, they cannot be developed independently. The two engineers creating the applications must coordinate with each other and spend valuable time that could be used within, rather than between, applications. The only alternative is to avoid sharing the file.

The iRMX 86 Operating System provides a type of object that can be used to obtain this kind of isolation. The object is called a job, and it has the following characteristics:

- Unlike tasks, jobs are passive. They cannot invoke system calls.
- Each job includes a collection of tasks and resources needed by those tasks.
- Jobs serve as useful boundaries for dynamically allocating memory. When two tasks of one job request memory, they share the memory associated with their job. Two tasks in different jobs do not directly compete for memory.
- An application consists of one or more jobs.
- Each job serves as an error boundary. When the application detects an error, or when the operator decides to abort an application, a job is a convenient object to delete.

FEATURES OF THE iRMX™ 86 OPERATING SYSTEM

Advantages of Multiprogramming

Multiprogramming provides your application system with two benefits:

- Multiprogramming increases the amount of work your system can do. By using your hardware a larger percentage of the time, it lets your system run several applications rather than one. This reduces the hardware cost of implementation.
- Because of the correspondence between jobs and applications, new jobs can be added to your system (or old jobs removed) without affecting other jobs. This makes your system much easier and faster to modify.

INTERTASK COORDINATION

The iRMX 86 Operating System provides simple techniques for tasks to coordinate with one another. These techniques allow tasks in a multitasking system to mutually exclude, synchronize, and communicate with each other.

Explanation of Intertask Coordination

As we have already seen, multitasking is a technique used to simplify the designing of real-time application systems that monitor multiple, concurrent, asynchronous events. Multitasking allows engineers to focus their attention on the processing of a single event rather than having to contend with numerous other events occurring in an unpredictable order.

However, the processing of several events may be related. For instance, the task processing Event A may need to know how many times Event B has occurred since Event A last occurred. This kind of processing requires that tasks be able to coordinate with each other. The iRMX 86 Operating System provides for this coordination.

Tasks can interact with each other in three ways. They can exchange information, mutually exclude each other, and synchronize each other. We'll now examine each of these.

- Exchanging Information

Tasks exchange information for two purposes. One purpose is to pass data from one task to another. For instance, suppose that one task accumulates keystrokes from a terminal until a carriage return is encountered. It then passes the entire line of text to another task, which is responsible for decoding commands.

The second reason for passing data is to draw attention to a specific object in the application system. In effect, one task says to another, "I am talking about that object."

FEATURES OF THE iRMX™ 86 OPERATING SYSTEM

The iRMX 86 System facilitates intertask communication by supplying objects called mailboxes along with system calls to manipulate mailboxes. The system calls associated with mailboxes are CREATE MAILBOX, DELETE MAILBOX, SEND MESSAGE, and RECEIVE MESSAGE. Tasks use the first two system calls to build and eradicate a particular mailbox. They use the second two calls to communicate with each other.

Let's see how tasks can use a mailbox for drawing attention and for sending information. If Task A wants Task B to become aware of a particular object, Task A uses the SEND MESSAGE system call to mail the object to the mailbox. Task B uses the RECEIVE MESSAGE system call to get the object from the mailbox.

NOTE

The foregoing example, along with all of the examples in this section, is somewhat simplified in order to serve as an introduction. If you want detailed information, refer to the iRMX 86 NUCLEUS REFERENCE MANUAL.

As mentioned previously, tasks can use mailboxes to send information to each other. This is accomplished by putting the information into a segment (an iRMX 86 object consisting of a contiguous block of memory) and using the SEND MESSAGE system call to mail the segment. The other task invokes the RECEIVE MESSAGE system call to get the segment containing the message.

Why don't tasks just send messages directly between each other, rather than through mailboxes? Tasks are asynchronous -- they run in unpredictable order. If two tasks want to communicate with each other, they need a place to store messages and to wait for messages. If the receiver uses the RECEIVE MESSAGE system call before the message has been sent, the receiver waits at the mailbox until a message arrives. Similarly, if the sender uses the SEND MESSAGE system call before the receiver is ready to receive, the message is held at the mailbox until a task requests a message from the mailbox. In other words, mailboxes allow tasks to communicate with each other even though tasks are asynchronous.

● Mutual Exclusion

Occasionally, when tasks are running concurrently, the following kind of situation arises:

- Task A is in the process of reading information from a segment.
- An interrupt occurs and Task B, which has higher priority than Task A, preempts Task A.
- Task B modifies the contents of the segment that Task A was in the midst of reading.
- Task B finishes processing its event and surrenders the processor.
- Task A resumes reading the segment.

The problem is that Task A might have information that is completely invalid. For instance, suppose the application is air traffic control. Task A is responsible for detecting potential collisions, and Task B is responsible for updating the Plane Location Table with the new X- and Y-coordinates of each plane's location. Unless Task A can obtain exclusive use of the Plane Location Table, Task B can make Task A fail to spot a collision.

Here's how it could happen. Task A reads the X- coordinate of the plane's location and is preempted by Task B. Task B updates the entry that Task A was reading, changing both the X- and Y-coordinates of the plane's location. Task B finishes its function and surrenders the processor. Task A resumes execution and reads the new Y-coordinate of the plane's location. As a direct result of Task B changing the Plane Location Table while Task A was reading it, Task A thinks the plane is at old X and new Y. This misinformation could easily lead to disaster.

This problem can be avoided by mutual exclusion. If Task A can prevent Task B from modifying the table until after A has finished using it, A can be assured of valid information. Somehow, Task A must obtain exclusive use of the table.

The iRMX 86 Operating System provides a type of object that can be used to provide mutual exclusion -- the semaphore. A semaphore is an integer counter that tasks can manipulate using four system calls: CREATE SEMAPHORE, DELETE SEMAPHORE, SEND UNITS and RECEIVE UNITS. The creation and deletion system calls are used to build and eradicate semaphores. The send and receive system calls can be used to achieve mutual exclusion.

Before discussing how semaphores can provide exclusion, we must examine their properties. As mentioned above, a semaphore is a counter. It can take on only nonnegative integer values. Tasks can modify a semaphore's value by using the SEND UNITS or RECEIVE UNITS system calls. When a task sends N units (must be zero or

FEATURES OF THE iRMX™ 86 OPERATING SYSTEM

greater) to a semaphore, the value of the counter is increased by N. When a task uses the RECEIVE UNITS system call to request M units (must be zero or greater) from a semaphore, one of two things happens.

- If the semaphore's counter is greater than or equal to M, the Operating System reduces the counter by M and continues to execute the task.
- Otherwise, the Operating System begins running the task having the next highest priority, and the requesting task waits at the semaphore until the counter reaches M or greater.

How can tasks use a semaphore to achieve mutual exclusion? Easy! Create a semaphore with an initial value of 1. Before any task uses the shared resource, it must receive one unit from the semaphore. Also, as soon as a task finishes using the resource, it must send one unit to the semaphore. This technique ensures the following behavior. At any given moment, no more than one task can use the resource, and any other tasks that want to use it await their turn at the semaphore.

Semaphores allow mutual exclusion; they don't enforce it. All tasks (there can be more than two) sharing the resource must receive one unit from the semaphore before using the resource. If one task fails to do this, mutual exclusion is not achieved. Also, each task must send a unit to the semaphore when the resource is no longer needed. Failure to do this can permanently lock all tasks out of the resource.

- Synchronization

As mentioned earlier, tasks are asynchronous. Nonetheless, occasionally a task must know that a certain event has occurred before the task starts running. For instance, suppose that a particular application system requires that Task A cannot run until after Task B has run. This kind of requirement calls for synchronizing Task A with Task B.

Your application system can achieve synchronization by using semaphores. Before executing either Task A or Task B, create a semaphore with an initial value of zero. Then have Task A issue RECEIVE UNITS requesting one unit from the semaphore. Task A is forced to wait at the semaphore until Task B sends a unit. This achieves the desired synchronization.

FEATURES OF THE iRMX™ 86 OPERATING SYSTEM

Advantage of Intertask Coordination

Every real-time multitasking system must provide for intertask coordination, so this coordination cannot be billed as an advantage. The true advantage arises from the flexible means that the iRMX 86 System provides for accomplishing coordination.

The intertask coordination supplied by the iRMX 86 Operating System is flexible and simple to use. Semaphores and mailboxes can accommodate a wide variety of situations. And your application system is not limited to some arbitrary number of mailboxes or semaphores. It can create as many as it needs.

EXTENDIBILITY

The iRMX 86 Operating System is extendible. It allows you to create your own object types and to add system calls to the Operating System.

Explanation of Extendibility

Something is extendible if you can add to it, and the iRMX 86 Operating System is extendible. Your system programming engineers can build their own types of objects and the system calls to manipulate those objects. These custom features become a part of the Operating System. From the point of view of the application programming engineer, there is no way to distinguish your custom objects from those supplied by Intel.

Advantage of Extendibility

The advantage of extendibility is that you can add your features to the iRMX 86 Operating System and obtain the same benefits as supplied by its object-oriented architecture. These benefits include the ability to send your custom-made objects to mailboxes and the ability to put them in object directories. Additionally, your application engineers can more quickly become familiar with your custom features. This shrinks your development time and costs, and it allows you to bring your application system to your users sooner.

INPUT/OUTPUT FEATURES

The iRMX 86 Operating System offers the power and flexibility of a general-purpose operating system. Input and output operations will be a large part of most applications, so the Operating System offers a collection of I/O features to speed development of application systems, and to make the I/O of those systems efficient.

CHOICE OF I/O SYSTEMS

To meet the I/O needs of a wide variety of applications, the iRMX 86 Operating System provides two I/O systems. You can use one system or the other, or both may be combined in one application.

Explanation of Choice of I/O Systems

Many features of the iRMX 86 Operating System are useful in most applications, but not all applications. This is especially true of features relating to input and output. The iRMX 86 Operating System provides two I/O systems: the Basic I/O System and the Extended I/O System.

EXTENDED I/O SYSTEM. The Extended I/O System is designed to be easy to use, and to be efficient for sequential I/O. The important features of the Extended I/O System are:

- AUTOMATIC BUFFERING OF I/O OPERATIONS

If you want to use multiple-buffered I/O, but do not want to be burdened with writing complex code to check and switch buffers, you can use iRMX 86 Extended I/O System calls. When the application program issues a system call to perform an I/O operation, the iRMX 86 Operating System performs the input or output and returns control to the user program after the data transfer is completed. But before returning control to the user program, the iRMX 86 Operating System starts reading or writing the next block.

For example, if the application is reading a file from disk, the following sequence will occur:

1. When the application program opens a file using an Extended I/O System call, the Operating System starts reading the first block of the file ("initiates" the input).
2. The Operating System returns control to the application program.

FEATURES OF THE iRMX™ 86 OPERATING SYSTEM

3. Later the program requests an Extended I/O System Read. The Operating System has already started reading this data. When the input is complete, the Operating System initiates a read of the next block of the file (called "reading ahead"), and returns control to the calling program.

In this way, whenever the user requests an Extended I/O System Read, the data is either immediately available, or is in the process of being read.

