TI-MIX 1983
International Symposium

April 5-8, 1983
New Orleans Hilton Hotel

Session Proceedings

Operating Systems

OPERATING SYSTEMS

TABLE OF CONTENTS

PAPERS NOT AVAILABLE BY PUBLICATION DEADLINE:

TI-MIX (Texas Instruments Mini/Microcomputer Information Exchange) is an organization for users of TI computers and related equipment. The purpose of TI-MIX is to promote the exchange of information between users and TI. Membership in TI-MIX is open to any person with an interest in TI computers or peripheral equipment. The international symposium provides a vehicle for direct interaction and information exchange with other users and with TI personnel. Acceptance of TI-MIX member papers for presentation at TI-MIX 1983 does not constitute an endorsement by TI-MIX or Texas Instruments Incorporated.

TI-MIX
M/S 2200
P.O. Box 2909
Austin, Texas 78769
(512) 250-7151

USES OF DNOS SEGMENTATION

by

S. Ken Culp
Texas Instruments
Austin, Texas

Operating Systems Session

1. SEGMENTATION FUNCTIONALITY

1.1 CURRENT OVERLAY ARCHITECTURE

On 990/10 and 990/12 mini-computers, the mapping hardware allows a user program to be divided into three separate pieces of physical memory. These three pieces of physical memory are combined into one contiguous logical address space of up to 65,536 bytes of memory. For many applications, this logical address space is insufficient which led to the concept of overlays being developed.

By operating system definition, one of the three segments is designated as the task segment, a unique one of which is required for every task. In addition to the task segment, one or two procedure segments can be added to the address space. Under DX10, these procedure segments must precede the task segment but may be shared by other tasks running concurrently. Figure Figure 1-1 below illustrates the task structure under DX10.

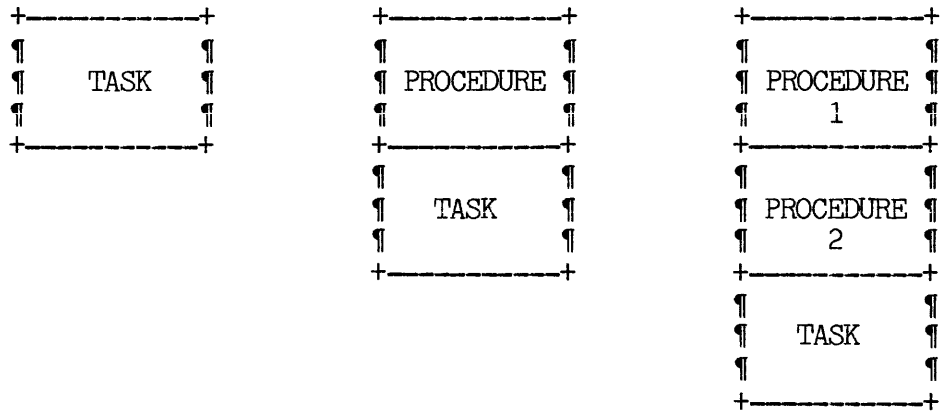


Figure 1-1 TASK STRUCTURE OPTIONS -- NO SEGMENTATION

Under DX10, the only escape from the address space restriction is an overlay. Overlays are a piece of code read from disk normally into the task segment and require a minimum of two disk accesses each time a new overlay is loaded. Since the loading of an overlay is simply the reading of a piece of a disk file over a portion of the task address space, the information previously stored at the overlay addresses is destroyed. Therefore, read/write data (DSEG's) must be saved external to the overlay area even if the data is only

required locally to the code in the overlay.

Overlays include the facility for automatic overlay loading by linking in the Overlay Manager and modifying the routine entry points to point to the Overlay Manager. Also, a hierarchical structure (Phases) is permitted.

1.2 GENERAL CAPABILITIES OF PROGRAM SEGMENTS

Under DNOS, procedure segments may precede the task segment as on DX10. Additionally, new segments may be mapped after the task segment and theseare called program segments. Furthermore, these segments may be changed for other segments at the option of the running task by simply issuing Change Segment Supervisor Calls (SVC's). Although the Change Segment SVC allows changing procedure segments preceding thetask, most segmentation changing is done after the task segment so the segments can be of variable length. (Swapping in variable length segments before the task segment would result in the relocation of the task segment with unworkable problems resulting). These segments need not necessarily be loaded from disk. In one case, the segments may be made memory resident at boot time and never loaded from disk. Also, the segments may remain cached in memory when not in use and then mapped in with zero disk accesses. In another case, an unmapped segment may not be in memory due to high memory requirements but can be remapped by the task in only one disk access. It is this characteristic of segmentation which allows the greatest amount of performance improvement through reduced disk activity.

Another feature of segments not available with overlays is the ability to modify DSEG's in segments, map the segment out, then map the segment in again without loss of the data in the DSEG. In this fashion, large blocks of data can be accessed by a task without using a disk file to buffer and stage the data.

The DNOS Link Editor does support linking multiple program segments in a single execution of the linker (one link control file), but does not support automatic segment loading or a hierarchical structure for those segments. Multiple segments are linked by providing multiple segment commands in the link control file (see the DNOS Link Editor Reference Manual, P/N 2270522-9701 *A). Figure Figure 1-2 below illustrates the task structure under DNOS.

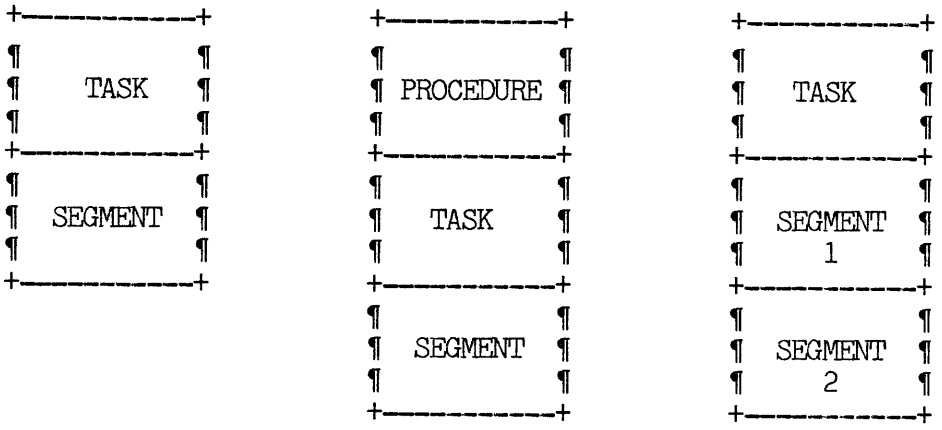


Figure 1-2 TASK STRUCTURE OPTIONS -- SEGMENTATION

1.3 SEGMENTATION: CAPABILITIES AND LIMITATIONS

DNOS program segments are normally loaded by specific request from the task. Program segments mapped after the task segment are not automatically loaded at task bid time but must be loaded by specific request (SVC) of the task. The map SVC does, however, allow all segments to be exchanged at runtime except the task segment (or a segment containing the Map SVC itself).

The DNOS linker supports linking multiple procedure segments in one link step but only at one map position in the address space That means that the task structure must be TASK/SEGMENT or PROCEDURE/TASK/SEGMENT. In terms of linker control commands, there cannot be both a SEGMENT 2 and SEGMENT 3 command in the same link control file.

Under DX10 or DNOS, the DSEG's and CSEG's contained in procedures linked before the task segment are migrated to the task segment to help in constructing reentrant procedure segments. In a similar fashion, the CSEG's referenced within multiple program segments linked following the task segment are promoted up to the task segment. DSEG's referenced in program segment links are, however, not promoted to the task segment nor reordered within the program segment. Therefore, if the DSEG must be in the task, it should be assembled separately, REF'd, and included in the task segment with an explicit include command. Alternately, the DSEG can be made into a CSEG which is promoted to the task segment (if referenced by multiple segments). The advantage of using a CSEG is that every label in the DSEG would have to be externally DEF'd and REF'd (if not included in the assembly step of the procedure referencing it).

If a program must be structured as TASK/SEGMENT/SEGMENT, multiple links will have to be used. Under these conditions, the actual load addresses of the segments would have to be specified in the link control stream on the SEGMENT command. Under these conditions, routines at the SEGMENT 2 level cannot reference labels defined at the SEGMENT 3 level and vice versa (and have those references resolved by the linker). If references are desired across the map locations, then a table of routine addresses could be placed in the task segment and subroutine calls be made indirect through that table.

The linker does not directly support linking multiple procedure segments preceding the task at the same map position (ie, linking multiple procedure 1's of Figure 1-2). Similar to linking segments at both map positions 2 and 3, these multiple segments can be linked with multiple link steps but references from the task to the procedure segments (and subsequent program segments) cannot be resolved by the linker.

Overlays can be linked and loaded into program segments but automatic overlay loading is supported only into the task.

1.4 CHARACTERISTICS OF SEGMENTS

Most of the following characteristics of segments are set via the Install Procedure Segment (IPS) SCI command or via the Modify Segment Entry (MSE) command. If a format image link is used, then the MSE approach is required.

1.4.1 Executable or Execute Protect.

This hardware option is definable on any DNOS but functions only on the 990/12 CPU. If execute protect is set and the Program Counter (PC) is transferred to the segment, a task error >A (execute protect violation) occurs. If the flag is not set, the segment may be executed.

1.4.2 Read Only or Read/Write.

This hardware option is definable on any DNOS system but functions only on the 990/12 CPU. If the segment is flagged as read only (as would be set for code or non-modifiable data) and an attempt is made to write to the segment, then a task error >B (write protect violation) occurs. All segments containing writable DSEG's should be flagged as Read/Write.

1.4.3 Sharable or Non-Sharable.

This software flag indicates to the O/S whether multiple tasks can simultaneously have the segment mapped into their respective address spaces. Most pure code segments that can be mapped by multiple tasks (at the same

logical address) would be flagged as sharable. Also, segments that contain data that is simultaneously needed by multiple tasks would be flagged as sharable. Private data for a task would not be flagged as sharable.

1.4.4 Replicatable.

This software flag indicates to the O/S whether multiple copies of a segment can exist at one time. Note that this flag interacts with the Sharable flag as follows: 1) if a segment is sharable, there would be no need for multiple copies; 2) if a segment is non-sharable, each task is allowed to use his own copy (as for routines with both code and local data) if the segment is replicatable; or may be excluded from having his own copy and be forced to wait for the one copy to become available if the segment is non replicatable.

1.4.5 Reusable or Non-Reusable.

This software flag indicates whether a segment that is mapped out and no longer needed by a task (see Reserve and Exclusive below) can be used by another task upon a map request. If the reusable flag is set, the segment may be cached when not in use. If reusable is not set, the segment is discarded when no longer in use.

1.4.6 Updatable.

The updatable option is set during IPS or MSE and indicates that the O/S may use the home program file as the location to swap a segment when that segment must be rolled out. This flag must be set if the Forced Write Segment SVC is used. This option is normally only needed for data segments whose contents must exist beyond the life of one task or system boot. Writing a segment to the home file is equivalent to writing a large record to a Relative Record file (where segment installed Id corresponds to the record number). Note that the DNOS Supervisor Call Reference Manual describes a method for using Segmentation SVC's for mapping records of an unblocked relative record file which accomplished the same function as updatable segments.

1.4.7 Memory Resident.

This software flag indicates that a segment is to be loaded into memory when the O/S is booted. These segments always remain in memory and their memory can never be used by other programs. This option should be used sparingly with performance being optained through segment caching.

1.4.8 Memory Based Segments.

The above flags are defined for disk based segments. DNOS supports the creation of segments at task runtime via the "create empty segment" SVC call. These segments are normally used for storage of data blocks. For these segments, there is no corresponding disk image.

1.4.9 Modified.

The modified option is a software flag set during mapping operations and indicates to the O/S whether the outgoing segment has been modified since it wasmapped in. If the segment is not flagged as modified, the O/S may discard the segment when it is mapped out since a correct copy can be loaded from the disk in one disk access. Pure code (Write Protect) segments should not be flagged as modified while data segments whose data is still required should be flagged modified. Modified segments not mapped in are saved to the roll file or the home file (see Updatable below).

1.4.10 Reserved.

The reserve option is a software flag set via a specific mapping call and indicates that the segment should not be discarded regardless of the setting of the modified flag. Segments so marked are saved to the roll file or home file when not mapped in and their physical memory is required. This flag is similar to the Exclusive flag except when a reserved segment is notmapped in to any task, any task can map the reserved segment back in.

1.4.11 Exclusive Use.

Exclusive use option is a software flag set via a specific mapping call or via aflag when a segment is mapped out (like the Modified flag). This flag indicates that the segment should not be discarded regardless of the setting of the modified flag. Segments flagged Exclusive and not mapped in can be mapped only by the task that marked the segment exclusive. Note that a segment can be flagged exclusive only once even by the same task without first resetting exclusive. A second exclusive map call (or mapping an exclusive segment out with the exclusive flag set) will result in an map SVC error >F9 (not currently documented).

1.5 USES OF SEGMENTS: PURE SHARED CODE

Solving the address space restriction for procedural code can be accomplished by linking pure code segments (normally sharable) to be loaded in the second or third map position. These segments can then be shared by multiple copies of the same task (or by different tasks if the segment is loaded at the same address). To reduce segment swapping and thereby improve performance, collect in one segment the routines that are normally accessed

together in a set.

To access segments so linked from a high level language (COBOL, Pascal), include a routine that maps the desired segment then call the subroutine in the segment normally. The example programs illustrate this method. Note that for pascal, pure code is linked into the segments and the exclusive bit need not be set. In the COBOL example, there is a DSEG associated with each subroutine so the segments cannot be sharable (but can be replicatable). However, since there is no local data (only LINKAGE references), the segments donot need to be reserved and can be Reusable. P1 USES OF SEGMENTS: DIRTY PROCEDURES) The most common usage for dirty procedures is for a set of subroutines that contain both code and data where the data stored must be preserved across multiple calls to the subroutines (not shown in the examples). This would occur in COBOL when the called subroutines contain WORKING-STORAGE data areas that must be maintained between calls. Since Pascal generates pure code subroutines, the only time this method would be needed for Pascal would be when assembly language DSEG's or any CSEG's were included in the segments. To reduce segment swapping and thereby improve performance, collect in one segment the routines that are normally accessed together in a set.

To access the segments, write a routine (see example 3) to load the segments and set exclusive access (first time only). The mapping routine uses segment installed id's to map the segment. These installed Id's can be hard coded into the source program (see example 1),or can be obtained via a CSEG from a FORMAT IMAGE link (see example 2). Once the segment is mapped in, call the subroutine normally (see example programs). Once the program is finished with the segments, a subroutine should be called to reset the Exclusive flag so another task could use the segments (if they are reusable). Note that if the task terminates, all exclusive access flags set by that task will be reset by DNOS making the segments available to other tasks.

1.6 USES OF SEGMENTS: PRIVATE DATA

The primary usage here is for data storage in excess of task addressability. For this purpose, break up the data storage into logically related pieces of information which will be accessed as a set. In the case of Pascal, a Record structure could be defined for the various segments, providing an alternative to the "NEW" function.

1.6.1 Access to Pre-Initialized Data Segments.

To access pre-initialized data segments, link and install the various segments and install on a disk file as execute protected, non-sharable, read/write (and replicatable as appropriate). Note that if pre-initialized data is stored in disk based segments and is modified by the task, these segments will not be in-memory reusable. For non-initialized segments (or

where the initialized data is not modified), the segments can be installed as reusable. To map the segments, save installed Id's in the same methods as noted above for code segments. Set exclusive use on each segment once before mapping the segment out. Reset exclusive when finished with the segment.

1.6.2 Access to Memory Based Data Segments.

To access memory based segments, provide a subroutine to create empty segments and save the runtime Id's (see example 3) in the task area. These runtime Id's are returned upon creating the segment. When creating the segment, set the Segment Attributes (see map SVC) for Read/Write, Execute Protect, and Share Protect. Also, before mapping out the segment, set exclusive access once.

1.7 USES OF SEGMENTS: SHARED DATA

Sharing data segments provides a method to pass a large block of data from one task to another without having to copy the data (via IPC). Access is via a map call using either installed Id's (disk based segments) or runtime Id's (memory based segments). For disk based segments, each task can store the installed Id's as noted above. For memory based segments, the runtime Id's must be passed from one task to another (via IPC or a dirty shared procedure segment). To prevent loss of segment data, one task should issue the reserve segment call once for all tasks using the segment. Exclusive should not be set because only the reserving task would then be able to access the segment.

If it is desired to prevent multiple tasks accessing a shared segment simultaneously, then set the non-sharable flag. Then when a task maps the segment and gets error >FA, the task can wait for the segment via wait on a semaphore (or could always wait on semaphore first then map the segment). When the task that is using the segment is finished with it, it will: 1) set the modified flag in the SVC block; 2) map out the segment; 3) post the semaphore, waking up the next task waiting on the segment.

1.8 NOTES ON USAGE OF SEGMENTS

1.8.1 Relocatable Segments.

If the segment contains no absolute address references, different tasks can map the same segment at different locations. Note that this would normally be used only for data storage and all data items would have to be accessed by pointer (Pascal) or base register (Assembly). It is, however,

possible to write assembly language pure procedural code that is position independent. This code could be linked into a segment that is mapped into multiple tasks at different addresses (output segment on only one task link, DUMMY on all rest). Note that the address of the segment within each task address space is returned in the map SVC block when the segment is mapped in.

Characteristics of position independent code include: 1) a base register set to point to the top of the subroutine; 2) no absolute branch instructions (use jumps or branch relative to subroutine base register [B (LABEL-START)(Rx)]; 3) all data references relative to registers.

1.8.2 Use of IPC Within Segments.

If the task is installed as software privileged, an IPC read or write can be initiated (via initiate event or setting initiate flag in I/O call block) and then the segment containing the buffer pointed to by the I/O call block mapped out. Note that neither the I/O call block nor the map SVC block can be in the segmentthat is to be mapped out (the map SVC and I/O SVC blocks would normally reside in the task segment).

Thus a server task can operate on multiple IPC channels with each channel or requestor using a unique segment per channel. This task need only wait for any I/O completion, then scan all the I/O SVC blocks to see which I/O completed. Then that segment can be mapped in and the request processed. The segment associated with the IPC channel could contain local data particular to the requestor that is to be saved across multiple requests.

1.8.3 Obtaining Segment Installed Id's.

There are three options for making installed segment Ids available to the source program: 1) define segment Id in link control source and use FORMAT IMAGE link; 2) let linker assign segment Id's with FORMAT IMAGE link; and 3) define segment Id's at IPS time with FORMAT ASCII links.

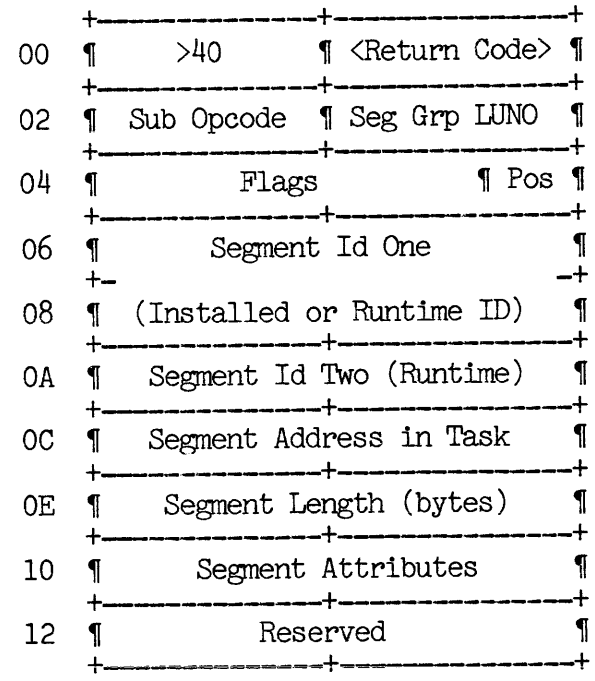
Case 1: When the segment Id's are defined within the link stream, these same Id's can be hard coded into the source program (example 1). However, the Id's can also be obtained by REF'ing the segment name in a common area (see below).

Case 2: When the linker assigns the segment Id's, the source program needs to access the installed Id's via REF'ing the segment name of the link as shown below (example 2).

```
SEGMENT 2, SEG1
INCLUDE . . .
   .
   .
SEGMENT 2, SEG2
   .
   .
            CSEG 'SEGTBL'
            DEF  SEGTBL
            REF  SEG1,SEG2,SEG3,SEG4,SEG5
      SEGTBL DATA SEG1,SEG2,SEG3,SEG4,SEG5
            CEND
```

Case 3: When segment Id's are set during IPS, the source program must have the installed segment Id's hard coded (into a table or within load subroutine call).

1.9 THE SEGMENT MANAGEMENT SVC



1.10 NOTES ON USING THE SEGMENT MANAGEMENT SVC

* Segment address is set after most segmentation SVC'S. This address can be used for accessing data segments as a base register of Pascal pointer. Thus the program need not have defined this load address within the code.

* Map installed segments by setting the position flag and setting the map location (1, 2 or 3) in the position bits.

* Segments can be replaced either by specifying the outgoing segment's position or runtime Id. The O/S sets the runtime id of the segment when it is map in so this runtime segment can be used to reference segments if desired. If both disk and memory based segments were being mapped by the same SVC block, it might be easier to use runtime Id rather than installed id.

* Tasks should release all segments no longer in use (reset exclusive or unreserve). The O/S will reset exclusive on all segments reserved by a task when that task terminates.

WARNING

IF A SEGMENT IS CACHABLE (LUNO ASSIGNED TO PROGRAM FILE AND "IN MEMORY REUSABLE" SET) AND A NEW COPY IS LINKED, AN EXECUTING PROGRAM MAY GET THE OLD COPY. ONLY BY IPL CAN THE NEW COPY BE GUARANTEED TO BE LOADED.

2. SAMPLE PROGRAMS

Four sample programs have been included to illustrate most of the principles discussed above. Three of these programs are Pascal as Pascal is the most difficult to use with segmentation. Some assembly language routines are included which are used to access the segment Id's or to map the segments.

For Pascal, there is a particular problem with using segments. That is the usage by the runtime of Get Memory SVC calls when the program issues NEW calls. As the size of the task segment cannot change (so segments will always map at the same location), the Get Memory SVC calls must be bypassed. As all the Get Memory calls are issued by a library routine called GET$ME, a special version of this routine is provided that allocates memory from a fixed common area (called STKHEP). When this fixed space is exhausted the standard overflow messages will be issued. GET$ME is the first module in the listings. Note also for Pascal that a common runtime was linked and used.

2.10.1 Example 1: Pascal, Pure Code Segments, Hard Coded Segment Id's.

Included in the listings is the Pascal source program, the link control file, and the output results. The source program illustrates the Map SVC record definitions and the mapping of the segments by installed Id's. Pay careful attention to the flag settings used by the routine. The CHANGESEG and INITSVC routines are used in conjunction with each other. That is, INITSVC is designed to use with mapping segments by installed Id. In the link control, segment Id's 2 and 3 are used for the segments and these Id's are set up in the first two lines of the main program.

2.10.2 Example 2: Pascal, Pure Code Segments, Segment Id's from Link.

The first listing for example 2 is an assembly language common called "SEGTBL" that will pick up the segment Id's from the link step. This common is designed to be used in conjunction with the CHANGESEG routine used in example 2 in that the procedure name is passed to CHANGESEG and CHANGESEG accesses SEGTBL to obtain an associated segment Id. Note that CHANGESEG does not issue the map SVC if the segment is already mapped in. In this fashion, the location of subroutines within the segments need not be known within the Pascal source. The link control and results are almost exactly the same as for example 1.

2.10.3 Example 3: Pascal, Private Data Segments, Runtime Segment Id's.

This example illustrates use of segments for stored private data. The method is to create an empty segment, move 100 integers into the segment, then

repeat this process three more times. Then each of the segments is mapped back in and the contents of the segments verified. Here the INITSVC routine is tailored for mapping segments by runtime Id and the map routine is called SWITCHSEG since it is radically different from either of the CHANGESEG routines of examples 1 and 2. Again note the use of the map SVC flags. The link control does not include any segments since these all segments referenced are created at runtime. The listings include the Pascal source, the link control, and the results of the program run.

