Publication Number B29-391

OS/32-MT
PROGRAM LOGIC MANUAL

The information contained in this manual is subject to design change and product improvement.

Subsidiary of PERKIN-ELMER
Oceanport, New Jersey 07757, U.S.A.

January 1975

OS/32 MT
PROGRAM LOGIC MANUAL

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

This Program Logic Manual (PLM) is a guide to the internal structure of the operating system OS/32 MT. It is intended for use by people involved in maintaining and modifying the system. It normally should be used with program listings.

This manual deals exclusively (and specifically) with OS/32 MT R00. Hence, specific methods of implementation of various functions are not to be construed to be the method of implementation to be used in all future releases of OS/32 MT.

Use of this manual requires the reader to be knowledgeable of the features, functions and conventions of OS/32 MT from the user's point of view as documented in Program Reference Manual, Publication Number 29-390, and Program Configuration Manual, Publication Number 29-389. Documentation for I/O drivers is in OS/32 Series General Purpose Driver Manual, Publication Number 29-384. The user should also be familiar with the 32-Bit Series architecture and its features.

OS/32 MT is an operating system that provides program management in a multi-tasking environment. System control via console operator, interrupt handling, I/O servicing, and inter-task communication/control are built-in functions of OS/32 MT. Disc file management features are also provided when the system is equipped with a disc, and as such, OS/32 MT is oriented towards a disc operating environment. A file directory and allocation bit map are maintained on each disc volume to allow for disc portability.

OS/32 MT is compatible on the program level with the serial task operating system OS/32 ST in most respects not related to multi-programming, and OS/32 ST can serve as a development tool and "test bed" for many OS/32 MT-oriented programs. Moreover, many system routines, I/O drivers in particular, are identical in both systems; many other routines are quite similar in structure.

Chapter 2 of this manual describes the general structure of OS/32 MT. Chapter 3 discusses the conventions followed by the system in terms of interfacing between modules, naming of fields and flags and structure of modules. Chapters 4, 5 and 6 contain a detailed technical description of the major module groupings in OS/32 MT. These chapters are designed to provide a technical overview of the system. Chapter 8 contains examples of system flow of control. Chapter 9 contains an explanation of Executive tasks and discusses user added extensions to OS/32 MT. Chapter 10 contains a list of CRASH and JOURNAL Codes and their meanings. Chapter 11 contains the format of system control blocks.

CHAPTER 2

SYSTEM STRUCTURE

This chapter describes, in a broad fashion, the general structure of OS/32 MT from a technical viewpoint. As illustrated in figure 2-1, OS/32 MT is composed of four major module groupings. These are Executive, Command Processor, File Manager and the OS/32 Series General Purpose drivers. This chapter discusses each of these module groupings and how they interact. I/O support is provided by the OS/32 Series General Purpose drivers together with major portions of the Executive, so the drivers are discussed in the context of this I/O subsystem.

2.1 EXECUTIVE

The executive contains logic for processing Supervisor Calls (SVCs) and other internal interrupts, a memory manager, task manager, Event Service handler, a crash handler, the loader, clock management routines, intertask coordination and control routines, general utility function routines and a system journal handler. Portions of the Event Service handler and SVC processor support the I/O subsystem and are discussed in section 2.2.

2.1.1 Task Management

All functions in OS/32 ST are performed on behalf of a task. A task is controlled through a Task Control Block (TCB). In OS/32 MT, the user, at sysgen time may decide upon the number of tasks his system is to contain. An OS/32 MT R00 system must contain at least 2 tasks (the System Task and a background task). The user may choose to have up to 253 other tasks in his system.

Each task in OS/32 MT has one of the following states associated with it; current, ready or wait. A task is in wait state when it requires the occurrence of an external event to continue its execution; for instance, the completion of an I/O operation. A task is ready when all external events have occurred that are necessary to let the task proceed. A task is said to be the "current" task when it is the highest priority ready task in the system.

OS/32 MT R00 requires that the System Task have a priority higher than the priority of any other task in the system.

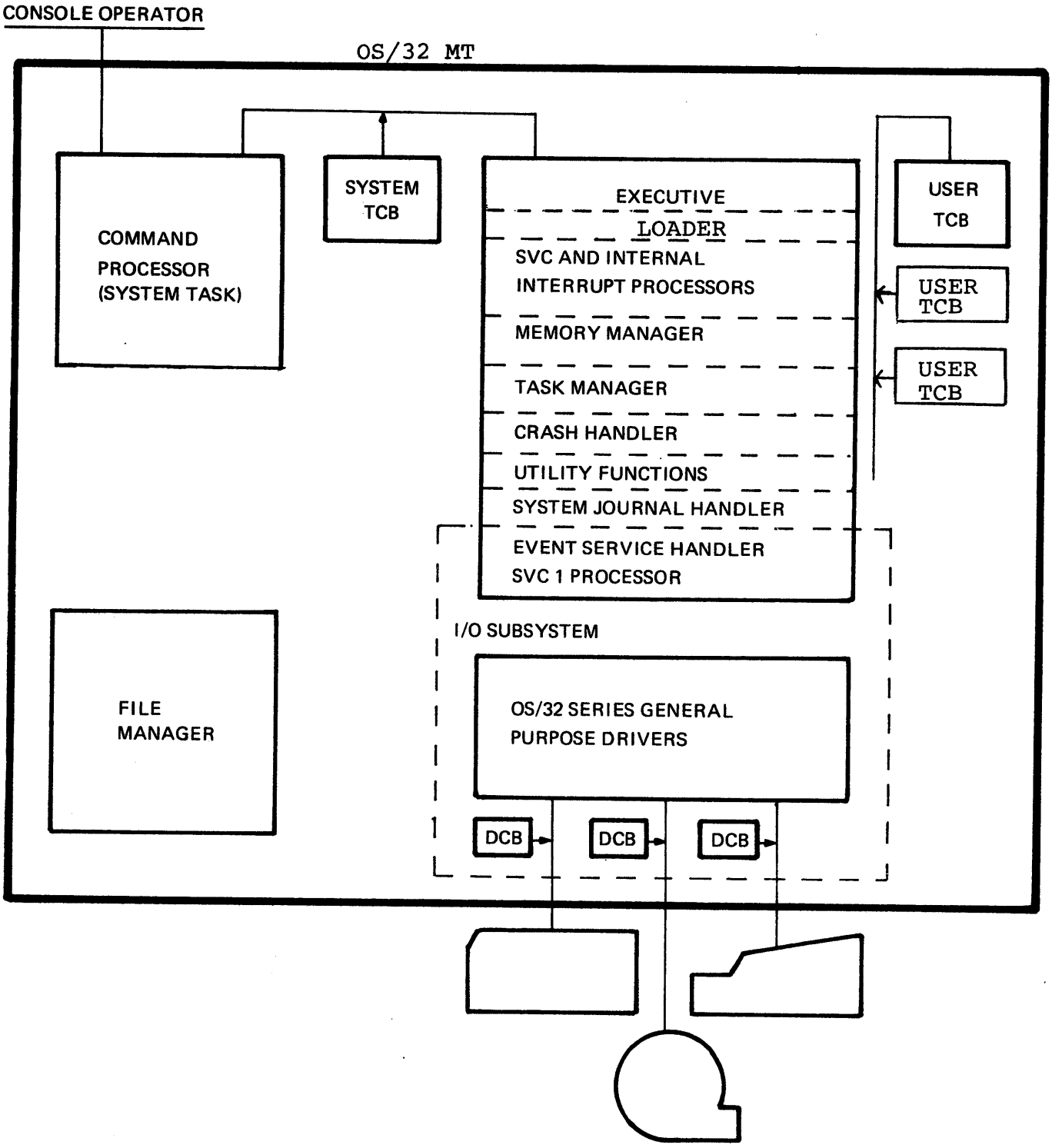


Figure 2-1. OS/32MT FUNCTIONAL BLOCK DIAGRAM

2.1.2 Executive Services

SVC handlers

All SVC interrupts cause entry to the SVC First Level Interrupt Handler. This module performs common preprocessing such as making a Journal entry for the particular SVC, checking the parameter block address for validity, saving the user registers, if necessary, and branching to the executor for the particular SVC. Some SVCs, such as SVC 2, have second level interrupt handlers to perform similar preprocessing for the different options.

Memory Manager

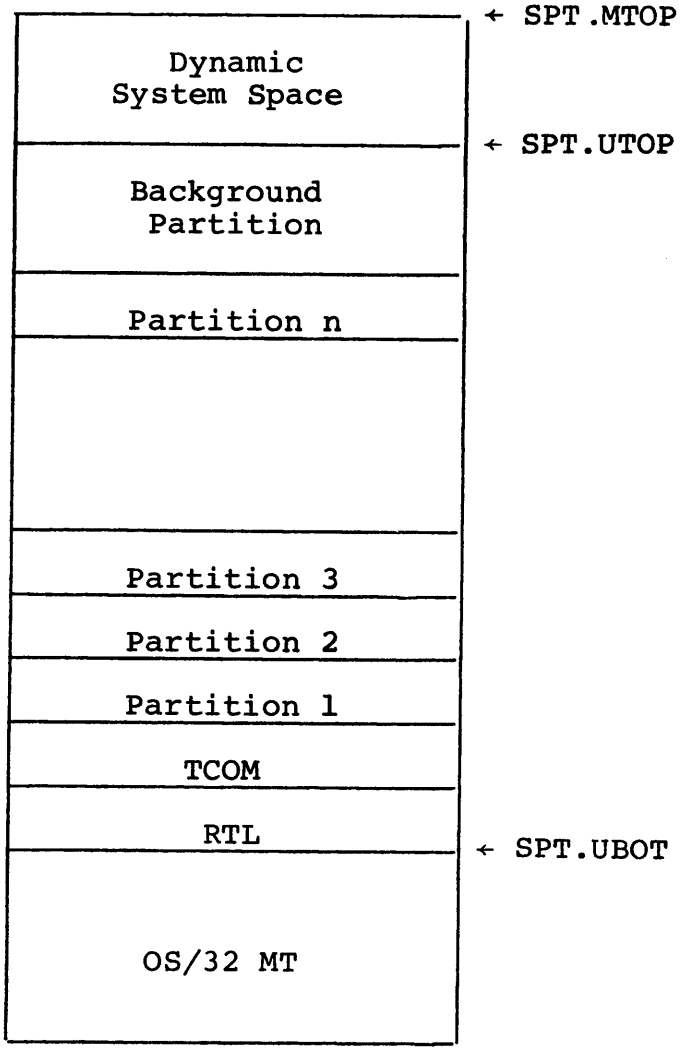
In the OS/32 MT R00 system two classes of memory are maintained; user space and system space.

User Space

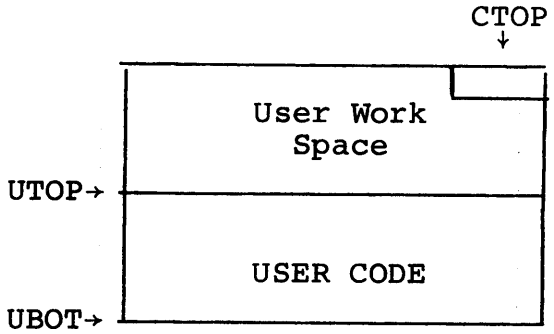
User space is divided into 4 classes of partitions. They are; foreground, background, task common, and resident library.

OS/32 MT R00 supports one task common partition and one resident library partition. Each of these partitions is limited in size to 64KB. The size of task common is established via the operator at any time the system is quiescent, and may be changed at any time the system is quiescent. The size of the resident library is determined when it is loaded. The resident library is loaded via operator command at any time the system is quiescent. A new resident library may be loaded, or an existing resident library deleted, via operator command, at any time the system is quiescent.

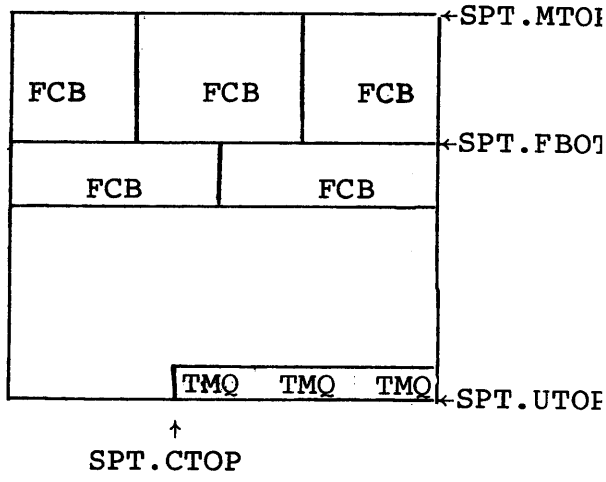
Foreground and background partitions are areas of memory associated with tasks. Each TCB in OS/32 MT R00 has an area of memory associated with it. The size of this area is selected at sysgen time, and may be changed via operator command at any time the system is quiescent. Any partition may be given a size of 0, thereby effectively deleting it from the system. The number of partitions created at sysgen time may never be increased. In OS/32 MT R00 there must always be a background partition, and the user may choose to create up to 253 foreground partitions. Tasks running in the background partition are limited in that they may not communicate or interfere with the foreground system. The tasks running in the foreground partitions have available to them the full range of OS/32 MT functions.