The equivalent output process is performed by "writing behind." When an application program requests an Extended I/O System Write, the iRMX 86 Operating System copies the data to a buffer maintained by the Extended I/O System, and returns to the calling program. Whenever this buffer is filled, the system initiates an output.

- EFFICIENT SEQUENTIAL I/O OPERATIONS

Another characteristic of the Extended I/O System is that when it does a "read ahead" operation, the Operating System assumes that a series of sequential reads are to be performed. For example, the Operating System will read data from disk address 23, then from disk address 24, and so on. So when your I/O is mostly sequential, (for example, when examining consecutive records of a file) Extended I/O System calls are particularly efficient. It is still possible to perform random access of a file with the Extended I/O System by preceding operations with a Seek call specifying the offset into the file.

- FREE OF TEDIOUS DETAILS

The system calls of the Extended I/O System have relatively few parameters and are easy to code. In many cases a single Extended I/O Call will serve the purpose of several Basic I/O System calls. This simplifies your application system, which reduces development time and reduces costs.

BASIC I/O SYSTEM. For some applications the performance or flexibility of the system is more critical than the time necessary to produce the system. For these applications, the iRMX 86 Operating System provides the Basic I/O System.

The Basic I/O System is the more flexible of the two I/O systems. It provides very powerful capabilities, and it makes few assumptions about the requirements of your application. The following features illustrate the flexibility of the Basic I/O System:

FEATURES OF THE iRMX™ 86 OPERATING SYSTEM

- **ALLOWS YOU TO DESIGN YOUR OWN BUFFERING ALGORITHM**

Rather than automatically providing a buffering algorithm, the Basic I/O System allows you to design and implement your own buffering technique. Using the Basic I/O System, you control the synchronization between I/O and processing.

- **APPROPRIATE FOR RANDOM I/O OPERATIONS**

Perhaps the I/O in your application is random access. This means that rather than reading or writing data in sequential blocks, the application accesses data in blocks that are not adjacent to each other. The Basic I/O System is more appropriate to these operations because of the explicit control the programmer has over I/O operations.

- **GIVES YOUR TASK CONTROL OF DETAILS**

The system calls of the Basic I/O System often have many parameters. Using these parameters, your tasks can closely tailor the behavior of each system call to match the performance requirements of your application system.

As these features show, the Basic I/O System emphasizes flexibility rather than ease of use. The Basic I/O System provides I/O features that are useful in time-critical or memory-critical applications, and allows the performance of a system to be optimized.

Advantages of Choice of I/O Systems

The iRMX 86 Operating System allows you to select the features you want. The Basic I/O System gives maximum control of I/O operations for applications requiring finely tuned performance, especially while doing random-access I/O. The Extended I/O System is easy to use. It saves development costs and development time, especially in applications that use sequential I/O.

Finally, remember that you can use both I/O systems when your application system uses I/O for several purposes, some of which are best accomplished by the Basic I/O System, and some of which are best accomplished by the Extended I/O System.

DEVICE-INDEPENDENT INPUT AND OUTPUT

The input and output capabilities of the iRMX 86 Operating System are device independent. This adds flexibility to your system by allowing you to easily reroute input or output to different devices.

Explanation of Device-Independent Input and Output

Device independence is a relatively simple yet powerful concept. A system provides device-independent I/O if it has one set of system calls for communicating with all I/O devices. The alternative to device independence is to provide different calls for each type of device. Let's first examine the alternative and then move on to device independence.

Consider an operating system that does not provide device independence. The system calls controlling input and output operations are explicitly related to the I/O devices being used. For instance, the system call for writing to the line printer might be PRINT, while the system call for writing to the terminal might be TYPE. Once you have written a procedure in such a system, the procedure is locked into a particular combination of devices. The only way you can reroute input or output is to edit the source code and recompile.

Now consider an operating system that is device independent: the iRMX 86 Operating System. Because the iRMX 86 System supports device-independent I/O, the system calls are not device dependent. The READ system call is always used for input, and the WRITE system call is always used for output. The device is specified by a parameter of the system call. Consequently, by using a variable as the parameter that selects the device, you can create I/O procedures that are completely independent of the devices they use.

Advantages of Device-Independent Input and Output

Device independence makes your application system very flexible. If you write a procedure to log events on a line printer, you can use the same procedure to log events on a terminal or, for that matter, on a disk. You need not recompile or otherwise modify your system.

HIERARCHICAL NAMING OF MASS STORAGE FILES

The iRMX 86 Operating System supports hierarchical naming of files on mass storage devices. This naming technique provides your application systems with additional flexibility by simplifying the process of organizing and naming files.

Explanation of Hierarchical Naming

Hierarchical naming is one of three common techniques used to name files on mass storage devices such as disks, bubble memories, or drums. The other two techniques are called simple naming and directory naming. The advantages of hierarchical naming become clear when that technique is compared to the other two. First we'll look at simple naming.

FEATURES OF THE iRMX™ 86 OPERATING SYSTEM

Simple naming allows you to provide files with a descriptive name. For instance, you might decide to name files ACCOUNTS PAYABLE, ACCOUNTS RECEIVABLE, TRANSACTIONS, and INVENTORY. These names are certainly descriptive, but what happens when a different application running in the same system also decides to use one of these names? This question is avoided by using a more powerful naming technique: directory naming.

Directory naming allows different applications (or different application engineers, for that matter) to use the same file name. Each application (or engineer) is given one special-purpose file, called a directory. This directory contains only file names; it does not contain data. Figures 4-2 and 4-3 provide examples of directories.

When application software refers to a specific file, it first names the directory and then names the file. For instance, in Figure 4-2, the TRANSACTIONS file associated with Engineering would be designated ENGINEERING/TRANSACTIONS. The comparable file for Marketing, in Figure 4-3, would be designated MARKETING/TRANSACTIONS.

The advantage of directory naming over simple naming is that directory naming allows the file names to reflect the relationships between files. In Figure 4-2, all the files pertaining to Engineering are in the directory called ENGINEERING. This grouping of related files is not supported by simple naming.

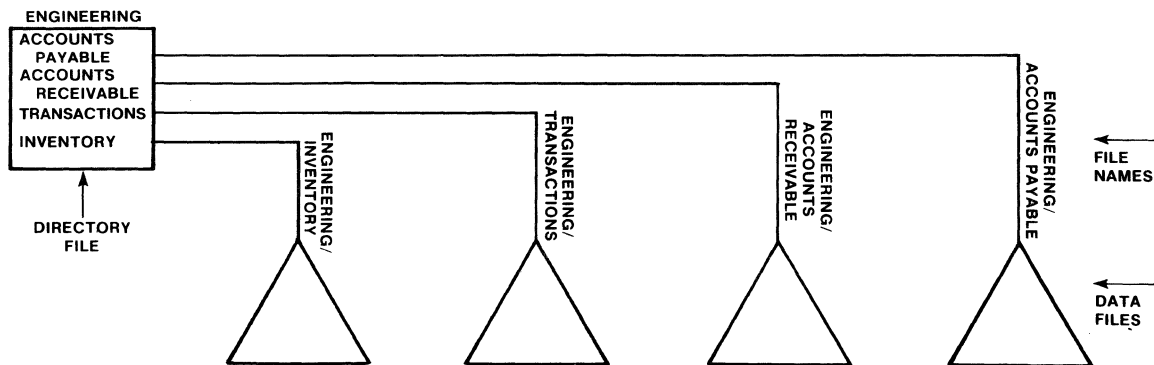


Figure 4-2. An Engineering Directory

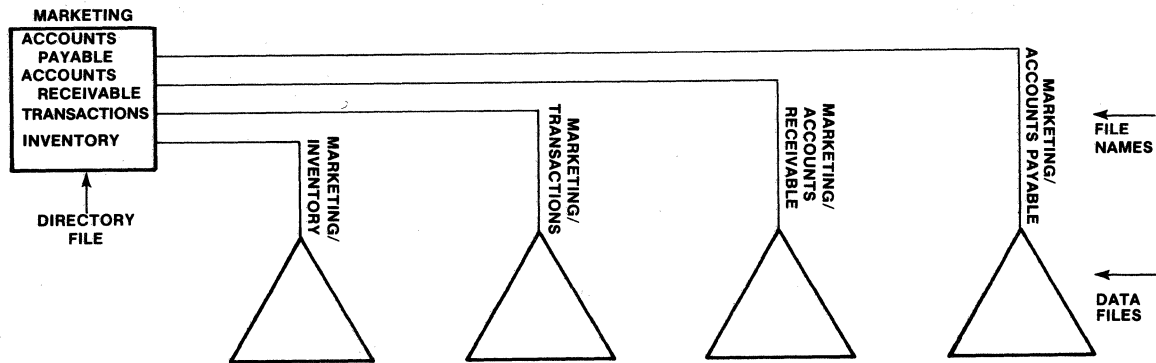


Figure 4-3. A Marketing Directory

What about situations in which more than one level of directory is required? This situation is illustrated in Figure 4-4. This figure differs from 4-3 only in that a second level of grouping has been included.

Just as Figure 4-4 shows that single-level directory naming is not sufficient for all collections of files, another figure could be constructed to show that two-level directory naming is not always sufficient. Consequently, the iRMX 86 Operating System supports any number of levels of directories. This n-level directory naming is called hierarchical naming of files.

Advantages of Hierarchical Naming

Hierarchical naming of files simplifies the process of adding new applications to your system. One concern about expanding your system is the naming of mass storage files associated with a new application. Names of new files must differ from names of existing files. If your system uses only a few mass storage files, you can expect little difficulty in assigning unique file names. But if your system uses a large number of files, the problem of ensuring uniqueness becomes more significant.

This uniqueness problem becomes particularly difficult if file names are assigned by an operator in a system having more than one operator.