2.10.4 Example 4: COBOL, Dirty Code Segments, Segment Id's from Link.

This is an example of a COBOL program that will map in pieces of dirty code (dirty since COBOL subroutine contains 48 byte DSEG used for workspace and temporary data). The assembly language common "SEGTBL" is used to access the installed segment Id's and a COBOL callable assembly language map subroutine "MAPSEG" accesses SEGTBL, maps the segments, and sets exclusive use. The MAPSEG subroutine is the first listing of example followed by the COBOL main followed by the called COBOL subroutines. The link control is similar to examples 1 and 2 except a COBOL runtime is used.

```
0001                    IDT  'GET$ME'
0002 0000               PSEG
0003                  * PROCEDURE GET$MEM(SZ:INTEGER; VAR PTR:MEMPTR)
0004                  *-------------------------------------------------------
0005                  * PURPOSE:
0006                  *   SIMULATES GET MEMORY SERVICE CALL TO GET ADDITIONAL
0007                  *   MEMORY BY ALLOCATING A FIXED STATIC BLOCK.  THE REGION
0008                  *   IS NOT CLEARED BY THE CODE.
0009                  * INPUTS:
0010                  *   SZ: SIZE (IN BYTES) OF THE REGION DESIRED.
0011                  * PROCEDURES CALLED:  NONE.
0012                  * OUTPUTS:
0013                  *   PTR: POINTER TO A NEW MEMORY REGION OF SIZE 'SZ' ROUNDED
0014                  *     UP TO AN EVEN NUMBER.
0015                  * EXCEPTIONS:
0016                  *   IF THE REGION COULD NOT BE OBTAINED, PTR=NIL IS RETURNED
0017                  * HISTORY:  01/10/83: ORIGINAL.
0018                  * NOTE:  LINK WITH INCLUDE FOR P$MAIN VICE MAIN FOR FIRST
0019                  *         MODULE IN LINK AS GET$MEM IS IN MAIN PARTIAL LINK.
0020                  *-------------------------------------------------------
0021                  *
0022       0024       RTNADR EQU  36
0023       0028       ARG1   EQU  40
0024       002A       ARG2   EQU  42
0025                  * PROCEDURE GET$MEM(SZ:INTEGER; VAR PTR:MEMPTR);
0026 0000    47       PROLOG TEXT 'GET$MEM '
0027 0008 0038'             DATA EPILOG,PROLOG
     000A 0000'
0028                        DEF  GET$ME
0029 000C 02A9  GET$ME      STWP R9                  GET ADDRESS CURRENT WORKSPACE
0030 000E 04E9              CLR  @RTNADR(R9)         FLAG SHORT LINKAGE TO DEBUGGER
     0010 0024
0031 0012 04C8              CLR  R8                  SET NIL RETURN IN CASE ERROR
0032 0014 C069              MOV  @ARG1(R9),R1        GET DESIRED SIZE
     0016 0028
0033 0018 0581              INC  R1                  ROUND UP TO EVEN
0034 001A 0241              ANDI R1,>FFFE
     001C FFFE
0035 001E A060              A    @B$NEXT,R1          GET ADDRESS LAST REQ BYTE
     0020 0000+
0036 0022 1807              JOC  EXIT                PUNT ON ADDRESS WRAP
0037 0024 0281              CI   R1,B$END            PAST END OF BLOCK?
     0026 3002+
0038 0028 1B04              JH   EXIT                YES, PUNT
0039 002A C220              MOV  @B$NEXT,R8          NO, SET ADDRESS
     002C 0000+
0040 002E C801              MOV  R1,@B$NEXT          UPDATE NEXT FREE AREA
     0030 0000+
0041 0032 C0A9  EXIT        MOV  @ARG2(R9),R2        GET ADDRESS OF RETURNED PTR
     0034 002A
0042 0036 C488              MOV  R8,*R2              RETURN ADDRESS OF BLOCK
0043 0038 0380  EPILOG      RTWP                     TO CALLER
0044 003A                   PEND
0045 0000                   CSEG 'STKHEP'
0046 0000 0002+ B$NEXT      DATA $+2                 ADDRESS NEXT FREE WORD
0047 0002                   BSS  >3000               FIXED MEMORY AVAILABLE BLOCK
0048       3002+ B$END      EQU  $                   END OF THE MEMORY BLOCK
0049 3002                   CEND
NO ERRORS.      NO WARNINGS
```

```
PROGRAM MAIN1;
(*-------------------------------------------------------------------*)
(*      THIS PROGRAM ILLUSTRATES THE USE OF DNOS SEGMENTATION TO     *)
(*      MAP IN PIECES OF PURE PASCAL CODE AND THEN EXECUTING THEM.   *)
(*      NOTE THAT IF MULTIPLE COPIES OF THIS PROGRAM WERE BEING      *)
(*      EXECUTED, THEN THE SEGMENTS COULD BE INSTALLED AS SHARABLE.  *)
(*      ALSO, SINCE THERE IS NO WRITABLE DATA IN THE SEGMENTS,       *)
(*      THERE IS NO NEED TO SET RESERVE OR SET EXCLUSIVE ACCESS.     *)
(*      THE LOAD ADDRESS OF THE SEGMENTS IS DETERMINED BY THE LINK   *)
(*      EDITOR AS IS THE ADDRESS OF PROC1, PROC2, PROC3, AND PROC4.  *)
(*      THE ID'S OF THE SEGMENTS IS FIXED IN THE LINK CONTROL FILE   *)
(*      AND SET WITHIN THE PROCEDURE SECTION OF THE MAIN ROUTINE.    *)
(*                                                                   *)
(*      THE MAP SVC BLOCK AND THE CONSTANTS WERE SET UP AS COPY      *)
(*      FILES SO THEY COULD BE USED FOR OTHER EXAMPLES.              *)

CONST
(*-------------------------------------------------------------------*)
(*                 COMMONLY USED CONSTANT DEFINITIONS                *)
(*-------------------------------------------------------------------*)

  MAP_SVC_CODE       = #40;      (* SVC CODE FOR MAP *)
  CODE_SEG_POS       = 3;        (* MAP POSITION FOR NEW SEGMENTS *)
  MAP_FLAGS_POSIT    = #C0;      (* MAP FLAGS: MAP BY POSIT / LUNO *)
  MAP_FLAGS_RESREL   = #40;      (* MAP FLAGS FOR RESERVE/RELEASE *)
  OWN_TASK_LUNO      = #FF;      (* LUNO FOR OWN PROGRAM FILE *)

" MAP SVC SUB OP-CODES:
    CHANGE_SEG       = #00;      (* CHANGE SEGMENT *)
    CREATE_SEG       = #01;      (* CREATE SEGMENT *)
    RESERVE_SEG      = #02;      (* RESERVE SEGMENT *)
    RELEASE_SEG      = #03;      (* RELEASE SEGMENT *)
    GET_SEG_STATUS   = #04;      (* GET SEGMENT STATUS *)
    FORCE_WRITE      = #05;      (* FORCE WRITE SEG. *)
    LOAD_SEG         = #09;      (* MAKE SEG. MEMORY RESIDENT *)
    UNLOAD_SEG       = #0A;      (* RELEASE MEMORY RESIDENT SEG. *)
    SET_EXCLUSIVE    = #0B;      (* SET EXCLUSIVE USE OF SEG. *)
    RESET_EXCLUSIVE  = #0C;      (* RESET EXCLUSIVE USE *)


TYPE
(*-------------------------------------------------------------------*)
(*                   COMMONLY USED TYPE DEFINITIONS                  *)
(*-------------------------------------------------------------------*)

  BYTE               = 0..255;
  NAME               = PACKED ARRAY [1..6] OF CHAR;

  SEGTBL_ENTRY       = PACKED RECORD;
    PROC_NAME        : NAME;
    SEG_ID           : INTEGER;
    END;
```

```
(*$ PAGE *)
(*-------------------------------------------------------------------*)
(*           DNOS 990 SEGMENTATION SVC FIELD DEFINITIONS             *)
(*-------------------------------------------------------------------*)

  T_MAPSVC          = PACKED RECORD;
    SVC_CODE        : BYTE;             (* SVC CODE = >40 *)
    ERR_CODE        : BYTE;             (* RETURNED ERROR CODE *)
    OP_CODE         : BYTE;             (* MAP OP CODE *)
    MAP_LUNO        : BYTE;             (* LUNO OF PROG FILE *)
    MAP_FLAG        : BYTE;             (* FLAGS             *)
    MAP_POS         : BYTE;             (* MAP POSITION (0,1,2) *)
    FILL1           : INTEGER;          (* FIRST WORD NEW SEG ID *)
    NEW_SEGID       : INTEGER;          (* NEW SEGMENT *)
    OLD_SEGID       : INTEGER;          (* OLD SEGMENT *)
    SEG_ADDR        : INTEGER;          (* RETURNED ADDR OF SEG *)
    SEG_SIZE        : INTEGER;          (* SIZE OF SEGMENT (I/O) *)
    SEG_ATTRIB      : INTEGER;          (* SEGMENT ATTRIBUTES *)
    FILL2           : INTEGER;          (* RESERVED *)
  END;  " T_MAPSVC "


VAR
  SEG1_ID           : INTEGER;          (* SEGMENT ID OF SEGMENT 1 *)
  SEG2_ID           : INTEGER;          (* SEGMENT ID OF SEGMENT 2 *)

COMMON
  MAPSVC            : RECORD
    MAP_SVCB        : T_MAPSVC;
    END; "

PROCEDURE SVC$(SVC_BLOCK : INTEGER); EXTERNAL;
```

```
(*$ PAGE *)
PROCEDURE INITSVC;
(*-------------------------------------------------------------------*)
(*      THIS ROUTINE INITIALIZED THE MAP SVC BLOCK AS INDICATED      *)
(*      IN THE COMMENTS BELOW.                                       *)
(*-------------------------------------------------------------------*)

ACCESS MAPSVC;

BEGIN " PROCEDURE INITSVC
  WITH MAPSVC, MAP_SVCB DO
    BEGIN " INITIALIZE MAP SVC BLOCK
      SVC_CODE := MAP_SVC_CODE;      (* SET SVC CODE *)
      MAP_LUNO := OWN_TASK_LUNO;     (* SEGMENTS FROM OWN PROG FILE *)
      MAP_FLAG := MAP_FLAGS_POSIT;   (* MAP BY POSITION AND LUNO *)
      MAP_POS  := CODE_SEG_POS;      (* LOCATION TO MAP SEGMENTS *)
      FILL1 := 0;                    (* FIRST WORD NEW SEG TO ZERO *)
      OLD_SEGID := 0;                (* INITIAL OUTGOING SEG TO ZERO *)
    END;  " INITIALIZE MAP SVC BLOCK
END;  " PROCEDURE INITSVC



PROCEDURE CHANGESEG(VAR SEG_ID : INTEGER);
(*-------------------------------------------------------------------*)
(*      THIS ROUTINE ISSUES THE MAP SVC SUPERVISOR CALL AFTER        *)
(*      SETTING THE NEW SEGMENT ID AND TYPE OF MAP SVC (OP_CODE).    *)
(*      SOME ERROR PROCESSING IS ILLUSTRATED BUT WOULD MOST LIKELY   *)
(*      BE INSUFFICIENT FOR MOST PROGRAMS.                           *)
(*-------------------------------------------------------------------*)

ACCESS MAPSVC;

BEGIN " PROCEDURE CHANGESEG
  WITH MAPSVC, MAP_SVCB DO
    BEGIN " INITIALIZE MAP SVC BLOCK
      OP_CODE  := CHANGE_SEG;        (* SET FOR CHANGE SEG *)
      NEW_SEGID := SEG_ID;           (* SET INSTALLED ID *)
      SVC$(LOCATION(MAP_SVCB));      (* MAP IN SEGMENT *)
      IF (ERR_CODE <> 0) THEN
        BEGIN " PUNT ON SVC ERROR
          WRITELN(' ERROR IN MAPPING SEGMENT; SEG ID = ',SEG_ID:2,
            ' ERROR CODE = ',ERR_CODE:4);
          ESCAPE MAIN1;
        END;  " PUNT ON SVC ERROR

    END;  " INITIALIZE MAP SVC BLOCK
END;  " PROCEDURE CHANGESEG
```

```
(*$ PAGE *)
(*-------------------------------------------------------------------*)
(*      THESE PROCEDURES WILL BE CALLED ONE AT A TIME FROM THE       *)
(*      MAIN ROUTINE.  EACH WILL PRINT ONE LINE TO STANDARD OUTPUT.  *)
(*-------------------------------------------------------------------*)

PROCEDURE PROC1;
  BEGIN " PROCEDURE PROC1
    WRITELN(' THIS LINE WRITTEN FROM PROCEDURE 1');
  END;  " PROCEDURE PROC1

PROCEDURE PROC2;
  BEGIN " PROCEDURE PROC2
    WRITELN(' THIS LINE WRITTEN FROM PROCEDURE 2');
  END;  " PROCEDURE PROC2

PROCEDURE PROC3;
  BEGIN " PROCEDURE PROC3
    WRITELN(' THIS LINE WRITTEN FROM PROCEDURE 3');
  END;  " PROCEDURE PROC3

PROCEDURE PROC4;
  BEGIN " PROCEDURE PROC4
    WRITELN(' THIS LINE WRITTEN FROM PROCEDURE 4');
  END;  " PROCEDURE PROC4



(*-------------------------------------------------------------------*)
(*      MAIN ROUTINE.  SETS SEGMENT ID VARIABLES THEN CALLS OTHER    *)
(*      SUBROUTINES.                                                 *)
(*-------------------------------------------------------------------*)

BEGIN
  SEG1_ID := 2;                        (* ID'S OF SEGMENTS SET WITHIN *)
  SEG2_ID := 3;                        (* CODE HERE AND ON LINK FILE. *)
  INITSVC;                             (* INITIALIZE MAP SVC *)
  REWRITE(OUTPUT);                     (* OPEN OUTPUT, WRITE FIRST MSG *)
  WRITELN(' THIS LINE WRITTEN FROM MAIN');
  CHANGESEG(SEG1_ID);                  (* MAP IN PROC1 AND PROC2 *)
  PROC1;                               (* CALL PROC1 *)
  PROC2;                               (* CALL PROC2 *)
  CHANGESEG(SEG2_ID);                  (* MAP IN PROC3 AND PROC4 *)
  PROC3;                               (* CALL PROC3 *)
  PROC4;                               (* CALL PROC4 *)
END.
```

```
        <<<<<<<<  COMMON TIP RUNTIME PROCEDURES  LINK CONTROL  >>>>>>>>
LIBRARY K.TIMIX.O
LIBRARY .TIP.OBJ
PARTIAL
PHASE 0, RUNTIME
INCLUDE (DSTR$$)                    INCLUDE (ENT$   )
INCLUDE (GO$    )                   INCLUDE (MESAG$)
INCLUDE (MM$DIR)                    INCLUDE (P$TERM)
INCLUDE (TERM$ )                    INCLUDE (MOV$N )
INCLUDE (FL$INI)                    INCLUDE (REWRT$)
INCLUDE (WRS$T )                    INCLUDE (WRLN$ )
INCLUDE (CLS$   )                   INCLUDE (ABEND$)
INCLUDE (DSTRY$)                    INCLUDE (ENT$MD)
INCLUDE (INIT$ )                    INCLUDE (RESUM$)
INCLUDE (RSUMR$)                    INCLUDE (SCIRTNS)
INCLUDE (MSG$   )                   INCLUDE (ENX$T )
INCLUDE (ENS$T )                    INCLUDE (DUMP$P)
INCLUDE (DMPP$H)                    INCLUDE (PB$INI)
INCLUDE (CLOSE$)                    INCLUDE (ENI$T )
INCLUDE (CUR$   )                   INCLUDE (P$$TRM)
INCLUDE (SCB$IN)                    INCLUDE (OPEN$ )
INCLUDE (REWND$)                    INCLUDE (WREOF$)
INCLUDE (IO$ERR)                    INCLUDE (TX$ERR)
INCLUDE (PUT$RC)                    INCLUDE (SCB$FR)
INCLUDE (FREE$ )                    INCLUDE (TIP$TC)
INCLUDE (GET$PA)                    INCLUDE (SVC$   )
INCLUDE (HEAP$T)                    INCLUDE (NEW$   )
INCLUDE (CREAT$)                    INCLUDE (INIT$1)
INCLUDE (P$INIT)                    INCLUDE (STACK$)
INCLUDE (WRX$T )                    INCLUDE (PRT$ME)
INCLUDE (DUMP$S)                    INCLUDE (CLS$FI)
INCLUDE (DIV$   )                   INCLUDE (SET$NA)
INCLUDE (OPN$FI)                    INCLUDE (PM$IO )
INCLUDE (PM$TIO)                    INCLUDE (GET$ME)
INCLUDE (STK$MA)                    INCLUDE (EOLN$ )
INCLUDE (PUTCH$)                    INCLUDE (WRC$T )
INCLUDE (CMP$ST)                    INCLUDE (WRI$T )
INCLUDE (S$NAME)                    INCLUDE (MAP$   )
END

<<<  MAIN 1 LINK CONTROL  >>>

FORMAT IMAGE,REPLACE
LIBRARY K.TIMIX.O
LIBRARY .TIP.OBJ
PROCEDURE RUNTIM
INCLUDE (RUNTIM)
PHASE0, MAIN1
INCLUDE (P$MAIN)
ALLOCATE
INCLUDE (MAIN1)
SEGMENT 3,SEG1,ID 2
INCLUDE (PROC1)
INCLUDE (PROC2)
SEGMENT 3,SEG2,ID 3
INCLUDE (PROC3)
INCLUDE (PROC4)
END
```

```
¶
¶
¶
¶  <<<  RESULTS OF MAIN1 RUN >>>
¶
¶  THIS LINE WRITEN FROM MAIN
¶  THIS LINE WRITEN FROM PROCEDURE 1
¶  THIS LINE WRITEN FROM PROCEDURE 2
¶  THIS LINE WRITEN FROM PROCEDURE 3
¶  THIS LINE WRITEN FROM PROCEDURE 4
¶
¶
¶
¶
¶
¶
¶
¶
¶
```

```
0001                    IDT  'SEGTBL'
0002 0000               CSEG 'SEGTBL'
0003                *
0004                *       THIS MODULE CONTAINS A TABLE OF SUBROUTINE NAMES AND
0005                *       THEIR ASSOCIATED SEGMENT ID'S.  THE SEGMENT ID'S ARE
0006                *       OBTAINED FROM THE FORMAT IMAGE LINK BY SPECIFYING
0007                *       THE SEGMENT NAME ON THE DATA STATEMENT.  NOTE THAT
0008                *       THIS TABLE AND THE LINK CONTROL STREAM MUST BE KEPT
0009                *       IN SYNC MANUALLY BY EDITING BOTH FILES AS MODULES
0010                *       ARE ADDED TO A PARTICULAR SEGMENT OR AS SEGMENTS ARE
0011                *       ADDED.
0012                *
0013                *       EACH TABLE ENTRY IS THE SIX CHARACTER TEXT NAME OF
0014                *       THE MODULE FOLLOWED BY A ONE WORD INTEGER SEGMENT
0015                *       ID.  THE FIRST WORD IN THE TABLE IS THE NUMBER OF
0016                *       ENTRIES IN THE TABLE.
0017                *
0018                *       ALL SEGMENT NAMES MUST BE EXTERNALLY REFERENCED:
0019                *
0020                            REF  SEG1,SEG2
0021 0000
0022 0000 0004                  DATA (TBLEND-$-2)/8     NUMBER OF TABLE ENTRIES
0023                *
0024                *                                   PROCEDURES IN SEGMENT 1
0025 0002   50                  TEXT 'PROC1 '             ROUTINE NAME
0026 0008 0000                  DATA SEG1                 PHASE NAME FOR SEGMENT
0027 000A   50                  TEXT 'PROC2 '             ROUTINE NAME
0028 0010 0008+                 DATA SEG1                 PHASE NAME FOR SEGMENT
0029                *
0030                *                                   PROCEDURES IN SEGMENT 2
0031 0012   50                  TEXT 'PROC3 '             ROUTINE NAME
0032 0018 0000                  DATA SEG2                 PHASE NAME FOR SEGMENT
0033 001A   50                  TEXT 'PROC4 '             ROUTINE NAME
0034 0020 0018+                 DATA SEG2                 PHASE NAME FOR SEGMENT
0035 0022
0036      0022+ TBLEND EQU  $                           END OF TABLE
0037 0022               CEND
NO ERRORS,     NO WARNINGS
```

```
PROGRAM MAIN2;
(*-------------------------------------------------------------------*)
(*      THIS PROGRAM ILLUSTRATES THE USE OF DNOS SEGMENTATION TO     *)
(*      MAP IN PIECES OF PURE PASCAL CODE AND THEN EXECUTING THEM.   *)
(*      NOTE THAT IF MULTIPLE COPIES OF THIS PROGRAM WERE BEING      *)
(*      EXECUTED, THEN THE SEGMENTS COULD BE INSTALLED AS SHARABLE.  *)
(*      ALSO, SINCE THERE IS NO WRITABLE DATA IN THE SEGMENTS,       *)
(*      THERE IS NO NEED TO SET RESERVE OR SET EXCLUSIVE ACCESS.     *)
(*      THE LOAD ADDRESS OF THE SEGMENTS IS DETERMINED BY THE LINK   *)
(*      EDITOR AS IS THE ADDRESS OF PROC1, PROC2, PROC3, AND PROC4.  *)
(*      THE ID'S OF THE SEGMENTS IS FIXED IN THE LINK CONTROL FILE   *)
(*      AND DETERMINED FROM THE LINK STREAM DYNAMICALLY.             *)
(*                                                                   *)
(*      THE MAP SVC BLOCK AND THE CONSTANTS WERE SET UP AS COPY      *)
(*      FILES SO THEY COULD BE USED FOR OTHER EXAMPLES.              *)

CONST
(*-------------------------------------------------------------------*)
(*                COMMONLY USED CONSTANT DEFINITIONS                 *)
(*-------------------------------------------------------------------*)

  MAP_SVC_CODE       = #40;      (* SVC CODE FOR MAP *)
  CODE_SEG_POS       = 3;        (* MAP POSITION FOR NEW SEGMENTS *)
  MAP_FLAGS_POSIT    = #C0;      (* MAP FLAGS: MAP BY POSIT / LUNO *)
  MAP_FLAGS_RESREL   = #40;      (* MAP FLAGS FOR RESERVE/RELEASE *)
  OWN_TASK_LUNO      = #FF;      (* LUNO FOR OWN PROGRAM FILE *)

" MAP SVC SUB OP-CODES:
    CHANGE_SEG       = #00;      (* CHANGE SEGMENT *)
    CREATE_SEG       = #01;      (* CREATE SEGMENT *)
    RESERVE_SEG      = #02;      (* RESERVE SEGMENT *)
    RELEASE_SEG      = #03;      (* RELEASE SEGMENT *)
    GET_SEG_STATUS   = #04;      (* GET SEGMENT STATUS *)
    FORCE_WRITE      = #05;      (* FORCE WRITE SEG. *)
    LOAD_SEG         = #09;      (* MAKE SEG. MEMORY RESIDENT *)
    UNLOAD_SEG       = #0A;      (* RELEASE MEMORY RESIDENT SEG. *)
    SET_EXCLUSIVE    = #0B;      (* SET EXCLUSIVE USE OF SEG. *)
    RESET_EXCLUSIVE  = #0C;      (* RESET EXCLUSIVE USE *)


TYPE
(*-------------------------------------------------------------------*)
(*                  COMMONLY USED TYPE DEFINITIONS                   *)
(*-------------------------------------------------------------------*)

  BYTE               = 0..255;
  NAME               = PACKED ARRAY [1..6] OF CHAR;

  SEGTBL_ENTRY       = PACKED RECORD;
    PROC_NAME        : NAME;
    SEG_ID           : INTEGER;
    END;
```

```
(*$ PAGE *)
(*-------------------------------------------------------------------*)
(*          DNOS 990 SEGMENTATION SVC FIELD DEFINITIONS              *)
(*-------------------------------------------------------------------*)

  T_MAPSVC          = PACKED RECORD;
    SVC_CODE        : BYTE;             (* SVC CODE = >40 *)
    ERR_CODE        : BYTE;             (* RETURNED ERROR CODE *)
    OP_CODE         : BYTE;             (* MAP OP CODE *)
    MAP_LUNO        : BYTE;             (* LUNO OF PROG FILE *)
    MAP_FLAG        : BYTE;             (* FLAGS             *)
    MAP_POS         : BYTE;             (* MAP POSITION (0,1,2) *)
    FILL1           : INTEGER;          (* FIRST WORD NEW SEG ID *)
    NEW_SEGID       : INTEGER;          (* NEW SEGMENT *)
    OLD_SEGID       : INTEGER;          (* OLD SEGMENT *)
    SEG_ADDR        : INTEGER;          (* RETURNED ADDR OF SEG *)
    SEG_SIZE        : INTEGER;          (* SIZE OF SEGMENT (I/O) *)
    SEG_ATTRIB      : INTEGER;          (* SEGMENT ATTRIBUTES *)
    FILL2           : INTEGER;          (* RESERVED *)
  END;  " T_MAPSVC "

VAR
  SEG1_ID           : INTEGER;          (* SEGMENT ID OF SEGMENT 1 *)
  SEG2_ID           : INTEGER;          (* SEGMENT ID OF SEGMENT 2 *)
  ROUTINE_NAME      : NAME;

COMMON
  MAPSVC            : RECORD
    MAP_SVCB        : T_MAPSVC;
    END;
  SEGTBL            : RECORD
    NUM_ENTRIES     : INTEGER;
    SEGLIST         : ARRAY [1..10] OF SEGTBL_ENTRY;
    END;

PROCEDURE SVC$(SVC_BLOCK : INTEGER); EXTERNAL;

PROCEDURE INITSVC;
(*-------------------------------------------------------------------*)
(*      THIS ROUTINE INITIALIZED THE MAP SVC BLOCK AS INDICATED      *)
(*      IN THE COMMENTS BELOW.                                       *)
(*-------------------------------------------------------------------*)
ACCESS MAPSVC;

BEGIN " PROCEDURE INITSVC
  WITH MAPSVC, MAP_SVCB DO
    BEGIN " INITIALIZE MAP SVC BLOCK
      SVC_CODE := MAP_SVC_CODE;      (* SET SVC CODE *)
      MAP_LUNO := OWN_TASK_LUNO;     (* SEGMENTS FROM OWN PROG FILE *)
      MAP_FLAG := MAP_FLAGS_POSIT;   (* MAP BY POSITION AND LUNO *)
      MAP_POS  := CODE_SEG_POS;      (* LOCATION TO MAP SEGMENTS *)
      FILL1 := 0;                    (* FIRST WORD NEW SEG TO ZERO *)
      OLD_SEGID := 0;                (* INITIAL OUTGOING SEG TO ZERO *)
    END;  " INITIALIZE MAP SVC BLOCK
END;  " PROCEDURE INITSVC
```

```
(*$ PAGE *)
PROCEDURE CHANGESEG(VAR ROUTINE_NAME : NAME);
(*-------------------------------------------------------------*)
(*      THIS ROUTINE TAKES THE ROUTINE NAME AND MAPS IT TO A   *)
(*      ID THROUGH THE COMMON SEGTBL.  THEN IF THAT SEGMENT IS NOT  *)
(*      MAPPED IN, IT IS MAPPED IN VIA A MAP SVC SUPERVISOR CALL.   *)
(*      THE NEW SEGMENT ID AND TYPE OF MAP SVC (OP_CODE) ARE SET.   *)
(*      SOME ERROR PROCESSING IS ILLUSTRATED BUT WOULD MOST LIKELY  *)
(*      BE INSUFFICIENT FOR MOST PROGRAMS.                     *)
(*-------------------------------------------------------------*)

VAR
  PROC_SEGID        : INTEGER;

ACCESS MAPSVC, SEGTBL;

BEGIN " PROCEDURE CHANGESEG
  WITH MAPSVC, MAP_SVCB, SEGTBL DO
    BEGIN " MAP SEGMENT IF REQUIRED
      PROC_SEGID := 0;              (* ASSUME SEGMENT NOT FOUND *)
      FOR I := 1 TO NUM_ENTRIES DO
        IF SEGLIST[I].PROC_NAME = ROUTINE_NAME THEN
          PROC_SEGID := SEGLIST[I].SEG_ID;
      IF PROC_SEGID = 0 THEN
        BEGIN " PUNT ON BAD PROC NAME
          WRITELN(' INVALID PROCEDURE NAME; NAME = ',ROUTINE_NAME);
          ESCAPE MAIN2;
        END;  " PUNT ON BAD PROC NAME

      IF NEW_SEGID = PROC_SEGID THEN
        ESCAPE CHANGESEG;

      OP_CODE  := CHANGE_SEG;       (* SET FOR CHANGE SEG *)
      NEW_SEGID := PROC_SEGID;      (* SET INSTALLED ID *)
      SVC$(LOCATION(MAP_SVCB));     (* MAP IN SEGMENT *)
      IF (ERR_CODE <> 0) THEN
        BEGIN " PUNT ON SVC ERROR
          WRITELN(' ERROR IN MAPPING SEGMENT; SEG ID = ',
            NEW_SEGID:2, ' ERROR CODE = ',ERR_CODE:4);
          ESCAPE MAIN2;
        END;  " PUNT ON SVC ERROR

    END;  " MAP SEGMENT IF REQUIRED
END;  " PROCEDURE CHANGESEG
```

```
(*$ PAGE *)
(*-------------------------------------------------------------------*)
(*      THESE PROCEDURES WILL BE CALLED ONE AT A TIME FROM THE       *)
(*      MAIN ROUTINE.  EACH WILL PRINT ONE LINE TO STANDARD OUTPUT.  *)
(*-------------------------------------------------------------------*)

PROCEDURE PROC1;
  BEGIN " PROCEDURE PROC1
    WRITELN(' THIS LINE WRITTEN FROM PROCEDURE 1');
  END;  " PROCEDURE PROC1

PROCEDURE PROC2;
  BEGIN " PROCEDURE PROC2
    WRITELN(' THIS LINE WRITTEN FROM PROCEDURE 2');
  END;  " PROCEDURE PROC2

PROCEDURE PROC3;
  BEGIN " PROCEDURE PROC3
    WRITELN(' THIS LINE WRITTEN FROM PROCEDURE 3');
  END;  " PROCEDURE PROC3

PROCEDURE PROC4;
  BEGIN " PROCEDURE PROC4
    WRITELN(' THIS LINE WRITTEN FROM PROCEDURE 4');
  END;  " PROCEDURE PROC4

(*-------------------------------------------------------------------*)
(*      MAIN ROUTINE.  SETS SEGMENT ID VARIABLES THEN CALLS OTHER    *)
(*      SUBROUTINES.                                                 *)
(*-------------------------------------------------------------------*)
BEGIN
  INITSVC;                              (* INITIALIZE MAP SVC *)
  REWRITE(OUTPUT);                      (* OPEN OUTPUT, WRITE FIRST MSG *)
  WRITELN(' THIS LINE WRITTEN FROM MAIN');

  ROUTINE_NAME := 'PROC1 ';             (* MAP IN PROC1 *)
  CHANGESEG(ROUTINE_NAME);
  PROC1;                                (* CALL PROC1 *)

  ROUTINE_NAME := 'PROC2 ';             (* MAP IN PROC2 *)
  CHANGESEG(ROUTINE_NAME);
  PROC2;                                (* CALL PROC2 *)

  ROUTINE_NAME := 'PROC3 ';             (* MAP IN PROC3 *)
  CHANGESEG(ROUTINE_NAME);
  PROC3;                                (* CALL PROC3 *)

  ROUTINE_NAME := 'PROC4 ';             (* MAP IN PROC4 *)
  CHANGESEG(ROUTINE_NAME);
  PROC4;                                (* CALL PROC4 *)
END.
```