Memory Map of Entire System



Memory Map of User Partition



Memory Map of Dynamic System Space

In dealing with memory arrangement, the background task plays a special role. Whenever the operator reconfigures other partitions, the extra memory needed, or the memory relinquished by a partition is given to or taken from the background partition.

System Space

System space is divided into two classes; that which holds OS/32 MT and its static data structures, and the area available for dynamic data structures. Dynamic data structures obtain their space from an area at the top of memory. This area is bounded by UTOP at its bottom and MTOP at its top. This dynamic area is divided into two pools, one of which is bounded by UTOP and CTOP and the other of which is bounded by FBOT and MTOP. Large blocks of dynamic space are taken from the FBOT-MTOP area, while small blocks are taken from the UTOP-CTOP area. This is done to decrease the effects of fragmentation in dynamic space. Memory in the FBOT-MTOP area is allocated from MTOP down, while memory in the UTOP-CTOP area is allocated from UTOP up. Each task has associated with it a maximum amount of system space it can obtain. This limit is set at Task Establishment time. If a task exceeds this limit, all further requests that require system space are rejected, until the task has relinquished a portion of the system space it has previously obtained.

Task Space

A user task is provided SVC calls with which he may manipulate his partition. Each partition has 3 pointers associated with it; UBOT, UTOP and CTOP. GET/RELEASE storage calls manipulate UTOP. UBOT and CTOP are fixed and may only be changed by reconfiguring partitions. EXPAND/CONTRACT allocation calls are ignored in OS/32 MT R00.

General Utility Functions

This package is primarily used for processing the miscellaneous SVC 2 routines, but it is also entered directly by the other system elements from time to time. Some of the features provided are:

UNPACK routine
EXECUTIVE MESSAGE routine

Any subroutine called by more than one system module is normally placed in this package.

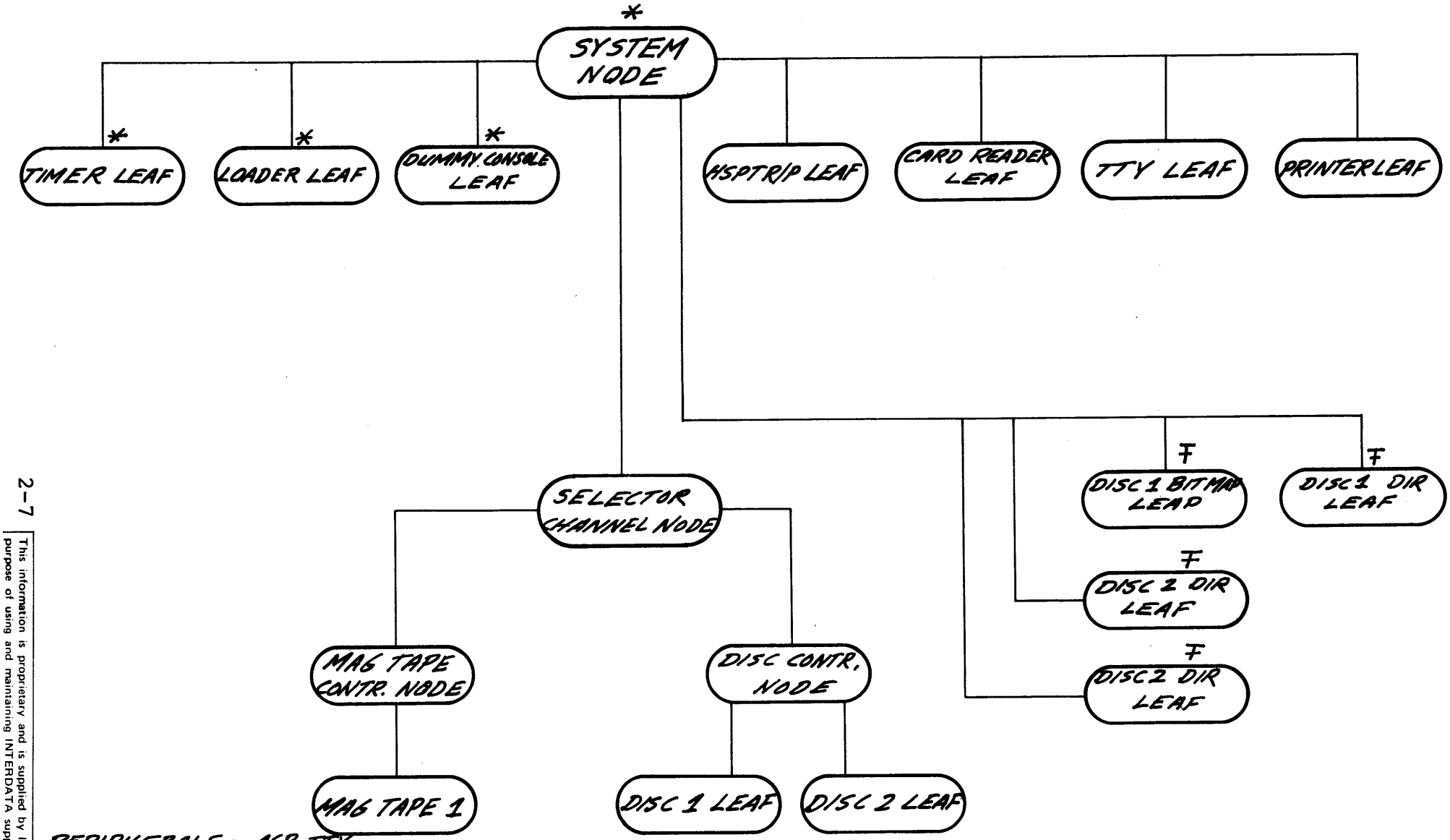
2.1.3 Internal Interrupt Handlers

This package handles interrupts due to machine malfunction, illegal instruction, and arithmetic fault. Illegal instructions, arithmetic faults or parity errors encountered within the executive cause an immediate entry to the crash handler. Arithmetic faults encountered while a task is running cause a message to be logged to the system console and the task to be PAUSEd or continued as specified by the user. Illegal instructions encountered while the task is running may enter the SYSGENable floating-point trap package to be executed. Otherwise, the task is PAUSEd via an entry into the task manager, after logging a message to the system console. Memory Access Faults cause the offending task to be PAUSE'd and a message logged to the system console. Memory parity machine malfunctions within a task cause the task to be aborted after logging a message to the system console. Power failure causes all registers to be saved, whereupon the system waits for power restoration. When power is restored, the following actions take place for all tasks:

All direct-access data transfers are retried.
All other I/O operations are terminated.
A message is logged to the system console, requesting the operator to reset peripherals if necessary. The system waits for the operator to enter 'GO'.
Tasks are PAUSEd or take Traps (depending on the value of the current TSW)

2.1.4 Event Service Handler

Coordination of system resources (mainly I/O devices) is controlled through the Event Coordination Table (EVT). The Event Service Handler contains routines to manage the EVT. The EVT is a tree structure consisting of nodes (entries with descendents) and leaves (entries without descendents). (Figure 2-3 illustrates an example of an EVT structure). Each path in the tree corresponds to a group of system resources that must be coordinated as one resource. For example, the system node, selector channel node and mag tape leaf path corresponds to all the resources that must be coordinated to control access to the magnetic tape. Coordination is implemented by providing routines to connect to, queue to, disconnect from, and release entries in the EVT. Only one task may be connected to an EVT entry at a time. The EVT is generated at SYSGEN time by the OS/32 MT Configuration Utility Program. A task is not connected to any required entry until it can be connected to all required entries, thus preventing deadlock conditions.



PERIPHERALS : ASR TTY
HIGH SPEED PTRP
LINE PRINTER
CARD READER
SELECTOR CHANNEL 1
MAG TAPE
DISC CONTROLLER
DISC 1
DISC 2

* ALWAYS GENERATED BY CUP F A DIRECTORY AND BIT RAP LEAF IS GENERATED FOR EACH DISC IN THE SYSTEM

2.1.5 Clock/Timing Facilities

OS/32 MT makes use of both a line frequency clock and a precision interval timer to provide user tasks with a flexible set of timer management/maintenance services. The following services are provided; time of day clock, day and year calendar, interval and time of day wait, interval and time of day trap, driver timeout. The clocks are coordinated through a leaf (TIMELV) and operate as Event Service subroutines of the system task.

The timer management routines require the use of dynamically allocated blocks of system space called Timer Queue Entries (TMQs) and obtain/release these by calls to GETSYS and RELESYS. The space used is deducted from the user's system space allocation, and if he cannot receive space (because he has exceeded his maximum allocation, or because none is available), a time call will be rejected. This does not affect the status of existing time items.

In addition, by sysgen option the operating system will display mmddhhmm on the processor display panel.

2.1.6 Loader

The OS/32 MT resident loader loads tasks, overlays and library segments. The input to the resident loader must be created by the OS/32 MT Task Establisher Task (TET). TET outputs 'load modules' which contain a Loader Information Block (LIB) followed by a core image of the task/library. The LIB enables the loader to derive the various parameters of the load module. All user tasks in OS/32 MT are established as though they were to be loaded at physical address 0 in the machine. Through the use of the Memory Access Controller (MAC) the task's program addresses will be relocated to correspond to the physical addresses occupied. This process is totally transparent to the user.

Executive tasks do not run with the MAC enabled, and hence must be written as totally position independent code. This will be discussed in Chapter 9. Overlays are loaded in the same manner as tasks. Library segments are loadable only through operator command when the system is quiescent. No task requesting use of a library can be loaded unless that library is present in memory.

2.1.7 Intertask Coordination/Communication

OS/32 MT R00 provides the foreground system of tasks with a means of intertask communication and control. The features provided include:

Load a task
Start/Delay Start a task
Cancel a task
Delete a task
Queue a parameter to a task
Change a task's priority
Obtain a task's status

These functions are performed via SVC 6 calls, and the desired function is specified via a parameter block. These calls may be directed towards another task or may be self-directed.

2.1.8 Task Handled Traps

OS/32 MT provides the user task with a mechanism whereby it may interrupt its normal execution, and enter a special subroutine upon the occurance of certain events. The events that will cause the user special subroutines to be entered are (in MT R00):

Receipt of a parameter on user task queue
Timer Completion
Power Restoration
I/O proceed completion
SVC 14

The routine to be entered, if any, is controlled by the current value of the Task Status Word (TSW) and of the task's User Dedicated Location (UDL). The TSW is a mask containing bits which are interpreted by OS/32 MT to enable (if set) or disable (if reset) the interrupt condition associated with the bit. The UDL contains addresses of special subroutines to be entered upon occurrence of an enabled event. The UDL also provides a storage area into which the value of the user's TSW and location counter previous to the event may be saved, so that the user may resume normal execution after completion of a special event handler.

The value of the TSW is manipulated through a Supervisor Call instruction (SVC 9). The addresses of subroutines, and the user status to be set when entering those subroutines may be set up by the task storing directly into its UDL.

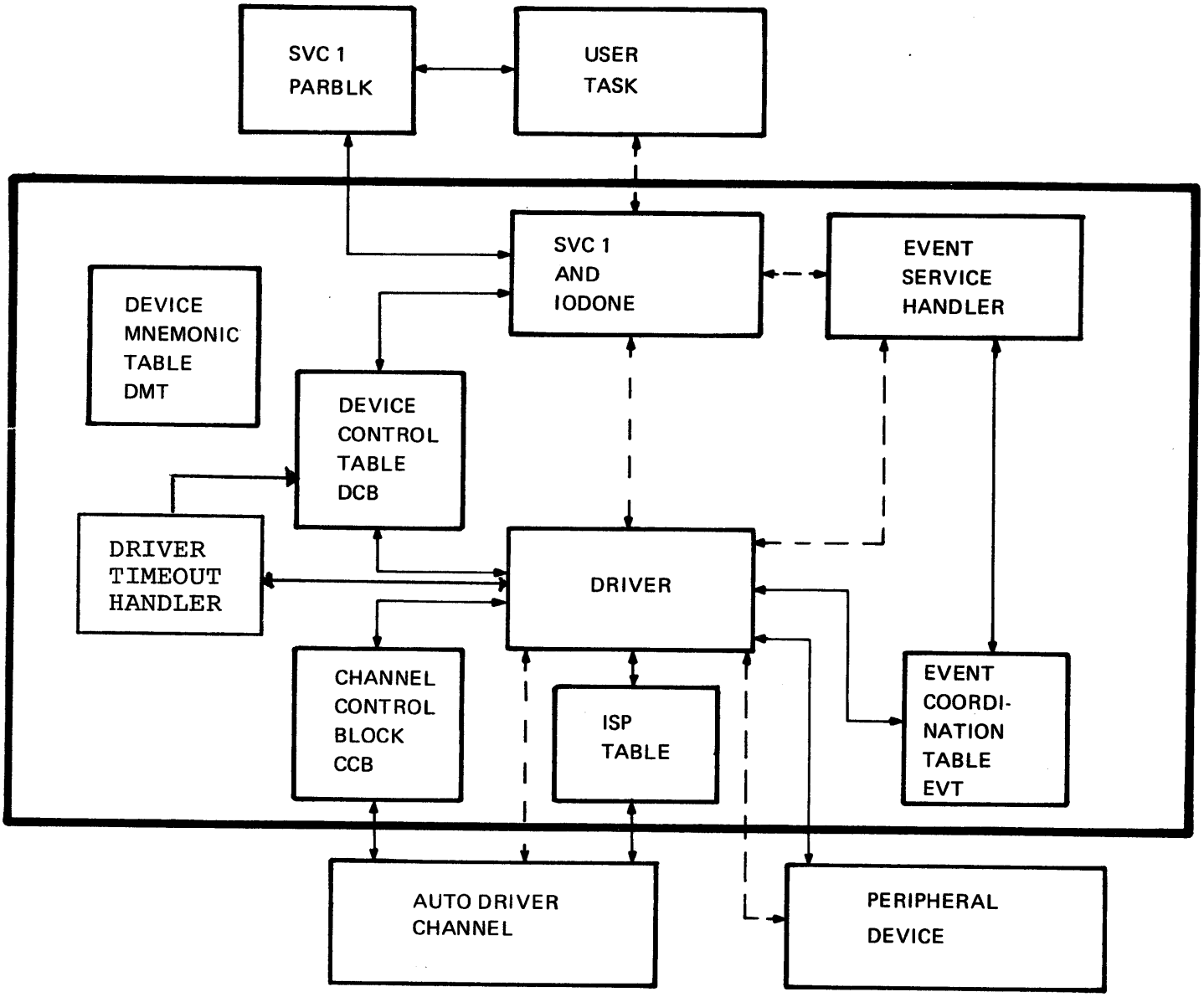


Figure 2-4. Elements of I/O System

2.1.9 Crash Handler

This routine is entered when the system cannot continue without the risk of destroying system or user information. A CRASH CODE is displayed on the display panel and is also stored in the SYSTEM POINTER TABLE (SPT) at SPT.CRSH. (See Chapter 10 for crash codes and meanings). System Initialization does not reset SPT.CRSH. Some of the conditions which cause the Crash Handler to be entered are:

Illegal Instruction within the system.
Invalid Item on the System queue.
Invalid TCB ID passed to Task Management.
Interrupt from undefined device.