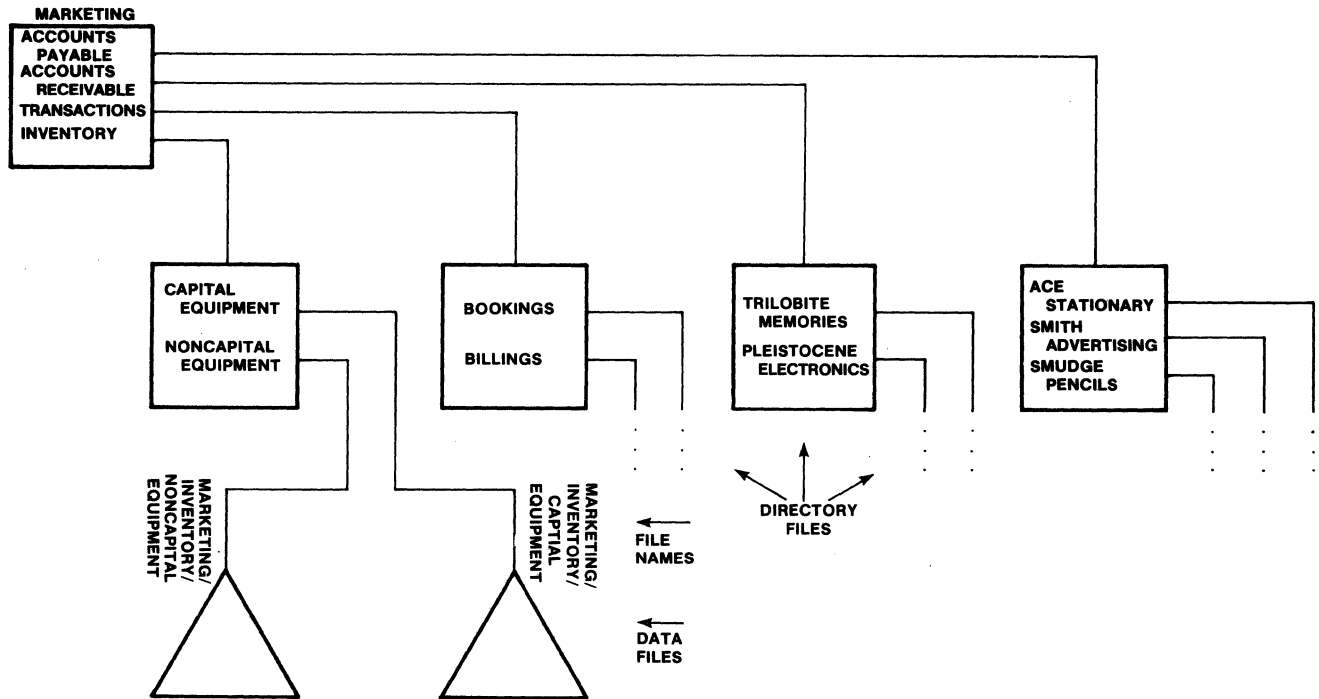


Figure 4-4. Hierarchical Naming Of Files

Hierarchical file naming eliminates this problem. Whenever you add a new application to your system, you can assign it a directory. The new application can then use this directory to provide unique names to any number of files. Also, each operator can be assigned a unique directory which can then be used to provide unique names.

FILE ACCESS CONTROL

The iRMX 86 Operating System allows your application system to control access to hierarchically named files. This facilitates file sharing while still preventing valuable data from being copied, modified, or destroyed by unauthorized users.

FEATURES OF THE iRMX™ 86 OPERATING SYSTEM

Explanation of File Access Control

In the multiprogramming environment provided by the iRMX 86 Operating System, the sharing of files can be useful. But the job that owns a file may wish to share it with only certain other jobs rather than all other jobs. Furthermore, the job owning a file may wish to restrict the nature of the shared access. For example, the owning job may wish to allow a particular file to be read but not written. The ability to specify how and with whom a file is shared is called file access control.

The iRMX 86 Operating System provides powerful file access control by allowing the owner of a file to specify who can use the file and how they can use it. In fact, a file's owner can even grant different combinations of access (reading only, writing only, reading and writing, etc.) to each user of a file.

Advantages of File Access Control

By controlling who can access a file and how they can access it, your system becomes more reliable and secure. There is less chance for an unauthorized task to accidentally modify a valuable file, and there is less opportunity for an unauthorized task to read a confidential file.

Your application software can, in fact, expand file access protection into a file security system. For instance, suppose that your application involves several operators accessing files on disk. By providing each operator with a password, so an individual's identity can be verified, your application software can strictly control which operators have access to which files.

CONTROL OVER FILE FRAGMENTATION

The iRMX 86 Operating System allows you to specify the granularity of each mass storage file. This lets you trade faster I/O for more efficient use of space on the mass storage device.

Explanation of File Fragmentation

When information is stored on a mass storage device, space is allocated in chunks rather than one byte at a time. These chunks, called granules, can be large or small, but all granules within one file must be the same size. This size is called the file granularity, and it is specified by the engineer who creates the file.

A file's granularity affects the use of a storage device in three ways.

FEATURES OF THE iRMX™ 86 OPERATING SYSTEM

- Data Transfer Rate

The file granularity directly affects the speed at which the Operating System can transfer information to or from the storage device. The larger the granularity, the faster the Operating System transfers data.

- Access Time

The smaller the granules, the more time is required to access a series of random locations in the file. Larger granules reduce access time.

- Wasted Device Space

The file granularity directly affects the amount of wasted space on the device. The larger the granularity, the more device space is wasted.

Here's an example. (For the sake of simplicity, we will ignore any information stored on the device on behalf of the Operating System.) Consider a file containing 20010 bytes. If the granularity is 10000 bytes, the file occupies three granules, each of which is 10000 bytes long. The first two granules are full and the third contains only 10 useful bytes. This file wastes almost 10000 bytes of storage space.

If we change the file granularity to 200 bytes, the file occupies 101 granules. Each of the first 100 granules is full and the last granule contains only 10 useful bytes. The file now wastes only 190 bytes of storage space.

Advantages of Control over Fragmentation

By allowing you to control the file granularity, the iRMX 86 Operating System lets you trade device space for performance. If your application has many mass storage units and space is readily available, you can specify a large file granularity. This provides you with faster average transfer rates and shorter access times, but it wastes some of your device space.

If, on the other hand, you have only one small mass storage unit, you might want to sacrifice some performance for better use of space. This trade would be particularly desirable if you do not use the device often enough to be concerned with the rate of data transfer.

SELECTION OF DEVICE DRIVERS

The iRMX 86 Operating System offers you your choice of Intel-supplied device drivers. It also allows you to write your own drivers.

FEATURES OF THE iRMX™ 86 OPERATING SYSTEM

Explanation of Device Drivers

A device driver is a software module that serves as the interface between a device's controller (which is hardware) and the iRMX 86 I/O System. The purpose of the driver is to make all devices look alike to the I/O System. In effect, the driver hides the idiosyncrasies of a device from the I/O System.

Advantages of Having a Selection

By selecting and creating device drivers, you can attach any device to your application system. This means that you are not limited to a few specific devices. You can select devices on any basis at all -- performance, cost, reliability, availability, whatever. The choice is yours.

FILE MAINTENANCE PROGRAMS

The iRMX 86 Operating System is delivered with programs which allow you to manipulate iRMX 86 files.

Explanation of File Maintenance Programs

As you develop an application, you will need to work with files. You can write programs to perform these file operations. But the Operating System already has programs to perform operations that are usually necessary in developing an application system. (The programs operate as commands to the system; why a program is a command is explained in the section Custom Interactive Commands.) These commands include:

- COPY, which copies files.
- FORMAT, which formats an iRMX 86 secondary storage device such as a disk or diskette.
- DIR, which displays a directory of the files on an iRMX 86 device.
- DOWNCOPY and UPCOPY, which are used to move files between ISIS (Development System) devices and iRMX 86 devices.
- RENAME, which allows you to rename files.
- CREATEDIR, which allows you to create a new file directory.
- SUBMIT, which automatically executes commands contained in an iRMX 86 file.

Advantages of File Maintenance Programs

The most important advantage of these programs is that you will save time and money in developing your application system, because you already have the software tools necessary to manipulate files during the development process. In addition, the file maintenance programs may be included as part of your application system if this is appropriate.

CUSTOMIZING FEATURES

The iRMX 86 Operating System is designed specifically for OEM applications. For this reason the application system as a whole can appear unique to the user. Certain features of the Operating System allow an application to be customized in its capabilities and in how it appears to the end user. Let's look at these features.

CUSTOM INTERACTIVE COMMANDS

People interact with your applications by entering commands and receiving information at terminals. The iRMX 86 Operating System allows you to define commands that are appropriate to the application and are meaningful to the operator. This command facility is called the Human Interface.

Explanation of Custom Interactive Commands

By designing commands which are appropriate to the type of people who use a system, you can control the human-to-application interface. This can make a system appear "friendly," it can give the application a unique character, and it can reduce operator errors.

CUSTOM COMMANDS = PROGRAMS. Because the first word in a command is the name of a program file on a mass storage device such as a disk, you are given great flexibility in defining commands. (In the example, TOXIN is the name of a program file.) When someone type a command at the terminal, the program having the command name is loaded from the disk, and is run by the Operating System. This means:

- You may add or modify commands simply by writing new programs.
- The number of custom commands for a system is not limited by program space in memory.
- You do not have to "rebuild" the system to change commands.
- Programs that are used infrequently do not take up memory space when they aren't being run.

COMMAND LINE PARSING. System calls are available for retrieving and interpreting parameters of a command. This process is called "parsing a command line."

FEATURES OF THE iRMX™ 86 OPERATING SYSTEM

Consider an application that monitors toxins in the blood of hospital patients. An operator (perhaps a nurse or doctor) can run a task that displays the toxin level of an individual patient, or of all patients being monitored. One approach would be to have the operator run the task with a command:

```
"RUN TOXIN.V3"
```

The program might prompt with:

```
"Display of which units? --"
```

A more "friendly" approach allows a person to use commands that are oriented to the application and to his or her skills, rather than to use computer-oriented commands. In the example, a better command is:

```
"TOXIN of John Doe"
```

The program TOXIN issues a system call to receive the parameter "John Doe". Because file names frequently are parameters for commands, specialized system calls are also available to interpret file name parameters.

Advantage of Custom Commands

The iRMX 86 Operating System makes it easy to design commands for operators who are not particularly familiar with computers. The ability to define commands enables you to create an application that is easy to use, is easy to understand, and is resistant to operator errors. New commands may be added by simply writing new programs, rather than making expensive changes to the system.

APPLICATION LOADING

The iRMX 86 Operating System allows your application to read programs from disk into memory and to run them. (This capability is briefly described under Custom Interactive Commands in this section.) Also, the Operating System allows a program to be broken into segments called overlays, so that the entire program does not have to be in memory at one time.

Explanation of Application Loading

The iRMX 86 Application Loader provides two powerful facilities: Load-Time Location, and Overlay Loading.

FEATURES OF THE iRMX™ 86 OPERATING SYSTEM

LOAD-TIME LOCATION. The explanation of Custom Interactive Commands mentioned that programs can be loaded from a mass storage device like a disk or bubble memory. The iRMX 86 Application Loader is designed so that programs may be loaded anywhere in available memory. The loader will modify the appropriate addresses in the program at the time the program is loaded. This capability, Load-Time Location, offers great flexibility in the design of application systems. As new programs are added, existing programs do not have to be rebuilt ("linked") in order to run together. Or if more memory is added to the system, the memory can be readily used.

OVERLAY LOADING. Occasionally a program is large enough that it is necessary to break it into pieces called overlays. Each overlay runs at a different time, and occupies the same area of memory. A program containing overlays consists of a "root" that is always present while the program is running, and of two or more overlays. The overlays are loaded by system calls issued from the root. An overlay facility allows programs to be run even if the programs are too large to fit in memory. Naturally, some care must be exercised to ensure that functions performed by separate overlays do not have to run simultaneously. Also, a program with overlays will execute somewhat slower than one that does not contain overlays.

Advantages of Application Loading

The iRMX 86 Application Loader gives a programmer great flexibility in the way programs use memory. The system can load programs anywhere in available memory, and programs can execute even though they are actually larger than the memory available.