<<< MAIN2 LINK CONTROL >>>

```
FORMAT IMAGE,REPLACE
LIBRARY K.TIMIX.O
LIBRARY .TIP.OBJ
PROCEDURE RUNTIM
INCLUDE (RUNTIM)
;
PHASE0, MAIN2
INCLUDE (P$MAIN)
INCLUDE (MAIN2)
INCLUDE (SEGTBL)
;
SEGMENT 3,SEG1,ID 5
INCLUDE (PROC1)
INCLUDE (PROC2)
;
SEGMENT 3,SEG2,ID 6
INCLUDE (PROC3)
INCLUDE (PROC4)
END
```

<<< RESULTS OF MAIN2 RUN >>>

```
THIS LINE WRITEN FROM MAIN
THIS LINE WRITEN FROM PROCEDURE 1
THIS LINE WRITEN FROM PROCEDURE 2
THIS LINE WRITEN FROM PROCEDURE 3
THIS LINE WRITEN FROM PROCEDURE 4
```

```
PROGRAM MAIN3;
(*-------------------------------------------------------------*)
(*      THIS PROGRAM ILLUSTRATES THE USE OF DNOS SEGMENTATION TO    *)
(*      MAP IN DIFFERENT DATA BLOCKS AND CONSERVE THOSE DATA        *)
(*      BLOCKS BY ISSUING THE "SET EXCLUSIVE ACCESS" MAP CALL.      *)
(*      NOTE THAT A NEW CHANGE SEGMENT ROUTINE IS PROVIDED THAT     *)
(*      WILL SWITCH SEGMENTS BASED UPON THEIR RUNTIME ID'S.  THIS   *)
(*      IS DUE TO THE FACT THAT MEMORY BASED SEGMENTS DO NOT HAVE   *)
(*      AN INSTALLED ID.                                            *)
(*                                                                  *)
(*      THE PROGRAM ITSELF WILL MOVE A SERIES OF CONSTANTS TO THE   *)
(*      DIFFERENT SEGMENTS AND THEN MAP THOSE SEGMENTS BACK IN AND  *)
(*      PRINT THE DATA OBTAINED.  THIS WILL ILLUSTRATE THAT THE     *)
(*      CONTENTS OF THE SEGMENTS ARE PRESERVED WHILE THEY ARE NOT   *)
(*      MAPPED IN TO THE TASK.  ALSO NOTE THAT THE DATA SEGMENTS    *)
(*      WILL ALL MAP AT THE SAME LOCATION SO THE POINTER TO THE     *)
(*      SEGMENTS WILL BE SET ONLY ONCE DURING INITIALIZATION.       *)

CONST
(*-------------------------------------------------------------*)
(*                COMMONLY USED CONSTANT DEFINITIONS           *)
(*-------------------------------------------------------------*)


  MAP_SVC_CODE       = #40;      (* SVC CODE FOR MAP *)
  CODE_SEG_POS       = 3;        (* MAP POSITION FOR NEW SEGMENTS *)
  MAP_FLAGS_POSIT    = #C0;      (* MAP FLAGS: MAP BY POSIT / LUNO *)
  MAP_FLAGS_RESREL   = #40;      (* MAP FLAGS FOR RESERVE/RELEASE *)
  OWN_TASK_LUNO      = #FF;      (* LUNO FOR OWN PROGRAM FILE *)

" MAP SVC SUB OP-CODES:
    CHANGE_SEG       = #00;      (* CHANGE SEGMENT *)
    CREATE_SEG       = #01;      (* CREATE SEGMENT *)
    RESERVE_SEG      = #02;      (* RESERVE SEGMENT *)
    RELEASE_SEG      = #03;      (* RELEASE SEGMENT *)
    GET_SEG_STATUS   = #04;      (* GET SEGMENT STATUS *)
    FORCE_WRITE      = #05;      (* FORCE WRITE SEG. *)
    LOAD_SEG         = #09;      (* MAKE SEG. MEMORY RESIDENT *)
    UNLOAD_SEG       = #0A;      (* RELEASE MEMORY RESIDENT SEG. *)
    SET_EXCLUSIVE    = #0B;      (* SET EXCLUSIVE USE OF SEG. *)
    RESET_EXCLUSIVE  = #0C;      (* RESET EXCLUSIVE USE *)


TYPE
(*-------------------------------------------------------------*)
(*                  COMMONLY USED TYPE DEFINITIONS             *)
(*-------------------------------------------------------------*)

  BYTE               = 0..255;
  NAME               = PACKED ARRAY [1..6] OF CHAR;

  SEGTBL_ENTRY       = PACKED RECORD;
    PROC_NAME        : NAME;
    SEG_ID           : INTEGER;
    END;
```

```
(*$ PAGE *)
(*-------------------------------------------------------------*)
(*          DNOS 990 SEGMENTATION SVC FIELD DEFINITIONS        *)
(*-------------------------------------------------------------*)

  T_MAPSVC          = PACKED RECORD;
    SVC_CODE        : BYTE;             (* SVC CODE = >40 *)
    ERR_CODE        : BYTE;             (* RETURNED ERROR CODE *)
    OP_CODE         : BYTE;             (* MAP OP CODE *)
    MAP_LUNO        : BYTE;             (* LUNO OF PROG FILE *)
    MAP_FLAG        : BYTE;             (* FLAGS              *)
    MAP_POS         : BYTE;             (* MAP POSITION (0,1,2) *)
    FILL1           : INTEGER;          (* FIRST WORD NEW SEG ID *)
    NEW_SEGID       : INTEGER;          (* NEW SEGMENT *)
    OLD_SEGID       : INTEGER;          (* OLD SEGMENT *)
    SEG_ADDR        : INTEGER;          (* RETURNED ADDR OF SEG *)
    SEG_SIZE        : INTEGER;          (* SIZE OF SEGMENT (I/O) *)
    SEG_ATTRIB      : INTEGER;          (* SEGMENT ATTRIBUTES *)
    FILL2           : INTEGER;          (* RESERVED *)
  END;  " T_MAPSVC "

  SEG_REC           = RECORD
    SEG_DATA        : ARRAY [1..100] OF INTEGER;
    END;

VAR
  SEG_PTR           : @SEG_REC;         (* POINTER TO SEGMENT DATA *)
  SEG_TABLE         : ARRAY [1..4] OF INTEGER;   (* RUNTIME SEG ID *)
  STATUS            : INTEGER;

COMMON
  MAPSVC            : RECORD
    MAP_SVCB        : T_MAPSVC;
    END; "

PROCEDURE SVC$(SVC_BLOCK : INTEGER); EXTERNAL;
```

```
(*$ PAGE *)
PROCEDURE INITSVC;
(*-------------------------------------------------------------------*)
(*      THIS ROUTINE INITIALIZED THE MAP SVC BLOCK FOR GET EMPTY      *)
(*      SEGMENT AND CHANGE SEGMENT CALLS BY RUNTIME ID.               *)
(*-------------------------------------------------------------------*)

ACCESS MAPSVC;

BEGIN " PROCEDURE INITSVC
  WITH MAPSVC, MAP_SVCB DO
    BEGIN " INITIALIZE MAP SVC BLOCK
      SVC_CODE := MAP_SVC_CODE;      (* SET SVC CODE *)
      MAP_LUNO := 0;                 (* RUNTIME SEGMENTS ONLY *)
      MAP_POS  := CODE_SEG_POS;      (* MAP IN THIRD SEGMENT *)
      FILL1 := 0;                    (* FIRST WORD NEW SEG TO ZERO *)
      OLD_SEGID := #FFFF;            (* INITIAL OUTGOING SEG TO -1 *)
    END; " INITIALIZE MAP SVC BLOCK
END;  " PROCEDURE INITSVC



PROCEDURE SWITCHSEG(VAR SEG_ID : INTEGER
                    VAR STATUS : INTEGER);
(*-------------------------------------------------------------------*)
(*      THIS ROUTINE ISSUES THE MAP SVC SUPERVISOR CALL AFTER        *)
(*      SETTING THE NEW SEGMENT ID AND TYPE OF MAP SVC (OP_CODE).    *)
(*      SOME ERROR PROCESSING IS ILLUSTRATED BUT WOULD MOST LIKELY   *)
(*      BE INSUFFICIENT FOR MOST PROGRAMS.                           *)
(*-------------------------------------------------------------------*)

ACCESS MAPSVC;

BEGIN " PROCEDURE SWITCHSEG
  WITH MAPSVC, MAP_SVCB DO
    BEGIN " INITIALIZE MAP SVC BLOCK
      OP_CODE  := CHANGE_SEG;        (* SET FOR CHANGE SEG *)
      NEW_SEGID := SEG_ID;           (* SET INSTALLED ID *)
      MAP_FLAG := #18;               (* MEMORY BASED SEGMENT, *)
                                     (*  EXCLUSIVE ON OUTGOING *)
      SVC$(LOCATION(MAP_SVCB));      (* MAP IN SEGMENT *)
      STATUS := ERR_CODE;

    END;  " INITIALIZE MAP SVC BLOCK
END;  " PROCEDURE SWITCHSEG
```

```
(*$ PAGE *)
PROCEDURE CREATESEG(VAR SEG_ID : INTEGER;
                    VAR SEGADR : INTEGER;
                    VAR STATUS : INTEGER;
                        SIZE : INTEGER);
(*-------------------------------------------------------------------*)
(*      THIS ROUTINE ISSUES THE MAP SVC SUPERVISOR TO CREATE A       *)
(*      MEMORY BASED SEGMENT.  INPUT TO THE ROUTINE IS THE SIZE      *)
(*      OF THE SEGMENT DESIRED.  OUTPUTS ARE THE ADDRESS OF THAT     *)
(*      SEGMENT AND THE RUN-TIME ID OF THE SEGMENTS.                 *)
(*      NOTE THAT THE SET EXCLUSIVE OPERATION IS ALSO ISSUED IN      *)
(*      THIS ROUTINE TO INSURE SEGMENT IS NOT TRASHED BY O/S.        *)
(*-------------------------------------------------------------------*)

ACCESS MAPSVC;

BEGIN " PROCEDURE CREATESEG
  WITH MAPSVC, MAP_SVCB DO
    BEGIN " CREATE_SEGMENT
      OP_CODE  := CREATE_SEG;          (* SET FOR CHANGE SEG *)
      NEW_SEGID := SEG_ID;             (* SET INSTALLED ID *)
      MAP_FLAG := #18;                 (* MEMORY BASED SEGMENT, *)
                                       (*  EXCLUSIVE ON OUTGOING *)
      SEG_SIZE := SIZE;                (* SET DESIRED SIZE *)
      SEG_ATTRIB := #8420;             (* READABLE, NON SYSTEM, *)
                                       (*  SHARE PROT, EXEC PROT *)
      SVC$(LOCATION(MAP_SVCB));        (* MAP IN SEGMENT *)
      IF (ERR_CODE = 0) THEN
        BEGIN " RESERVE & RETURN SEG INFO
          OP_CODE := SET_EXCLUSIVE; (* SET EXCLUSIVE USE      *)
          NEW_SEGID := OLD_SEGID;   (* MOVE SEG ID TO SEG 1   *)
          SVC$(LOCATION(MAP_SVCB));
          IF (ERR_CODE = 0) THEN
            BEGIN " RETURN SEG INFO
              SEG_ID := OLD_SEGID;  (* RETURN SEGID TO CALLER *)
              SEGADR := SEG_ADDR;
            END;  " RETURN SEG INFO
        END;  " RESERVE & RETURN SEG INFO
      STATUS := ERR_CODE;
    END;  " CREATE SEGMENT
END;  " PROCEDURE CREATESEG
```

```
(*$ PAGE *)
BEGIN
(*-------------------------------------------------------------------*)
(*      MAIN ROUTINE.  MAPS FOUR EMPTY SEGMENTS, MOVES 100           *)
(*      INTEGERS IN TO THE SEGMENTS, MAPS EACH SEGMENT BACK IN,      *)
(*      THEN VERIFIES THE DATA IN THE SEGMENTS.                      *)
(*-------------------------------------------------------------------*)

  INITSVC;                              (* INITIALIZE MAP SVC *)
  REWRITE(OUTPUT);                      (* OPEN OUTPUT, WRITE FIRST MSG *)


  FOR I := 1 TO 4 DO
    BEGIN " CREATE AND FILL EMPTY SEGMENTS
      CREATESEG(SEG_TABLE[I], SEG_PTR::INTEGER, STATUS, 200);
      IF STATUS = 0 THEN
        FOR J := 1 TO 100 DO
          SEG_PTR@.SEG_DATA[J] := I
      ELSE
        WRITELN(' ERROR IN MAPPING SEGMENT; ERROR CODE = ',
          STATUS:4);
    END;  " CREATE AND FILL EMPTY SEGMENTS

  IF STATUS <> 0 THEN ESCAPE MAIN3;

  FOR I := 1 TO 4 DO
    BEGIN " REMAP SEGMENTS AND VERIFY SEGMENT CONTENTS
      SWITCHSEG(SEG_TABLE[I], STATUS);
      IF STATUS = 0 THEN
VS:     BEGIN " VERIFY SEGMENTS
          FOR J := 1 TO 100 DO
            IF SEG_PTR@.SEG_DATA[J] <> I THEN
              BEGIN " WRITE MESSAGE
                WRITELN(' SEGMENTS DO NOT VERIFY ');
                ESCAPE VS;
              END;  " WRITE MESSAGE
          WRITELN(' SEGMENT NUMBER ',I:1,' VERIFIED');
        END   " VERIFY SEGMENTS
      ELSE
        WRITELN(' ERROR IN MAPPING SEGMENT NUMBER ',I:1,
          '; ERROR CODE = ', STATUS:4);
    END;  " REMAP SEGMENTS AND VERIFY SEGMENT CONTENTS
END.
```

```
<<< MAIN3 LINK CONTROL >>>
FORMAT IMAGE,REPLACE
LIBRARY K.TIMIX.O
LIBRARY .TIP.OBJ
PROCEDURE RUNTIM
INCLUDE (RUNTIM)
;
PHASEO, MAIN3
INCLUDE (P$MAIN)
ALLOCATE
INCLUDE (MAIN3)
END
```

```
<<< RESULTS OF MAIN3 RUN >>>

SEGMENT NUMBER 1 VERIFIED
SEGMENT NUMBER 2 VERIFIED
SEGMENT NUMBER 3 VERIFIED
SEGMENT NUMBER 4 VERIFIED
```

```
0002                    IDT  'MAPSEG'
0003 0000               CSEG 'SEGTBL'
0004 0000        SEGTBL BSS  2+(10*8)
0005 0052               CEND
0006             *
0007             *      THIS ROUTINE TAKES A SIX-CHARACTER SUBROUTINE NAME
0008             *      PASSED AS AN ARGUMENT AND LOADS THE ASSOCIATED SEGMENT
0009             *      IF THAT SEGMENT IS NOT ALREADY IN MEMORY.  THIS
0010             *      ROUTINE REFERENCES THE COMMON "SEGTBL" FOR THE NAMES
0011             *      AND SEGMENT ID'S OF THE SEGMENTS.
0012             *
0013             *      CALLING SYNTAX:
0014             *
0015             *           WORKING-STORAGE SECTION.
0016             *           01 SEG-NAME-1  PIC X(6) VALUE "PROC1 "
0017             *           01 STATUS      PIC 9(4) COMP-1.
0018             *              .
0019             *              .
0020             *              CALL "MAPSEG" USING SEG-NAME-1, STATUS.
0021             *
0022             *      RETURN STATUS:
0023             *           >FFFF:  SUBROUTINE NAME NOT FOUND IN LIST
0024             *           ALL OTHERS:  MAP SVC ERROR CODE (AS INTEGER)
0025             *
0026             *      REGISTER USAGE:
0027             *           R0 - SCRATCH
0028             *           R1 - WORKING COPY OF DESIRED ROUTINE NAME
0029             *           R2 - COUNT ENTRIES IN SEGTBL COMMON
0030             *           R3 - ADDRESS OF DESIRED ROUTINE NAME
0031             *           R4 - ADDRESS OF RETURN STATUS CODE
0032             *           R5 - INDEX INTO SEGTBL COMMON
0033             *
0034                    DXOP SVC,15
0035 0000        MAPSEG EVEN
0036                    DEF  MAPSEG
0037 0000 0000"         DATA WS,MAP000
     0002 0004'
0038 0004        MAP000 EVEN
0039 0004 C09D          MOV  *R13,R2          GET ADDRESS ARGUMENT LIST
0040 0006 C032          MOV  *R2+,R0          GET BYTE LENGTH OF ARGS
0041 0008 0280          CI   R0,4             IF NOT TWO, JUST RETURN
     000A 0004
0042 000C 1622          JNE  MAPXIT
0043 000E C0F2          MOV  *R2+,R3          GET ADDRESS OF SEGMENT NAME
0044 0010 C112          MOV  *R2,R4           GET ADDRESS OF STATUS FIELD
0045 0012 04D4          CLR  *R4              ASSUME NO ERRORS NOW
0046 0014 0205          LI   R5,SEGTBL        GET ADDRESS OF SEGMENT TABLE
     0016 0000+
0047 0018 C0B5          MOV  *R5+,R2          GET NUMBER ENTRIES IN SEGTBL
0048 001A        MAP010 EVEN
0049 001A 0200          LI   R0,6             SET BYTE LENGTH OF NAME
     001C 0006
0050 001E C043          MOV  R3,R1            WORKING COPY ROUTINE NAME
0051 0020        MAP020 EVEN
0052 0020 8C75          C    *R5+,*R1+        SAME NAME
0053 0022 1611          JNE  MAP060           NO, TO NEXT ENTRY
0054 0024 0640          DECT R0               REDUCE BYTE COUNT
0055 0026 16FC          JNE  MAP020
```

```
0057                    *                               <<< HAVE MATCHING NAME IF HERE
0058                    *                                    R5 POINTS TO SEG ID
0059 0028
0060 0028 8815                  C     *R5,@NEWSEG           IS THIS SEGMENT ALREADY IN
     002A 0028"
0061 002C 1312                  JEQ   MAPXIT                YES:  DONE NOW
0062 002E C815                  MOV   *R5,@NEWSEG           NO:   SET SEGMENT ID
     0030 0028"
0063 0032 2FE0                  SVC   @MAPSVC               ISSUE MAP SVC
     0034 0020"
0064 0036 C020                  MOV   @MAPSVC,R0            TEST ERROR CODE
     0038 0020"
0065 003A 0240                  ANDI  R0,>00FF
     003C 00FF
0066 003E 1309                  JEQ   MAPXIT                NO ERROR:  NORMAL EXIT
0067 0040 04E0                  CLR   @NEWSEG               ERROR:  SET NO SEG IN
     0042 0028"
0068 0044 1005                  JMP   MAPERR                RETURN SETTING ERROR CODE
0069 0046
0070 0046               MAP060  EVEN
0071                    *             IF HERE, R0 IS 2 MORE THAN REMAINING LENGTH OF
0072                    *             NAME.  BY ADDING R0, WILL SKIP OVER REST OF NAME
0073                    *             AND SEGMENT ID FIELD.
0074                    *
0075 0046 A140                  A     R0,R5                 SELECT NEXT ENTRY
0076 0048 0602                  DEC   R2                    LOOP FOR NEXT NAME
0077 004A 16E7                  JNE   MAP010
0078                    *                               <<< NO MATCH IF HERE
0079 004C 0200                  LI    R0,>FFFF              SET ERROR CODE
     004E FFFF
0080 0050               MAPERR  EVEN
0081 0050 C500                  MOV   R0,*R4                RETURN ERROR CODE
0082 0052               MAPXIT  EVEN
0083 0052 0380                  RTWP                        RETURN TO CALLER
0084 0054                       PEND
0085 0054
0086 0000                       DSEG
0087 0000               WS      BSS   32
0088 0020
0089 0020               MAPSVC  EVEN                        MAP SVC BLOCK
0090 0020 4000                  DATA  >4000                   SVC CODE
0091 0022   00                  BYTE  >00                     OP-CODE = CHANGE SEG
0092 0023   FF                  BYTE  >FF                     LUNO = OWN PROGRAM FILE
0093 0024 C003                  DATA  >C003                   FLAGS:  MAP BY LUNO,
0094                    *                                             INSTALLED ID,
0095                    *                                             MAP POSITION 3
0096 0026 0000                  DATA  0                       NEW SEGMENT ID (WD 1)
0097 0028 0000          NEWSEG  DATA  0                                      (WD 1)
0098 002A 0000                  DATA  $-$                     RETURNED SEG ADDRESS
0099 002C 0000                  DATA  $-$                     RETURNED SEG LENGTH
0100 002E 0000                  DATA  $-$                     RETURNED SEG ATTRIBUTES
0101 0030 0000                  DATA  0                       RESERVED
0102 0032                       DEND
NO ERRORS,     NO WARNINGS
```

```
LINE  DEBUG PG/LN  A...B.....................................................

   1              IDENTIFICATION DIVISION.
   2              PROGRAM-ID. COBMAN.
   3              AUTHOR. S. KEN CULP.
   4              ENVIRONMENT DIVISION.
   5              CONFIGURATION SECTION.
   6              SOURCE-COMPUTER.  TI-990-10.
   7              OBJECT-COMPUTER.  TI-990-10.
   8              INPUT-OUTPUT SECTION.
   9              FILE-CONTROL.
  10                  SELECT LIST-FILE ASSIGN TO PRINT, "OUTPUT".
  11              DATA DIVISION.
  12              FILE SECTION.
  13              FD   LIST-FILE LABEL RECORDS OMITTED.
  14              01   DATA-RECORD    PIC X(34).
  15              WORKING-STORAGE SECTION.
  16              01   SEQ-RECORD.
  17                   02  HEADER     PIC X(28) VALUE
  18                           " PROCEDURE NAME ENTERED IS: ".
  19                   02  PROC-NAME  PIC X(6).
  20              01  SNL              PIC X(6).
  21              01  MAP-STATUS           PIC 9(5)  COMP-1.
  22              01  PROC1-NAME       PIC X(6)  VALUE  "PROC1 ".
  23              01  PROC2-NAME       PIC X(6)  VALUE  "PROC2 ".
  24              01  PROC3-NAME       PIC X(6)  VALUE  "PROC3 ".
  25              01  PROC4-NAME       PIC X(6)  VALUE  "PROC4 ".
  26              PROCEDURE DIVISION.
  27  >0000       MAIN-01.
  28  >0000           OPEN OUTPUT LIST-FILE.
  29  >0006           MOVE "MAIN  " TO PROC-NAME,
  30                  WRITE DATA-RECORD FROM SEQ-RECORD.
  31  >0016           CALL "MAPSEG" USING PROC1-NAME, MAP-STATUS.
  32  >0018             IF MAP-STATUS = ZERO
  33                      CALL "PROC1" USING SNL,
  34                      MOVE SNL TO PROC-NAME,
  35                      WRITE DATA-RECORD FROM SEQ-RECORD.
  36  >0030           CALL "MAPSEG" USING PROC2-NAME, MAP-STATUS.
  37  >0032             IF MAP-STATUS = ZERO
  38                      CALL "PROC2" USING SNL,
  39                      MOVE SNL TO PROC-NAME,
  40                      WRITE DATA-RECORD FROM SEQ-RECORD.
  41  >004A           CALL "MAPSEG" USING PROC3-NAME, MAP-STATUS.
  42  >004C             IF MAP-STATUS = ZERO
  43                      CALL "PROC3" USING SNL,
  44                      MOVE SNL TO PROC-NAME,
  45                      WRITE DATA-RECORD FROM SEQ-RECORD.
  46  >0064           CALL "MAPSEG" USING PROC4-NAME, MAP-STATUS.
  47  >0066             IF MAP-STATUS = ZERO
  48                      CALL "PROC4" USING SNL,
  49                      MOVE SNL TO PROC-NAME,
  50                      WRITE DATA-RECORD FROM SEQ-RECORD.
  51
  52  >007E           CLOSE LIST-FILE.
  53        ZZZZZZ END PROGRAM.                              *** END OF FILE
```

```
LINE  DEBUG PG/LN  A...B.....................................................

   1               IDENTIFICATION DIVISION.
   2               PROGRAM-ID. PROC1.
   3               AUTHOR. S. KEN CULP.
   4               ENVIRONMENT DIVISION.
   5               CONFIGURATION SECTION.
   6               SOURCE-COMPUTER.  TI-990-10.
   7               OBJECT-COMPUTER.  TI-990-10.
   8               DATA DIVISION.
   9               LINKAGE SECTION.
  10               01  SNL              PIC X(6).
  11               PROCEDURE DIVISION USING SNL.
  12  >0000        MAIN-01.
  13  >0000            MOVE "PROC1 " TO SNL.
  14  >0006        MAIN-02.
  15  >0006            EXIT PROGRAM.
  16        ZZZZZZ END PROGRAM.                              *** END OF FILE
```