2.1.10 System Journal

The system journal is a circular list of historical data maintained by the system. Each entry on the journal consists of five fullwords of information: the task id of the task which was active at the time of the entry, the reason for making the entry (Journal Code) and information pertinent to that call. The system journal is established at SYSGEN time by OS/32 MT Configuration Utility Program. System Journal processing may be eliminated at SYSGEN time. See Chapter 10 for a list of the Journal Codes and their meanings.

2.2 I/O SYSTEM

The I/O system consists of system routines and control blocks necessary to provide a device independent facility for performing I/O requests. It is composed of the SVC 1 executor, IODONE, device drivers, Device/Volume Mnemonic Tables (DMT/VMT), Device Control Blocks (DCB), Channel Control Blocks (CCB), Interrupt Service Point Table (ISPTAB), Logical Unit Table (LTAB), the Event Coordination Table (EVT), and the Event Service Handler (see Figure 2-4).

2.2.1 Device/Volume Mnemonic Tables

All devices and direct-access volumes are referred to throughout the system either by logical unit or by an ASCII identifier. These tables, DMT and VMT, bind these ASCII identifiers to the devices' DCB's.

2.2.2 Logical Unit Table

This table is physically present in all TCBs. It is of interest to the I/O subsystem and the file manager. It consists of a table of DCB or FCB addresses, one for each logical unit. If the logical unit is not assigned to any device, the entry is set to zero. The size of the Logical Unit table for all user tasks is the same, and is fixed at SYSGEN time. Access privileges are placed in a Logical Unit entry at ASSIGN time.

2.2.3 Device Control Block (DCB)

A DCB is provided for each device in the system. This control block contains device-dependent information such as the attributes of the device, flags and register save areas if needed. Pointers are provided to the driver initialization, interrupt service, and termination phases, as well as the event service leaf which coordinates access to this device (see below).

2.2.4 Channel Control Block (CCB) and Interrupt Service Pointer Table (ISPTAB)

CCB's and the ISP table are used to control I/O requests through the Auto Driver Channel capability of the 32-bit series processor.

2.2.5 SVC 1 Processor

The SVC 1 processor saves the user's registers, picks up the user's parameter block for the driver, and then makes several error checks. These are done primarily through the mechanism of checking the attributes bytes in the device control block against the function code specified in the call. If the call is in order, the system enters a reentrant state, places the data from the parameter block into the DCB and vectors to the appropriate driver.

2.2.6 Drivers

These are the same as the OS/32 ST drivers and are consequently fully reentrant, with the exception of the interrupt-handling phase. The initiation phase of an OS/32 driver runs as a reentrant subroutine of the task, i.e., using the user registers and with queue service enabled. The interrupt-handling phase runs with all interrupts inhibited, except for illegal instruction and machine malfunction. The termination phase of the driver runs in a reentrant state although no task may be executing more than one termination phase subroutine at a time.

2.2.7 Termination Event Coordination Table

The OS/32 Termination Event Coordination Table (EVT) is used to coordinate access to all devices, controllers, selector channels and bus switches in the system as well as other system resources that must have controlled access, such as bulk storage directories and allocation bit-maps. This table contains busy flags for all devices and pointers for each device that requires coordination, to the controller, channels and bus switches with which that device must be coordinated. See Section 2.1.4.

2.2.8 Trap Generating Devices

There exist a class of devices, called Trap Generating Devices (TGDs), whose interrupts require the scheduling of a task to perform a service in response to that interrupt. In OS/32 MT the means provided to respond to these interrupts is to have the driver queue a parameter to the appropriate user task, and awaken him, so that he may respond to the event. The Instrumentation Society of America (ISA) has established standard calls for handling these devices, and OS/32 MT supports:

Connect a task to a TGD (Connect)
Enable interrupts from a TGD (Thaw)
Disable interrupts from a TGD (Freeze)
Disconnect a task from a TGD (Break)

In addition, OS/32 MT supplies the user with a facility to simulate the occurrence of an interrupt from one of these devices (SINT).

2.3 COMMAND PROCESSOR

The Command Processor (which is also the System Task) provides the operator interface to OS/32 MT. It executes as a task in OS/32 MT and is designed so that many functions are performed through Supervisor Calls. The Command Processor contains routines to support the Command Substitution System (CSS), routines to do memory partitioning and routines to support Direct Access devices. The command processor controls all I/O requests to the Console and Log devices.

2.3.1 Command Processing

The Command Processor accepts commands from the system console device, decodes them and calls the appropriate executor. Some commands are executed via Supervisor calls (e.g., EXPAND, ASSIGN) while others are executed by the Command Processor routines (e.g., MARK, DISPLAY). The Command Processor contains logic to provide the console operator with informative messages in case of error.

2.3.2 Command Substitution System (CSS)

The Command Substitution System routines provide the ability to build, execute and control files of OS/32 MT operator commands. CSS consists of routines to execute CSS operator commands, to manage the CSS buffers and to provide the command parameter substitution facility. The CSS buffers are established at SYSGEN time by the OS/32 MT Configuration Utility Program.

2.3.3 Direct Access Support

The Command Processor provides the operator with the command functions necessary to allocate and delete files, display files, perform functions such as Rewind, backspace record to a file assigned to the user task and for mounting and dismounting direct access volumes. Most of these functions are executed via SVC 1 and SVC 7 calls.

2.3.4 Console Support

The Command Processor controls the user task communication with the keyboard/printer device used as the system console. This is accomplished via a dummy driver which intercepts all log messages and SVC 1 requests to the console device and executes them for the user task. Because of this feature and the structure of Task Management, most commands can be entered and executed while a user task is active, even if the task has assigned the console device.

2.4 FILE MANAGEMENT

The file management routines handle all access to bulk storage files, either by the user task or by the system. There are four basic modules in this package: the directory and bit-map handler, the Contiguous file access method, the Chained file access method, and the SVC 7 processor. In addition, there are a set of utility programs.

2.4.1 SVC 7 Processor

This package processes all SVC 7 calls. It calls on the directory and bit-map handler when a file is assigned, allocated, deleted, or check-pointed. When a file is closed, it calls the disc driver as required to make sure all valid data is written on the disc. Protection keys are checked by this module, which also performs all assignment of devices to logical units.

2.4.2 Directory and Bit-Map Handler

This package handles all access to and modifications of the directory and bit-map for each bulk storage device. Entries are provided to look up a file in the directory, to enter a new file name in the directory, to modify or delete a directory entry, to allocate one or more contiguous sectors of storage, or to release allocated bulk storage.

2.4.3 Contiguous File Access Method

This package is entered when an I/O request is made to a Contiguous file. It performs sector address computations and enters the disc driver.

2.4.4 Chained File Access Method

This package is entered when an I/O request is made to a Chained file. It handles all buffering and unbuffering, calls the disc driver for read or write whenever a buffer is filled/ emptied, and allocates new space on the appropriate bulk storage device as required for file expansion.

2.4.5 Disc Utility Programs

Along with OS/32 MT the user is provided with a set of Utility tasks which perform miscellaneous non-SVC 7 functions. This includes disc dump, disc initialization, bit map/directory initialization, defective sector flagging, etc.

2.5 Floating Point

At sysgen time the user may specify which type of floating point support his system is to contain. The choices are: no floating point, hardware floating point, or software simulated floating point. If no floating point is selected, than no task in the system may execute floating point instructions. If a task does execute a floating point instruction, it will be treated as an illegal instruction. The OS/32 MT resident loader will not load a task in a system without floating point support, if the LIB of its Load Module (see 2.1.6) indicates that Floating Point is required.

If software support is selected, a series of software routines to simulate floating point instructions is included in his system. Each time an illegal instruction interrupt occurs, control is passed to this simulation package. If it determines that the illegal instruction was, indeed, a floating point instruction, the package will perform the appropriate operation; if not, control is passed to the illegal instruction handler.

If either software or hardware floating point is specified, and a task's options indicate it uses floating point, the operating system will save and restore the current contents of the task's floating point registers when the task is stopped and restarted. This means that each task has its own unique copy of the floating point registers.

CHAPTER 3

SYSTEM CONVENTIONS

3.1 MACHINE STATES

OS/32 programs, tasks and routines run in one of eight well-defined states. These states are differentiated by a combination of PSW bits and status bits of an active task. Any state not defined below is not permissible. At any given instant in time, the processor is executing code in one of these states. They are, in increasing order of priority and privilege:

1) User Task (UT)
2) Executive Task (ET)
3) Reentrant System (RS)
4) Reentrant System, Alternate Save Area (RSA)
5) Event Service (ES)
6) Nonreentrant System (NS)
7) Nonreentrant System, User Registers (NSU)
8) Interrupt Service (IS)

The definition of these states in terms of PSW and TCB status bits is shown in Figure 3-1.

3.1.1 User Task State - UT

The UT state is the state in which all user tasks run. The PSW Protect and memory relocation bits are set. Internal and external interrupts are enabled, with the possible exception of the Arithmetic Fault interrupt bit, which is the only interrupt bit that is under user control. This state may only be exited via interrupt or execution of SVC. The User Register set is active.

3.1.2 Executive Task State - ET

The ET state is the state in which all executive tasks (E-tasks) run (see Chapter 9). Protect mode and memory relocation are disabled. All interrupts other than Arithmetic Fault are enabled. Arithmetic Fault is under control of the E-task. All SVC's are permitted. The User register set is active. This state should only be exited via interrupt or execution of SVC.

3.1.3 Reentrant System State - RS

The RS state is the state in which reentrant system code is executed on behalf of a task. Machine constraints are the same as for the ET state; however, software constraints are

more stringent: No code may be executed which would cause the RS state to be entered (e.g., a TYPE II SVC - see Section 3.2). Note that RS, RSA and ES code is executed on behalf of a task and is scheduled and dispatched as though it were a privileged routine of the task. For this reason, Queue Service interrupts are enabled. The User register set is active; the previous contents of the User registers is assumed to have been stored by the task manager in the RS Save area of the calling task's TCB. This state may be exited in several ways:

Branch to task manager to return to UT/ET state
or enter RSA state
LPSW or EPSR that enters NS or NSU state
I/O or internal interrupt

3.1.4 Reentrant System State, Alternate Save Area - RSA

This state is identical to RS state except that the previous contents of the user register set have been stored in an "alternate save area" (other than in the TCB). System code executing in RSA state may execute a full range of SVC calls (calls that enter RS state), since the RS save area is considered unused. Tasks in RSA state may also execute calls that cause the current contents of the user register set to be put into another (unused) alternate area. Alternate save areas are linked together as shown in figure 3-1. . Each new alternate save area is added at the beginning of the chain. RSA code must exit from each successive RSA state in an orderly manner. The state may be exited via:

LPSW or EPSR that enters NS or NSU state
I/O or internal interrupt
SVC
Branch to task manager to return to previous RSA state, RS state
Branch to task manager to enter another level of RSA

3.1.5 Event Service State - ES*

This state is identical to RS state in all respects except that the ES (Event Service) status bit in the task's TCB is set, disabling the dispatching of an event for that task (see Section 4.2). It is used only in drivers, principally in driver termination code. This state may be exited only via a call to the Return from Event routine in the event handler package.

---

* ES state was previously known as RSN state. Any reference to RSN should be interpreted as a reference to ES.