START-UP SYSTEM

The iRMX 86 release package contains a system that is ready to be used. This ready-to-run system is called the start-up system.

Explanation of Start-up System

The iRMX 86 Operating System is a "building block" operating system, with pieces you can put together to create your system. However, the release package that you receive includes a "start-up system", built and ready to run on appropriate hardware. The release package also includes the files which were used to create this start-up system.

Advantages of Start-up System

The start-up system provides these specific advantages:

FEATURES OF THE iRMX™ 86 OPERATING SYSTEM

- You can become familiar with the Operating System immediately, and perhaps even run your application software without rebuilding the Operating System.
- The start-up system has commands to perform common file operations. (See File Maintenance Programs earlier in this chapter.) These commands can be used in developing your application, and may themselves be included in your application.
- The actual files used to create the start-up system provide an example of how to put together an iRMX 86 system.

RUN-TIME BINDING

The iRMX 86 Operating System uses run-time binding. This provides your system with three kinds of flexibility. It allows tasks in different jobs to share objects; it lets your procedures use logical names for files and devices; and it simplifies the process of attaching your application software to the iRMX 86 Operating System.

Explanation of Run-Time Binding

Before we look into run-time binding, let's consider binding as it relates to a program. Binding is the process of letting each program know the locations of the variables and procedures that it uses.

Binding can be performed several times during the development and execution of a program. Some binding takes place during the process of compilation. As a program is being compiled, its references to variables and procedures are resolved (that is, converted into machine language) whenever the compiler has sufficient information. Sometimes, however, a program refers to variables or procedures that are part of a separate program. When this happens, the compiler cannot resolve the reference, and binding must be delayed.

Some binding also takes place during linking. Linking is the process of combining several programs that are compiled separately. The purpose of linking is to allow a program to refer to variables and procedures defined in a different program. (Such references are called external references because they refer to information outside of the program under consideration.) When the linking process resolves an external reference, it performs binding that cannot be completed during compilation.

Run-time binding means binding while the system is actually running. The iRMX 86 Operating System provides three kinds of run-time binding:

- o Binding objects to tasks.
- o Binding files and devices to tasks.
- o Binding your application software to the Operating System.

FEATURES OF THE iRMX™ 86 OPERATING SYSTEM

The first two kinds of run-time binding are based on the use of object directories. An object directory is an attribute of a job that allows tasks to provide ASCII names for objects. Tasks use the CATALOG OBJECT, LOOKUP OBJECT, and UNCATALOG OBJECT system calls to define, lookup, or delete the name of an object. In each case, the task using the system call must specify the job whose object directory is to be accessed.

Now we'll look more closely at each type of run-time binding.

BINDING OBJECTS TO TASKS. When two tasks use a mailbox to pass information, they obviously must both access the same mailbox. But if the programs for the two tasks are compiled and linked independently of one another (as they probably would be if they are in separate jobs), the tasks must use run-time binding to access the same mailbox.

The run-time binding of objects to tasks is accomplished as follows. When a task creates an object that it wishes to share with other tasks, the creator task catalogs the object in an object directory. Other tasks can then access the cataloged object if they know its ASCII name.

Engineers can control the sharing of objects by selectively broadcasting object names. If two engineers wish to share an object, they must agree on both the name and the object directory that is to contain the name. One task then creates the object and the other accesses it through the object directory.

BINDING OF FILES AND DEVICES TO TASKS. Suppose you wish to code an application utility program that takes input from any supported input device or from a disk file. Runtime binding can help accomplish this. The utility program can be coded to lookup an input connection under a particular name. Then any program that needs the utility program can create the input connection, catalog it under the agreed-upon name, and invoke the utility program. In effect, the ASCII name in the object directory is the logical name of the input file.

BINDING OF APPLICATION SOFTWARE TO OPERATING SYSTEM. The iRMX 86 Operating System uses a third type of run-time binding to allow your application software to communicate with the Operating System. Whenever your application software invokes a system call, an INTEL-supplied interface routine converts the call into a software-generated interrupt. This interrupt causes control to be transferred to a procedure within the iRMX 86 Operating System that performs the desired function. In other words, the software interrupts bind the system calls of your application software to the iRMX 86 procedures.

Advantages of Run-Time Binding

Runtime binding provides your application system with flexibility. By allowing your system to name objects, the iRMX 86 Operating System provides a means of sharing dynamically created objects between jobs. By supporting logical names for files and devices, the iRMX 86 System allows tasks to work with any combination of files and devices rather than with a single, fixed combination. By using software interrupts to bind your application software to the Operating System, you can reconfigure the Operating System without having to recompile or relink your application software.

ERROR HANDLING

The iRMX 86 Operating System allows your application system to specify an error handling procedure for each task.

Explanation of Error Handling

Error handling is the process of detecting and reacting to unexpected conditions. The iRMX 86 Operating System supports error handling by doing a substantial amount of validity testing and condition checking within system calls, but it cannot detect every error.

Nonetheless, the iRMX 86 Operating System does protect your system from most types of errors. The concepts involved in the iRMX 86 error handling scheme are exception codes, and exception handlers. We'll look at these one at a time.

- Exception Codes

Whenever a task invokes a system call, the iRMX 86 Operating System attempts to perform the requested function. Whether or not the attempt is successful the Operating System generates an exception code. This code indicates two things. First, it shows whether the system call succeeded or failed. Second, in the case of failure, the code shows which unexpected condition prevented successful completion.

- Exception Handlers

An exception handler is a procedure that the Operating System can invoke when a task receives an exception code indicating failure of the function requested. As each task is created, it is assigned an exception handler; either one written by the programmer, or a default error handler provided by the Operating System. The alternative to using exception handlers is to process exception codes in the procedure that issued the system call.

FEATURES OF THE iRMX™ 86 OPERATING SYSTEM

Because you can write the error handler, you can control the behavior of an application when it receives an exception code. The handler can recover from the error, delete the task containing the error, warn the operator of the error, or ignore the error altogether. The choice is yours.

In summary, exception handling works as follows. The Operating System generates an exception code for each system call. If the code indicates successful completion, the Operating System detected no problems. If the code indicates an exceptional condition, the exceptional condition can be processed either of two ways: within the procedure that invoked the system call, or by the task's exception handler, which is invoked by the Operating System. The technique used is a characteristic of the task, and is established when the task is incorporated into the system.

Advantage of Error Handling

Error handling provides your application system with several methods for reacting to unusual conditions. One of these methods, having the Operating System automatically invoke your task's error handling procedure, greatly simplifies error processing. The other method, dealing with some or all unusual conditions within your application task, allows you to provide special processing for unusual circumstances. The iRMX 86 Operating System allows your application system to use both methods.

DYNAMIC MEMORY ALLOCATION

The iRMX 86 Operating System supports dynamic allocation of memory. This allows you to reduce your implementation costs by building systems in which applications share memory. It also allows your applications to change the amount of memory they use as their needs change.

Explanation of Dynamic Memory Allocation

Although there are numerous techniques for assigning memory to jobs, each technique falls into one of two classes: static allocation or dynamic allocation. Let's look briefly at static allocation first.

Static memory allocation entails assigning memory to jobs when the system is starting up. Once the memory is allocated, it cannot be freed to be used by other jobs. Consequently, the total memory requirements of the system is always the sum of the memory requirements of each job.

Dynamic memory allocation, on the other hand, allows jobs to share memory. Memory is allocated to jobs only when tasks request it. And when a job no longer needs the memory, one of its tasks can free the memory for use by other jobs.

FEATURES OF THE iRMX™ 86 OPERATING SYSTEM

Dynamic allocation also is useful within a job. Some tasks can use additional memory to improve efficiency. An example of this is a task that allocates large buffers to speed up input and output operations.

Advantage of Dynamic Memory Allocation

The dynamic allocation of memory provides your application system with reduced implementation costs. If your application system runs more than one application, chances are fair that memory demands for various jobs will be out of phase. That is, one job will be freeing memory while another needs more. Dynamic memory allocation allows jobs to take advantage of this. Consequently, your application system requires less memory than it would using static allocation.

SOFTWARE INTERFACE

The iRMX 86 Operating System may be used to run various languages translators (PASCAL, FORTRAN, PL/M-86, ASM86 Macro Assembler, etc.). A standard, flexible protocol, the Universal Development Interface (UDI), allows language translators, language run-time packages, and other software development tools to run on the iRMX 86 Operating System.

Explanation of Software Interface

The UDI protocol consists of a set of system calls by which language software uses the Operating System. (Language processors might be compilers, interpreters, assemblers, or run-time systems.) Any language may be run on the iRMX 86 Operating System if the language processor uses the UDI standard system calls. In addition, the same language processor can, without modification, be run on any other operating system which includes the UDI system calls. (Intel markets a variety of operating systems which use UDI for language support.)

Advantages of Software Interface

There are at least two major advantages to the UDI software interface.

- A language processor can use well-defined, appropriate, standard calls to communicate with the iRMX 86 Operating System. Existing languages can be adapted easily to run on the Operating System.
- Any language processor or software tool using UDI system calls can run on several Intel operating systems. This feature is commonly termed "portability," and is becoming a major consideration in software design because of obvious economic benefits.

OBJECT-ORIENTED DEBUGGER

The iRMX 86 Operating System provides a special debugger that is attuned to iRMX 86 objects. This debugger simplifies the process of removing the bugs in the interaction between tasks of the application system. It also facilitates debugging in a real-time environment.

Explanation of an Object-Oriented Debugger

We have already discussed the object-oriented architecture of the iRMX 86 Operating System. Reviewing briefly, each iRMX 86 job is a community of tasks, and each task can manipulate objects. A special set of objects (mailboxes and semaphores) provides a means for tasks to coordinate with one another.

The iRMX 86 Debugger has two capabilities that greatly simplify the process of debugging in a multitasking environment. First, the Debugger allows you to debug several tasks while the balance of the application system continues to run in real time. Second, the debugger lets you see which tasks or objects are queued at mailboxes and semaphores.

These two capabilities help you debug your application system at two levels. You can look into the behavior of an individual task, and you can examine the interaction between tasks. Both levels must be thoroughly debugged before your system is fully implemented.

Advantage of an Object-Oriented Debugger

The object-oriented Debugger gives your application system flexibility while simultaneously providing economic benefits.

- Added Flexibility

By allowing you to debug several tasks while the system continues to run in real time, the Debugger lets you check out new tasks in a running system. This simplifies the process of adding new tasks to an existing application system.

- Economic Benefits

By simplifying the process of debugging the interplay between tasks, the Debugger lessens the amount of time needed to debug your application system. This directly reduces the time to market, the cost of implementation, and the cost of maintenance.

FEATURES OF THE iRMX™ 86 OPERATING SYSTEM

ON-TARGET PROGRAM DEVELOPMENT

You may develop an iRMX 86 application system on a "host" computer system, and then transfer the system to a "target" computer. An alternative is to develop programs on the target computer. With certain hardware and software options, the iRMX 86 Operating System can provide an ideal program development environment. Although on-target program development is not practical for all applications, for some applications it is very worthwhile.