NOTE: PROCEDURES CPROC2, CPROC3, AND CPROC4 ARE THE SAME AS THIS PROC EXCEPT FOR RETURNED STRING IN SNL AND NAME OF ROUTINE.

```
<<< MAIN  LINK CONTROL >>>

FORMAT IMAGE,REPLACE
NOAUTO
LIBRARY K.TIMIX.O
LIBRARY .S$SYSLIB
PROCEDURE CRUNTM
INCLUDE (RCBPRC)
;
PHASE0, CMAIN1
INCLUDE (RCBTSKD)
INCLUDE (RCBMPD)
INCLUDE (COBMAN)
INCLUDE (MAPSEG)
INCLUDE (SEGTBL)
;
SEGMENT 3,SEG1,ID 7
INCLUDE (CPROC1)
INCLUDE (CPROC2)
;
SEGMENT 3,SEG2,ID 8
INCLUDE (CPROC3)
INCLUDE (CPROC4)
END
```

```
<<< RESULTS OF MAIN  RUN >>>

PROCEDURE NAME ENTERED IS: MAIN
PROCEDURE NAME ENTERED IS: PROC1
PROCEDURE NAME ENTERED IS: PROC2
PROCEDURE NAME ENTERED IS: PROC3
PROCEDURE NAME ENTERED IS: PROC4
```

OPERATING SYSTEM SUPPORT FOR ASYNCHRONOUS TERMINALS

OPERATING SYSTEMS

Daniel Gillen
Texas Instruments
Austin, Texas

INTRODUCTION

A Device Service Routine (DSR) must provide support for two hardware units, the peripheral device and the 990 chassis resident controller. A DSR structure will be discussed which separates the software support for these units into two major DSR code elements. The first is a Peripheral Service Routine (PSR) module providing support for the peripheral device independent of the controller. The second is a Hardware Service Routine (HSR) providing controller support.

The goals leading to the DSR design and the problems addressed by the design are presented. The DSR structure will be discussed in terms of functionality, logic and data flow and module interfaces. A specific implementation will be presented as an example of the design philosophy.

TERMINOLOGY

At Texas Instruments the peripheral device support software is linked with the operating system and is called a Device Service Routine (DSR). The DSR provides a software interface between the application software and the peripheral hardware. Two terms key to this discussion are controller and peripheral device. These terms will be defined in a somewhat restricted way for purposes of this paper. The definitions are oriented around Texas Instruments Business System products.

Peripheral Device

Peripheral device is the term used to refer to input and/or output hardware capable of being connected to a computer. My use of this term assumes a separate hardware unit, a controller, is required to interface this peripheral device to the computer. The terms peripheral device, device and peripheral will be used interchangeably in this paper.

Controller

A controller is a hardware unit which interfaces directly to a CPU. Typically the controller is a printed circuit board which resides in the computer chassis. Two types of controllers will be considered for Texas Instruments Business Systems, CRU controllers interfacing to the Communications Register Unit (CRU) and TILINE controllers interfacing to the TILINE data bus. This paper will consistently use the term controller for this hardware unit though historically many other terms have been used. Some of these other terms include interface, interface module, board and card.

A few examples are listed here for purposes of clarification.

Texas Instruments Business Systems peripheral devices:

1. Omni 800 Model 810 printer
2. Opti 900 Model 940 Video Display Terminal
3. WD800 Winchester Disk Drive

Texas Instruments Business Systems controllers:

1. CI421 - S300 Two Channel Communications Option Board
2. CI401 - S600/S800 Communications Interface Module
3. TPBI - S600/S800 TILINE Peripheral Bus Interface
4. TMS9902 UART on the 990/10A and S300 processor boards

DSR STRUCTURE

Three major functional levels exist between an application and an I/O peripheral device. The application interfaces to an Input/Output subsystem which, in turn, interfaces to I/O device hardware.

APPLICATION <---> I/O SUBSYSTEM <---> I/O DEVICE

Looking one level lower at the I/O subsystem structure we see essentially three separate functions being performed. The operating system pre-processes the application request before passing the request to a DSR. The DSR executes the I/O request then passes it to the operating system post processing element which reports completion to the application. This process is illustrated as follows.

OS PRE-PROCESSING ---> DSR ---> OS POST-PROCESSING

Looking still one level lower at the DSR we have three more interfaces. There is an interface to the operating system, an interface to the I/O device and an interface to the controller illustrated as follows.

OS I/F <---> DEVICE I/F <---> CONTROLLER I/F

This paper analyzes the DSR structure. The three functions of the DSR will be given names. The operating system interface will be denoted OSI. The peripheral device interface will be called a Peripheral Service Routine (PSR) and the controller interface will be called a Hardware (Controller) Service Routine (HSR). Figure 1 pictures the levels of a Device Service Routine (DSR).

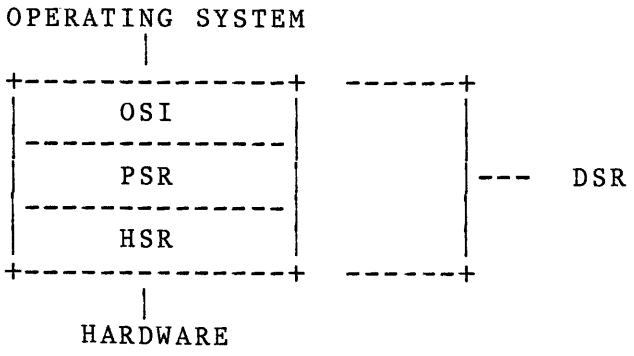
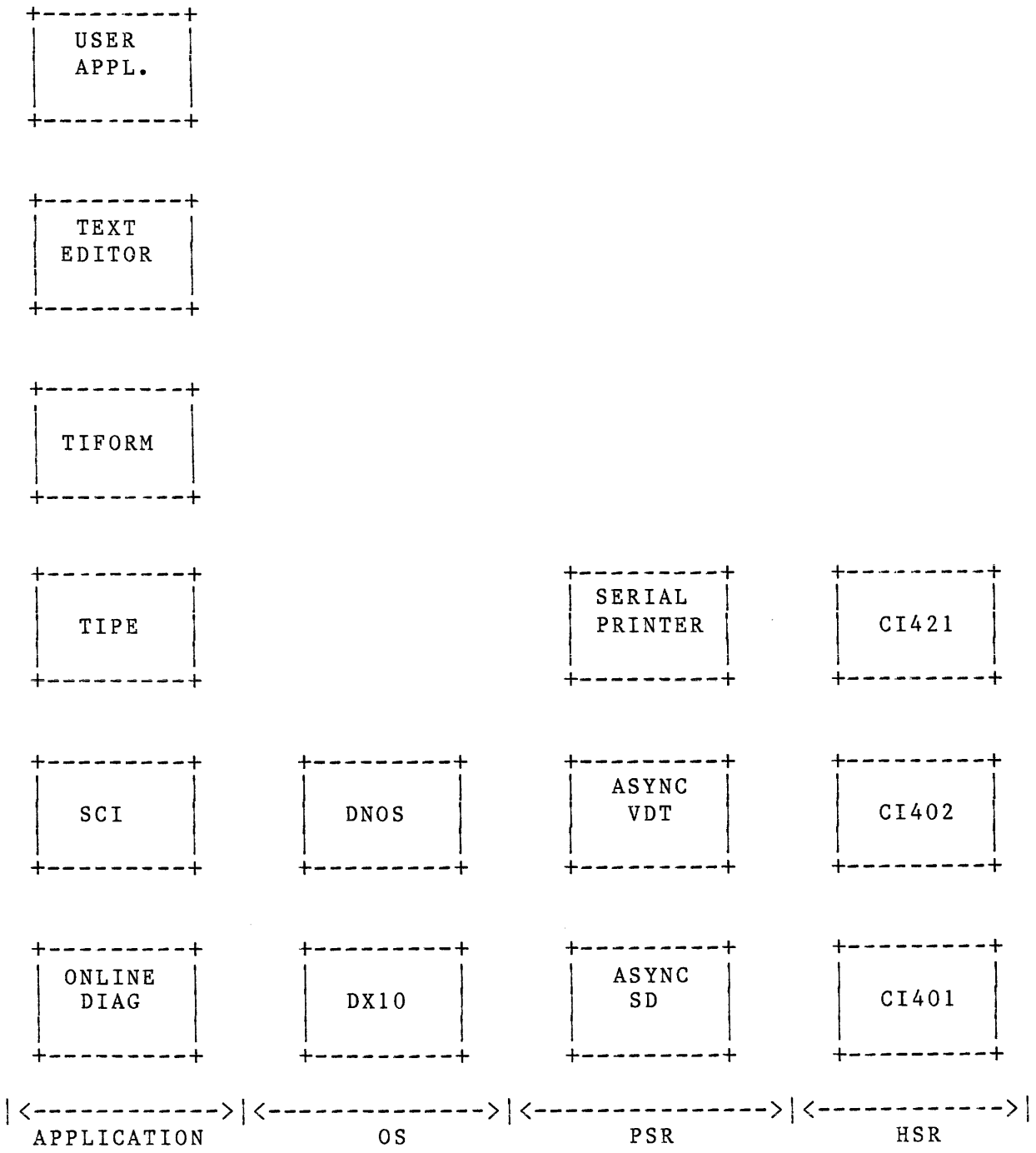


Figure 1 DSR STRUCTURE

Table 1 lists examples of software elements at the various levels. Each column of the table identifies a class of software. The column entries indicate specific examples within the class. The key point to consider in the table is that a unique path from application to I/O device is defined by choosing one entry from each column. The table has only a few entries for each column but it should be apparent that there are a large number of paths when all possible combinations are considered. Several issues deserve mention relative to the table.

The OSI level does not appear explicitly as a separate column in Table 1. The OS interface to the two operating systems is very similar and our implementation did not include a separate module for the OS interface. The the OS interface logic is embedded in the PSR for this implementation. Assume, for purposes of this discussion, the two operating systems are identical. This is an over simplification but will allow us to concentrate on other DSR interfaces.

Table 1 SOFTWARE ELEMENTS



Advantages of isolating controller support to the HSR module and device support to the PSR should be obvious. Only one software module is required for each device and one for each controller. This pays dividends in development as well as in sustaining during the life of the product. Similarly, applications have an identical, ignoring the OS, device interface across all controllers. When a group of peripheral devices are supported by DSR's using this design, the device support may easily be moved to a new controller. Only one HSR module must be developed to support all current peripheral devices on a new controller.

The key to the design is the definition of the interface to the HSR. The first step taken was to bound the problem. The goal was to develop an interface scheme for asynchronous peripheral devices. Thus, the set of controllers was limited to asynchronous controllers. Next, a set of functions supported by asynchronous controllers was compiled. Finally, a generic interface was specified to provide access to these controller functions.

The specification of the generic interface was an iterative process and had to satisfy several parameters. Some key parameters included:

1. Controller independence

2. Access to full controller functionality

3. Emulation of buffered controller for output

4. Provide PSR required services

The decision to emulate a buffered controller was based partially on experience gained supporting non buffered controllers. The buffering referred to here is not the one or two character buffering typically done by UART chips but buffers in the 32-128 character range. The purpose was to allow better separation between PSR and HSR output processing. One benefit of this approach was minimizing the amount of output processing with interrupts masked. This is a critical issue for VDT's where the ratio of output data to input data at the CPU is heavily weighted in the output direction.

The functions supported by the HSR are numerous. They can be grouped into the following classes.

1. Controller initialization.
2. Read/write operational parameters.
3. Set/reset output signals or functions.
4. Read input signals or functions.
5. Status change notification.
6. Read/write data characters.
7. Timer services.
8. Controller interrupt processing.

AN IMPLEMENTATION

Now an implementation of a set of Device Service Routines (DSR's) will be discussed. The DSR's fit the basic structure introduced in the previous portion of the paper. They supported asynchronous devices attached to asynchronous controllers. The emphasis will be on logic flow and software interfaces. Figure 2 illustrates the DSR structure, logic and data flow. This implementation separated the PSR level into two modules. One will be referred to as the TSR and the other as the Interrupt Service Routine (ISR). Table 2 indicates functions performed by the DSR modules for this implementation.

Table 2 DSR FUNCTIONS

TSR - TERMINAL SERVICE ROUTINE	- All DSR entry points except interrupt entry (Request/Initial, Power up, Abort, Timeout, and Delayed Reentry) - Request and completion reporting I/F to OS - Runs in PDT workspace - Provides software interface to terminal - Terminal dependent logic
ISR - INTERRUPT SERVICE ROUTINE	- Contains interrupt entry of the DSR - I/F to HSR for interrupt processing - High priority receive character processing - Runs in DSR interrupt workspace
HSR - CONTROLLER SERVICE ROUTINE	- Generic (subroutine) software interface to the controller hardware - Contains all controller dependent logic - Contains all direct access to controller - Emulation of buffered controller * Software FIFO's - Maintains controller status and statistics

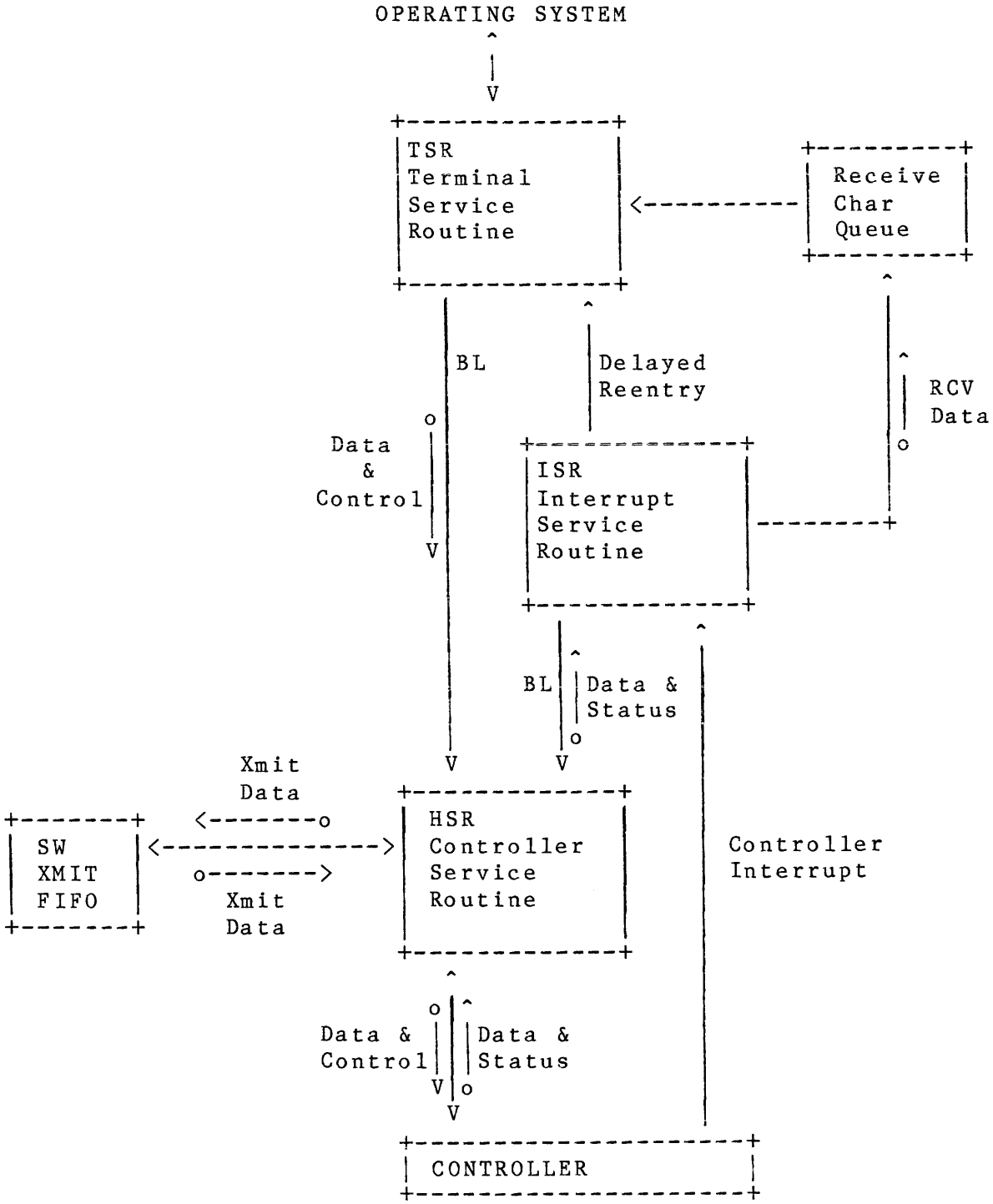


Figure 2 DSR LOGIC FLOW

The TSR module contains all DSR entry points except the interrupt entry. It accepts requests from the OS and reports completions to I/O subsystem of the OS. The primary function of the TSR is to provide a software interface to the peripheral device. The actual functions vary considerably based on the type of device. The primary device types supported in this implementation were VDT's, and serial printers. The TSR combined with the ISR to support the peripheral device.

The TSR performs initial processing for all requests. The TSR calls the HSR for the output of data. The HSR stores output data in a transmit FIFO until the controller is prepared to accept the data. The HSR can not accept data when the transmit FIFO fills with data waiting to be transmitted. In this event the TSR requests notification when the HSR can accept data. The HSR will notify the ISR when it can accept more transmit data and the ISR will "schedule" the TSR using the DSR reentry support of the OS. Typically the TSR considers the output request complete before all the data has actually been transmitted.

Read requests are processed, by the TSR, entirely from a receive character queue. The receive data characters are stored in this character queue by the ISR routine. Other requests are processed primarily by the TSR with the aid of the ISR if required.

The ISR module contains some functions which might be considered device support and some that might be considered controller support. The ISR module was developed based on several decisions during the design phase. The ISR module contains the interrupt entry to the DSR and uses an interrupt workspace different than the Physical Device Table (PDT) workspace. This routine runs with controller interrupts masked. If the controller interrupts the CPU at interrupt level 8, the ISR has interrupt mask set at 7. The ISR calls the HSR to actually service the controller interrupt.

For the most part, ISR processing is independent of request processing of the DSR. Receive data is stored in the receive character queue even when no read request is active at the DSR. Error recovery action must be taken when the receive character queue becomes full. The ISR processes events requiring immediate attention. Some examples of ISR processing for keyboard devices are biding an application task, halting output, aborting I/O and aborting tasks. It also schedules TSR level elements to start or resume processing.

The HSR level does not support the concept of a read request. The HSR will decode the controller interrupt and report the cause for the interrupt to the ISR. If the cause of the interrupt was a received data character, the data character is also passed back to the ISR. There is no storage of receive data at the HSR level.