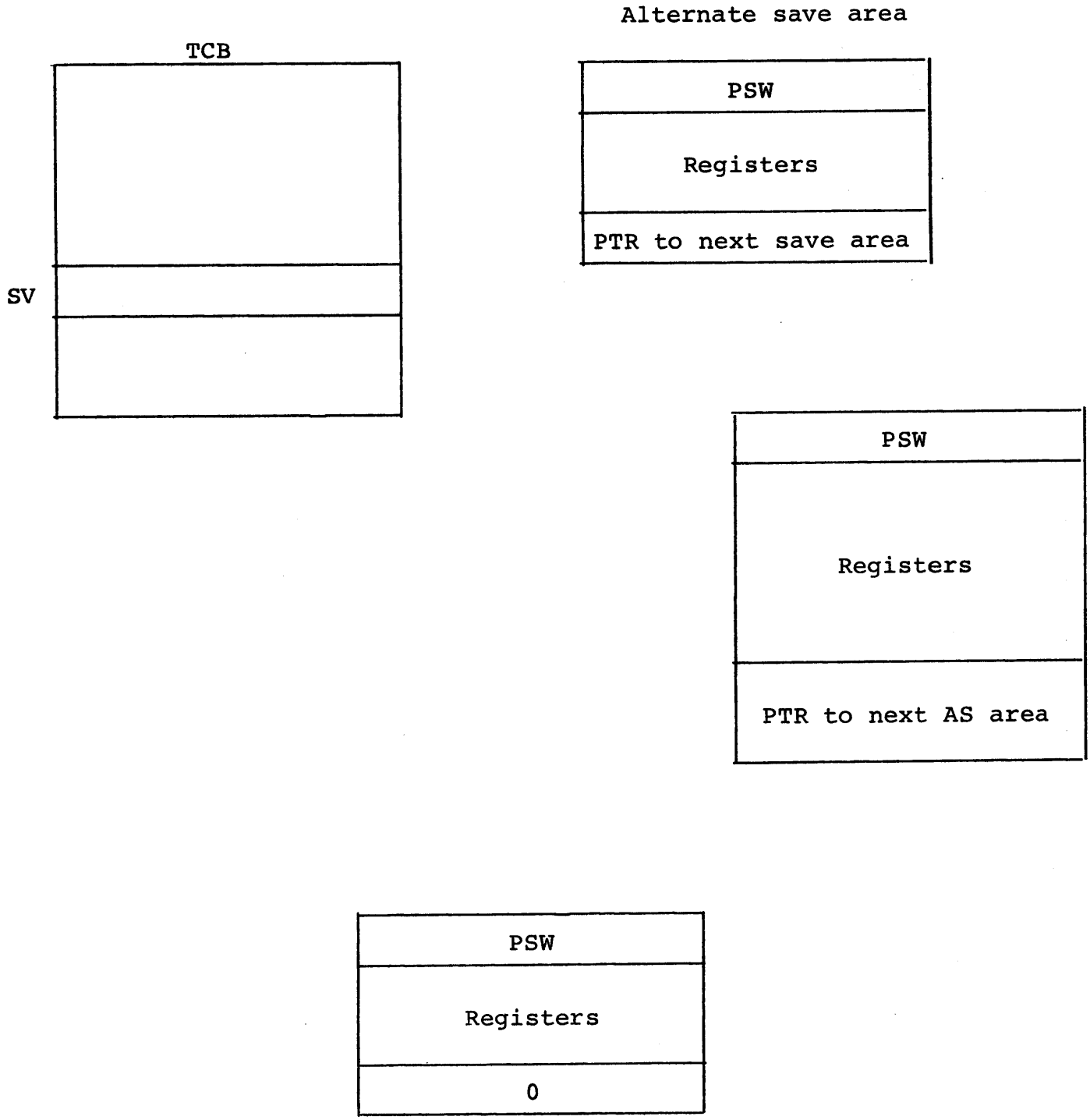


FIGURE 3-1

3.1.6 Nonreentrant System State - NS

The NS state is the state in which the system executes system code which changes critical system information such as EVT,TCB. This code is nonreentrant and Queue Service interrupts are disabled. The Executive register set is active, but NS is restricted in that the code may use only registers 8-F. As no new task may be dispatched while the system is in this state, routines that run in this state must necessarily be short and quick to execute. No SVC's may be executed. This state is exited via LPSW, EPSR, or external interrupt.

3.1.7 Nonreentrant System State User Register Set - NSU

This state is identical to the NS state (see Section 3.1.6) except that the User register set is enabled. It is used when the user registers are stored in a TCB or alternate save area. NSU code may use registers 0-F of the user set.

3.1.8 Interrupt Service State - IS

The IS state is used only for interrupt service routines within drivers and in the machine malfunction handler. All interrupts are disabled except machine malfunction. The Executive register set is active, of which IS code may use registers 2-7. This state is only exited via LPSWR on register 0 and 1. Since all interrupts are disabled, it is extremely important that all IS code be as brief as possible.

	PSW Status Bits						TCB Status			TCB Option
	I 17	MM 18	AF 19	MR 21	QS 22	P 23	R 24:27	ES 0	RS 1	ET 0
UT	1	1	d	1	1	1	F	0	0	0
ET	1	1	d	0	1	0	F	0	0	1
RS/RSA	1	1	1	0	1	0	F	0	1	d
ES	1	1	1	0	1	0	F	1	d	d
NSU	1	1	1	0	0	0	F	d	d	d
NS	1	1	1	0	0	0	0	d	d	d
IS	0	1	1	0	0	0	0	d	d	d

0 means bit must be zero
1 means bit must be one
d means bit may be zero or one
F means all 4 bits are one
I - Immediate Interrupt
MM - Machine Malfunction
AF - Arithmetic Fault
MR - Memory Relocation
QS - System Queue Service
P - Protect
R - Register Select
ES - Event Service State
RS - Reentrant System State
ET - Executive Task

FIGURE 3-1 System States

3.2 SVC DEFINITIONS AND CONVENTIONS

SVC		Function	Type
1		I/O	II
2	code 1	Pause	II
	2	Get Storage	II
	3	Release Storage	I
	4	Set Status	I
	5	Fetch Pointer	II
	6	Unpack	II
	7	Log Message	II
	15	Pack	II
	16	Pack File Descriptor	II
	17	Mnemonic Scan	II
	18	Move Characters	II
	19	Peek	I
	20	Expand Allocation	I
	21	Contract Allocation	I
3		End of Job	II
5		Fetch Overlay	II
6		Intertask Communication	II
7		File Management	II
9		TSW Swap	II
14		User SVC	II

All SVC interrupts cause the system to enter NS state. Each SVC enters a separate entry point in the First Level Interrupt Handler (FLIH). FLIH decodes the SVC number and passes control to the appropriate executor. There are two types of SVC executors: those that are short and do not require access to the user register set (Type I) and those that are lengthy or require access to the user register set (Type II). Type I SVC's execute in NS state, thus eliminating the overhead of saving the user registers. Type II SVC's execute for some portion in RS or RSA state.

FLIH passes control to an SVC executor with:

1) address of the Task Control Block of the invoking task in register 9.
2) unrelocated address of parameter block in register 12
3) address of the SVC parameter block in register 13.
4) resume PSW in registers 14 and 15.

Entry is in the state (NS or RS) indicated in a table contained in FLIH. On entry to the executor, the parameter block has been checked to insure it is on a fullword boundary and the address is between UBOT and CTOP+2. It is the responsibility of the executor to perform validity checking of any addresses passed in the parameter block.

3.3 INTERNAL INTERRUPT CONVENTIONS

Internal interrupts cause control to be passed to the individual interrupt handler in NS state. In all cases, a message is output to the system console indicating the nature of the interrupt and the address at which it occurred. In addition to the interrupts generated by the 32-bit architecture, illegal SVC calls and invalid addresses passed in SVC calls are handled by the internal interrupt handler as illegal instructions.

3.4 SUBROUTINE CONVENTIONS

Two levels of subroutine linkage are defined for the reentrant system states, RS and RSA, and for non-reentrant state, user register set, NSU. One level of subroutine linkage is defined for non-reentrant system state, NS.

3.4.1 RS, RSA and NSU Subroutines

The mainline level is allowed to use the full register set U0-UF. First level subroutines are linked through register 8 and may use registers U8-UF without save/restore. Second level subroutines are linked through register 12 and may use registers UC-UF without save/restore. This is a general definition used as a guideline; individual modules may violate this definition.

3.4.2 NS Subroutines

The mainline is allowed to use registers E8-EF of the executive register set. The first level subroutine is linked through register 8 and may use register E8-EB without save/restore. Subroutines that may be called from either NSU or NS must be written as an NS subroutine.

3.4.3 Calling Sequences

Parameters are passed in registers or in memory. Parameters may be passed in memory immediately following the BAL instruction only if the parameters require halfword alignment. Parameters may be passed in system tables such as SPT, TCB, etc.

3.4.4 Exits

The normal exit from a subroutine should be to the address contained in the link register, or to a specified number of halfwords past the address contained in the link register. Alternate exits must be to locations passed as parameters. Exits to unlabeled addresses are not permitted.

3.5 GENERAL NAMING CONVENTIONS

3.5.1 Data Structures

All data structures (defined by CAL STRUC statements) in OS/32 are named with three character symbolic names, e.g., TCB, SPT, DMT. All fields within these structures are defined by a name of the form SSS.FFF, where SSS is the structure name, and FFF is the field ID. (See Chapter 11 for structure definitions).

3.5.2 Bits

Certain fields in a data structure contain flag bits to denote information. These flag bits are manipulated with either bit instructions (e.g., TBT, SBT, RBT) or logical immediate instructions (e.g., THI, OHI, NHI). For each flag bit there are two definitions - one for the bit number and one for the mask. These definitions are of the form SFFF.XXB and SFFF.XXM where S is a character which refers to the structure ID, FFF are three characters which refer to the field, XX identifies the function of the flag bit, B denotes a bit number, and M denotes a bit mask. For example, in the TCB there is field TCB.OPT which contains the option bits; Bit 0 = 1 means the task is an EXEC TASK (E-TASK), Bit 0 = 0 means that the task is a USER TASK (U-TASK). The bit number and bit mask definitions of this flag are:

```
TOPT.ETB    EQU    0
TOPT.ETM    EQU    X'8000'
```

CHAPTER 4

EXECUTIVE DESCRIPTION

4.1 TASK MANAGEMENT

4.1.1 Task Control

In OS/32 MT a task may be in wait state or in ready state. Wait state indicates that some external event must take place before the task may proceed. Ready state indicates that all such necessary events have taken place. The task manager controls tasks through the use of several control blocks (see Figure 4-1).

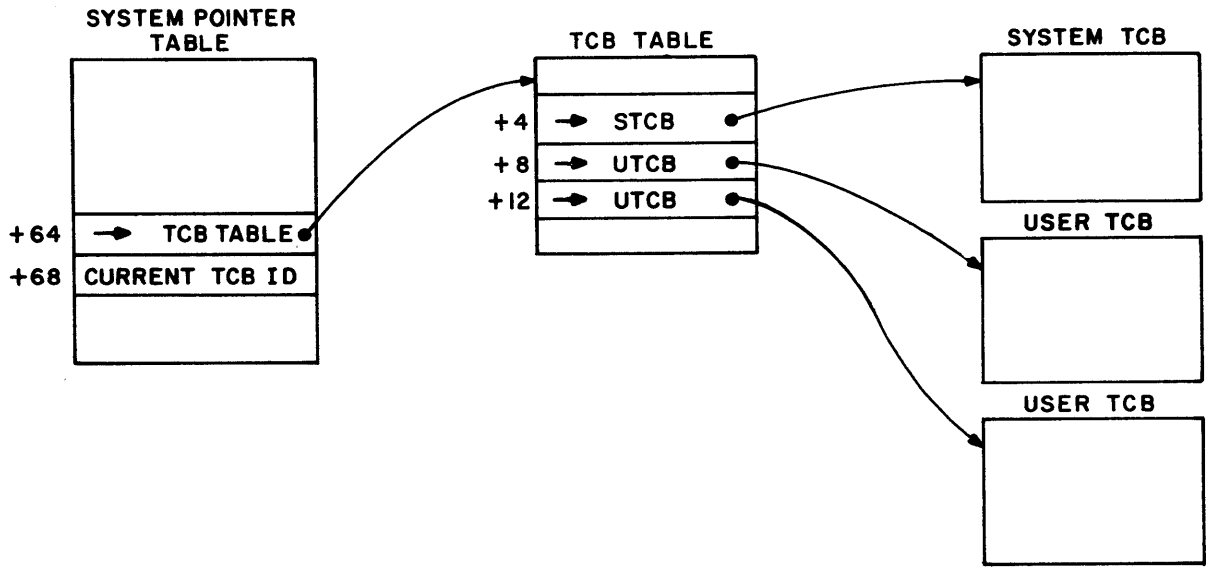


FIGURE 4-1 TASK CONTROL

Each task is described by a Task Control Block (TCB). The addresses of the TCB's are maintained in the TCB table which is pointed to by the System Pointer Table (SPT). A chain, is maintained of the tasks that are in ready state. This task ready chain is maintained in priority order. In OS/32 MT the System Task has priority 1 (highest) and the user tasks may be assigned priority between 10 and 249, inclusive. TCBs are referenced by their address or by their id. TCB id is the index, starting at 1, of the TCB in the TCB table. The System Task is TCB 1.