Explanation of On-Target Program Development

The emphasis of most of this chapter has been on building a specialized software product using the iRMX 86 Operating System as a basis for the application. Typically the programs you write are developed on an Intellec Microcomputer Development System, and the system is then "migrated" onto an iSBC 86,88 board.

In contrast, you can develop programs directly on the iSBC 86,88-based hardware by using certain capabilities of the iRMX 86 Operating System. Each of these features has already been described, and here is how they combine to provide a program development system.

- File support. The iRMX 86 file system supports creation of source, object, and loadable files. Many programmers can use the same disk because of the hierarchical structure and protection mechanisms of the iRMX 86 file system.
- Languages and software tools. The Universal Development Interface makes it easy to support language processors and run-time support systems. You can perform all phases of program development using editors, linkers, and other tools of the programming trade; these software tools are available from Intel and run on the iRMX 86 Operating System.
- Convenience. The Application Loader makes it easy to load and execute software. Also, the Human Interface provides a powerful facility for parsing the names of files that are used by language processors, editors, and linkers.
- Debugging. Programs developed on an iRMX 86 Operating System can be debugged using the iRMX 86 object-oriented debugger.

Advantages of On-Target Program Development

On-target program development using the iRMX 86 Operating System can be attractive for these reasons:

- If your application system has spare resources (processing time, memory, mass-storage space) you can use the system more efficiently.
- Programmers can make changes on-site, which has economic and scheduling advantages.

FEATURES OF THE iRMX™ 86 OPERATING SYSTEM

BOOTSTRAP LOADING

The iRMX 86 Operating System contains a bootstrap loader that allows your application system to reside on disk and be loaded into RAM (random-access memory).

Explanation of Bootstrap Loader

A bootstrap loader is a program that resides in ROM on your application hardware. When your iAPX 86,88 microprocessor is reset the bootstrap loader receives control and loads the rest of the software, including the iRMX 86 Operating System and the application software.

Advantages of a Bootstrap Loader

The iRMX 86 Bootstrap Loader provides your application system with two major advantages:

- Reduced Manufacturing Costs

By placing the iRMX 86 Bootstrap Loader in ROM, you can shift the rest of your application system to RAM. Since the rest of your system is probably one or two orders of magnitude larger than the Bootstrap Loader, this displacement substantially decreases the amount of ROM required to implement your application.

This decrease in the amount of ROM required for your application leads directly to reduced manufacturing costs. ROM, unlike RAM, requires that information be "burned" or masked into memory. By decreasing the amount of ROM in your system, the Bootstrap Loader reduces your masking or "burning" expenses.

- Reduced Software Maintenance Costs

The iRMX 86 Bootstrap Loader simplifies the process of providing updated software to your customers. Rather than shipping ROMs containing the modified software, you can ship diskettes. This greatly reduces the cost of updating your software.

CONFIGURABILITY

The iRMX 86 Operating System is configurable. By allowing you to select only the parts of the Operating System that you need, configurability reduces the amount of memory required for your application system.

Explanation of Configurability

A system is configurable if you can select the pieces of it that you want and discard the pieces that you don't want. For example, Figure 4-5 shows a system that consists of six parts. During the process of configuration, you select the desired parts and combine them to form the system.

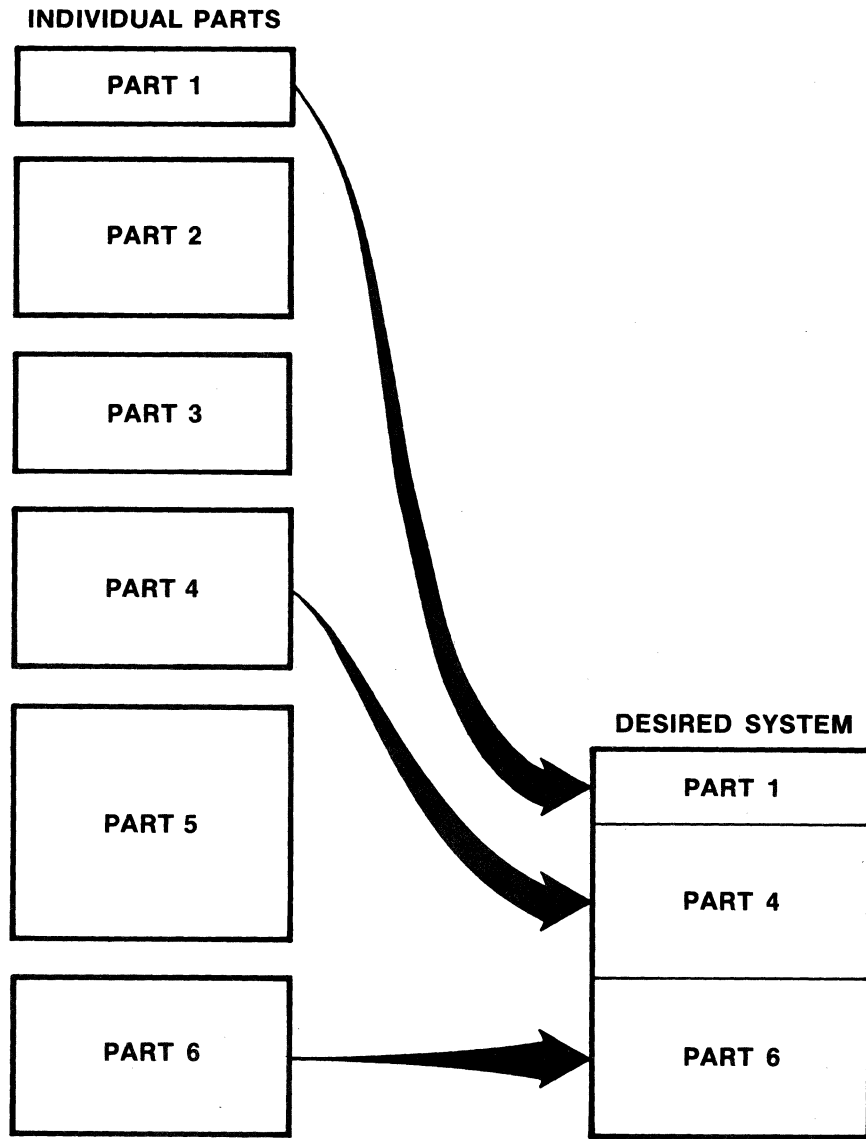


Figure 4-5. Configuration Of A Hypothetical System

FEATURES OF THE iRMX™ 86 OPERATING SYSTEM

When you configure an application system that is based on the iRMX 86 Operating System, you have two objectives. First, you select the parts of the iRMX 86 Operating System that your application system needs. And second, you combine those parts with your application software to form the complete application system. These two objectives are depicted in Figure 4-6.

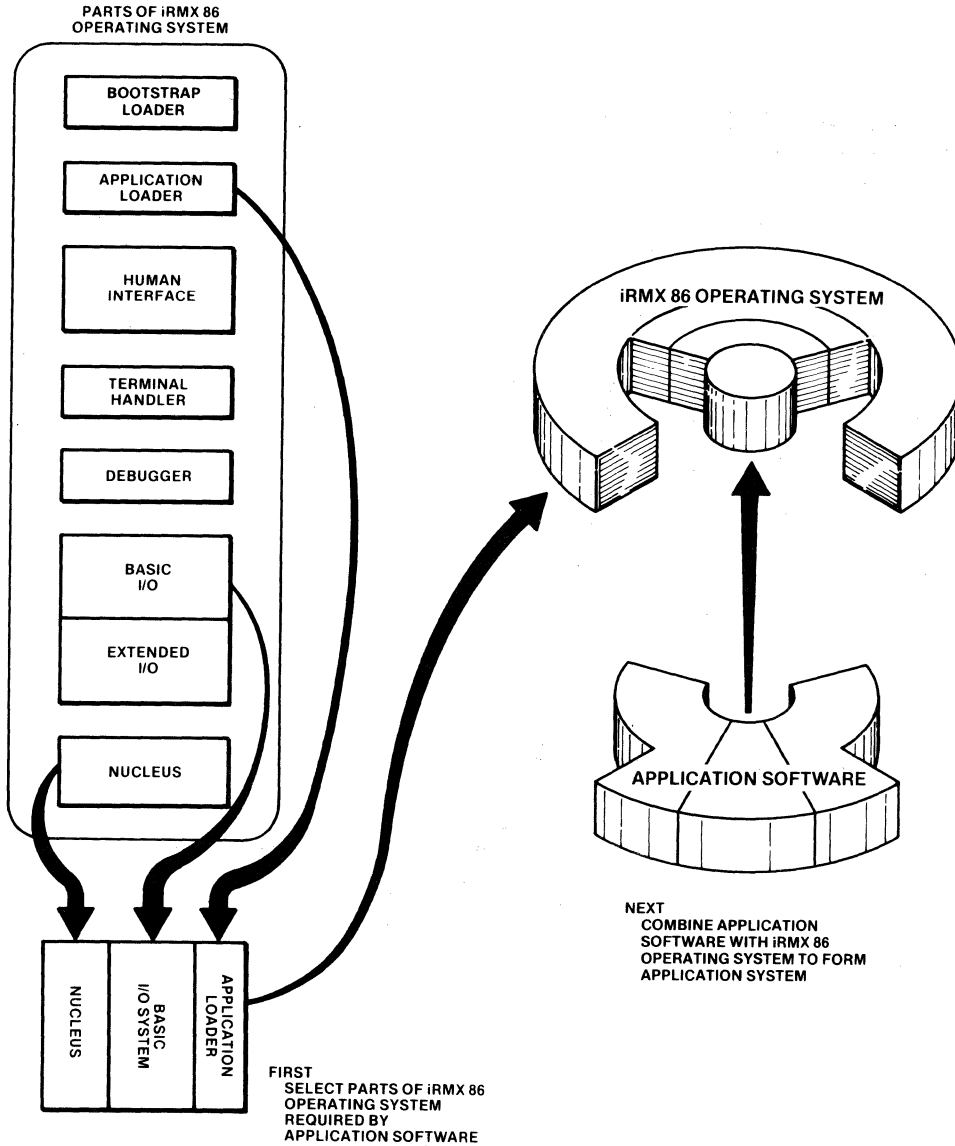


Figure 4-6. The Dual Objective Of iRMX™ 86 Configuration

FEATURES OF THE iRMX™ 86 OPERATING SYSTEM

The iRMX 86 Operating System consists of seven major parts. During the process of configuration you must specify which of these parts are to be included in your application system. The seven parts are:

- The Nucleus

The Nucleus is the heart of the iRMX 86 Operating System. All of the other pieces of the Operating System use the Nucleus, so it must be included in every application system built upon the iRMX 86 Operating System.

- The I/O System

The I/O System provides file management and the device-independent interface to input and output devices. The I/O System is an optional component of the iRMX 86 Operating System, so it can be excluded from the Operating System if it is not needed. The I/O System actually consists of two systems: the Basic I/O System and the Extended I/O System. The user may include the Basic I/O System without including the Extended I/O System. The Extended I/O System requires the Basic I/O System.