The generic interface to the HSR consists primarily of two mechanisms. The HSR is a set of subroutines with a branch and link (BL) call interface. A subroutine implements one or more generic functions for the specific controller in use. A "set DTR" subroutine call is made by the TSR. The HSR for a CRU controller might implement this as a "SBO DTR" CRU instruction but the HSR for a TILINE controller might implement the same subroutine using a "SOC @DTR,@OUTSIG(R12)" instruction to access the TILINE Peripheral Control Space (TPCS) for the controller. Identical requests from the TSR/ISR will invoke identical functions for all controllers. Provision is made for controller hardware differences. A "not supported" return is provided for each HSR routine. This return is taken when the requested function is not supported by the controller hardware.

A new requirement exists when one peripheral device type is supported by different controller types. The second part of the generic interface mechanism addresses linking one TSR with multiple HSR's. The HSR's would each contain subroutines with identical names. Our solution to this problem was to provide a subroutine branch table in each HSR. The address of each generic subroutine was located at the same position, relative to the beginning of the branch table, in each HSR. The TSR maintains a pointer to the HSR branch table. This pointer is in the table (PDT or PDT extension) associated with a specific peripheral device. The reentrant TSR software accesses the proper HSR based on the contents of the tables for the peripheral device. Figure 3 illustrates the linkage. If SETDTR is an index to the "set DTR" subroutine entry in the HSR branch table then the instruction sequence

```
MOV  @PDTHSR(R4),R5
MOV  @SETDTR(R5),R6
```

places the "set DTR" subroutine address in R6. When R4 contains the address of the PDT and PDTHSR is the index of the HSR branch table pointer from the start of the PDT.

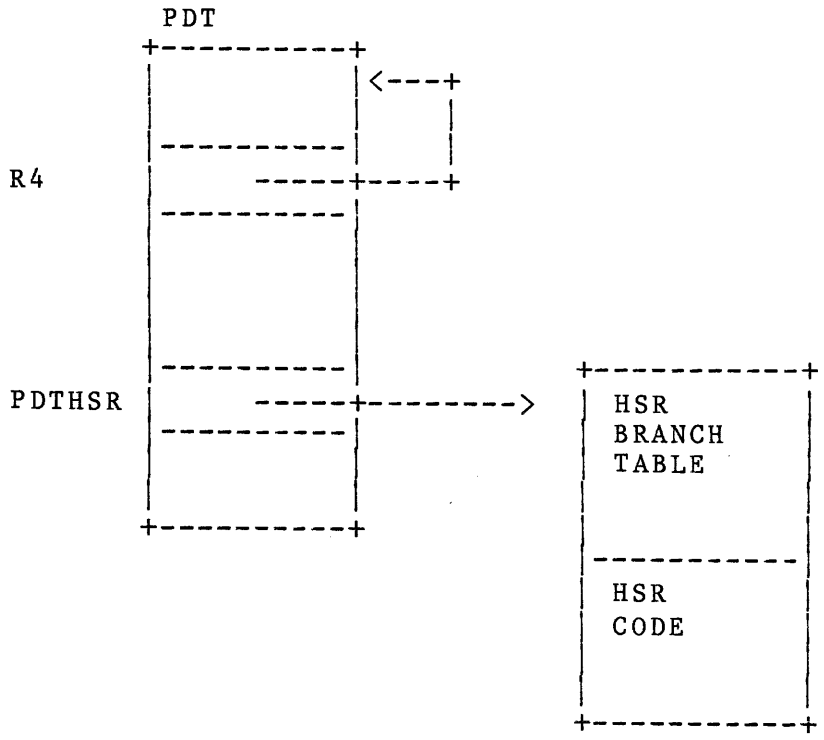


Figure 3 TSR/HSR LINKAGE

SUMMARY

The philosophy of separating controller and device support at the DSR level is not a new concept. It has been implemented in many DSR's in many different ways. The specific implementation discussed in this paper fit our purposes but may not fit another set of objectives. The specific implementation shows significant benefits when only one interface (controller/device) is isolated. Expanding this concept should prove even more beneficial. One goal of the implementation described was to define a HSR generic interface supporting all asynchronous controllers (TI Business Systems). New controllers may require extensions of the generic interface definition to satisfy this goal, but the benefits seem to make it worth the effort.

DNOS FILE SECURITY
A Presentation for TI-MIX 1983
Operating Systems Session

by

Michael P. Simpson
Texas Instruments
Austin, Texas

1 INTRODUCTION

File security will be available as a SYSGEN option with DNOS 1.2. It will complement existing DNOS security features such as logon passcodes, user IDs and SCI privilege levels. The system is designed to be effective in a cooperative environment and easy to use. It will have little or no effect on users who choose not to include it in their system. Using a set of new SCI commands, a user will be able to define groups of users and specify which groups of users may access his files, as well as how the files may be accessed. The ability to secure program files, batch streams, SCI command procedures (procs), and data files provides a system manager with a high level of control over access to sensitive system components.

This paper introduces the scope, concepts, and functionality of DNOS file security from the point of view of the user. An illustration is provided as an example to clarify several important new concepts. These concepts are interrelated and must be understood before one can establish a secure environment.

2 SCOPE AND PURPOSE

DNOS file security provides a means to prevent access or destruction of secured files by unauthorized individuals. The extent to which this is successful depends on at least three factors: the skill and determination of the individual, the software tools available to him, and physical security measures. It would be difficult, if not impossible, to design a security system that would protect against a determined attempt by a skilled individual with access to powerful tools such as the SCI debugger. Physical security measures appear to be much more effective against this type of security threat. On the other hand, a reasonably effective level of security protection can be achieved by controlling access to the system, controlling access to powerful tools, and placing control of this access in the hands of responsible individuals.

3 CONCEPTS

There are several aspects of DNOS file security that must be understood separately. The two most important are the concepts of access groups and access rights. Access to a file is granted or denied based on the relationship between the access groups associated with a user and the access rights associated with a file. Access groups and access rights are defined and discussed below.

3.1 Access Groups.

An access group is simply a group of users. Users associated with an access group are called members of that access group. Any user can create an access group and specify which users are members of that group. When a file is secured, one specifies which type of access will be granted to which access groups.

3.2 Access Rights.

There are five access rights: read, write, delete, execute, and control. An access group may be given any combination of these access rights. A new SCI command will be available to assign and modify access rights to individual files.

Read access is more than just the right to read data from a file. If the file is an SCI batch stream or procedure, read access is the right to execute the batch stream or the procedure. Read access on a program file allows a user to issue the Map Program File (MPF) SCI command.

Write access is the right to write data to a file. It includes the ability to write over the existing data as well as write new data. Write access to a program file represents the right to install or delete tasks, segments, procedures, and overlays. Write access to a key indexed file includes the right to delete records from that file.

Execute access only has meaning when it is associated with a program file. It represents the right to execute tasks, segments, procedures, and overlays within the program file. Powerful tasks can be protected by installing them in secured program files.

Delete access is the right to delete a file. Delete access is also required in addition to write access to text edit a file.

Control access is the right to change the security associated with a file. This includes the right to change the set of access groups associated with a file, as well as change the access rights associated with each access group. Each secured file has one and only one access group with control access.

3.3 Access to a Secured File.

Each secured file can have access rights for as many as nine access groups. A different set of access rights can be defined for each access group. The access rights associated with an access group determine how the members of that access group can access the file. For example, assume a secured file has the read access right for the access group named MANAGER. Any user who is a member of the access group named MANAGER is granted read access to the file. Establishing access group membership and access rights for a file will be discussed in detail later.

A user's access groups are established at the time he logs on. Any changes to his access group membership are recorded on disk and do not take effect until the next time he logs on.

Access rights to a file are established when a LUNO is assigned. Any changes to the access rights associated with a file will not affect access rights through LUNOs currently assigned. Access rights are checked for each individual file operation and are enforced only for files. They have no meaning for directories. Any attempt to secure a directory will result in an error.

3.4 The Access Group Leader.

When an access group is created, the creator becomes the leader of the access group. The leader of an access group has the right to add users to the access group, delete users from the access group, assign leadership of the access group to another user, or delete the access group. Only one leader is allowed for each access group. If leadership is assigned to another user, that user becomes the only leader.

3.5 Predefined Access Groups.

There are two predefined access groups that exist on all secured systems. They are named PUBLIC and SYSMGR.

PUBLIC is an access group which has all users as members. It has no leader, it cannot be deleted, and its membership changes automatically as user IDs are added or deleted from the system. A secured file is unsecured by specifying all access rights for the access group named PUBLIC

SYSMGR is an access group which is created automatically and can never be deleted. Any user who is a member of SYSMGR has full access to any file and leadership capabilities for any access group. Due to the nature of this access group, members of SYSMGR cannot be a member of any other access group.

3.6 File Creation Access Group.

Every user has an associated file creation access group. If one has not been specified, PUBLIC is assumed. When a file is created, all access rights are assigned for the file creation access group of the creator. If a user's file creation access group is PUBLIC, all files he creates will be unsecured. There is an SCI command which allows a new file creation access group to be defined.

4 OVERVIEW OF A SECURED SYSTEM

A simplified representation of a secured system is illustrated on the following page. The system has only two access groups and three files. The example depicts the relationship between membership in an access group and access rights to a secured file. The paragraphs following the illustration describe the details of this relationship.

S I M P L E S E C U R E D S Y S T E M

A C C E S S G R O U P S

ACCESS GROUP NAME	MANAGERS
MEMBERS	FRED MARY

ACCESS GROUP NAME	CLERKS
MEMBERS	BETTY FRED BILL

S E C U R E D F I L E S

FILENAME	SALES	
ACCESS GROUPS	MANAGERS	READ, CONTROL, DELETE
	CLERKS	READ, WRITE

FILENAME	PAYROLL	
ACCESS GROUPS	MANAGERS	READ, WRITE, CONTROL
	CLERKS	READ

FILENAME	INV PROC	
ACCESS GROUPS	MANAGER	READ, WRITE,CONTROL

The access group named MANAGERS has members Fred and Mary. The access group named CLERKS has members Betty, Fred, and Bill. The files named SALES and PAYROLL have access rights defined for MANAGERS and CLERKS. The file which contains the INV proc has access rights defined only for MANAGERS.

Mary can write to the file named PAYROLL because she is a member of the access group named MANAGERS and write access is defined for that access group. However, if Mary attempts to write to the file named SALES, she will get an error because she is not a member of any access group with write access to the file. It is important to note that if Mary really needed to write to the file named SALES, she could issue the command to change her access rights. She can issue that command because she is a member of the access group named MANAGERS and control access is defined for that access group. Similarly, Bill can write to the file named SALES because he is a member of the access group named CLERKS and write access is defined for that access group. However, if Bill tried to write to the file named PAYROLL he would get an error because he is not a member of an access group with write access to that file. If he really needed to write to the file named PAYROLL, there are two things he can do. He could ask the leader of the access group named MANAGERS to make him a member of that access group or he could ask any member of the access group named MANAGERS to change the security on the file to give write access to an access group of which Bill is a member. Bill is a new employee and likes to try new commands. Luckily, when he tries the INV command he will get an error because he is not a member of an access group with read access to the INV proc.

5 USER INTERFACE

A set of new SCI commands is provided for file security. Most require the user to verify his identity by entering his logon passcode. SCI will not echo the password either interactively or in the batch stream listing. Passwords imbedded in batch streams may be represented by a synonym and must be protected by file security on the batch stream and listing.

5.1 Access Group Commands.

Several SCI commands are provided for creating, deleting, listing, and changing the membership of access groups. Most can only be issued by the leader of the access group. An attempt to issue a command which requires leadership by anyone other than the leader will result in an error.

Access groups are created by issuing the Create Access Group (CAG) command. Any user who has access to the proc can issue the CAG command. The access group will be created and the user issuing the command will become the leader.

Only the leader of an access group can issue the Modify Access Group (MAG) command. The command is used to add users to the access group, delete users from the access group, or assign a new leader of the access group. Each user ID to be added and the new leader's user ID must be valid.

Only the leader of an access group can issue the List Access Group Members (LAGM) command. It lists user IDs of all users which are members of the access group.

Only the leader of an access group can issue the Delete Access Group (DAG) command. It is the responsibility of the user issuing the command to insure that no files exist which permit access only to this access group. If such a file is accidentally overlooked, it becomes accessable only to the SYSMGR access group.

Any user may issue the List Access Group(LAG) command. It lists all access groups of which the user is a member. The output will indicate which access group is the user's file creation access group and those groups for which the user is the leader.

Any user may issue the Set Creation Access Group (SCAG) command. It allows a user to specify which access group will automatically have full access to files he may create. The user must be a member of any access group he specifies as a file creation access group. To prevent conflict between batch or background and foreground SCI file creation, this command updates the creation access group recorded on disk. The new file creation access group is not effective until the next time a user logs on under that user ID.

5.2 Access Rights Commands.

There are two commands provided to manipulate access rights. They list or modify the access groups and their corresponding access rights for an individual file. To issue

these commands, one must be a member of an access group with the control access right to the file.

The List Security Access Rights (LSAR) command lists the access groups and access rights associated with a particular file. The user must enter his logon passcode to verify his identity. The user specifies the file pathname and the output displays all access groups with access rights to the file. It also indicates which access rights are associated with each access group.

The Modify Security Access Rights (MSAR) command modifies the security on an individual file. It prompts the user for his logon passcode to verify his identity. The user specifies the file pathname, access group name, and which access rights are to be given to the access group. If the access group named PUBLIC is entered and all access rights are specified, the file becomes unsecured.

6 IMPACT TO EXISTING APPLICATIONS

Existing applications and utilities can be adapted to run in a secure environment with no code changes. There are different approaches to establishing a secure environment for an application. One can secure the application program, secure the files it accesses, or both. Which approach one chooses will depend on the nature of the application or utility. Utilities such as Initialize New Volume (INV) are inherently powerful and should be secured. Utilities such as Show File (SF) can probably be unsecured but protection on individual files will limit what files can be shown.

6.1 Securing An Application.

To control access to a powerful application or utility one can secure the program file, command procedure, batch streams, or any combination of these. This may be accomplished by creating an access group and adding as members, each user ID which can access the application. Execute access must be specified for this access group on the program file. Read access must be specified for this access group on the proc or the batch stream.

6.1.1 Securing Sensitive Files.

There are two things that must be considered when securing sensitive files. One must insure that unauthorized users cannot access the file. One must also insure that applications or utilities that must access the file have the necessary access rights to do so.

There are two categories of applications from the point of view of file security: applications that run in their own job, and applications that run in the user's job. The steps involved in establishing a secure environment depend on the category of the application.

Applications or utilities that run in their own job will automatically inherit the access rights of the user ID of the job. One must create an access group with that user ID as a member. One must also insure that all files the application must access are permitted to that access group with the appropriate access rights.

Applications or utilities that run in the user's job will inherit the access groups of the user. Any attempt by the application or utility to access a file in a way not allowed for the user will result in an error.

6.2 Security Bypass.

In certain circumstances it may be desirable for an application or utility to have access to a file but undesirable to give that access to a user. For example, a data base may be maintained by an application. The application needs write access to the data base files; However, it may be undesirable to give write access to users because that allows them to write to the file with programs other than the application which manages the data base. A new task attribute called security bypass is provided for circumstances such as this. A task that is installed with security bypass will be granted all access to any file. It is the responsibility of the task to enforce security and the responsibility of the system manager to insure the integrity of such tasks. A separate utility is provided to assign the security bypass attribute to a task. Access to this utility can be controlled by securing the proc and the program file in which it is installed.

7 DOCUMENTATION

The use of file security will be carefully documented in a new manual entitled DNOS Security Manager's Guide . It will include a thorough description of the role and responsibilities of the security manager. The DNOS System Command Interpreter Reference Manual will describe the security implications if any, in the descriptions of the individual commands.

NOTES

INTERPROCESS COMMUNICATION IN DNOS
A Presentation for TI-MIX 1983
Operating Systems Session

by

Lori Mohr Stuart
Texas Instruments
Austin, Texas

1. Interprocess Communication Mechanisms

There are many programming applications that require the use of synchronization or communication between processes. The Texas Instruments 990 operating systems provide several mechanisms by which processes can exchange signals or messages. These mechanisms are:

* Shared procedures (DNOS and DX10)
* Intertask message queues (DNOS and DX10)
* Semaphores (DNOS)
* Shared segments (DNOS)
* Event Synchronization (DNOS)
* Interprocess communication channels (DNOS)

Shared procedures and segments suffer from the limitation that the communicating tasks themselves must coordinate their use of the shared data. Message queues have only a rudimentary synchronization capability and no access control other than a usage convention which is not enforced by the operating system. DNOS semaphores are used for synchronization, but only between tasks in the same job. Event synchronization is a global mechanism by which one task can signal another, provided the signalling task knows the run-time ID and the job ID of the task to be signalled. A message passing facility may be necessary for the signalling task to obtain the job and run-time identifiers.

The interprocess communication (IPC) channel facility of DNOS is the most versatile of all of these mechanisms. It is a means of global communication between any two or more tasks in the system. Tasks exchange messages by reading and writing over IPC channels that are created by the system at the request of the user and exist independently of the tasks using them. The IPC facility can be used for both message exchange and synchronization, and it provides access control by which channel creators and users can limit the availability of a channel.

Every IPC channel has an owner task which is specified at the time the channel is created. Every message exchange is between the channel owner and some other task. Tasks communicate over an IPC channel by assigning logical unit numbers (LUNOs) to the channel and then using ordinary I/O supervisor calls to read and write the messages. As with other I/O supervisor calls, if a read or write to a channel cannot be processed immediately, i.e., if there is no matching channel request from another

task, the task issuing the read or write is optionally suspended until a matching channel request is issued. This allows synchronization between cooperating tasks. Furthermore, the use of I/O SVCs as the means of communication to channels means that the access control imposed on opens to files and devices is also imposed on open operations to channels. A task which successfully issues an open operation with exclusive write access to a channel is guaranteed to be the only requester task writing to the channel.

2. Uses of IPC

IPC channels can be used to implement several programming functions:

* Task synchronization
* Queue service
* Intermediate processing of data
* Sending and receiving messages

When IPC is used for task synchronization, the existence of a message may be more important than the message contents. Tasks may require synchronization in order to regulate access to shared resources or to guarantee that a series of operations are performed in a certain order. Like other I/O operations, IPC operations can suspend the issuer until the request completes. If the initiated I/O bit is set in the I/O call block or if the request is initiated by the Initiate Event SVC, the issuing task will continue to execute and can determine whether the request has completed by using a Wait for any I/O SVC, a Wait on Event SVC, or simply checking the busy bit in the request block. Either way, the tasks have a way of determining whether a message has been exchanged and can synchronize their actions accordingly.

IPC channels can also be used to implement queue servers. An IPC channel and a server task would be created for each provided service. Tasks would submit requests for service by writing to the IPC channel. The server task would read the requests from the channel and then process the requests.

Because the principal means of channel communication is resource-independent I/O SVCs, tasks that perform resource-independent I/O to files or devices require little or no change to use channels as sources of input or destinations of output. It is possible to use IPC to implement filters--tasks that perform intermediate processing of data. A task that writes its output to a VDT or file could just as easily write the data to a channel. The task that reads the data from the channel could perform some additional processing on the data and then output the data to a VDT or a

file or even to another channel for further processing. Because channel I/O usually requires no disk access, using channels to pipeline data can speed processing time.

There are two types of channels--symmetric and master-slave. Symmetric channels permit the use of resource-independent I/O SVCs to send messages between tasks. The channel interface of a symmetric channel owner is much the same as that of any other task, hence the name "symmetric". Symmetric channels are particularly useful for implementing filters.

Master-slave channels provide a mechanism for simulating I/O. A non-owner task (a slave) performs resource-specific I/O to the channel. The owner task (master) receives the full supervisor call block of the SVC issued by the slave, including any data that is being written. The master processes the slave's I/O request much like a Device Service Routine or file server processes requests. The master then returns the call block to the system, including any data being returned to the task. The master's interface to the channel is very different from that of a slave. The slave task may not even need to know that the resource to which it is sending requests is actually a channel. The master uses a special set of channel interface commands to obtain the requester's call block and return it to the system. A more detailed description of symmetric and master-slave channels follows below.

3. Accessing Symmetric Channels from Pascal

There are two ways to perform I/O to channels from tasks written in TI Pascal. The task can either use the standard Pascal I/O functions or can issue the supervisor call directly using the SVC$ routine. Resource-independent I/O to symmetric channels from either an owner task or a non-owner task can be done with standard Pascal text file I/O functions.

To use standard Pascal I/O to access a channel, a file variable of type TEXT should be defined. The file variable is then associated with the channel. This can be done with the SET$ACNM function, which associates a file variable with a pathname, or the SETLUNO function, which associates a file variable with a LUNO. Either of the pre-defined text files, INPUT or OUTPUT, may be used to access a channel. The synonym INPUT or OUTPUT must have been externally defined as the channel name.

The following Pascal functions can be used to access the channel:

* RESET(F) -- Open text file (channel) F for input.

* REWRITE(F) -- Open text file (channel) F for output.

* READLN(F) -- Read the next record from file F. This operation performs a read to the channel.

* WRITELN(F) -- Write the current contents of the line buffer to file F. This operation writes the buffer to the channel.

* EOF(F) -- Result is TRUE if the last READLN matched to a Write EOF operation.

Other Pascal functions that can be issued to a text file are listed in the TI Pascal Programmer's Guide. The READ and WRITE functions to a symmetric channel will not cause a read or write SVC to be issued to the channel. READ and WRITE only read from and write to a local line buffer. For the read or write to actually go to the channel, a READLN or WRITELN must be issued. Task termination causes the LUNO to the channel to be closed. Before issuing the close, the Pascal task will issue a Write EOF. This will match a read operation to the channel performed by another task and cause an EOF function performed by the other task to return the value TRUE.

An example program that implements a filter is shown in Figure 1. The program accepts input from a resource-independent source, processes the input, and then outputs the data to a resource-independent destination. In this case, the default text files INPUT and OUTPUT are used. Either of the two files or both of them could be channels. This task could either be a channel owner or a non-owner. Instead of using the default input and output files, the task could have obtained the filenames from a synonym, which could either be already known to the task or could have been passed in as a parameter.

4. Accessing Master-Slave Channels from Pascal

I/O operations from slave tasks can either be issued by standard Pascal I/O routines or by the SVC$ routine. Since master-slave channels are intended to simulate I/O, a slave task may perform any kind of I/O operation to the channel, provided that the type of I/O is compatible with the resource type of the channel and that the channel master is written to handle that type of I/O. The channel interface of a master of a master-slave channel must be written using direct supervisor calls with SVC$. The special I/O operations used by the channel master to obtain and return call blocks are not supported by intrinsic Pascal I/O functions. The master's channel interface is explained below.

5. Creating and Using IPC Channels

5.1 Creating and Deleting Channels

When a channel is created, a disk-resident channel descriptor is built. The channel has a pathname and is located in a directory like a file. Rebooting the system does not delete the channel. The channel has no memory resident representation until a task assigns a LUNO to it. A channel can either be created by a task by issuing the Create Channel SVC (I/O subopcode >9D) or from SCI by issuing the Create IPC Channel (CIC) command. In either case, the following information must be provided:

* Channel name, which must be a valid pathname.

* Program file which contains the owner task. The program file must be in the same directory as the channel. If the channel name is ".A.B.C", the program file must be in the directory ".A.B".

* Installed ID or name of the owner task in the program file.

* Channel type -- symmetric or master-slave.

* Channel scope -- global, job-local, or task-local.

* Channel message length -- the maximum number of bytes that can be transferred in one message.

* Channel type -- shared or non-shared.

* Default resource type (master-slave only)

Channels can be deleted from a task by the Delete Channel SVC (I/O subopcode >9E) or from SCI by the Delete IPC Channel (DIC) command.

IPC provides several options by which users can tailor a channel to meet their particular needs. A channel can either be symmetric or master-slave, can be available at a system-wide (global) level, a job-local level, or a task-local level, and can either be shared or non-shared. Each of these options is explained in the following sections and guidelines for choosing the type of channel are presented.

5.2 Symmetric Channels

Symmetric channels are suitable for applications that require only a simple exchange of data buffers or messages. If the application requires more complex I/O operations, as is frequently the case when I/O to a device or file is being simulated by I/O to a channel, the channel will have to be master-slave. If the application requires that the channel be multiplexed (used simultaneously by more than one task) and that the data exchange between the owner and requester be bi-directional, master-slave channels will be necessary. The reason for this will become clearer in the following discussion of shared and non-shared channels.

Every message transfer on a symmetric channel is either to or from the owner task. Each task opens the channel, performs read or write operations to the channel to accomplish the exchange of data, and then closes the channel. Each read operation to the channel must be matched by a write operation from another task. The actual data which is exchanged is the contents of the write buffer which is transferred to the read buffer of the task performing the read operation. In Figure 2, the supervisor call blocks for channel operations from two tasks are shown. The message written to the channel by task B is read by task A.