4.1.2 Task Management Facilities

The task manager provides subroutines to:

Dispatch the current task
Suspend the current task
Put a task on the ready chain
Remove a task from the ready chain
Cause a task to enter RS, RSA or ES state
Dispatch from top of EVT
Remove a wait condition from a task
Start the user task

In order to maintain the control of a task as it is in various states, the task manager uses three PSW and register save areas in the TCB, the dispatch save area, the RS save area and the ES save area. The state of these save areas is indicated by the ready chain, RS and ES bits in the status field of the TCB.

4.1.2.1 Dispatch Current Task (TMDISP, TMRDISP)

The task manager dispatches the current task by deciding which is higher priority, the task at the top of the ready chain or the task at the top of the EVT queue (see Section 4.2). The top of the ready chain is maintained in the SPT. If this TCB ID is zero, there is no task ready and if there is also no task at the top of the EVT queue the task manager places the system in an enabled wait state. If the TCB ID is non-zero, and that task's priority is higher than the priority of the task at the top of the EVT queue, the task manager loads the user register set from the TCB dispatch save area, loads the saved floating point registers, and then passes control to the task by loading the resume PSW in the dispatch save area.

4.1.2.2 Suspend the Current Task (TMSTOP)

This task manager facility is called to prepare the current task for removal as current task. The contents of the user's registers are saved in the dispatch save area of the TCB, the task's floating point registers are saved in the TCB, following the LU table, and the task's resume PSW is saved in the dispatch PSW save area of the TCB. This facility is used before placing the task in a wait state or when an event has occurred which has made a task of higher priority ready.

4.1.2.3 Chain (TMCHN)

When a task has become ready it is placed on the ready chain in priority order. In OS/32 MT, the ready chain may have any or all of the tasks in the system on it.

4.1.2.4 Unchain (TMUCHN)

When a condition exists that prevents a task from proceeding until an external event occurs, the task manager removes the TCB from the ready chain. The task need not have been the current task.

4.1.2.5 Enter System State (TMRSIN, TMRSNIN, TMRSAIN)

All reentrant system routines execute as privileged subroutines of the invoking task. On entry to one of these routines the task manager saves the current PSW and user register values in one of three places: the RS save area in the TCB if the task is entering RS state, the ES save area of the TCB if the task is entering ES state, and the specified alternate save area if the task is entering RSA state. The condition of these various save areas is indicated by the state of the corresponding bits in the TCB status field; if set, then the save area contains valid data. These routines must be entered from NS state.

4.1.2.6 Exit From System State (TMRSOUT, TMRSNOUT, TMRSAOUT)

On exit from one of the reentrant system states, RS, RSA or ES, the task manager restores the state of the task to the environment saved in the appropriate save area. The corresponding bit in the status field of the TCB is reset to indicate no valid data in that save area. An exception to this is when multiple RSA levels are in existence. In this case, the RS and RSA bits are not reset until the last RSA state has been exited. These routines also check for pending bits in the status field of the TCB, and if set, put the task into the corresponding wait state by moving the PSW and User registers from the specified save area to the TCB dispatch save area saving the floating point registers, removing the TCB from the ready chain, and branching to TMDISP.

4.1.2.7 Dispatch from Top of EVT Queue (EVTDISP)

This function of the task manager is discussed more fully in Section 4.2.4.5. In brief, if the task at the top of the EVT queue is of higher or equal priority to than the task at the top of the ready chain, it is chained and then dispatched as the current task.

4.1.2.8 Remove Wait (TMREMW)

This facility of the task manager removes the specified wait conditions from the specified task by resetting the corresponding bits in the wait field of the TCB. If no wait conditions remain the task is placed on the ready chain (TMCHN).

4.1.2.9 Start User Task (TMSTART)

The task manager starts the specified task by constructing the start PSW from the options field in the TCB and the specified location. This PSW is placed in the dispatch save area of the TCB and the TCB is chained on the ready chain. When the task becomes top of ready chain it is dispatched at the saved PSW with all the user registers set to zero, if the task had been just loaded.

4.1.3 Task States

The state of a task is defined by the settings of the bits in status and wait fields of the TCB and the value of the current TCB field of the SPT. The following is a list of detailed task states and their meanings:

State	Indication	Meaning
Dormant	Dormant bit TCB wait field	Partition associated with this has not been loaded.
Ready	Ready chain bit TCB status field	Task on ready chain
Current	TCB ID in SPT current TCB field	Task is the executing task.
RS	RS bit in TCB status field	TCB RS save area contains valid data. Task may be Ready or in wait state.
RSA	AS bit in TCB status field	Save area(s) pointed to by TCB contains valid data. Task may be ready or in wait. RS bit must also be set in TCB status field.
ES	ES bit in TCB status field	TCB ES save area contains valid data. Task must be Ready. I/O wait bit may also be set.
Wait	Ready chain bit reset TCB status field	Task needs external event before it may proceed.

4.2 EVENT SERVICE HANDLER

4.2.1 Event Coordination Table - EVT

Coordination of system resources is controlled through the Event Coordination Table. The EVT is a tree structure consisting of nodes (entries with descendants) and leaves (entries without descendants). Each path in the tree corresponds to a group of system resources that must be coordinated as one resource. Figure 4-2 illustrates a simple portion of an EVT. All paths in the tree are descendants of the system node.

4.2.2 System Queue

The 32 bit architecture provides for the facility of a system queue. This queue is maintained in the standard list format, and is pointed to by a fixed location (X'80') in low memory. Whenever the status portion of the PSW is updated with the Queue Service bit set, an internal interrupt is generated if there is an entry on this queue. OS/32 MT uses the system queue to schedule events coordinated by the EVT. The entries made to the system queue are always in the form of an address of a leaf in the EVT. When a system queue service interrupt occurs, the leaf is said to have evented. In OS/32 MT there should be at least as many system queue slots as there are devices in the system.

4.2.3 Coordination

In order to explain the Event Service Handler, the following terms must be defined:

4.2.3.1 Connection

In order to assume control of a system resource reflected in the EVT a task must be connected to the resource. A task is connected to a leaf and ancestor nodes, up to system node, by placing the task ID and priority in the leaf and the task ID, the task priority and the connected leaf address in the upper nodes. An unconnected leaf has a TCB ID of X'00' and a priority of X'FF'; unconnected nodes have, in addition, a connected leaf address of 0. Only one task may be connected to a leaf or node at a time. A task must be connected to all entries between a connected leaf and the highest connected node in the path. Referring to Figure 4-2, a task could be connected to the following EVT entries:

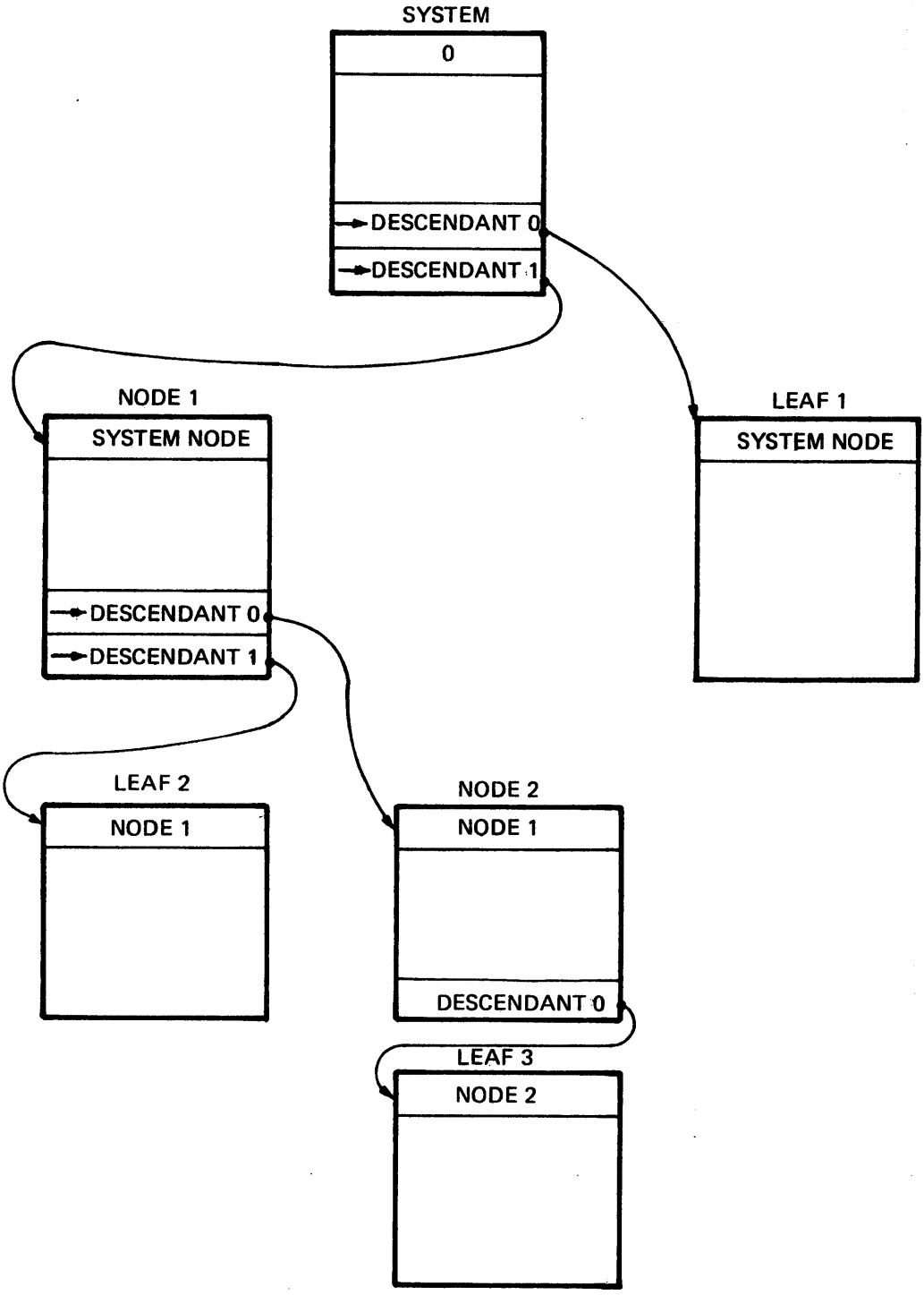


Figure 4-2. Portion of EVT

Leaf 1
Node 1, Leaf 2
Node 1, Node 2, Leaf 3
Node 2, Leaf 3
Leaf 2
Leaf 3

A task could not be connected to just Node 1. Tasks never connect to the system node.

4.2.3.2 Queueing

Before connection is made to an EVT entry, all upper nodes must be unblocked (not connected). If an upper node or the leaf being connected to is blocked, the Event Service Handler, upon request, will queue the task for the leaf. This leaf queue is maintained in priority order. While a task is on the leaf's queue, it is placed in a "connection wait" state. Each unblocked node maintains a pointer to the descendant subtree of the highest priority. Thus the highest priority task on a leaf queue is queued to the highest unblocked upper node.

4.2.3.3 Assertion

Although a task must be initially connected to an entire path in the EVT, it can release upper nodes if they are not necessary for some portions of an operation. When the task again requires these upper nodes, it is said to be asserting reconnection and an assert flag is set in the highest connected entry of this path, a pending flag is set in the leaf and the priority of the highest connected entry is propagated. When the asserting task becomes top of EVT, the dispatcher reconnects required upper nodes before dispatching the Event Service Routine (see Section 4.2.4.5).

4.2.4 Event Service Facilities

The Event Service Handler provides subroutines to:

Connect a task to a path in the EVT
Disconnect a task from EVT entries
Release upper nodes
Service System Queue Service interrupts
Dispatch from top of EVT
Return from event
Propagate a priority up the EVT

All Event Service Handler routines execute in NS state except for return from event.

4.2.4.1 Connect (EVCON, EVQCON)

Requests for connection always specify a leaf address to be connected to. The connection routines check all upper nodes up to system node. If any upper nodes are connected in some other path, a condition code of X'F' is returned (EVCON) or the task is placed on the leaf's queue and into connection wait (EVQCON). If the path is unblocked, the task is connected to the leaf and the leaf is added to the task's connected leaf chain and a condition code of X'0' is returned.

The connected leaf chain is maintained as a bi-directional list of leafs pointed to by the TCB. The connection wait chain is maintained as a bi-directional list of TCBs pointed to by the leaf. A task can be connected to a leaf and in connection wait for it at the same time. Refer to Figure 4-3 for an example of a task connected to two leaves and both that task and another task in connection wait for the first leaf. This would occur if the connected task and the queued task issued I/O requests to a device prior to the completion of a previous I/O and proceed request by the connected task to the same device.