- The Human Interface

The Human Interface provides for controlling the application system with commands entered at a terminal. You can create commands, and the Human Interface provides some file-manipulation commands. Like the I/O System, the Human Interface is an optional component and can be left out of the Operating System if it is not required. If the Human Interface is included, it requires the Application Loader and the Extended I/O System.

- The Application Loader

The Application Loader allows your application to load programs and overlays from disk into main memory. The Application Loader is an optional part of a system, but if included requires one or both I/O Systems.

- The Debugger

The Debugger is also an optional component of the iRMX 86 Operating System. While the application system is being developed, the Debugger is a very useful tool. By including it in your system during the development period, you can take advantage of its powerful capabilities. Then, once development is completed, you can remove the Debugger and reduce the size of your finished application system.

- The Bootstrap Loader

The Bootstrap Loader is an optional component of the iRMX 86 Operating System. The Bootstrap Loader reads your application system from disk into main memory whenever you reset the hardware of the application system.

- Terminal Handler

The Terminal Handler is a piece of software that allows you to use a terminal without using the I/O System or Human interface. It is possible to configure the terminal so that it is only an output device.

Advantages of Configurability

Figure 4-6 shows the advantage of configurability. In this figure, an iRMX 86 Operating System consisting of the Nucleus, Human Interface, and I/O System is being combined with application software. By excluding from your finished product the subsystems of the iRMX 86 System that you don't need, you reduce the amount of memory needed by your system.

CHAPTER PERSEPCTIVE

In this chapter we discussed some features of the iRMX 86 Operating System. We also saw some of the advantages that each feature lends your application system. Next we'll see how some of these features work together.

CHAPTER 5. A HYPOTHETICAL SYSTEM

In the previous chapter, you were shown some of the features of the iRMX 86 Operating System. The features were discussed individually. This chapter revisits some of these features using a hypothetical system to show you how features combine to form a powerful environment for your application software.

During the following discussion, a hypothetical application system is used to illustrate the relationship between your application software and the iRMX 86 Operating System. The system monitors and controls a variable number of kidney machines in a hospital. These machines remove toxins from the blood of patients whose kidneys are not functioning correctly.

The system, which is portrayed in Figure 5-1, consists of three main hardware components.

- Intel iSBC 86,88 Single Board Computer

The single board computer provides the intelligence for the entire system. It contains the software to monitor and control the function of all the machines in the system.

- Bedside Units

One of these units is located at the side of each patient's bed. Connected by cable to the iSBC 86 Single Board Computer, each of these units performs four distinct functions:

- Measuring the level of toxins in the blood as the blood enters the unit.
- Displaying information so medical personnel at the bedside can monitor the dialysis process.
- Accepting commands from the bedside personnel.
- Removing toxins from the blood.

Each bedside unit performs these functions under the control of the single board computer. That is, commands and measurements are sent to the single board computer which then adjusts the rate of dialysis and generates the bedside display.

- Master Control Unit

The system's Master Control Unit (MCU) consists of a terminal with a screen and a keyboard. This terminal, which allows one individual to monitor and control the entire system, operates under control of the single board computer.

A HYPOTHETICAL SYSTEM

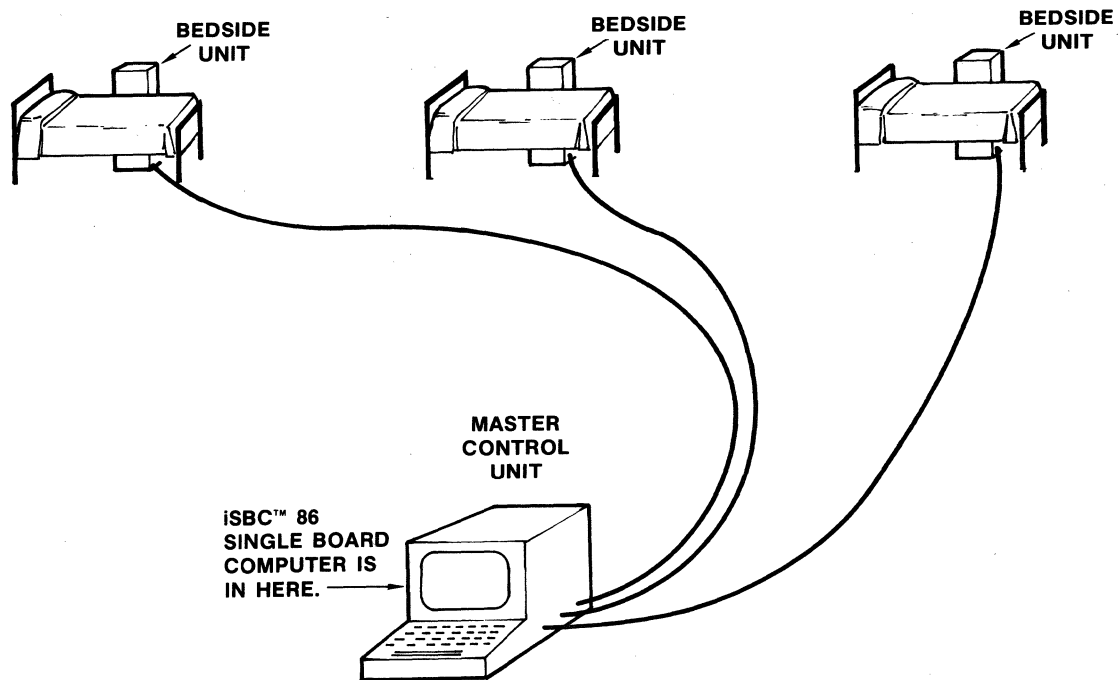


Figure 5-1. The Hardware Of The Dialysis Application System

So, in summary, the system consists of one Master Control Unit and a variable number of bedside units, all operating under control of the software within an Intel iSBC 86 Single Board Computer. Now let's look at the software.

The application software controls the dialysis process. In order to do this, the software must:

- Obtain commands from the Master Control Unit.
- Obtain commands (if there are any) from each of the active bedside units.
- Reconcile the commands from the MCU and the commands of the active bedside units.
- Obtain a toxicity level from each of the active bedside units.
- Create a display at each active bedside unit.
- Create a display at the MCU.
- Control the rate of dialysis at each of the active bedside units.

A HYPOTHETICAL SYSTEM

Now that we have roughly examined the nature of the system, let's investigate how the iRMX 86 Operating System fits in. Let's start with interrupt processing.

INTERRUPT PROCESSING

Two kinds of information flow from the bedside units to the single board computer -- commands and toxicity levels. Before we delve into the technique used to process this information, we must know more about the form of the information.

The toxicity levels, measured as the blood enters the bedside unit, are not subject to violent change. The machine slowly removes toxins from the blood while the patient's body, even more slowly, puts toxins back in. The result is a rather steadily declining toxicity level.

This means that the toxicity levels must be monitored regularly, but not too frequently. Let's suppose that each bedside unit computes the toxicity levels once every ten seconds and sends a signal when the computation is complete. When the signal line goes high, the levels are available until the signal line goes low and then high again for the next computation.

The command information changes less predictably than the toxicity levels. Persons at the patient's bedside can enter commands through the bedside unit. Let's suppose that after encoding the information they press a button labeled ENTER, and that this button sets a line high. When the line goes high, the command information is available until the ENTER button is pressed for the next command.

Now let's see how the interrupt processing of the iRMX 86 Operating System fields the commands. (The toxicity levels can be fielded in precisely the same manner so, for the sake of brevity, they are not discussed.) By attaching all the signal lines to a Multibus interrupt line, we convert the signal into an interrupt level. Each interrupt level has an interrupt task that is executed when the level goes high. So, when the ENTER line from any bedside unit goes high, the interrupt task for bedside commands begins running.

You must write the interrupt tasks for your system's custom devices, so the bedside-command task may serve as an example for you. In brief, the task performs the following steps.

- It determines which bedside unit received the command.
- It puts the command information, along with the number of the bedside unit that received the command, into a message.
- It sends the message to a predetermined mailbox. The only task that waits at this mailbox is the task that reconciles bedside commands with the commands from the Master Control Unit.
- It surrenders the processor to the iRMX 86 Operating System.

A HYPOTHETICAL SYSTEM

One advantage of interrupt processing now becomes clear. Instead of wasting time polling the bedside units to see if a command has been issued, the application system can do other things until interrupted by one of the units. When an interrupt (an event) does occur, it is quickly converted into a message and is placed into a mailbox for processing by a task. The system then returns to its normal priority-based, preemptive scheduling. This technique enables your system to deliver more throughput.

Interrupt processing also provides the application system with flexibility. For instance, you can add more bedside units without modifying the system's software at all.

HUMAN INTERFACE

The iRMX 86 Human Interface allows interaction between medical personnel and the system to be "human engineered". Information can be requested and displayed at the MCU, and at bedside units, in a form that is meaningful to the operators of the system. Also, new capabilities may be added to the system by simply adding new programs.

MULTITASKING

The entire application system is based on the multitasking capability of the iRMX 86 Operating System. Tasks are run using the preemptive, priority-based, scheduling that was discussed in Chapter 4. This allows the more important tasks (such as those controlling the bedside units) to preempt lower priority tasks (such as those of the Terminal Handler).

INTERTASK COORDINATION

The only form of intertask coordination used in our hypothetical dialysis system is intertask communication. The system uses a number of mailboxes to send information from one task to another. The simplicity of mailboxes allows engineers to divide the system into tasks on the basis of modularity, rather than on the basis of minimizing intertask communication.

MULTIPROGRAMMING

Although multiprogramming has not yet been of use in our hypothetical example, its potential is high. Suppose that we extend the example to include cardiac monitoring in addition to dialysis. The two functions could advantageously be performed in different jobs. Why? Because they need share very few resources.

A HYPOTHETICAL SYSTEM

If the cardiac application has very little to do with the kidney application, there is no need for them to share mailboxes, tasks, or any other objects. By splitting them into two different jobs, there is less chance that one application can affect the environment of the other.

But what happens if the two applications need to share only a little information? How can the shared data be passed from one job to another without losing the benefits of isolation? The iRMX 86 Operating System provides for this contingency in its implementation of run-time binding.

RUN-TIME BINDING

As mentioned earlier in this manual, run-time binding provides a means for tasks of different jobs to share objects. As tasks create objects to be shared, the tasks catalog the objects in an object directory. Then the tasks that need the objects can look them up by using their ASCII names.

Runtime binding also allows you to change the configuration of the iRMX 86 Operating System without recompiling or relinking your application software. For instance, suppose you have been including the iRMX 86 Debugger in systems delivered to your customers. The advantage in doing this is that it allows some debugging on systems as they are being used. But now, a year or so after you started delivering systems, your product has stabilized. Virtually no new bugs are being found. If you delete the Debugger from your system, you can reduce the amount of memory required in any new systems you sell. The run-time binding of the system to your application software allows you to remove the Debugger from your system without making any changes to your application software.

MASS STORAGE FILES

As specified, the hypothetical system does not require mass storage files. However, a very reasonable extension of the current specification could include recording information about patients.