The operations allowed to symmetric channels are the same for both owners and requesters. The allowed operations are:

00	Open
01	Close
05	Read device status
09	Symmetric read
0B	Symmetric write
0D	Write EOF

The following sub-opcodes are allowed and perform operations identical to those shown:

Sub-opcode	Operation	Identical to
0A	Read direct	Symmetric read
0C	Write direct	Symmetric write
02	Close, write EOF	Close
03	Open Rewind	Open
04	Close and unload	Close

Open operations issued from requester tasks are queued until the owner task has opened the channel. Once the channel has been opened by the owner and one or more requesters, message transfer can take place. Symmetric reads and writes that cannot be processed immediately because there is no matching operation are queued. The channel requests are processed in a first come-first served manner. The next owner request is matched to the next requester request. If the operations are of different type (read-

write), the message transfer is performed. If the operations are of like type (read-read or write-write), both the owner and requester operations are returned with an error. A write EOF request matches a read, setting the EOF flag in the read call block and zeroing the actual read count.

One important difference between the owner's channel interface and that of the requester is that an owner may only have one operation outstanding to a particular channel at a time. If the owner initiates a request to the channel, the owner will have to wait until the first request completes before issuing another request. If the owner issues a second request before the first completes, the second request will receive an error.

5.3 Master-Slave Channels

Resource-specific channel I/O is performed by master-slave channels. In symmetric channel I/O, the actual data transferred between communicating tasks is the contents of the data buffer of the symmetric write. In master-slave channel I/O, the actual data transferred is the entire requester call block. Unlike symmetric channel I/O, where each owner operation matches one requester operation, master-slave channel I/O requires two owner operations to match each slave operation. One owner operation reads in the requester call block; the second owner operation writes the call block back to the requester. The returned call block will contain any returned data or error codes. All requester I/O operations to the channel, including opens and closes, are passed to the channel master. I/O utility operations, such as Assign and Release LUNO, and Abort I/O SVCs can optionally be passed to the master as well. The option can be specified when the channel is created.

In symmetric channel communication, owners and requesters are allowed the same set of limited operations. In master-slave communication, slave tasks can issue any I/O command to the channel. IPC supports the full set of resource-specific I/O to master-slave channels. When a master-slave channel is created, a channel resource type is specified. If the resource type is a file type, file I/O operations to the channel are allowed. If the resource type is a device type, device-specific operations to the channel are allowed. If the resource type is channel, only resource-independent I/O to the channel is allowed.

The operation used by a channel master to read a requester call block is the Master Read. The data buffer of the master read operation will contain five words of header information followed by the full requester call block after the master read operation completes. The call block is written back to the channel with a Master Write operation. The data buffer of the master write operation contains the header information (unchanged), the requester call block, and any data being returned to the slave task. The master may only have one master read operation outstanding at any one

time. However, the master may do any number of master write operations while a master read is pending.

Figure 3 shows a requester read operation before the supervisor call is issued. After the requester has issued the request and the master has issued the master read, the master read call block and its data buffer will appear as shown on the left side of Figure 3. Figure 4 shows the call block of the subsequent master write. The channel master has updated the actual character count in the requester call block and is returning data. The left side of Figure 4 shows the requester call block and data buffer after the master write completes.

It was stated in the previous section that a symmetric channel cannot be simultaneously multiplexed and bi-directional. It may not be apparent how a master-slave channel can serve this function either, since a master cannot independently send a message to a slave task. A master can only process and return slave operations. One way to achieve this message exchange is by using a write with reply operation. The resource type of the channel would have to be VDT, since the write with reply operation is only meaningful for terminals. The requester does a write with reply to the channel and the master returns a message to the requester in the reply block. The program in Figure 5 implements a channel master which serves a queue of requests from various slave tasks and returns information to each slave task that issues a request.

5.4 Channel Scope

IPC channels are either global, job-local or task-local. The scope of a channel determines whether a channel and channel owner are replicatable and whether an assign to the channel results in the channel owner being bid automatically. The characteristics of each of these types is as follows:

* Global -- There is only one instance of a global channel at a time and it can be accessed by any task in the system. The channel owner must be the first task to assign a LUNO to a global channel.

* Job-Local -- A job-local channel may be replicated--one instance per job. The channel is accessible to any task in the job. Either the owner task or another task may be the first to assign a LUNO to the channel. If a task other than the owner is the first task to assign a LUNO to the channel, the owner task will be automatically bid.

* Task-Local -- A task-local channel and its owner task are replicated for each task that assigns a LUNO to the channel. Each non-owner task that assigns a LUNO to the channel gets its own instance of the channel. The owner task is automatically bid by the Assign LUNO. A task-local channel owner may not be bid

directly.

If the two tasks that need to communicate can be anywhere in the system, or if the channel owner is not replicatable, the channel will have to be global. If the communicating tasks will always be in the same job or if they can be replicated in many jobs, the channel should be job-local or task-local. A task-local channel is appropriate if the function performed by the channel owner can safely be performed simultaneously by more than one instance of the channel owner. Task local channel owners are replicated for every LUNO assigned to them. If the channel owner is controlling a resource that is available to the entire job, the channel should be job-local so that only one task is accessing the resource at a time.

5.5 Channel Type -- Shared or Non-Shared

Before discussing the difference between shared and non-shared channels, a discussion of channel states is in order. A channel is always in one of three states relative to a task which has assigned a LUNO to it.

* Closed -- The task must open the channel before performing reads or writes to the channel.

* Open -- The task may issue reads or writes to the channel.

* Dormant -- The task must issue a close, and then may reopen the channel. If there are any outstanding operations to the channel at the time the channel becomes dormant, the operations will be returned with an error. Any subsequent operations (except a close) will be returned with an error. The dormant state only applies to symmetric channels.

For all types of channels, open operations performed by non-owners are queued until the channel owner's open has completed. An owner close always puts the channel into the dormant state relative to all requesters. The requesters must close the channel and then may reopen the channel.

Shared and non-shared channels have the following characteristics:

* Shared -- A shared channel may be accessed simultaneously by any number of requesters. A close issued by a requester does not change the state of the channel relative to the owner.

* Non-Shared -- A non-shared channel may only be opened by one non-owner task at a time. This is true regardless of the access privileges requested by the requester open. If a second requester issues an open to a non-shared channel before the first requester closes its LUNO, the second open request will receive an error.

After a requester task has closed its LUNO to the channel, that task or any other requester task may open a LUNO to the channel. A requester close operation causes a symmetric channel owner to become dormant relative to the channel owner, i.e., any outstanding or subsequent operations (except a close) will be returned with an error. The owner should close the channel and reopen it. Another data exchange may now take place.

The unlimited accessibility of shared channels is usually acceptable for master-slave channels because the output of a master write always returns to the slave task that originally performed the request to the channel. The header information provided to the master in the data buffer of the master read allows the master to differentiate between requesting tasks. Furthermore, since master-slave channels are usually used for the purpose of simulating I/O, the standard access control imposed by the operating system on LUNO opens is the most useful means of limiting access to a particular channel.

Shared channels are not sufficient for many symmetric channel applications. An owner of a shared symmetric channel has no way to direct a message to a particular task. The owner operation to the channel will be matched to whatever requester request happens to be next on the queue. Non-shared symmetric channels are intended for applications requiring an extended message exchange between two tasks. Once a requester has successfully opened the channel, the owner is guaranteed that there is only one requester sending and receiving data. The owner will also know when a session with a particular requester has ended because of the error code received on an owner operation after the requester has closed its LUNO to the channel.

An example of a bi-directional message exchange using a non-shared symmetric channel is shown in Figure 6.

The non-shared attribute is much less useful for master-slave channels, but it can be used to limit access to the channel to one slave task at a time. A master-slave channel does not become dormant to the master in the case of a requester close. The master knows that a session has ended when a requester close is master read.

6. Executing and Debugging Tasks That Use IPC Channels

Most Pascal tasks that use channels can be bid from SCI by the Execute Pascal Task (XPT) command or by the SCI primitives, BID, DBID and QBID. Synonyms which specify the input and output of the task can be defined. Task-local channels owners must be handled differently. Task-local channel owners will never be bid by SCI, but will instead be bid by the system as a result of an Assign LUNO to the channel. Therefore, the input and output access names cannot be obtained from synonyms. The access names could be

specified directly in the task code. This may be acceptable in the case of the channel name, since it won't change for the channel owner. Logical names could also be used to specify the input or output. The value of the logical name can be obtained from a Map Name SVC. The input and output access names could also be read from a known file.

Debugging programs that use IPC channels presents some special problems. If there is more than one terminal available for debugging and if the scope of the channel allows both tasks to be bid from SCI, debugging programs that use IPC should not be different than debugging any other program. The owner task is bid in debug mode from one terminal and a non-owner task is bid in debug mode from another terminal. Both tasks can be controlled by the debugger. If the channel is job-local, the owner task should be bid first and both stations should be connected into the same job, through the reconnect capability.

If the channel is task-local, the owner task cannot be bid from SCI. Even for global and job-local channels, only one of the communicating tasks can be bid in debug mode if there is only one terminal. Tasks bid in debug mode from SCI are background tasks, and there can only be one background task per station. If the channel is task or job-local, the owner task can be bid by assigning a LUNO to the channel. If the channel is global, one of the communicating tasks must be bid with a Bid Task SVC.

There are several ways to get a task which was not bid in debug mode into a state where it can be controlled and debugged. First, the task must be put into an unconditionally suspended state (state 6). This can be done by issuing a Suspend Task SVC (SVC 6) from the task which is to be debugged or by using the Modify Program Image (MPI) SCI command to temporarily change an instruction in the task to the assembly language instruction XOP 15,15 (>2FCF) before bidding the task. Once the task is in state 6, the Pascal debug commands can be used to set breakpoints in the task, resume execution of the task, and inspect the stack and memory. Two or more tasks can be controlled from the same station. Each of the debug commands requests a run ID of the particular task to be inspected. The run ID for each task can be obtained by a Show Task Status (STS) command. The Execute Debugger (XD) command and all of the other debug commands that XD enables (Set Simulated Breakpoint, for example) will only work for one task at a station. Some commands that can be used for more than one task at a station are: Show Panel (SP), Show Pascal Stack (SPS), List Pascal Stack (LPS), Assign Breakpoint (AB), Assign Breakpoint - Pascal (ABP), Proceed from Breakpoint (PB), Proceed from Breakpoint - Pascal (PBP), Modify Memory (MM), and Modify Internal Registers (MIR).

One very useful technique to debug tasks that use channel I/O is to set a breakpoint before and after the SVC$ routine. The only parameter to the SVC$ routine is a pointer to the supervisor call block about to be issued. By examining the call block before and after the SVC$ routine, problems related to the channel interface can be found.

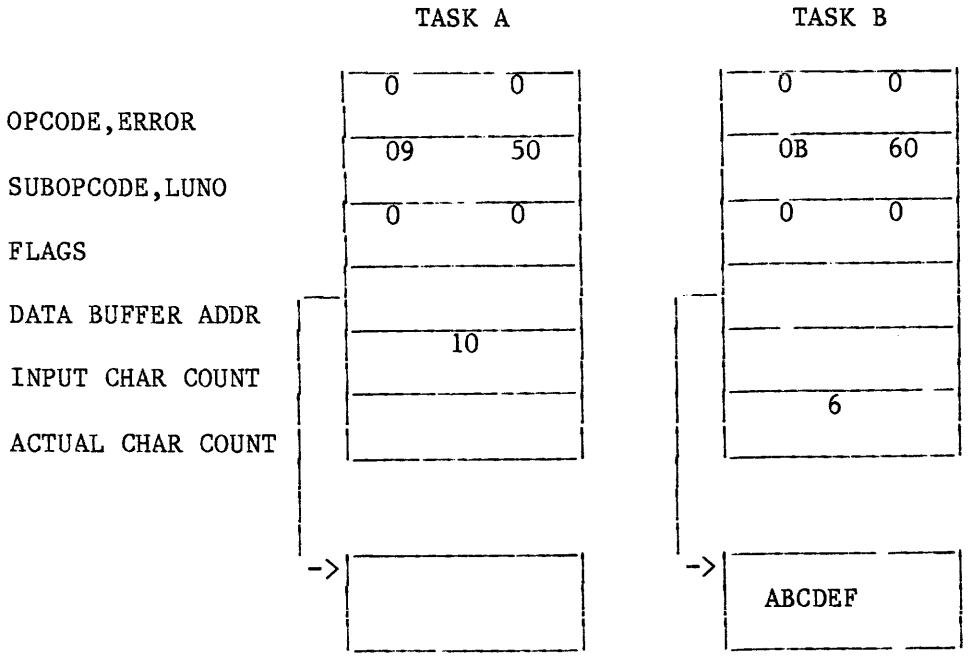
One problem commonly encountered while debugging tasks that use IPC channels is finding a task hanging in state 9 (waiting on I/O). When the task is in this state, Show Pascal Stack and Show Panel will not work. This makes it difficult for the programmer to determine what caused the hang. The problem of tasks hanging in state 9 is usually caused by an error in synchronization between two tasks. One task is expecting a message and the other task is no longer communicating to the channel. One way to debug this situation is to print a copy of the SVC block to a terminal or file before the SVC is issued. If a hang occurs, the last SVC block will then be available.

7. Summary

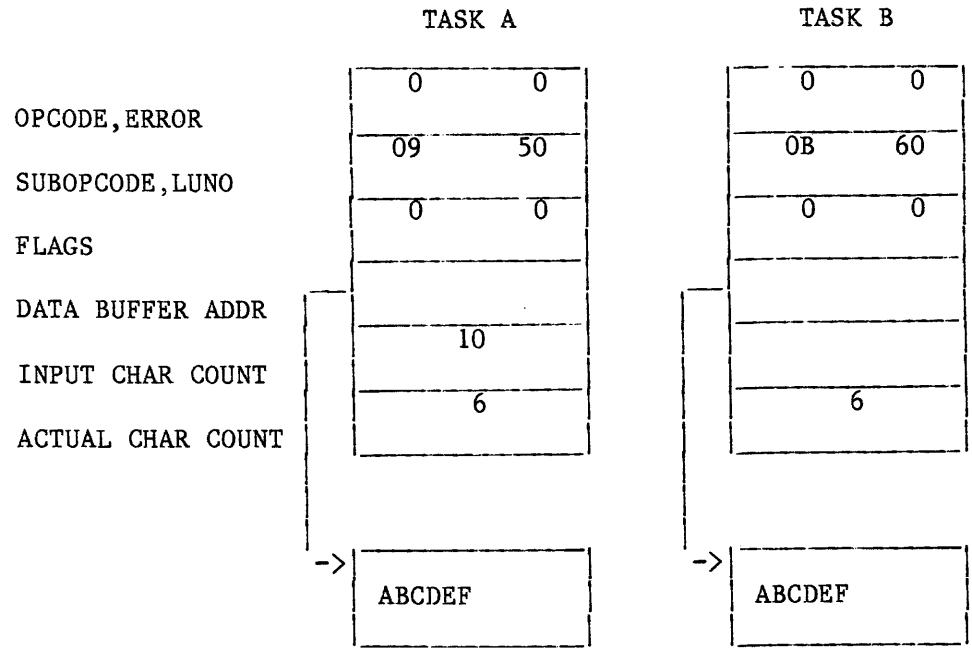
IPC channels can be used in most applications requiring communication between tasks. More detailed information regarding the use of Pascal I/O and IPC channels can be found in the Model 990 Computer DNOS TI Pascal Programmer's Guide and the DNOS Supervisor Call Reference Manual.

```
PROGRAM FILTER;
(* READ DATA FROM STANDARD INPUT, PROCESS DATA, AND WRITE
   TO STANDARD OUTPUT *)
VAR   PHRASE:PACKED ARRAY[1..50] OF CHAR;
BEGIN
   RESET(INPUT);
   WHILE NOT EOF(INPUT) DO
        BEGIN
        READ(INPUT,PHRASE:50);
        READLN(INPUT);
        (* PERFORM INTERMEDIATE PROCESSING HERE *)
        WRITELN(OUTPUT,PHRASE:50)
   END
END.
```

FIGURE 1 -- SAMPLE PASCAL SYMMETRIC CHANNEL TASK PROGRAM

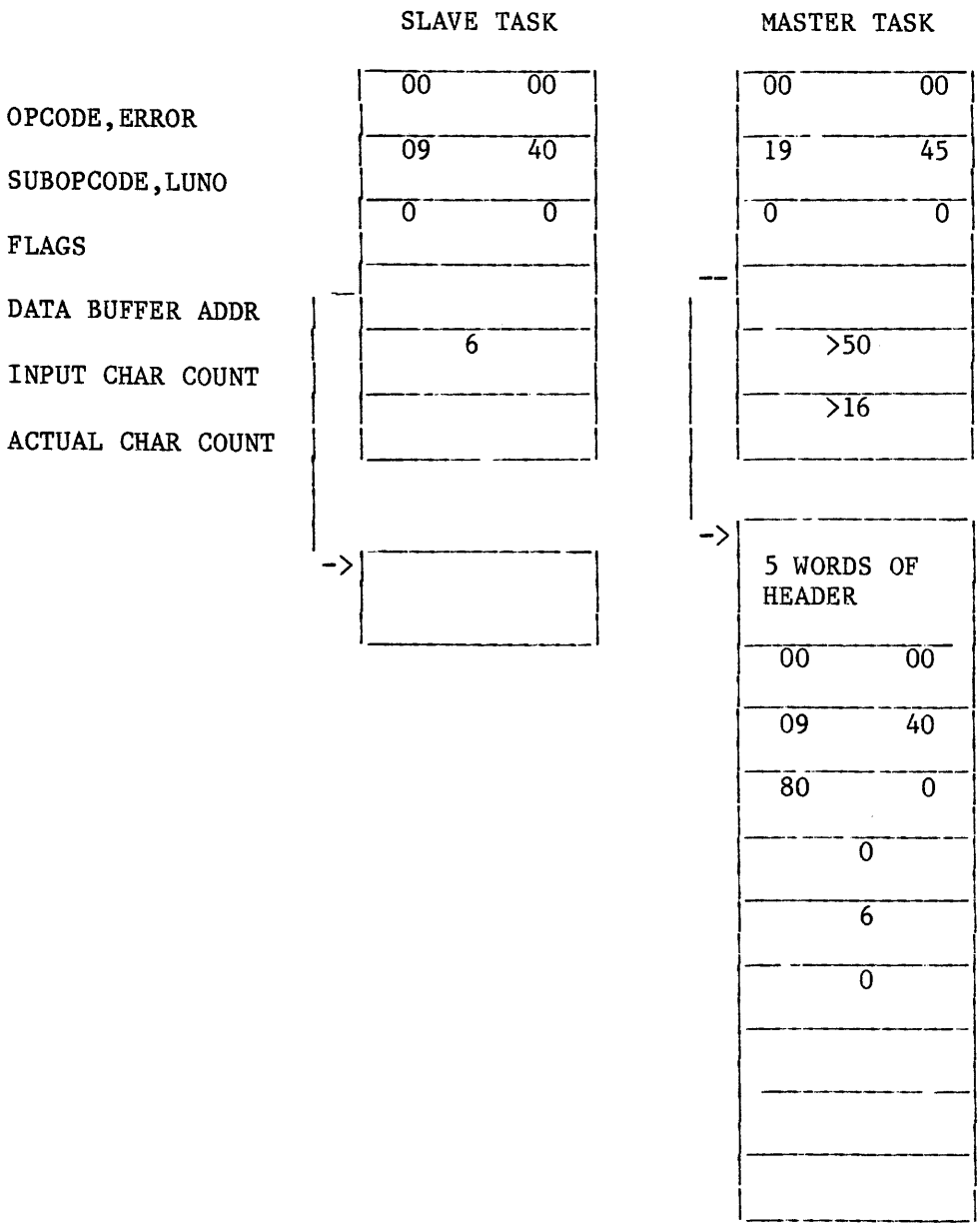


BEFORE SUPERVISER CALL IS ISSUED



AFTER SUPERVISER CALL IS ISSUED

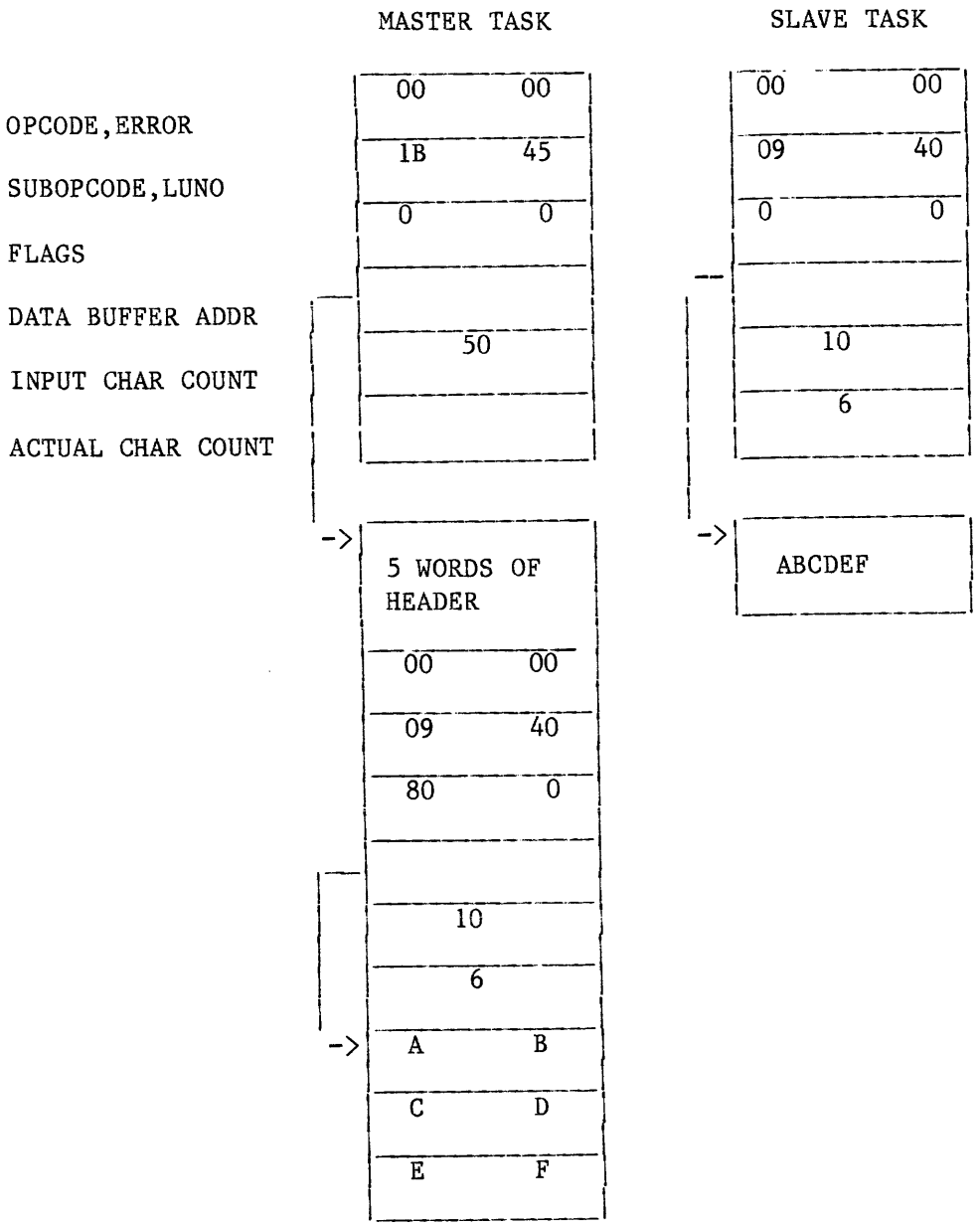
FIGURE 2 -- SYMMETRIC CHANNEL MESSAGE EXCHANGE



SLAVE'S CALL BLOCK BEFORE SUPERVISOR CALL

MASTER'S CALL BLOCK AFTER MASTER READ

FIGURE 3 -- MASTER-SLAVE DATA EXCHANGE



SLAVE'S CALL BLOCK AFTER MASTER WRITE

MASTER'S CALL BLOCK AFTER MASTER WRITE

FIGURE 4 -- MASTER-SLAVE DATA EXCHANGE

```
PROGRAM MASTER;
(* MASTER READ A REQUEST. IF IT IS A WRITE WITH REPLY, READ THE
   INCOMING DATA AND RETURN A REPLY.  THIS PROGRAM ASSUMES THAT
   A SHARED JOB-LOCAL OR GLOBAL MASTER-SLAVE CHANNEL WITH THE
   PATHNAME '.CHAN' HAS BEEN CREATED.  THE CHANNEL MUST HAVE A
   DEFAULT RESOURCE TYPE OF VDT.
 *)
CONST P_OPEN = 0; P_CLOSE = 1;          (* DEFINE OPERATION CODES *)
      P_ALUNO = #91; P_WRITE = #0B;
      MASTER_READ = #19; MASTER_WRITE = #1B;
      INVALID_CALL_ERR = 1;             (* DEFINE ERROR CODES *)
TYPE  BYTE = 0..#FF;
      BUFFER = PACKED ARRAY[1..50] OF CHAR; (* BUFFER DEFINITION*)
      BUFPTR = @BUFFER;
      RPY    = PACKED RECORD            (* REPLY BLOCK DEFINITION *)
               RPYBUF:  BUFPTR;
               RPYICC:  INTEGER;
               RPYOCC:  INTEGER
               END;
      RPYPTR = @RPY;
      ACNM   = PACKED RECORD              (* PATHNAME DEFINITION *)
               CH: PACKED ARRAY [0..5] OF CHAR
               END;
      PNAPTR = @ACNM;
      SVCBLK = PACKED RECORD              (* SVC BLOCK DEFINITION *)
               SOC,ERR:     BYTE;
               OC,LUN:      BYTE;
               SFLAG:       INTEGER;
               DBA:         BUFPTR;
               ICC:         INTEGER;
               OCC:         INTEGER;
               RPY:         RPYPTR;
               RES1:        INTEGER;
               FLG:         INTEGER;
               RES2,RES3:   INTEGER;
               PNA:         PNAPTR;
               REST:        PACKED ARRAY[1..6] OF INTEGER
               END;
      SVCPTR = @SVCBLK;          (* MASTER READ BUFFER DEFINITION *)
      MRB    = PACKED RECORD
               HEADER:      PACKED ARRAY[1..5] OF INTEGER;
               REQUEST:     SVCBLK;
               DATABUF:     PACKED ARRAY[1..70] OF BYTE
               END;
      MRBPTR = @MRB;
```