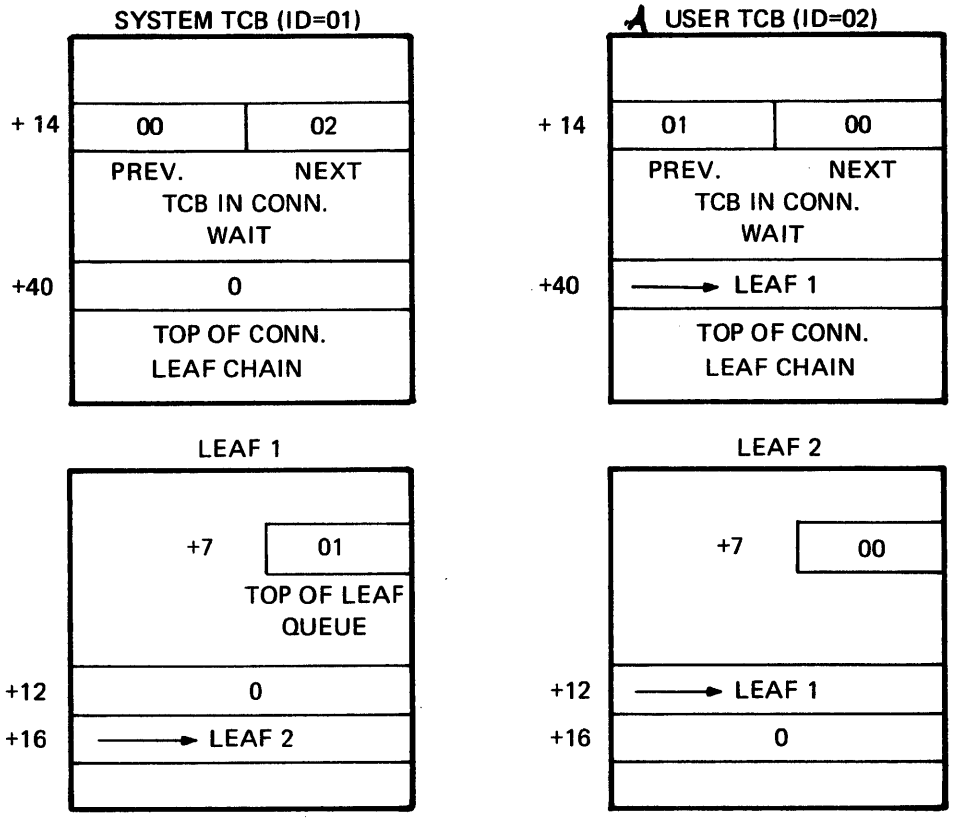


FIGURE 4-3 CONNECTION AND CONNECTION WAIT

4.2.4.2 Disconnect (EVDIS)

Requests for disconnection always specify a leaf address. A task is disconnected from the specified leaf and all upper nodes. After each node is unblocked, the priority of the task on top of the queue for that node, if any, is propagated up to the node.

4.2.4.3 Release (EVREL)

Requests for release specify a leaf address and the level above the leaf that is to be released. All nodes from the specified level up are unblocked as in disconnection. For example, the disc driver requires only the disc for a seek operation, so the nodes from level 2 up can be released (the disc controller node and the selector channel node).

4.2.4.4 System Queue Service (SQS)

SQS is entered on a queue service interrupt from the microcode. The leaf address on the bottom of the System Queue is removed. SQS obtains the address of the connected TCB from the TCB id stored in the leaf. The event count in the leaf and in the connected TCB are incremented by one. SQS checks the status flag to see if the task is eventable. If the task is already in ES state it is non-eventable and SQS simply loads the PSW at the time of the interrupt. The non-zero occurrence counts in the leaf and the TCB queue the event to the task. If the task is eventable, SQS sets a pending flag in the leaf and an assertion flag in the highest connected node. The priority of the task is then propagated up the tree from the highest connected node (see Section 4.2.4.7). If the priority was able to propagate up to the top of the EVT (system node), then the current task, if any, is suspended by saving the current PSW and user registers in the dispatch save area of its TCB and the event service routine for the connected task (which may be the current task) is scheduled by EVTDISP (Section 4.2.4.5), unless the current task is higher priority. If the current task is of higher priority, it is redispatched and the connected task is dispatched into the event service routine when it becomes top of ready chain.

4.2.4.5 Dispatch From EVT (EVTDISP)

Every routine that causes the current task to be changed branches to the task manager routine TMDISP to determine the next task to be dispatched. TMDISP determines the next current task by comparing the priority of the task at the

top of the ready chain, if any, with the priority of the task queued to the system node, if any. If the top of EVT priority is equal to or higher than the priority of the task at the top of ready chain, EVTDISP is entered to dispatch the task queued to the EVT. The task queued to the top of the EVT is queued for one of two reasons: it is in connection wait or it is queued for the dispatching of an event service routine (asserting reconnection).

EVTDISP walks down the EVT by loading the pointer of the highest queued descendant at each level until it reaches a leaf or entry with the assert flag set. If it reaches a leaf, the task at the top of the queue for that leaf is in connection wait. If this task is also currently top of the ready chain then TMRDISP is entered to redispatch it where it was interrupted. Otherwise, EVTDISP removes the TCB from the queue, connects the task to all upper nodes, removes the task from connection wait (thus putting it on the ready chain) and branches to the task manager to dispatch the task.

If EVTDISP reached an entry with the assert flag set, it resets the assert flag, resets the pending flag in the connected leaf, connects the task to all upper nodes and branches to the task manager routine TMRSNIN which puts the task in the ready chain, if necessary, saves the current state of the task in the ES save area of the TCB, decrements the event count in the leaf and the TCB by one, and dispatches the task at the event service routine pointed to by the leaf.

4.2.4.6 Return From Event (EVRTE)

All event service routines terminate through EVRTE. EVRTE checks the event count in the TCB in case an event has occurred for the task during the event service routine. If no events are queued, EVRTE simply passes control to the task manager routine TMRSNOUT to exit from ES state. If the TCB event count is non-zero, EVRTE searches the connected leaf chain for the first leaf with a non-zero event count. If the task is connected to all upper nodes, the task is redispatched in ES state at the event service routine pointed to by the leaf. If it is not connected to all upper nodes, the pending flag is checked. If the leaf's pending flag is set, then the task has been propagated for this leaf and the routine continues to search the connected leaf chain of the task for a leaf with a non-zero event count.

If a leaf with a non-zero event count is found without the pending flag set, pending is set and EVRTE walks up the EVT to the highest connected node, sets the assert flag in that node and propagates the priority up the EVT.

When all leaves in the task's connected leaf chain have been processed, EVRTE passes control to TMRSNOUT to exit from ES state.

4.2.4.7 Propagation (EVPROP)

Whenever the priority that is queued to an entry in the EVT is changed, that priority is propagated up the EVT to insure that the highest priority task is always queued to the top of the EVT. Since a task is never connected to an entry in the EVT until it is able to be connected to all the entries in the path required, propagation insures that the highest priority task will be connected first at the time the path is free. Requests for propagation specify the address of the starting EVT entry and the priority to be propagated up from that entry.

The priority is propagated by stepping up the EVT one level at a time and comparing the propagating priority to the highest queued priority of that node. If the propagating priority is lower than the priority of the top of the node's queue, EVPROP returns normally. If the propagating priority is higher, EVPROP replaces the node's highest queued priority with the propagating priority and stores the descendant number of the entry just stepped up from in the node's highest queued descendant pointer. This continues until the priority has been propagated up to the system node or a blocked node is encountered. If the priority is propagated up to the system node, EVPROP returns with a top-of-tree indication.

If a blocked node is encountered, EVPROP stops queuing descendants but if the propagating priority is greater than the priority of the task connected to the node, it replaces the connection priority and the propagation process continues.

4.2.5 Dispatch Priority

In OS/32 MT each task has two priorities associated with it: the task priority and the dispatch priority. The ready chain is maintained in dispatch priority order. The system task priority is 01 and a user task's priority is established at TET time. In most cases the dispatch priority of the task is equal to the task priority. However, if a task is connected to an EVT entry that is blocking a path requested by a higher priority task, then the blocking task will have its dispatch priority raised to the priority of the blocked task for the time it is connected to that entry.

In OS/32 MT, certain leafs exist which are non-eventing. These leafs are used to coordinate and control access to certain system logic paths or data structures (such as loading or bit map access). A subroutine executing on behalf of a task, that is connected

to these leafs might block execution of a higher priority task. In that case, the dispatch priority of the connected task will be raised to the level of the highest priority task that it is blocking for the duration of its connection to non-eventing leafs.

A third priority is maintained (TCB.NPRI) when a task is connected to non-eventing leafs. This indicates the highest priority that a task connected to a non-eventing task is blocking. If this priority is higher than the dispatch priority or the priority of another blocked leaf, the task is dispatched at this priority. Each time a non-eventing leaf is released, a search is made for the new highest priority non-eventing leaf the task is connected to, and the ready chain is readjusted to reflect the task's new priority.

4.3 SVC HANDLER

4.3.1 First Level Interrupt Handler (FLIH)

Each SVC interrupt vectors to a separate entry point in the First Level Interrupt Handler, in NS state. FLIH stores the SVC number in a save area and branches to a common SVC preprocessing routine. FLIH maintains a table which controls the preprocessing for each SVC. Most require a parameter block. The address as passed, unrelocated by the microcode, in register 13 is checked to insure that it is between UBOT and CTOP+2 and is on a fullword boundary. The end of the parameter block is checked except for SVC 2 which may have different length parameter blocks, and SVCs 3 & 14 which have no parameter blocks. FLIH then makes an entry in the system journal and checks the table for the entry state required by the requested SVC executor. If the executor requires NS entry, FLIH branches to executor; if the executor requires RS entry, FLIH calls the task management routine TMRSIN to pass control in RS state. On entry to an executor, register 9 contains the TCB address of the invoking task and registers 14 and 15 contain the resume PSW. On entry to selected routines, register 12 will contain the unrelocated parameter block address and register 13 will contain the relocated parameter block address.

4.3.2 SVC 1 Executor (SVC1)

Entry to the SVC 1 executor is in NS state. The executor checks the Logical Unit (LU) specified and if it is valid it loads the address of the DCB (for a device) or FCB (for a file) from the TCB of the invoking task. Since the fields of the DCB or FCB used by SVC 1 are identical, processing is independent of the device or file being referenced. SVC 1 checks the validity of the request for the device or file by comparing the request against the

attributes in the TCB LU table (for data transfer requests) or the attributes field of the DCB/FCB (for command function requests). Requests to the null device are returned with normal status immediately. If any errors are detected in the request, the appropriate status is placed in the parameter block and control is returned to the invoking task.

If the request is for wait only or test I/O complete, SVC 1 processes the request in NS state by loading the address of the leaf in the EVT associated with the device from the DCB. If the connected TCB ID of the leaf is different from the TCB ID of the invoking task, normal status is returned immediately to the task. If the TCB ID is the same, then the task has I/O proceeding on the specified device. If it is a test I/O complete request, SVC 1 sets the condition code to X'F' and returns; if it is a wait only call, SVC 1 checks to see if the incomplete I/O is to the same LU as specified and, if not, returns normal status to the task. If the I/O is to the same LU then SVC 1 places the task into I/O wait for the I/O request by setting the I/O wait bit in the TCB wait field, removing the TCB from the ready chain,manipulates the function code field of the DCB to reset the command function bit, and sets the wait (X'08') bit, and exits to the task manager routine TMDISP.

SVC 1 then enters either RS state (requests for device or unbuffered file) or RSA state (request to a buffered file) depending on the state of the buffered access method flag in the flags field of the DCB/FCB. It then processes the information in the parameter block.

For data transfer requests, SVC 1 checks the validity of the start and end addresses and if the request is not for a buffered file, it calls the event service handler routine EVQCON to connect the task to the EVT entries pointed to by the leaf address in the DCB for requests with the unconditional proceed bit reset, or it calls the routine EVCON for requests with unconditional proceed set. If EVCON cannot connect the task to the requested path in the EVT it returns a non-zero condition code to SVC 1 which sets the condition code in the resume PSW in the RS save area of the TCB to X'F' and branches to the task manager routine TMRSOUT to return control to the invoking task. If EVQCON cannot connect the task to the requested path, it puts the task in connection wait. When the task is connected, control returns to the instruction following the EVQCON call in SVC 1. If the leaf address in the DCB is zero, no connection is performed. After processing the required connections, SVC 1 stores the relocated start and end addresses and random address from the parameter block into the DCB and branches to the driver or file manager entry point specified in the DCB/FCB. If the request is for I/O and wait, before branching to the driver,

SVC 1 sets the I/O wait pending flag in the status field of the TCB. Upon exit from the driver or file manager initialization routine, the task manager puts the task into I/O wait. For command function requests, SVC 1 performs a call to EVQCON for all device requests. For all requests, SVC 1 then passes control to the command function entry point specified in the DCB/FCB.

4.3.3 SVC 1 Termination (IODONE)

Drivers and the file management access routines exit to IODONE to complete I/O requests. IODONE is entered in ES state (IODONE) or RS state (IODONE2) with a DCB address and a leaf address. IODONE places the status returned in the DCB into the parameter block if there is one, calls EVDIS to disconnect the task from the specified leaf and its upper nodes, removes the I/O wait condition if this call is an I/O and wait call, and branches to EVRTE if called from termination routine of a driver, or to TMRSOUT if called from an initialization routine at entry IODONE2.

If an I/O proceed call was made, IODONE calls SV9.ATQ to add to the task's queue, and cause an I/O proceed trap (see Section 4.3.8) if these conditions are enabled.