The iRMX 86 I/O System allows you to record information in files on flexible and hard disks. The system provides device handlers, disk formatting and allocating, and provides a way to move information between main memory and the disk. Your application software need not include code to perform these functions.

If mass storage devices were added to the system, it would be possible to do program development on the system, so that new programs could be written and tested at the sit. This is a powerful addition to a system, although it is not appropriate for every application.

A HYPOTHETICAL SYSTEM

DEVICE INDEPENDENCE

Even if the application system uses mass storage devices, device independence is not necessarily required. But, if the application is extended to allow the operator at the MCU to direct recorded data to any of several devices (say teletypewriter, line printer, magnetic tape or disk), device independence becomes more important. The device-independent I/O System lets you implement recording without adding code specific to each possible device.

CHAPTER PERSPECTIVE

In this and the previous chapters, you were introduced to some of the features and benefits associated with the iRMX 86 Operating System. If you want more detailed information, you will find the next chapter very useful. It contains descriptions of all the iRMX 86 technical manuals.

CHAPTER 6. iRMX™ 86 LITERATURE

This chapter lists the iRMX 86 Operating System documentation, describes each manual in the set, and correlates the manuals with the features described in previous chapters. The first six manuals listed here are: the manual you are reading, plus manuals describing each major "layer" of the Operating System. Otherwise the order in which the manuals are described is arbitrary.

Manual	Number
- G Introduction to the iRMX™ 86 Operating System	9803124
- ✓ iRMX™ 86 Nucleus Reference Manual	9803122
- ✓ iRMX™ 86 Basic I/O System Reference Manual	9803123
- ✓ iRMX™ 86 Extended I/O System Reference Manual	143308
- ✓ iRMX 86 Human Interface Reference Manual	9803202
- ✓ iRMX™ 86 Loader Reference Manual	143318
- ✓ iRMX™ 86 System Programmer's Reference Manual	142721
- ✓ iRMX 86 Installation Guide	9803125
- ✓ iRMX™ 86 Configuration Guide	9803126
iRMX 86 Debugger Reference Manual	143323
- G iRMX™ 86 Programming Techniques	142982
- G Guide to Writing Device Drivers for the iRMX 86™ and iRMX 88™ I/O Systems	142926
✓ iRMX™ 86 Terminal Handler Reference Manual	143324
- iRMX 86™ Disk Verification Utility Reference Manual	144133
✓ Run-time Support Manual for iAPX 86,88 Applications	121776

Each of these manuals is designed for a well-defined set of readers and each has a different purpose. This chapter describes the readers at whom each manual is aimed and the purpose of each manual. (Table 6-1 correlates features with manuals.) Also, the chapter provides some time-saving tips to bear in mind as you read the documentation.

The following descriptions deal with engineers in two classes -- system programmers, and application programmers. System programmers are responsible for configuring the system, extending the Operating System, writing interrupt handlers, and performing other functions that affect the entire application system. Application programmers, on the other hand, are responsible for writing application software. This distinction is drawn because the actions of the system programmer have a more global affect.

Specifically, some system calls can, if used improperly, cause problems for all the tasks in your system; other system calls can affect only the task invoking the call. As a matter of policy, the more powerful system calls should be used only by system programmers and, even then, only within Operating System extensions. To emphasize this distinction, the more powerful system calls are all explained in one manual, the SYSTEM PROGRAMMER'S REFERENCE MANUAL.

READING TIPS

The following pointers can save you a substantial amount of time:

- No one individual need become intimately familiar with all of the documents associated with the iRMX 86 Operating System. Read only the documents that relate to your responsibilities.
- Before reading one of the documents, read its preface and scan its table of contents to see if the manual contains the kind of information you seek.
- Read the Introductory Manual before reading any of the others.

By following these tips, you can quickly focus your attention on the information that is of most value to you.

INTRODUCTION TO THE iRMX 86 OPERATING SYSTEM

This manual, the one you are presently reading, is aimed at a wider variety of readers than any of the other iRMX 86 manuals. Being an introduction, it can be understood by anyone who has some experience programming or managing programming projects. It is designed to introduce managers and engineers to the iRMX 86 Operating System.

iRMX 86 NUCLEUS REFERENCE MANUAL

This reference manual is written for any engineers planning to use the iRMX 86 Operating System. It is the information warehouse for the Nucleus. It contains concise but detailed discussions of the following topics:

- The nature of objects in general and of tasks, jobs, semaphores, mailboxes, and segments in particular.
- Error processing.
- Interrupt processing.
- The requirements and behavior of the Nucleus system calls available to all engineers.

Each of these topics is presented in a manner suitable for reference purposes and, to a lesser degree, for instructional purposes.

iRMX 86 BASIC I/O SYSTEM REFERENCE MANUAL

This manual describes the iRMX 86 Basic I/O System. The manual contains descriptions of the I/O operations a programmer may use with the iRMX 86 Operating System, and of the types of files supported (named, stream, and physical). The heart of the manual is a chapter describing all the information necessary to use each Basic I/O System call, including:

- calling sequence,
- description of parameters, and
- explanation of exception codes which may be returned by each call.

The Basic I/O System is the more flexible of the two I/O systems delivered with the iRMX 86 Operating System. Because most of the system calls are asynchronous, the programmer is given full control of how I/O and processing will be synchronized in an application.

iRMX 86 EXTENDED I/O SYSTEM REFERENCE MANUAL

This manual describes the iRMX 86 Extended I/O System, the easier-to-use I/O system. It includes descriptions of the types of iRMX 86 files, and a general description of the Extended I/O System. Because the manual's chief use is as a reference (although it also serves as a tutorial), the most useful part of the manual is a chapter describing all the information necessary to use each Extended I/O System call, including:

- calling sequence
- description of parameters, and
- explanation of exception codes which may be returned by each call.

The Extended I/O System is specifically designed to unburden programmers from details of I/O operations. In particular, Extended I/O data transfers are synchronous, meaning that the operating system performs multiple-buffering operations, automatically synchronizing I/O operations with processing.

iRMX 86 HUMAN INTERFACE REFERENCE MANUAL

The iRMX 86 Human Interface is an optional layer of the iRMX 86 Operating System. The Human Interface allows operators to load and run programs from a console terminal, and provides facilities for custom commands. In addition, the Human Interface has a number of file-maintenance commands (COPY, FORMAT, RENAME, etc.).

If you will want to use the powerful facilities provided by the Human Interface, the manual provides the information you will need, including:

- Description of all Human Interface commands.
- Descriptions of Human Interface system calls. The system calls are used for parsing of custom commands, control of programs run by the Human Interface, sending and receiving messages, and general command control functions.
- Explanation of logical naming of devices and files, and descriptions of standard logical names.

iRMX 86 LOADER REFERENCE MANUAL

This manual describes the Bootstrap Loader and the Application Loader. The Bootstrap Loader loads an application system from secondary storage (disk, diskette, bubble-memory) into memory, and transfers control to the system. It works in conjunction with a program in ROM which is run when the iAPX 86,88 Processor is reset. The Loader Reference Manual describes how to use the Bootstrap Loader, and secondary storage devices from which systems can be loaded.

iRMX™ 86 LITERATURE

The Application Loader is used for two purposes:

1. To load and run programs that reside on secondary storage. These programs are invoked by Human Interface commands.
2. To load overlays by invoking system calls.

The Loader Reference Manual describes the types of code which may be loaded by the Application Loader (absolute, position-independent, and load-time locateable). The Manual also describes the information necessary to use the Application Loader system calls.

If your application uses either the Bootstrap Loader or the Application Loader (most applications use one or both) then you will want to use this manual. As with most of the iRMX 86 manuals, this one serves primarily as a reference manual, secondarily as a tutorial.

iRMX 86 SYSTEM PROGRAMMER'S REFERENCE MANUAL

This manual describes aspects of the iRMX 86 Operating System that should be used only within extensions of the iRMX 86 Operating System. If these features are used by application software that has not been incorporated into the Operating System, your application system might not be compatible with future releases of the iRMX 86 Operating System.

This reference manual discusses the following topics:

- The creation and deletion of extensions to the Operating System.
- Region exchanges and related system calls.
- Enabling and disabling the deletion of the objects.
- Attaching and detaching devices.
- Creation and deletion of user objects.
- Adding new types of objects to the Operating System.
- Using the Bootstrap Loader.

iRMX 86 INSTALLATION GUIDE

The INTELLEC Microcomputer Development System is a general purpose tool for programming microcomputers. When you purchase the iRMX 86 Operating System, you receive the iRMX 86 software on several flexible disks and you receive the iSBC 957B package. The iRMX 86 INSTALLATION GUIDE tells you how to use the software and the iSBC 957B package (or the 957A if you have an earlier release) in conjunction with your Microcomputer Development System and an iSBC 86,88 Single Board Computer.

iRMX™ 86 LITERATURE

The manual describes everything necessary to get the start-up system running. This includes instructions for installing wire jumpers on an iSBC 86/12 board, loading and running the start-up system, and using the Human Interface commands included in the start-up system. If you are familiar with the Intellec Microcomputer Development System, this manual will prove very useful.

iRMX 86 CONFIGURATION GUIDE

As you build an application system upon the iRMX 86 Operating System, you must decide which optional iRMX 86 features you want in your system. For instance, if your system uses no features of the I/O System, you can save a substantial amount of memory by excluding the iRMX 86 I/O System.

Once you have made these decisions and have written your application software, you must configure your system. Configuration is the process of building a complete system from the iRMX 86 Nucleus, your application software, and iRMX 86 options that you have selected. The iRMX 86 CONFIGURATION GUIDE leads you through the process of configuration.

The readers of this manual must be thoroughly familiar with the ISIS Operating System and the language translators being used to create the application software. Furthermore, they must know which parts of the iRMX 86 System are used by the application software, and they must be fluent in the vocabulary of the iRMX 86 Nucleus, Terminal Handler, Debugger, I/O System, Bootstrap Loader, Human Interface, and Application Loader.

iRMX 86 DEBUGGER REFERENCE MANUAL

This manual describes the debugger used with the iRMX 86 Operating System. It explains the requirements and behavior of the Debugger commands available to all engineers.

iRMX 86 PROGRAMMING TECHNIQUES

This manual provides system and application programmers with techniques that can save time and avoid mistakes during system development.

GUIDE TO WRITING DEVICE DRIVERS FOR THE iRMX 86 AND iRMX 88 I/O SYSTEMS

This manual gives detailed instructions for writing a device driver that is compatible with both the iRMX 86 I/O System and the iRMX 88 I/O System. System programmers can use this manual to add new devices to application systems. Readers of this manual must be very familiar with the I/O System and the Nucleus.

iRMX 86 TERMINAL HANDLER REFERENCE MANUAL

If you wish to use the iRMX 86 Operating System without using the iRMX 86 I/O System, you might need software to communicate with a terminal. This is the function of the Terminal Handler. The iRMX 86 Terminal Handler provides basic functions:

- Line editing (special handling of RUBOUT key, certain control keys, and carriage return).
- Character echoing as a line buffer is filled.

The Terminal Handler Reference Manual describes the use of special keys, how the Terminal Handler communicates through mailboxes, and an option called Output-Only Terminal Handler.