FIGURE 5 -- MASTER-SLAVE OWNER EXAMPLE PROGRAM

```
VAR   CALLBLK : SVCPTR;
      PATH    : PNAPTR;
      MRBP    : MRBPTR;
      NAME    : PACKED ARRAY[1..5] OF CHAR;
      I       : INTEGER;
      MSGLEN  : INTEGER;
      MSGPTR  : BUFPTR;
      RPYP    : RPYPTR;
PROCEDURE SVC$(P:SVCPTR);EXTERNAL;
PROCEDURE ERROR PROC;
   BEGIN
   (* ERROR PROCESSING *)
   END;
(* SET UP AND ISSUE MASTER WRITE *)
PROCEDURE MWRITE(P:SVCPTR);
   BEGIN
   P@.OC := MASTER_WRITE;
   P@.OCC := 100;
   P@.SFLAG := 0;
   SVC$(P);
   IF P@.ERR <> 0 THEN ERROR_PROC;
   END;
(* BEGIN MAIN PROGRAM *)
BEGIN
      NEW(CALLBLK);       (* GET SVC BLOCK *)
      NEW(PATH);
      NEW(MRBP);
      NAME:='.CHAN';
      PATH@.CH[0]:= '#0A';
      FOR I:=1 TO 5 DO
         PATH@.CH[I]:=NAME[I];
      WITH CALLBLK@ DO     (* BUILD ASSIGN LUNO *)
        BEGIN
          SOC:=0;
          ERR:=0;
          OC:= P_ALUNO;
          FLG:=#0400;    (* AUTOGENERATE LUNO *)
          PNA:=PATH
        END;
      SVC$(CALLBLK);
      IF CALLBLK@.ERR <> 0 THEN  ERROR_PROC;
      CALLBLK@.OC:= P_OPEN;    (* OPEN LUNO *)
      CALLBLK@.SFLAG := 0;
      SVC$(CALLBLK);
      IF CALLBLK@.ERR <> 0 THEN  ERROR_PROC;
```

FIGURE 5 -- CONTINUED

```
    WHILE (TRUE) DO      (* DO FOREVER *)
      BEGIN
      WITH CALLBLK@ DO BEGIN
          OC := MASTER_READ;   (* SET AND EXECUTE MASTER READ *)
          ICC := 100;
          SFLAG := 0;
          DBA := MRBP :: BUFPTR
        END;
      SVC$(CALLBLK);
      IF CALLBLK@.ERR <> 0 THEN ERROR_PROC;
      (* PROCESS REQUESTER CALL BLOCK.  MASTER WRITE OPENS AND
         IMMEDIATELY.  PROCESS WRITE BUFFER OF WRITE WITH REPLY
         AND PROVIDE A REPLY.  IF THE REQUEST IS NOT AN OPEN,
         CLOSE OR WRITE, RETURN IT WITH AN ERROR.  *)
      CASE MRBP@.REQUEST.OC OF
        P_OPEN: MWRITE(CALLBLK);
        P_CLOSE: MWRITE(CALLBLK);
        P_WRITE: BEGIN
           MSGLEN := MRBP@.REQUEST.OCC;
           (* MRB ADDRESSES ARE BYTE OFFSETS FROM BEGINNING OF MRB*)
           MSGPTR::INTEGER:=MRBP@.REQUEST.DBA::INTEGER
                              + MRBP::INTEGER;
    (* *****  REQUESTER MESSAGE CAN BE PROCESSED HERE. MSGPTR
              IS A POINTER TO THE INPUT BUFFER. MSGLEN IS THE
              LENGTH OF THE MESSAGE.  *)
           RPYP::INTEGER:=MRBP@.REQUEST.RPY::INTEGER
                            + MRBP::INTEGER;
    (* *****  RETURN REPLY HERE.  RPYP IS THE POINTER TO THE REPLY
              BLOCK IN THE MRB.  THE REPLY BLOCK CONTAINS A BUFFER
              POINTER (WHICH WILL HAVE TO BE ASSIGNED HERE, A
              MAXIMUM INPUT COUNT AND AN ACTUAL READ COUNT (WHICH
              WILL ALSO HAVE TO ASSIGNED HERE).  FOR CONVENIENCE,
              USE THE SAME BUFFER THAT WAS USED FOR THE INPUT
              MESSAGE TO STORE THE REPLY.  THERE MSGPTR POINTS TO
              THE OUTPUT MESSAGE BUFFER.   *)
           RPYP@.RPYBUF:= MRBP@.REQUEST.DBA;
           RPYP@.RPYOCC:= 10;  (* OR WHATEVER THE LENGTH IS *)
           MWRITE(CALLBLK)
           END
        OTHERWISE BEGIN
           CALLBLK@.ERR := INVALID_CALL;
           MWRITE(CALLBLK)
           END
        END
      END
END.
```

FIGURE 5 -- CONTINUED

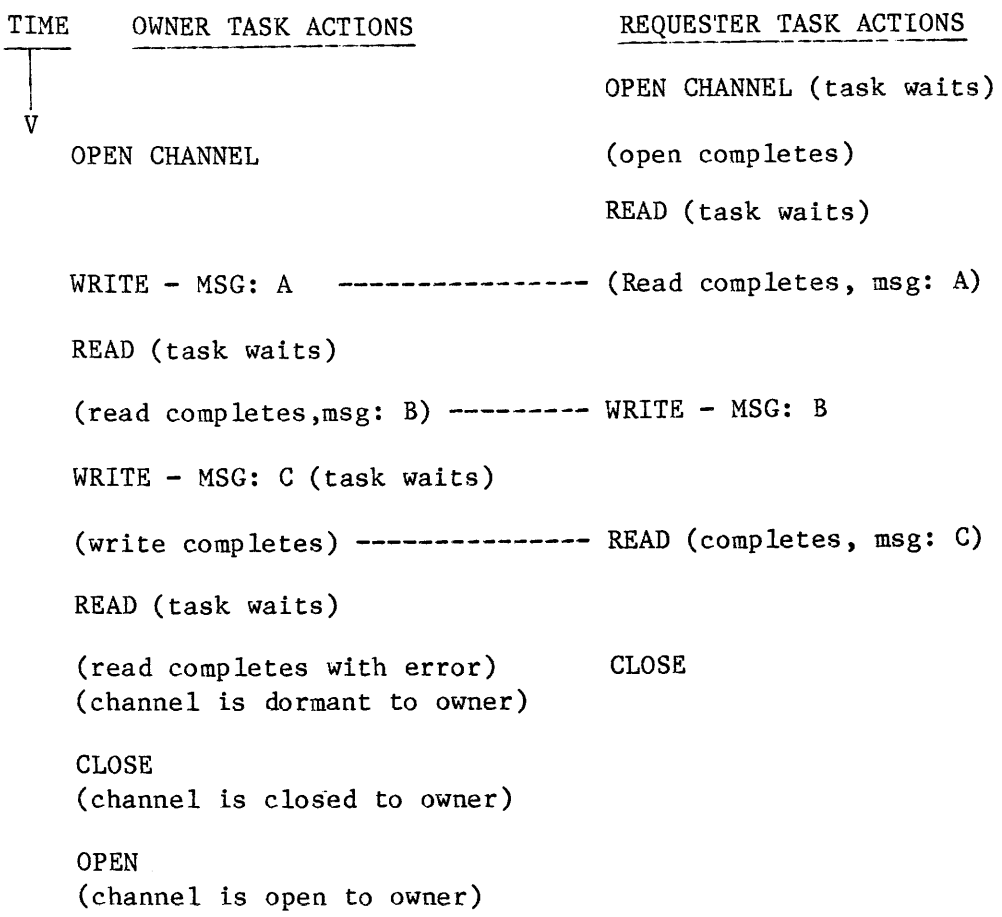


FIGURE 6 -- NON-SHARED SYMMETRIC CHANNEL MESSAGE EXCHANGE

NOTES

TI-MIX 1983

Operating Systems Session

NEW UTILITIES FOR DATA BACKUP

by

Harold Wilensky

Texas Instruments Incororated

Austin, Texas

TWO NEW UTILITIES FOR DATA BACKUP

by

Harold Wilensky

1 Introduction - The Tortoise and the Hare

I am sure that you remember the story of the Tortoise and the Hare. One of the lessons that can be learned from that story is that faster is not necessarily better. On that particular day and in those particular circumstances the faster animal did not win the race. I would like for you to keep that lesson in mind as I talk about some new data backup utilities currently in development. They are faster than the others that we support and for many installations that is the overriding concern. However, because some limitations have been placed on these utilities to achieve greater speed, they will probably not be perfect for everyone.

2 Definitions

Before we get into the details of these utilities I would like to define four terms that will be used quite often and sometimes cause confusion.

1. Backup - A sequential representation of a directory structure, e.g. the output of Backup Directory

2. Copy - A duplicate of a directory structure, e.g. the output of CVD or CD.

3. Disk Compression - Avoidance of secondary allocations and a contiguous packing of data

4. File Compression - Making allocated but unused space available

3 Our goals and how we achieve them

For some time now we have been receiving input from our customers concerning the functionality of the data backup utilities currently supported. After consulting with some of our customers, the Customer Support Line, our marketing personnel, and the TIMIX Systems Committee we developed some goals that should be met by any new data backup utilities:

1. Speed - This was the overwhelming concern. As larger disks became available speed became an overriding concern. It is unreasonable to expect someone to spend a half day to copy a disk. The way to increase speed is to maximize parallelism between CPU activity and I/O and overlap I/O as much as possible. To achieve much greater speed, some trade-offs have to be made with regard to flexibility. These trade-offs exact a certain price. That price is to disallow any selectivity options such as backup by date or "copy this directory but exclude file X." This is the sort of trade-off I had in mind when I mentioned the story of the Tortoise and the Hare.

2. The utility must do its own verification if requested. This is very highly recommended. When copying between disks or backing up to disks, the verification is performed in parallel with data transfer. This is faster than making a separate pass.

3. All media supported by the DS990 and Business System Series products must be supported. This includes non-error free media, error correcting and bad track avoidance disks. Many of the newer disk types use technologies that are very sensitive to disk surface abnormalities. These disks cannot be guaranteed to be error-free; therefore, a physical, track-by-track copy such as DCOPY (which is quite fast) won't work.

4. User friendliness - Keep the user informed about what is happening and give informative error messages. As much as possible the user interface should be through SCI. If a series of copies or backups (analogous to multiple runs of DCOPY or CVD) is needed, all information about the copy or backup should be requested through SCI. This would provide an interface compatable with the rest of the DX10 or DNOS system. The user should be kept informed of the progress while the utility is active and receive informative messages when error conditions are encountered.

5. Single fixed/removable drive systems - The utility must be useable on systems in which there is a single disk drive, one of whose platters is fixed, e.g. the CD1400. This implies that the utility must be able to run without a system disk.

6. System disks created by a copy must be immediately bootable.

7. One should be able to copy between different kinds of disks regardless of sector size, ADU size or physical record length.

8. Disk and file compression should be performed. In the copy process files should be compressed to the end of used space when possible, secondary allocations should be minimized, and program files should be compressed.

How are we meeting these goals? We are developing two new utilities that will compare to the directory utilities(BD, CD, etc.) in much the same way that the Hare compared to the Tortoise: They will be much faster and in many ways more attractive; but keep in mind that they may not be perfect for every environment. These two new utilities will be released with the next releases of DX10 and DNOS. They are Copy Volume(CV) and Backup Directory to Device(BDD). The information I will present about these two utilities is subject to change because they are still under development.

4 CV - Copy Volume

Copy Volume copies an entire disk volume to another disk volume regardless of sector size, physical record size, or any other disk characteristic. If the source disk has more data on it than the destination disk is capable of containing, then CV will copy as much as possible. CV performs optional data verification in parallel with its other activities. We highly recommend that you request verification. When copying to a disk with a different physical record length than the source, you may request that physical record length conversion take place for sequential and/or relative record files. This makes much more efficient use of the destination disk. An optimal physical record length is calculated by CV. CV performs both disk and file compression on most files. The exceptions are any file that is created non-expandable, Key Indexed Files, and Program Files. KIFs are not compressed at all. Program files are always compressed to the extent that unused space at the end of the file is released. "Holes" created in a program file by previously

deleted tasks are recovered in many instances. CV has the ability to perform a series of copies in much the same way as DCOPY and CVD. CV, however, requests all of the information about the copy (or copies) to be performed through SCI. Once the copy is under way the user with a VDT is kept informed about the progress of the copy. CV also has the ability to run without a system disk so that systems with a single fixed/removable drive can copy data volumes. CV does not support any selectivity options such as copy by date.

5 BDD - Backup Directory to Device

Backup Directory to Device backs up any one directory and all of its files and sub-directories to disk or tape. Data backed up with BDD is restorable with the Resore Directory command. BDD performs optional verification. When backing up to a disk(or multiple disks) the verification is performed in parallel with other activity. When backing up to a tape(or multiple tapes) the verification takes place on a second pass of the data. BDD performs the same kind of file compression as CV. BDD has the same user interface as CV. Like CV it keeps you informed of its progress. It also has the ability to run without a system disk. BDD does not support any selectivity options.

6 Technical Overview

In the preceding discussion of CV and BDD much was made of their speed and lack of flexibility. This technical discussion of their implementations should give you some insight concerning those issues. Before going into that discussion a little background is necessary.

Background. As you know the DX10 and DNOS file structure consists of disk volumes which contain one or more directories. Each directory may contain zero or more files and zero or more directories. A directory is really a special case of a relative record file. Each record is one sector in length and contains a File Descriptor Record(FDR), Alias Descriptor Record(ADR) or Key Indexed File Descriptor Record(KDR). For DNOS a Channel Descriptor Record(CDR) is also possible. The records in a directory must be contiguous. An FDR contains information about a particular file or directory. That information includes the starting disk address of the file or directory, its size and disk allocation information. For the purpose of this discussion we will ignore ADRs, KDRs and CDRs since they have little impact on the algorithm. The following diagram should help to explain this

file structure.

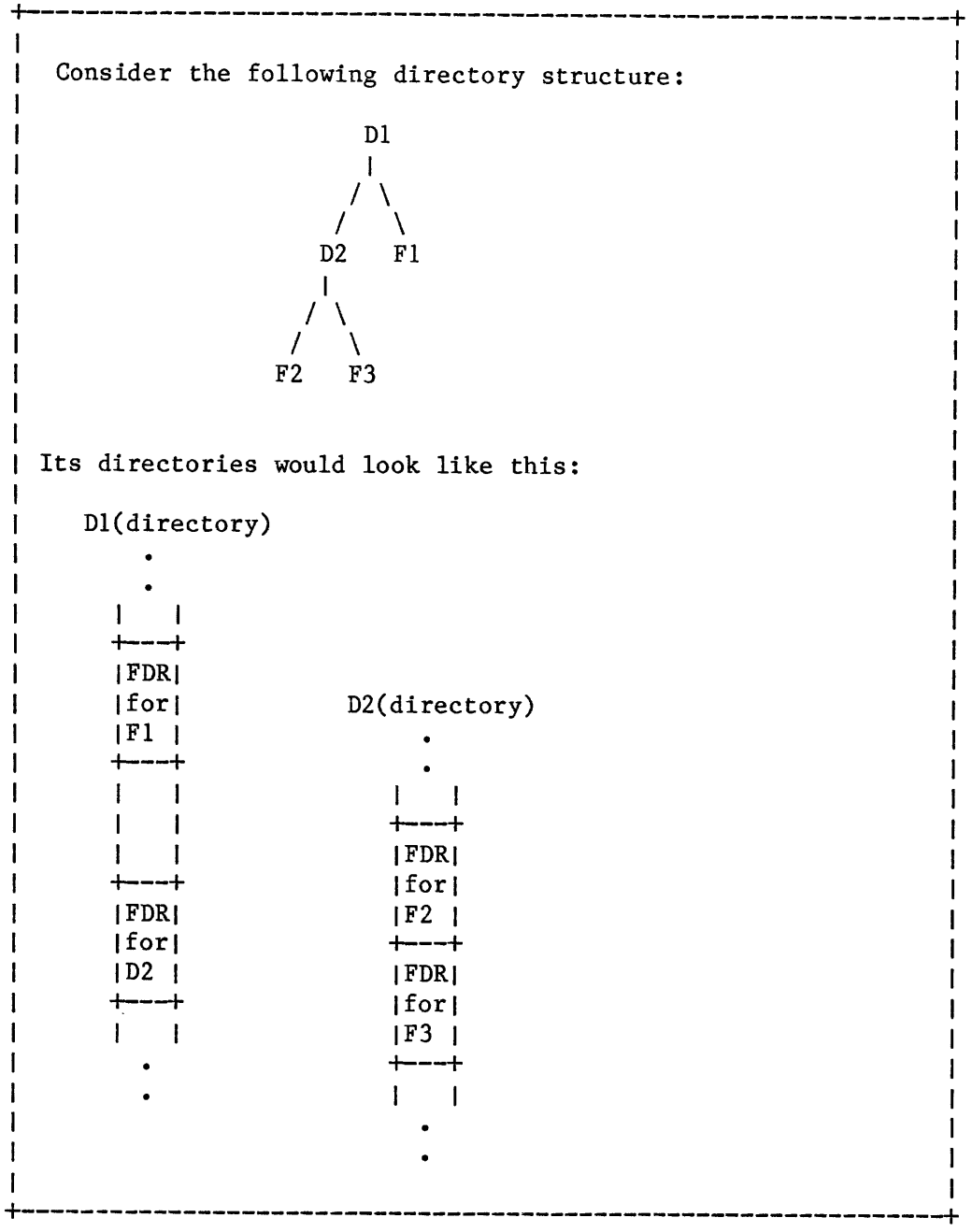


Figure 1 Sample Directory and File Structure

Copy Volume. The implementation of CV is based on the idea that minimizing the number of FDR accesses can buy a great deal of speed. Specifically, the CV algorithm reads and writes several FDRs at one time. This minimizes I/O operations because FDRs within a given directory exist contiguously on the disk. CV keeps the FDRs in an internal buffer and uses the information contained in the FDR to determine the absolute disk address of each allocation of a source file. CV then reads the source data into one of its data buffers. If the entire file will not fit, it reads as much as possible. If more than one file will fit then as many files as will fit are read into the buffer. It is also possible for the last part of one file and the first part of another file to be in the buffer at any give time. The data is then written to the destination disk. When all of the FDRs in the FDR buffer have been processed, the FDR buffer is written to the destination disk in an area previously allocated for the directory that these FDRs belong to. CV uses an internal stack of directory information to keep its place in the directory hierarchy.

CV uses five buffers for I/O: Two input, two output and one verify buffer. Whenever possible, the reading of source data is done in parallel with the writing to the destination disk of previoulsy read data. Verification, which consists of a read from the destination disk into the verify buffer and a comparison of that data with data in the read buffer takes place in parallel with other activity.

All I/O is done using direct disk I/O which bypasses the File Manager. This allows the CV program to calculate its own absolute disk addresses and thereby minimize disk head movement.

It is the heavy parallelism and the ability to copy data from more than one file at a time that imposes the restrictions that have been mentioned.

Backup Directory to Device. BDD creates as output sequential data that can be restored by Restore Directory(RD). In other words BDD creates the same output that Backup Directory(BD) creates.

Like CV, BDD copies many FDRs at a time in order to minimze I/O. Unfortunately, BDD cannot gain as much I/O parallelism as CV because its output is sequential in nature. It does, however, overlap its CPU activity with its I/O quite heavily.

BDD maintains its place in the source directory hierarchy by the way it keeps FDRs in the FDR buffer. BDD's FDR buffer is a one to 23 level stack (because the longest pathname allowed by the system is 48 characters or 23 nodes) of queues. Each queue

entry is an FDR of a given directory. The length of the queues varies depending on the number of files in a given directory.

BDD builds buffers that contain data formatted for the output media. Essentialy this is a sequential representation of the source directory hierarchy and consists of a file's FDR followed by the file. Like CV, data from more than one file may be in the output buffer at any given time.

BDD uses direct disk I/O to read from the source disk. If the destination device is a disk then direct I/O is used to write the output data.

7 Differences between old and new

The following tables describe the differences between the currently supported copy utilities and CV.

Table 1 Speed and Flexibility Comparison of Copy Utilities

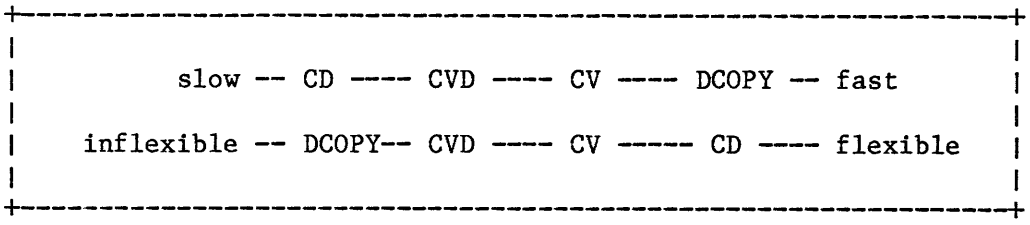


Table 2 Functional Comparison of Copy Utilities

	CD	CVD	CV	DCOPY
Keeps user informed of progress	Yes	No	Yes	No
Copy sub-directory or file	Yes	No	No	No
Copy to tape	No	No	No	Yes
Copy between different disk types	Yes	No	Yes	No
Self-Verification	No	Yes	Yes	Yes
Requires that system disk remain installed	Yes	No	No	No
Tolerates media errors	Yes	Yes	Yes	No
Select particular files to be copied	Yes	No	No	No
Performs disk and file compression	Yes	Yes	Yes	No

The following tables describe the differences between the currently supported backup utilities and BDD.

Table 3 Speed and Flexibility Comparison of Backup Utilities

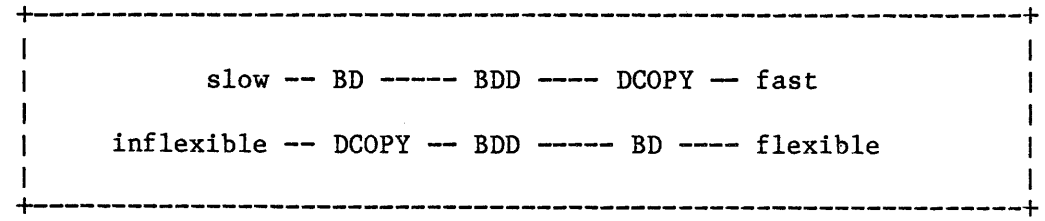


Table 4 Comparison of Backup Utilities

	BD	BDD	DCOPY
Keeps user informed of progress	Yes	Yes	No
Backup sub-directory or file	Yes	Yes	No
Backup to multiple disk or tape volumes	Yes	Yes	Yes
Backup to sequential file	Yes	No	No
Restore to disk with a sector size different from original source	Yes	No	No
Restore to disk with same sector size as original source but different ADU and physical record size	Yes	Yes	No
Self-verification	No	Yes	Yes
Requires that system disk remain installed	Yes	No	No
Tolerates media errors	Yes	Yes	No
Select particular files to be backed up	Yes	No	No

Performs disk and file compression	Yes	Yes	No

8 Examples

NOTE

At this point some examples of CV and BDD will be presented.