4.3.4 SVC 2 Executors (SVC2 and SVC2.xx)

SVC 2 requests are all vectored to the SVC 2 second level interrupt handler SVC2 for common preprocessing. SVC2 maintains a table of valid SVC 2 codes indicating the type of preprocessing required and the entry state required by the individual executors. SVC2 checks the validity of the code and then performs validity checking on the register specifications passed in the parameter block if necessary. For SVC 2 codes that require parameters to be passed in registers, SVC2 assumes 2 formats of the parameter block:

0	1	2	3
OPTIONS	CODE	X'0'	REGISTER #

for parameter blocks requiring one register specification, and:

0	1	2	3
OPTIONS	CODE	REGISTER #	REGISTER #

for parameter blocks requiring two register specifications.

SVC2 then branches to the SVC 2 executor for the specified code for NS entry executors, or branches to the task manager routine TMRSIN to enter RS entry executors.

4.3.5 SVC 3 Executor (SVC3)

SVC3 is entered in NS state from the first level interrupt handler. In NS state, SVC3 stores the specified return code in the TCB and goes down the connected leaf chain pointed to by the TCB and halts any read requests by calling the routine TIMEOUT (see Section 7.6). It then enters RS state for the rest of processing. SVC3 issues an SVC 7 checkpoint call for memory resident tasks, or an SVC 7 close call for non-resident tasks, for each LU in the TCB. This insures that all writes have been normally completed, that the timeout of all reads is also complete and that all files are in a safe condition. If the task is currently connected to a trap-generating device, the connection is frozen and disconnected via a call to TGD.8. It then puts the return code in the end of task message and sets up the TCB and the message function code so that the task is put into I/O wait while the message is printed on the system console and that control is returned to the SVC 3 processor in NSU state on completion of the message. After issuing the end of task message, SVC 3 walks the time of day and the precision interval time chain. The TCB ID field is zeroed for each TQE pertaining to this task. Then SVC 3 removes the TCB from the ready chain and if non-resident, from the peer task chain. It sets the dormant bit in the TCB wait field and zeroes out the TCB. SVC 3 then exits to the task manager.

4.3.6 SVC5 - Load Overlay

SVC5 is called to load an overlay. It is entered in RS state. The logical unit specified in the parameter block is obtained, and a flag is set if the options field specifies rewind. SV6.LOD1 is called to load the overlay. Upon return, the status of the load is determined. If the load was successful, TMRSOUT is called; if not, the status field of the SVC5 parameter block is set to the appropriate error code and then TMRSOUT is called to return to the user task.

4.3.7 SVC 6 - Intertask Service Functions

4.3.7.1 Decode SVC 6 Options (SV6.MAIN)

The basic function of SV6.MAIN is to test each bit of the SVC 6 function code (SV6.FUN), store status and priority in the SVC 6 parameter block and call the appropriate executor.

SV6.MAIN is entered in RS state. Upon entry the error status field of the parameter block (SVC6.STA) is zeroed out. The task-id of the task issuing the SVC 6 is tested for '.BG'. If the calling task was the background task, the SVC 6 flag in its TCB is tested. Either TMRSOUT is called to return to the user level (if SVCCONTINUE is specified), effectively treating the SVC 6 as a "no-op". If SVCPAUSE was specified, ISHRS is called to indicate an illegal SVC 6 call.

SV6.MAIN checks the direction field in the function code to determine if the SVC6 call is a self-directed call. In such a case, a flag is set to indicate to executors that the SVC6 call is issued on behalf of the current task. A pointer to the executed function is maintained.

CHECK.ID is called to validate the syntax of the task name. If the name is invalid, SV6.ERR is called. SV6.SCAN is called to determine to which task the specified task-id belongs. That task's status and current priority are stored into the SVC6 parameter block.

SV6.TAB is a table of 32 fullword entries. Each entry corresponds to a bit in the function code. The entry is either the address of the corresponding executor, or zero if the bit specifies an illegal function code.

When all bits have been examined, TMRSOUT is called to return to user level.

4.3.7.2 Executor Design

An executor may be added to the SVC 6 package at any time by including its address in the executor address table in the location corresponding to its function code bit.

All executors have the following responsibility;

Test the condition code upon entry for presence of called task; branch to appropriate entry in SV6.ERR if necessary.

4.3.7.3 SVC 6 Error Handling (SV6.ERR)

SV6.ERR is called whenever an error is detected during execution of an executor except SV6.LOAD. It stores the error code and current position pointer in SVC6.STA and then calls TMRSOUT to exit to UT/ET state. The SV6.ERR2 entry point is called to return to RS state before storing the status. SV6.ERR3 is the entry point for illegal function code.

4.3.7.4 Find a TCB (SV6.SCAN)

SV6.SCAN enters NSU state and then searches the TCBs for the TASKID given in the parameter block (SV6.ID). If the task is found, condition code is set to non-zero to indicate task is present; if the task is not found, the condition code is set to zero to indicate task is not present.

SV6.SCAN returns to RS state.

4.3.7.5 Cancel Task - (SV6.CAN)

SV6.CAN is entered in NSU state and calls CANEOJ. If this executor was called for a self-directed call, it returns to UT/ET state.

4.3.7.6 Delete Task (SV6.DELE)

SV6.DELE is entered in NSU state. It resets the memory resident bit in TCB.OPT to indicate task is no longer resident. SV6.DELE than calls SV6.CAN.

4.3.7.7 Queue Parameter - (SV6.QPAR)

This routine is entered in NSU state. The parameter to be added to the task queue of the called task is picked up from the parameter block (SV6.PAR). SV9.ATQ is called with this parameter reason code 1, and the TCB ID of the task whose queue is to be added to (i.e., the called task). If the condition code returned by SV9.ATQ is negative, SVC.ERR2 is called with the error code set to 12.

4.3.7.8 Change Priority - (SV6.PRIO)

SV6.PRIO is entered in NSU state. The priority in the parameter block (SV6.PRI) is compared to the maximum priority of the called task (TCB.MPRI), and the lesser of the two is stored in TCB.PRI. If the specified priority is less than 10 or greater than 249, SV6.ERR2 is called with error code 5 to indicate illegal priority.

4.3.7.9 Trap Generating Devices - (SV6.TGD)

This module is called for any of the five trap-generating device (TGD) functions in the function code. DMTLOOK is called to obtain the address of the device control block. If the device is not found, an error exit is taken. The DCB is checked to see if the device is "SVC 6 connectable" (DFLG.S6B=1). If it is not an SVC 6 connectable device, an error exit is taken. The function code pointer is used to index ISATAB, a five byte table with each entry corresponding to the TGD functions.

CONNECT - EVCON1 is called to connect the device leaf to the specified task. If the device is presently connected to any task, an error exit is taken. The queue parameter (SVC6.PAR) is saved in DCB.PBLK. Return is made to SV6.MAIN.

If thaw, freeze, or SINT is specified, the function code X'C0', X'90', or X'A0' respectively is stored in DCB.FC. EVL.CTCB is compared to called task's number to determine whether device is still connected to the task. If device is no longer connected, an error exit is taken. The driver is entered at the "command entry point" (DCB.FUNC). Control is returned by the driver to SV6.MAIN.

If unconnect is specified, a "freeze" call is performed. The device is now disassociated from the connected task (by calling EVDIS1).

4.3.7.10 Start Task (SV6.STAR)

SV6.STAR is entered in NSU state. If a self-directed call is being made, an error exit is taken.

The start location is obtained (from SVC6.SAD in parameter block) or TCB.SLOC, if SV6.SAD is zero.

If the task is not dormant (TWT.DMB is not set), an error exit is taken. A carriage return is stored at TCB.UTOP.

TMSTART is called to put the task on the ready chain. If the current task is still at the top of the ready chain, TMRSOUT is called to return to UT/ET level.

If another task is now at the top of the ready chain, then the dispatch PSW is set up to return to RS state, and TMDISP is called.

4.3.7.11 Delay Start - (SV6.STAD)

SV6.STAD is entered in NSU state. An error exit is taken if a self-directed call is being made. An SVC 2,23 instruction is built in UDL.AIDS. This is followed by a branch to the starting address (SV6.SAD) if non-zero, or TCB.SLOC. A branch is made to STA.4 in SV6.STAR, where the dormant bit is tested.

4.3.7.12 The Resident Loader (SV6.LOAD)

This routine is used to load tasks, overlays, and library segments that have been created by the Task Establishment Task (TET/32). It is entered in one of 3 ways; via an SVC 6 load call, via an SVC 5 load overlay request and via the Command Processor (both the Command Processor and SVC 5 enter at SV6.LOD1). SV6.LOAD is entered in RS state. It calls EVQCON to obtain control of the loader leaf (LDRLV). TMRSRSA is called to enter RSA state, and TMRSARS is called before exiting to return to RS from RSA.

Loading a Library

The loader reads the Loader Information Block to determine what type of segment is being loaded (see TET/32 Manual for LIB description, also chapter 11).

The Loader checks the amount of space available (as passed by the Command Processor). It moves the RTL name to SPT.RTLN and then reads in the RTL. It returns indicating the size of the library that was loaded.

Loading a Task

NSU state is entered to determine that the specified task-id is not present. The task options are checked to determine that the task being loaded is consistent with the system being loaded into (i.e., tasks using floating point may not be loaded into systems not supporting floating point). An appropriate partition is found (if not specified). The task is loaded, and the segmentation registers are set up for the RTL and TCOM, if the task uses any.

The following parameters are copied into the TCB: MPRI, PRI, DPRI, MXSP, USSP, OPT, CTSW, SLOC, UTOP. The loader puts a task in the peer task chain.

Loading An Overlay

Before reading the LIB, an SVC 1 rewind command is issued if the call requested it. A test is made to determine if available storage for the overlay exists. The overlay name is tested to verify that the overlay name in the LIB is the same as the name in SVC 5 parameter block. The physical address at which to load the overlay is determined and it is loaded.

If LOD.ERR is called on behalf of an SVC 6 load call the error status is stored in SVC 6 parameter block, and TMRSOUT is called to return to UT/ET level. If LOD.ERR is called on behalf of the Command Processor or SVC 5, the condition code is set to non-zero before returning to the Command Processor. TMRSARS is called since any call to LOD.ERR is issued when the loader is in RSA state. LOD.ERR disconnects the loader leaf by calling EVDIS. If necessary, the task name is removed from TCB.NAME.

4.3.8 Task Traps (SVC 9)

Associated with each task is a task status word (TSW), which is the task level analogue of the PSW. Specific bits in the TSW control:

Trap wait	
Trap Enable for	Power restoration
	SVC 14
	Task Queue Service
Enable Queue Entry for	TGD Device Interrupt
	SVC 6 Queue Parameter request
	Timeout Completion
	I/O Proceed

Queue entries are handled by SV9.ATQ; traps are performed by SV9.STSW. SVC 9 updates the TSW and the location for resuming execution of the user level program.

TSW Update/Establishment (SVC 9)

SVC 9 is entered in RS state. The new TSW which was pointed to by the SVC argument is stored in TCB.CTSW. The current PSW condition code is changed to the condition code found in the new TSW. The new TSW "loc", which follows the TSW status fullword in the SVC 9 parameter block is tested. If zero, the current PSW location is unchanged; otherwise, the PSW location is changed to the location specified by the TSW. The PSW status and location are saved in the RPSW save area in the TCB, since SVC 9 was entered in RS state. The task queue service trap enable bit in the "new" TSW is tested. If it is not set, TMRSOUT is called. If the trap bit is set, the task queue (the address is in UDL.TSKQ) is examined. If the queue is empty, TMRSOUT is called and a trap is not taken. If there are any items on the queue, SV9.STSW is called to perform TSW swap. Upon return, TMRSOUT is called to return to UT/ET level.

4.3.8.1 Add to Task Queue (SV9.ATQ)

SV9.ATQ is entered in NSU state. It is passed the reason code parameter and the TCB id of the task whose queue is to be added to. The task queue reason codes are:

Code bits 0-7	Meaning of Code	Parameter bits 8-31
0	Device Interrupt	Param. assoc. with device
1	SVC 6 Queue Param	Param. Spec. in call
8	I/O Proceed Complete	Addr. of SVC 1 param. blk.
9	Timer Termination	Parameter spec. in call

SV9.ATQ tests the validity of the reason code, and then determines whether the appropriate Queue Entry Enable bit in TCB.CTSW is set. If set, UDL.TSKQ is tested for the address of the task queue. If the address is not equal to zero, ADCHK is called to return the physical address of the task queue as well as to ensure the task queue is in a valid, writeable segment. If the task queue address is valid, the end slot in the queue is also tested by calling ADCHK. If the queue exists and is in a valid writeable segment, the reason code and parameter are added to the bottom of the task queue. The condition code set by the ABL instruction is tested to determine if the queue was full. If the queue was updated, the Task Queue Service Trap Enable bit in the TSW is tested. If set, SV9.STSW is called to perform a TSW swap. SV9.ATQ sets one of the following condition codes upon return:

C	V	G	L	Meaning
0	0	1	0	Item queued; TSW swap occurred
0	0	0	0	Item queued; no TSW swap (bit not enabled)
0	0	0	1	Item not queued (no TSW swap) for any of the following reasons: Invalid reason code Queue Entry bit not set No task queue Task queue not in valid, writeable segment Task queue full

4.3.8.2 Cause a Task To Take a Trap (SV9.STSW)

When it has been determined that a task is to take a trap, SV9.STSW is called. It is entered in NSU state and is passed the address of the TCB for whom the swap is being performed, and the physical address of the UDL swap area.