If your application is not using the I/O System or Human Interface to communicate with terminals, you will be interested in the information in this manual.

iRMX 86 DISK VERIFICATION UTILITY REFERENCE MANUAL

This manual documents the Disk Verification Utility, a software tool that runs as a Human Interface command, verifying and modifying the data structures of iRMX 86 disk structures (files, directories, and physical volumes). The manual describes the how to invoke the utility and contains detailed descriptions of all utility commands. Because users of the Disk Verification Utility must be familiar with the structure of iRMX 86 volumes in order to use the utility intelligently, the manual describes in detail the structure of iRMX 86 file and directory structures.

You will use this manual in maintaining disk file volumes, and perhaps in recovering files which are "lost" or corrupted.

RUN-TIME SUPPORT MANUAL FOR iAPX 86,88 APPLICATIONS

This manual describes facilities that Intel provides to develop an application system on the iAPX 86,88 processors, and describes how to use these facilities. The organization of the manual reflects the concept of application system "layers." Subjects discussed include language translators and utilities, the development process using Intel languages, and interaction between hardware and operating systems (including the iRMX 86 Operating System).

You will be interested in this manual if you need a general introduction to the application development process. Of particular interest is a discussion of Intel's Universal Development Interface (UDI), by which language translators and utilities use operating systems.

Table 6-1. Correlation of Manuals and Features

TITLE	FEATURE
iRMX 86 Nucleus Reference Manual	Object-Oriented Architecture Multiprogramming Multitasking Interrupt Processing Preemptive, Priority-based Scheduling Error Handling Dynamic Memory Allocation Intertask Coordination Run-time Binding
iRMX 86 Basic I/O System Reference Manual and iRMX 86 Extended I/O System Reference Manual	Choice of I/O Systems Hierarchical Naming of Mass Storage Files File Access Control Control of File Fragmentation Application Loading Device-Independent I/O
iRMX 86 Human Interface Reference Manual	Custom Interactive Commands File Maintenance Utilities
iRMX 86 Loader Reference Manual	Bootstrap Loader Application Loading
iRMX 86 Installation Guide	Start-Up System Configurability
iRMX 86 Configuration Guide	Configurability
iRMX 86 System Programmer's Reference Manual	Intertask Coordination Expandibility Selection of Device Drivers Run-time Binding File Access Control Device Independence
iRMX 86 Debugger Reference Manual	Object-oriented Debugger
Guide to Writing Device Drivers for the iRMX 86 and iRMX 88 I/O Systems	Selection of Device Drivers Device-Independent I/O
Run-time Support Manual for iAPX 86,88 Applications	Software Interface

INDEX

For your convenience, the primary reference of topics having multiple references are underscored.

access time 4-21
application 1-2, 4-7
application loading (Application Loader) 4-25 to 4-26, 4-33, 4-37, 6-4
application software 1-2
application system 1-2
ASM 86 Macro Assembler 4-31
asynchronous 2-1, 4-4, 4-11
attaching devices 6-5

Basic I/O System 4-13, 4-14 to 4-15, 6-3
binding 4-27 to 4-29, 5-5, 6-8
bootstrap loading 4-34, 4-37, 6-4
bubble memory 4-16, 4-26

commands 4-22, 4-24 to 4-25, 6-5, 6-8
compilers 4-31
concurrency 4-4, 4-11
configuring 4-34 to 4-38, 6-6
COPY (command) 4-22
cost of implementation 3-2
costs after development 3-2
CREATEDIR (command) 4-22
custom interactive commands 4-24 to 4-25
customizing features 4-24

data transfer rate 4-21
debugging 2-3, 4-32, 4-33, 6-6
detaching devices 6-5
development time 3-1
device drivers 4-21 to 4-22, 6-6
device independence 2-2, 4-15 to 4-16, 5-6, 6-8
device selection 2-2, 4-16, 4-21 to 4-22
DIR (command) 4-22
directories
 file 4-16 to 4-19, 4-33, 6-3, 6-7
 object 4-27 to 4-28, 5-5, 6-3
disk, diskette 4-16, 4-20, 4-22 to 4-23, 4-25, 4-33, 4-34, 6-7
documentation 6-1 to 6-8
DOWNCOPY (command) 4-22
dynamic memory allocation 2-2, 4-7, 4-30 to 4-31

INDEX (continued)

editing 4-33
error processing 2-1, 4-29 to 4-30, 6-3
events 2-1, 4-4 to 4-5, 4-8 to 4-9
example of application system 5-1 to 5-6
exception codes 4-31, 6-3 to 6-4
exception handlers 4-29 to 4-30
exchanging information 4-8 to 4-9
Extended I/O System 4-13 to 4-14, 6-4, 6-8
extending the Operating System 4-12, 6-5

file
 access control 2-3, 4-19 to 4-20, 6-3 to 6-4, 6-7
 fragmentation 4-20 to 4-21, 6-8
 granularity 2-2, 4-20 to 4-21
 naming 4-16 to 4-19, 6-3
 maintenance 4-22 to 4-23
 protection 2-3, 4-19 to 4-20
 sharing 2-3, 4-7, 4-19 to 4-20, 4-33
FORTRAN 4-3, 4-31

granularity of files 2-2, 4-20 to 4-21, 6-8

hierarchical file naming 4-16 to 4-19, 6-8
human engineering 2-3, 4-24
Human Interface 4-24 to 4-25, 4-33, 5-4, 6-4

iAPX 86,88 microcomputer iii, 1-1, 6-7
iSBC 86,88 single-board computer 1-1, 4-33, 4-34, 5-1
iRMX 86 Operating System
 architecture 4-2 to 4-4
 benefits 3-1 to 3-2
 characteristics 1-1
 customers 1-1
 documentation 6-1 to 6-8
 features 4-1 to 4-38, 6-8
 installation 6-5
 purpose 1-3
iRMX 88 Operating System 6-6
implementation cost 3-2
input and output 4-13 to 4-23, 4-37, 5-5 to 5-6, 6-3 to 6-4, 6-8
installation of iRMX 86 6-5
interactive terminal operations 4-24 to 4-25, 5-4, 6-4
interpreters 4-31
interrupts 4-5 to 4-6, 5-3 to 5-4, 6-3, 6-8
intertask coordination 4-8 to 4-12

job 4-7 to 4-8, 4-30, 5-5, 6-3

languages (computer programming) 4-31
literature 6-1 to 6-8
loading
 application 4-25 to 4-26, 6-4 to 6-5, 6-8
 bootstrap 4-34, 6-5
load-time location 4-26
logical names 4-27

INDEX (continued)

nucleus 4-37

mailbox 4-9, 4-32, 5-4, 6-3

maintenance of application system 3-2

maintenance of files 4-22 to 4-23

manuals 6-1 to 6-8

mass storage (devices and files) 2-2 to 2-3, 4-16, 4-20 to 4-21, 4-24

memory allocation 2-2, 4-30 to 4-31, 6-8

multiprogramming 2-2, 4-7 to 4-8, 5-5, 6-8

multitasking 4-4 to 4-5, 4-8, 5-4, 6-8

mutual exclusion 4-7, 4-10 to 4-11

object directories 4-28, 5-5

object names 4-27, 5-5

object-oriented architecture 4-2 to 4-4, 4-32

objects (types of)

- general discussion 4-2, 4-32, 5-5, 6-3
- custom (extension) 4-12, 6-5
- job 4-7 to 4-8
- mailbox 4-9
- region 6-5
- semaphore 4-10 to 4-11
- task 4-5 to 4-6

OEM 1-1

on-target development 4-33

operator 4-3, 4-7, 4-18 to 4-20, 4-24 to 4-25, 4-30, 5-4, 5-6, 6-4

original equipment manufacturer (OEM) 1-1

overlays 4-25 to 4-26, 6-4, 6-8

parsing 4-24 to 4-25

PASCAL 4-31

PLM/86-86 4-31

polling 4-5, 5-4

portability 4-31

program development 4-33

protocol (UDI) 4-31

priority-based scheduling 4-6, 5-4, 6-3, 6-8

read-ahead 4-14

real-time programming 2-1

real-time software 2-1

regions 6-5

RENAME (command) 4-22

runtime binding 4-27 to 4-29, 5-5, 6-3, 6-8

runtime support 4-31, 4-33, 6-7

scheduling 2-1, 4-5, 4-6 to 4-7, 5-4, 6-3, 6-8

segment 4-10,

semaphore 4-10 to 4-11, 4-32, 6-3

software development tools 4-31, 4-33, 6-7

software interface (UDI) 4-31, 4-33, 6-7, 6-8

software interrupt 4-28

start-up system 4-26 to 4-27, 6-5, 6-8

SUBMIT (command) 4-22

synchronization 4-11 to 4-12

system calls 4-3, 4-13 to 4-15

INDEX (continued)

task 4-2, 4-5 to 4-6, 4-8, 4-32, 5-4 to 5-5, 6-3
terminal handler 4-38, 6-7, 6-8
time-slicing 4-6
types of objects 4-3, 4-12, 6-5

Universal Development Interface (UDI) 4-31, 4-33, 6-7, 6-8
UPCOPY (command) 4-22
user 1-2
user objects 6-5

Very Large Scale Integration (VLSI) 3-1
VEU 1-1
volume end users 1-1

write behind 4-14



REQUEST FOR READER'S COMMENTS

Intel Corporation attempts to provide documents that meet the needs of all Intel product users. This form lets you participate directly in the documentation process.

Please restrict your comments to the usability, accuracy, readability, organization, and completeness of this document.

1. Please specify by page any errors you found in this manual.

2. Does the document cover the information you expected or required? Please make suggestions for improvement.

3. Is this the right type of document for your needs? Is it at the right level? What other types of documents are needed?

4. Did you have any difficulty understanding descriptions or wording? Where?

5. Please rate this document on a scale of 1 to 10 with 10 being the best rating. _____

NAME _____ DATE _____

TITLE _____

COMPANY NAME/DEPARTMENT _____

ADDRESS _____

CITY _____ STATE _____ ZIP CODE _____

Please check here if you require a written reply.

WE'D LIKE YOUR COMMENTS . . .

This document is one of a series describing Intel products. Your comments on the back of this form will help us produce better manuals. Each reply will be carefully reviewed by the responsible person. All comments and suggestions become the property of Intel Corporation.



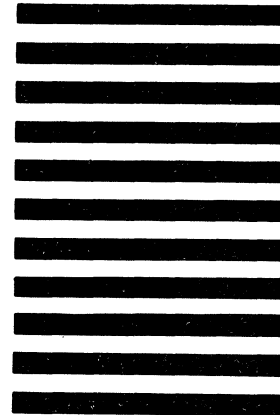
**NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES**

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 79 BEAVERTON, OR

POSTAGE WILL BE PAID BY ADDRESSEE

Intel Corporation
5200 N.E. Elam Young Pkwy.
Hillsboro, Oregon 97123

O.M.S. Technical Publications







INTEL CORPORATION, 3065 Bowers Avenue, Santa Clara, California 95051 (408) **987-8080**

Printed in U.S.A.