NOTES

TI-MIX 1983: OPERATING SYSTEMS PANEL Q&A

The following questions were submitted to TI-MIX 1983 registration forms prior to March 15. TI has addressed these questions in writing below. Additional questions will be fielded during each panel discussion at TI-MIX 1983.

Submitted by Scott H. Jaffe, Sedata Systems, Inc., Seville, OH:

Would you consider adopting a standard DSR to support TI Personal and/or Home Computers as local/remote terminals under DX10?

TI Answer: There is an effort under way to support the TI Professional Computer as a VDT under DX10 via an emulation package written for the Professional Computer. The home computer should be able to function as a KSR into the TPD DSR under DX10, but this has not been verified. There are no activities now under way for Home Computer access to DX10 as a VDT.

Submitted by David Machanick, Consultant, Dallas TX:

I suggest adding memory file capability to DX10. Fix the "offline" printer problem so that DX10 does not have to be re-IPLed to bring the printer back in service.

TI Answer: It is unclear what you mean by "memory file." Both DX10 and DNOS provide a mechanism for sending data between tasks (ITC on DX10 and IPC or ITC on DNOS). On DNOS this mechanism is supported via standard I/O calls through the IPC channel, and in both systems the data is buffered into memory. No provision is made for creating an entire "file" in memory.

We have fixed all known printer restart problems in both DX10 3.5 and DNOS 1.1. If for some reason the printer "hangs" in an unknown hardware state (perhaps caused by static discharge) it can sometimes be cleared by doing direct CRU writes to re-enable it. Do an HO on the output device and turn the printer power off. Wait approximately 30 seconds to allow the capacitors in the printer to discharge. For serial printers write a hexadecimal value of 4600 to the CRU address associated with that device and for parallel printers write a hexadecimal FFFF to the CRU address. Turn the printer power on, put it online and do an RO command. If this does not work the interface will probably require an "I/O reset" which will require an IPL. See a member of the OS panel at TI-MIX to discuss your specific problems.

Submitted by M. D. Korkut, Korkut Engineers, Inc., Metairie, LA:

Is there a systems package that will allow a 990 to address more than 64K at one time?

TI Answer: The DNOS segmentation support allows memory resident segments to be exchanged under user control. Large applications have been developed using this feature. There are no other plans to directly support applications larger than 64K on 990s.

Submitted by Bruce E. Murtha, Shepard Steel Co., Hartford, CT:

Can we look forward to an enhancement which would limit the access of a user to a particular disk drive or a particular directory?

TI Answer: DNOS 1.2 will provide a file access security subsystem. Individual files may be secured and secure disks will not be accessible on non-secure systems.

Submitted by Donald McMunn, Nova Systems, Nashville, TN:

When a task is suspended until the completion of I/O and the device involved (i.e. tape) encounters a device error (door not shut, etc.), how may the suspended task be cancelled?

TI Answer: Device errors should be reported back to the task. If there are specific instances where this is not true, let the Customer Support Line know about it.

Submitted by Rick Nebel, Southwest Baptist University, Bolivar, MO:

Does TI have plans for new TI hardware and software products in the areas of multichannel MUXs (7 or 15 channels), new releases in remote terminals for multidropped and polling networks, and programming and operational aids such as program and/or report generators, etc.?

TI Answer: There will be three new products announced at TI-MIX 1983 in this area. The CI-403 four-channel RS-232C multiplexor and the CI-404 four-channel fiber optic multiplexor join the CI-402 two channel interface to provide chassis-based Business Systems cost-effective FCC-compliant RS-232C ports. No effort is now under way for either 7 or 15 channel multiplexors. The 931 terminal will also be announced at this time, providing a remote terminal capability via its RS-232C interface. This allows connection either via modems, or via third party statistical multiplexor devices. No specific multidrop or polling products are in our current plans.

Submitted by David Teagarden, Moore Business Forms, Inc., Denton, TX:

When is Query, or will it, have substring manipulation capabilities?

TI Answer: Query already supports the ability to search character (CH) data for specified substrings. Possible enhancements would be to extend this capability to the character numeric (CN) and character numeric signed (CS) data types. Alternatively, a capability to define "edit masks" to insert literal information into subfields of character fields could be provided.

We have no plans to provide these enhancements at this time, although sufficient user demand could generate some.

Submitted by Alexander Gelbman, Coulter Electronics, Hialeah, FL:

How long will you continue to expand and enhance DX10?

TI Answer: We will continue to support DX10 as long as customers continue to enthusiastically purchase 990 systems. Since DX10 is a very mature product, it is difficult to make major expansions or enhancements without changing the internal design. The DNOS operating system was implemented to allow us to redesign the internals of our 990 operating system and make major enhancements without perturbing the extensive base of DX10 customers who like DX10 just the way it is. Each system provides some unique attributes and the customer must pick the one which best suits his needs.

Submitted by Robert J. Mateer, Los Angeles City Schools, Los Angeles, CA:

Error messages are often misleading. Many hours spent chasing up wrong trees. What is being done to improve messages?

TI Answer: You may write STRs against documentation as well as software. Document specific error messages which are misleading and submit an STR against that message. If you have suggestions as to how the message can be improved, be sure to include them.

Submitted by Darris Chivers, Automated Services, Salt Lake City, UT:

Will we be able to, at sysgen, specify the size wanted for the synonym table area and keyword area? Could we have "TAGS" or "LABELS" within a proc so we could start at a certain place like on a restart.

TI Answer: DNOS supports a much larger synonym area than DX10. The fixed size synonym space is an integral part of the design of DX10 SCI and would be very difficult to change. A restart capability for batch streams has been considered before, and this too, would be a very extensive change to SCI which we currently do not plan to do.

Submitted by Duane A. Schley, Computer Sharing Corporation, Minneapolis, MN:

Why doesn't the TPD DSR send X-on and X-off to the remote terminal when data is sent too fast for the CPU to get it? (It now cannot get it fast enough.)

TI Answer: The TPD DSR provides KSR support for a wide range of keyboard devices (743/5, 78X, 820). We have not experienced any problems with accepting data from the keyboard of such devices. The TPD DSR also provides ASR support for the 763/5 family. As such, it takes the necessary steps to accept data from the bubble memory. Without specific information about the system configuration and devices, we cannot determine the problem stated here. By sysgening a large enough character queue, we have been able to overcome any problems suppporting

TI devices. We will investigate what would be necessary to add general purpose X-on/X-off support.

Submitted by Vickie Staples, Prodata Computer Marketing, Seattle, WA:

What is the maximum baud rate at any given time for remote terminals on an S372 (e.g., 110 has 9600 max and 112 has 1920 max)? How soon before S300s are a 7 terminal system?

TI Answer: The intent of this question is not quite clear. The S300 using the TI-provided DSRs can currently support three terminals, each of which is running at 9600 baud. The CI-422 four channel option board will be available in late 2Q83 to provide an additional four ports, for a total of seven. The RS-232C ports on the S300 should not be clocked in excess of 9600 baud each. We have tested a seven terminal S300 configuration, and found no problems. We are currently pursuing tests of six terminals (at 9600) with 3780, and with 3270/ICS. Our intent is to identify and fix any problems which we may find in DX10 3.6 User written DSRs may not follow the conventions as TI does, thus this answer does not apply to such environments.

Submitted by L. Allan Butler, Associated Medical Devices, Inc., Denver, CO:

Will AMPL be supported under DNOS? If so, when?

TI Answer: AMPL will not be supported under DNOS.

Submitted by Gordon Alley, Automatic Control Electronics Co., San Antonio, TX:

Are there any DNOS/M systems running? Will networking systems support DNOS/M for booting, remote file access, etc.?

TI Answer: There are no DNOS/M systems running, and DNOS/M will not be supported in the future.

Submitted by Stephen D. Jungersen, Data Concepts, Inc., Morton Grove, IL:

Are there any plans for a 32-bit and/or multiprocessing system in the works?

TI Answer: We are planning an advanced architecture product which will have an addressing capability greater than 16 bits. We have no plans to develop close-coupled multiprocessing but continue to see Local Area Networks as the key to a distributed processing strategy.

Submitted by Santiago Montejo, Hidroestudios, Bogota, Columbia, S.A.:

We have problems estimating CPU time. Is there any utility to help us solve this problem?

TI Answer: The DNOS accounting file records exact CPU times used by tasks during execution. The SMM display on DX10 and the XPD display on DNOS give CPU utilization figures which can be used in stand-alone environments to compute actual CPU times. Come and see an appropriate member of the OS panel at TI-MIX to discuss your specific situation to see if other tools may be available.

Submitted by William J. Callahan, Service Engineering, Inc., Dracut, MA:

Are there any plans to migrate UNIX to TI equipment?

TI Answer: We have no plans to migrate UNIX to 990-based systems. We are aware of the pervasiveness of UNIX in the marketplace and will certainly evaluate the possibility of offering it on future products.

Submitted by Victor M. Louden, General Electric Supply Co., Bridgeport, CN:

When will TI allow two (or more) processors to work together?

TI Answer: See the above answer to Stephen Jungersen's question.

Submitted by Fred W. Powell, Powell and Associates, Staunton, VA:

In the following list of questions and comments for the Operating Systems Question and Answer session, any comment is to be taken to mean "can it be done, and when will it be scheduled for implementation?" All questions and comments pertain to DX10.

(* Questions are new ones; other questions are repeats, with possible rewording from previous discussions. The repeats are included to indicate that they are still applicable, and to get a status report on those which were to be included in future releases.)

1. Positive Comments

   *1. Many of the items discussed at TI-MIX 1982 and in this session in particular, have found their way into future releases. This fact has gone unnoticed and is certainly appreciated, "Keep up the good work." (Maybe we can even do it faster).

   *2. The revised manuals for 3.5 show a great improvement.

2. Regarding KIF files and record locking:

   a. The delete operation should require that the record be locked by the task performing the delete, just as in a write operation. This would prevent two tasks from attempting to delete the same record.

b. Problems arise when two tasks attempt to insert records with the same key at the same time (after already determining that the record was not there). This could be avoided if there existed an op code to "read and insert if not found."

c. How do you protect the currency in one task when another task deletes the record which is pointed to by the currency block of the first task? This should not be left for the user to worry about.

d. How do you define keys which are made up of noncontiguous fields? It is wasteful of storage and just plain annoying to have to create duplicate fields just to construct the necessary keys.

*e. Why is it necessary that the primary key always be non-modifiable when using sequential placement? Many times this is a real nuisance.

TI Answers:

2a. This request is currently in our backlog of design requests. We can currently make no commitment as to when resources will be available to implement the changes required.

2b. The desired result can be accomplished by defining the key as "no duplicates allowed" and then just using INSERT. If this is not a valid solution for your specific situation, see an appropriate member of the OS panel at TI-MIX in order to clarify the exact problem.

2c. This is in our backlog of design requests, but it is a very difficult problem to fix. The currency information would have to be maintained in system space instead of task space. All tasks which maintain their own currency would be adversely affected.

2d. Our original KIF requirements have their roots in support of ANSI standard COBOL. COBOL only allows one data field for a key definition which precludes accessing a key of non-contiguous fields. We currently have no plans to modify KIF to add this feature since it would be a major change to the KIF internal logic.

2e. The COBOL ANSI standard asserts that the primary key on an indexed file will be non-modifiable.

---

3. Regarding task edit keys and text editing:

a. It is often desirable to duplicate the following line as well as the preceeding line (in a proc or in a text edit). Control 1 could be used for this.

b. A Dup function which dups one character at a time would be very useful. This could be used with the repeat key to dup only the desired part of the line. (Control 2).

c. The other function which is needed is to position the cursor at the end

of the current line (last nonblank plus 1). This could be done with Control 4.

*d. Find string, replace string, and delete string should (*optionally) use line numbers (starting and ending), (*) in addition to the number of occurences, to control its range of application.

e. Replace string (or maybe a new command called add string) should have the ability to insert a string at a specified position in the line.

*f. A new function is needed that can "grab" a part of a line, and put it somewhere else.

TI Answers:

3a. This request is currently entered in our design backlog and will be considered as funding and resources permit.

3b. This request is currently entered in our design backlog and will be considered as funding and resources permit.

3c. This request is currently entered in our design backlog and will be considered as funding and resources permit.

3d. This request is currently entered in our design backlog. If TAB settings are set appropriately the user should be able to tab very close to the desired position with very few keystrokes.

3e. This is feature which is usually found in block-oriented editors as opposed to a line oriented editor such as the one we support. It would require a significant amount of change to the existing editor to provide this feature.

3f. See 3e.

---

4. Regarding the SCI:

a. The ability to access two proc libraries was an invaluable enhancement, but two is not enough. We need at least three, and probably four. (Three allows the following breakdown: application, installation, system).

b. The protection of primitives (via .OPTION) was also a valuable enhancement. This needs to be complemented by an option which restricts the direct user execution of procs to the first proc library, thus preventing direct use of those in the second (and third and fourth) libraries. Under this option, proc in the second (and greater) libraries would be accessible only from procs in the first library.

c. In general, there are two attributes for any output file which should be incorporated into all appropriate procs. These are "file status" and "open mode". "File status" has these values: BLANK = don't care or unknown, OLD = file must already exist, and NEW = file must not already exist. Note that REPLACE = NO means NEW, but there is no way

to specify OLD. "Open mode" has these values: OPEN, OPEN REWIND, AND OPEN EXTEND. For consistency, all procs which generate output should allow these options. This includes SVL, XB, LD, SVS, etc., just to name a few.

*d. The proc language should allow "else if" as a control structure.

*e. The proc language should allow logical operators (and, or, not) in expressions on if (and else if) statements.

*f. A task should be able to send a message to a station, similar to the CM command? (*)

*g. When an error occurs in a proc (while testing a new proc), it is difficult to know where you are, what parameter is missing, etc. Better diagnostics are needed.

h. An HBT command to halt the background task would be useful.

*i. Keyword table overflow is a frequent problem. Can the table be made any larger?

*j. A proc can cause an overflow of the TCA parameter table without generating an error. Can that be checked and an error produced?

*k. SVC service errors (that really are not errors, e.g. file not found by delete operation) can be annoying to an end user. Could something be done to optionally turn off the display of such error messages.

*l. It is often the case that you wish to return to the proc library(s) and menu from which you came (via .USE and .OPTION), but you have no way of knowing that state in all cases. The best solution would be for SCI to automatically stack (and allow the user to unstack) these states. Alternately, (but not as desirable), would be a mechanism to set a synonym with value equal to the current proc and menu state.

TI Answers:

4a. DX10 3.6 and DNOS 1.2 will both allow a maximum of 5 PROC libraries.

4b. With DNOS File Security it will be possible to secure certain procedures to specific access groups. We currently do not have any plans to add the specific feature you request to DX10 or DNOS SCI since it would require major internal changes.

4c. The (EXTEND,ADD,REPLACE) options are a part of SVL. We will evaluate other commands of this type as resources permit.

4d. We agree but currently do not have resources to implement this feature. This is in our design backlog.

4e. We agree but currently do not have resources to implement this feature. This is in our design backlog.

4f. In DNOS it is possible to send a message to the "operator" from a task but we have no plans to implement a general message capability from tasks.

4g. In DNOS SCI the line number in which an error occurs is given as part of the error message.

4h. This can be accomplished with an STS command and the HT command.

4i. In DX10 the synonym table cannot easily be made any larger and we currently have no plans to make changes in this area. In DNOS the table is 12288 bytes long compared to 864 for DX10.

4j. We need to know the specific instance of overflow you are referring to. Either see a member of the OS panel at TI-MIX or submit an STR exactly describing the situation.

4k. This is not easily done on DX10 since each processor task for the separate commands controls the displaying of the errors. We will enter this as a design request and investigate a more general solution to the problem.

4l. This would be another major change to SCI and we currently have no plans to implement this feature. On DNOS 1.2 the current set of PROC libraries is stored in a well known synonym ($$CL) and thus the user can save and restore the procedure libraries using the .SYN primitive.

---

5. Directory and Volume Utilities

a. Either LD or MD (short form) should include the record length as part of the listing.

b. MD needs the following changes/additions: "Top Level Only" should be "Number of Levels"; "Directory Nodes Only" should be "List Types" with values S,R,K,D,P,I,A. These are obviously file types, i.e. D = Directory nodes only, and A = All.

c. The control files for CD, BD, etc. would be much more useful if the names in an INCLUDE or EXCLUDE statement could be a pattern (as well as a specific name). Thus EXCLUDE S$_____ would mean that all names beginning with S$ were to be excluded, and INCLUDE___TEST would mean that all names ending with TEST were to be included. (*) Range values would also be useful, i.e. INCLUDE A_____:M_____ means all files with names that begin with A through M. (How do you copy half of a directory to another directory? Also, it would be useful if DD and MD allowed the use of control files.

*d. Why does CD not convert physical record lengths for KIF files?

*e. Is it intended that VC terminates on the first >0011 error rather than indicating the error and continuing with the next file?

*f. It is confusing as to how MVI uses a control file, and how the synonym "$$DSC$" gets set. Also, it would be useful if another disk could be specified within MVI without having to Quit and then invoke it again.

*g. Scan Disk (SD) will allow its output to be sent to a device, (only a file), and it opens the file with exclusive access making it impossible to view while the task is executing. Also, what will auto-correct do (and not do)?

TI Answers:

5A. The MD command of a given file can be used to obtain the logical record length of that file. Our experience has shown that if we change the output of a particular utility we break utilities written to accept that file as input and are requested to put it back the way it was. For instance, the May issue of MIX-TIPS will contain a number of instances requiring MD output to remain constant.

5b. This is in our current design request backlog and will be considered as time and resources permit.

5c. We currently have no plans to implement this feature in our backup utilities. The utilities were not designed with these features in mind and adding these features would force us to essentially rewrite major portions of the logic.

5d. The internal structures of KIF files are based on physical record length. The directory utilities do not understand the internal structures of KIF files and make no attempt to do physical record level conversion. The CKR utility may be used to copy one KIF file into another pre-created KIF file with a different physical record length.

5e. VC should continue after errors rather than stopping on the first error. Have you submitted an STR on this problem? We will investigate it.

5f. MVI uses a control file as if it is reading from a TTY device interactively. Each input must be on a separate line of the file. We do not understand the context of your question concerning the synonym $$DSC$. Please see an appropriate member of the OS panel and ask your question to them. We will enter your request to change the disk name dynamically into the design data base and consider it for future releases.

5g. SD has been updated in DX10 3.6 and DNOS 1.2. It will run in foreground and has the option of displaying its search progression as it runs. The SD utility is written in FORTRAN which opens input and output files with exclusive use which precludes looking at a file from another station. The new code will allow output to a device on both DX10 and DNOS.

---

6. Tasks and Lunos

a. In CPI, it is desirable to replace a task by name without having to know its installed ID.

b. If procedures could be made station local (optionally), then a family of cooperating tasks could share data through a dirty procedure. This is impossible now since all procedures are global.

c. XT needs the option DISP = NO, as in AL and AGL.

*d. The station number should be a parameter (with default "me") when assigning a station local luno. This is needed when a task is initiated at one station but is associated with another. The release operation should be handled similarly, including RAL.

*e. How can a task spawn another task and continue execution, and at some point, suspend itself pending completion of the spawned task? What is needed is a suspend SVC which is conditional on the completion of a specified task, i.e. the parent task should be able to place itself in a state >17 pending completion of the specified task.

*f. Why can't a task or station local luno be associated with phantom station, i.e. a station which was not specified in the sysgen? Giving an error for the sake of giving an error is not to be considered just cause.

*g. It would be very useful to have a new "kill task" command (or option on the current one), that "kills" all tasks associated with the specified station.

TI Answers:

6a. We agree this is a desirable feature and will attempt to implement it if funding and resources are available. This is already in our design request backlog.

6b. This is in our design request backlog. It is possible to communicate with all tasks associated with a particular station if the station ID is stored as part of the message in the dirty procedure.

6c. This problem will be fixed on DNOS 1.2 since it is merely a change to the proc but will not be fixed in DX10 3.6 since development is source frozen already.

6d. If a task is associated with a given station (even if it is bid from a different one) and issues a station local Assign Luno SVC, the luno will be associated with the station the task is currently associated with. That is, if the "station local" bits are set in the SVC call block the feature should already exist as you have requested. If this is not true see an appropriate member of the OS panel at TI-MIX and let's discuss the exact situation.

6e. This feature as described would require some major changes in the DX10 kernel, and we currently do not anticipate making this change. The feature you desire can be accomplished by recording the ID of the spawned task after doing a normal bid task then periodically doing Poll Task Status calls on that task until it is determined that the spawned task has terminated. Incidentally, the DNOS semaphore mechanism provides this feature if you are interested.

6f. Most users want to know if a LUNO has been assigned to a non-existent device. Errors are given because it is assumed if the user is assigning a LUNO to a device he expects the device to be there not "just to give an error for the sake of giving an error." The system has no way of determining whether a user expected that device to actually exist or not. The only devices known to the system are those included during a SYSGEN.

6g. This is not a feature we have a great number of requests for but will consider it. The DNOS job structure provides a Kill Job command which kills all tasks under that job.

---

7. Print Utilities

*a. The "page eject" if PF and CC should be a parameter for both before and after printing, so that each user can tailor it to suit his needs.

b. PF should also have parameter to allow a halt between pages, (*) as well as a restart at specified line or page capability.

TI Answers:

7a. The current method we have chosen for page ejects protects the unsophisticated user from printing on the last page of a previously printed file as well as handling embedded ANSI carriage control correctly. We will consider this feature as time and resources permit, and it is currently in our design backlog.

7b. This capability exists in the DNOS spooler but we currently have no plans to incorporate this feature into DX10 PF.

---

8. Link Editor

a. The link edit control file should allow a copy command to include frequently used command sequences, or for example, the INCLUDEs for a shared procedure. It should also allow a substitute command to substitute one external reference name for another.

TI Answer: A COPY command is an idea we have considered in the past, but have not had the resources to do as of yet. We will continue to consider this capability. We would like to know more about the SUBSTITUTE command since we have not had many requests for such a feature.

---

*9. Communications

*a. The TPD DSR has capabilities that are not directly user accessible, e.g. parity options, the end-of-record character, etc. These options should be settable in the system, and modifiable by MHPC or equivalent. Additionally, some option should be provided such that the DSR does not attempt to interpret any characters received, such as a DC3 (>13). This could be done using a special terminal type, or by specifying no terminal type.

TI Answer: It appears that some people are trying to utilize the TPD DSR (teleprinter DSR) for third-party devices and "home-brew" protocols. While this is possible in many cases, the list of exceptions would quickly get larger than the rules if we tried to add every feature necessary for a completely general purpose DSR. We are making some changes for DX10 3.6 in the area of 8-bit data support that may address Fred's desire for not interpreting certain characters. We may be able to better document some of the facilities that are present to make them "user accessible", and will investigate doing this.

---

*10. Sysgen

*a. Prior to release 3.5, sysgen showed the current values of device parameters when in change mode for that device, and used those values as defaults. That was a useful feature which seems to have disappeared.

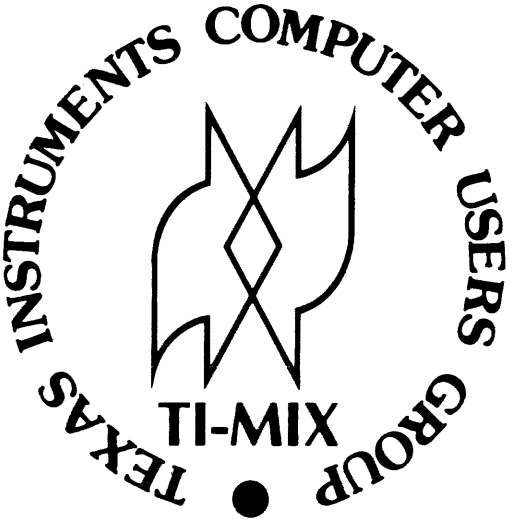
TI Answer: The XGEN utility was totally rewritten in Pascal between 3.4 and 3.5. Even though the default feature disappeared, the user was given the ability to show, print, and text edit (with care) the configuration file which we feel offsets the lack of displaying defaults in "change" mode. We will investigate what it would take to add that feature back.

---

*11. Miscellaneous

*a. We always seem to have a significant increase in system crashes (>20, >27, >A0) following a new OS release. This slowly returns to normal after several patch updates. Why does this happen?

TI Answers: We spend a great deal of time testing our newly released software, but the number of hardware configurations, software packages, and timing problems to be handled by an operating system are enormous. We cannot create every possible configuration and test every software package against it. We do test thoroughly against representative configurations, but many times problems are not found until a particular situation is created by a customer. As these problems are reported, they are patched in a patch release, and the system becomes more stable over time.


TEXAS INSTRUMENTS COMPUTER USERS GROUP
TI-MIX