The current TSW status, and current value of the location counter are stored in the "old TSW" portion of the swap area. The new status and location at which to redispatch the task are obtained from the "new TSW" portion of the swap area. The UT/ET level PSW is located, and the new location is stored there.

If the task is not in trap wait, SV9.STSW returns. If it is in trap wait, the new TSW is checked to determine if it also has trap wait set. If the task is to be taken out of wait, TMREMW is called. If the top-of-chain is changed by this call, TMSTOP is called to suspend the task that was previously top-of-chain, and TMDISP is exited to, to dispatch the appropriate task.

4.3.9 User SVC (SVC 14)

SVC 14 is entered in RS state. The SVC 14 trap enable bit of the current TCB.CTSW is checked. If the bit is not set the SVC 14 call is considered illegal and a branch to ISHRS is taken. If the trap enable bit is set, the SVC 14 argument address is stored in UDL.SV14, which is the area in the UDL reserved for use by SVC 14. Then NSU state is entered and SV9.STSW is called to perform the TSW swap. Upon return TMRSOUT is called to return to UT/ET level.

4.3.10 ADCHK

All SVC executors check any addresses passed to insure that the address lies between UBOT and CTOP+2. This prevents the Operating System from being misled by a user task into overwriting another task, or an RTL segment. This checking is performed by the routine ADCHK. Entry to ADCHK is made in one of two places, at ADCHK, for checking addresses from all states other than RSA state, and to ADCHK1 to check addresses from calls in RSA state. The address passed is manipulated to determine its logical segment, and then that logical segment is checked to see if it is present. If it is not present, ADCHK returns with the C bit set in the condition code. If it is present, ADCHK checks to see if the address specified is within the limits of the specified segment. If not, ADCHK returns with the C bit set in the condition code. The address is then relocated, and the logical segment determined. The G and L bits are set if the segment is non-writeable or non-executable, respectively. If the address checked is not valid, the SVC executor routine branches to MEMFAULT (from NS) or MEMFLTRS (from RS) to process the error (see Section 4.7) and suspend the offending task.

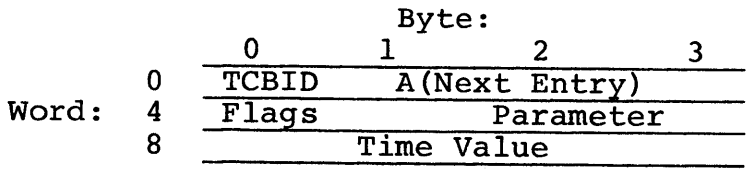
4.4 TIMER MANAGER

OS/32-MT requires a Universal Clock, which consists of a Line Frequency Clock (LFC) and a Precision Interval Clock (PIC). The LFC is enabled when the operator issues a SET TIME command, whereas the PIC is only enabled when a task requests for a non-zero time interval wait or trap. When the LFC is enabled, the time-of-day is kept in seconds since midnight in a fullword SPT.TIME. Should the system be built with the display panel option, the month, day, hour and minute will be displayed on the front panel and updated every minute on the minute. Furthermore, the month and day are displayed according to either the U.S. or European option.

In OS/32-MT tasks can request time-of-day and time interval waits and traps. Every interval entry requests system space, for a block that is kept in the Interval Chain. The same is done for the time-of-day entries, and a separate chain is maintained for these. A pointer to the head of the interval queue is kept in (SPT.IQHD) as is a pointer to the head of the time-of-day chain (SPT.TQHD).

4.4.1 Structure and Management of The Timer Chains

The interval timer chain and the time-of-day chain are identical in structure, each element being formatted as follows:




```
TMQ          STRUC
TMQ.TCB      DS     1
TMQ.NEXT     DS     3
TMQ.FLGS     DS     1
TMQ.PRAM     DS     3
TMQ.TIME     DS     4
ENDS
```

The A(Next Entry) field points to the next entry in the chain. It is zero for the last entry in the chain.

The Flags field is defined as follows:

Bit 0 = 1	already evented
Bit 1 = 0	Trap on termination
Bit 1 = 1	Wait for termination
Bit 2 = 1	TMQ in use
Bits 3-7:	Undefined, must be zero

The TCBID field is the TCB identifier of the affected task.

The Parameter field is used only for Trap calls, and contains the datum to be added to the Task Queue of the affected task on termination of the timeout.

The Time value field represents the number of seconds or milliseconds, respectively, for the timeout. This entry is incremental; that is, it refers to the elapsed time from the execution of the previous chain entry.

Therefore, if the interval chain contains entries to be executed respectively 10, 20 and 50 milliseconds from now, the time value fields of these entries would be 10, 10 and 30. In order to find out the time to a given interrupt, the time value fields of that entry and of all the preceding entries on the chain must be summed.

The treatment of the time value field for the first entry on the chain differs between the Interval and Time-of-Day chains. The value of the first entry of the Time-of-Day chain is decremented each second; therefore, its value always indicates the number of seconds from the present time. The value in the first entry of the Interval chain is not so decremented. Its value refers, instead, to the number of milliseconds from the time the PIC was most recently started at which the interval is completed. In order to find the number of milliseconds from "now", therefore, the PIC must be read and its count value added tp tje va;ie om the Interval chain's first entry.

4.4.2 Handling LFC Interrupts (ISRLFC)

The LFC when enabled interrupts at twice the line frequency. For example, in the United States, where the line frequency is 60 Hz., the LFC interrupts at 120 times per second.

This routine keeps a count which is initialized at twice the line frequency. At every interrupt, the count is decremented by 1. When it reaches zero (which means one second has elapsed) an event is scheduled to be serviced, and the count is reset to twice the line frequency.

4.4.3 Handling PIC Interrupts (ISRPIC)

The PIC only interrupts when a task's requested time interval has timed out or at time intervals of 4095 milliseconds, whichever is shorter.

This routine searches down the interval timer queue to determine which entries have timed out (the timed out entries have time values of zero). The search ends when the first non-zero time value is found or when the chain ends. Each timed out entry is then scheduled to be serviced. When all the entries in the queue have been processed and the queue is empty, then the PIC is stopped, otherwise, the next time interval is used to set up for the next PIC interrupt.

4.4.4 Handling of Completed Time/Interval Waits, Clock Maintenance (TIMESR)

The timer ESR is scheduled by either ISRLFC or ISRPIC when a time/interval event has occurred. If an LFC event has occurred, SPT.TIME is incremented by 1 second. The month, day, year are updated as appropriate, according to the Gregorian calendar. In case the object sysgen required the date and time to be displayed on the front panel, then every minute on the minute the following will appear on the panel (in decimal):

SOPT.USB=0 (mmddyy)	0	Month mm	Day dd	Hour hh	Minute mm

SOPT.USP=1 (ddmmyy)	0	Day dd	Month mm	Hour hh	Minute mm

Each entry in the device timer chain, if any, is decremented by 1. If a device has timed out, its device leaf address is added to the system queue (SQ).

The time value of the first entry of the time-of-day chain (if any) is decremented by 1. If it results in zero, the wait/interval time has elapsed. In this case, if it was a wait, the task is removed from the wait state by a call to TMREMW, its TMQ is released, and the next entry in the chain is processed in the same manner. If it was a trap, it is processed the same way as a wait except a call is made to SV9.ATQ (in NSU state) to add the specified parameter to its task's queue.

For a PIC event to have occurred, the head of the interval chain must have timed out. This means that the time value is zero. It is treated in the same fashion as the time-of-day entry that has completed. Note that the only difference is that the head of the time-of-day queue is decremented, whereas the head of the interval queue is not. PIC or LFC event service exit via a call to EVRTE.

4.4.5 SVC 2 Timer Calls

SVC 2 code 8, 9, 10, 11 and 23 deal with the clock routines. SVC 2 code 8 and 9 fetch time and fetch day respectively. SVC 2 code 10 sets up for a time-of-day wait, SVC 2 code 10 sets up for an interval wait. SVC 2 code 23 includes the functions of SVC 2 codes 10 and 11 and also does time-of-day and interval traps.

When a task requests an interval wait or trap, the time interval is compared to the time value sent from the PIC (if the PIC is enabled). If the requested interval is less than the interval established, the PIC is set to interrupt the request interval

and the requested interval will become the entry at the head of the queue. Otherwise, the appropriate place in the chain to place this request is determined by stepping down the chain, and subtracting from the value requested the successive outstanding requests, until the appropriate location is found. Note that the chain is linked by means of the TMQ.NEXT field, so not only are the time values updated, the next entry fields are also updated. Each time interval is relative to the one preceding it. In other words, the chain is such that the time interval between now and the head of the queue is the shortest, and the time interval between now and the end of the chain is the longest.

Since the PIC can only accept at most 4095 millisecond intervals, all entries with time intervals longer than 4095 milliseconds will only be able to set up the PIC to interrupt at 4095 millisecond intervals. Note that though the PIC will interrupt after each 4095 msec interval, ISRPIC will not schedule an event service till the entry's time value has reached zero.

When a wait is requested, the task is put in wait state before exit. All the adjustments to the timer chains are done in NSU state. If a zero time interval wait is requested, it is ignored. If a zero interval trap is requested, the trap will be taken immediately.

When a time-of-day wait or trap is requested, the present time (seconds since midnight) is subtracted from the time specified in the user's SVC parameter block. If the resultant time value is equal or less than zero, then 86400 seconds (or 24 hours) is added to it, to indicate that the interval is to be completed at the same time the next day.

This entry is put in the time-of-day chain in much the same way as for the interval chain except the time values are in seconds, and every second the time value of the top entry is decremented by one.

4.5 SYSTEM JOURNAL

OS/32 MT provides a facility for recording significant events in the system in a system journal. The journal is a standard circular list with a length specified at Configuration Utility Program time. The address of the journal is kept in the System Pointer Table. Entries to the journal are made from system routines by executing a BAL instruction to the journal routine followed by a halfword journal code. Each entry in the journal consists of five fullwords of information in the following format:

	0	1	2 + 3
WORD 1	TCB ID	X'0'	JOURNAL CODE
WORD 2	CONTENTS OF REGISTER 12		
WORD 3	CONTENTS OF REGISTER 13		
WORD 4	CONTENTS OF REGISTER 14		
WORD 5	CONTENTS OF REGISTER 15		

where TCB ID is the ID of the current task at the time of the journal call, and the last four words are the contents of registers 12-15 at the time of the journal call. Entry to the journal routine must be in NS state. When the journal list is full, the journal routine resets the slots-used field and reuses the list, thus maintaining the most recent entries. For a complete list of journal codes made by the system, see Chapter 10.

In order to allow the system task and other Executive tasks (see Chapter 9) to make journal entries from other than NS state, an SVC 2 code 0 call is provided. The parameter block for SVC 2 code 0 is:

```
+0          X'0', X'JOURNAL CODE'
+4      VALUE 1
+8      VALUE 2
+12     VALUE 3
+16     VALUE 4
```

where values 1-4 are stored in the second through fifth word of the journal entry. The journal code is OR'd with X'8000' before being stored in the journal to identify it as a user code. This SVC is only valid from a task executing in privileged mode (bit 23 of the PSW status reset, and Executive Task).

4.6 EXECUTIVE MESSAGES

Since the executive routines cannot issue SVC calls, all messages output by the executive are processed by the system task (command processor). This is accomplished by connecting to the dummy leaf (see Section 5.7), storing the start and end address of the message in the dummy DCB and branching to the dummy driver. All messages are processed in this way by branching to the executive message subroutine EXECMSG.

4.7 CRASH HANDLER

Throughout OS/32 MT are checks for normally impossible states of the system, such as invalid leaf address on the system queue or illegal instruction interrupt in system code. When such a condition is found the system brings itself to a halt before further destroying the conditions that led up to the impossible situation. This is done by entering the crash handler.

The crash handler is entered by issuing a SINT instruction to device number 0 followed by a halfword crash code. The first entry in the ISP table is set by SYSINIT (see Section 4.8) to branch to the crash handler, CRSEP. CRSEP on entry loads the address of the system journal into register 5 of the executive set, the address of the last entry made to the system journal into register 6 of the executive register set, displays the crash code on the display panel and loads a PSW with only the wait bit and the machine malfunction enable bit set, thus stopping the system in an uninterruptable state. See Chapter 10 for a complete list of crash codes and their meanings.

4.8 INTERNAL INTERRUPT HANDLERS

The internal interrupt handlers process the interrupts generated by the microcode for illegal instruction, arithmetic fault, memory parity error, power fail and power restore. In addition, this package processes illegal SVC calls and invalid addresses passed in SVC calls.

4.8.1 Machine Malfunction Handler (MMH)

On detection of memory parity error, power fail or power restore, the machine malfunction handler (MMH) is entered. Entry is in a state with all interrupts masked off. The condition code is used to determine the type of interrupt and the appropriate routine is entered.

On parity error in the system, the crash handler is entered. If the parity error is detected while the user task is executing, MMH loads a pointer to the memory parity error message and enters the illegal instruction handler for common interrupt processing.

On power fail detect, MMH tests an internal flag to see if a power restore sequence was in execution at the time of the power fail. If this is so, MMH simply loads an enabled wait PSW to wait for the power restore interrupt. If a power restore sequence was not in execution, the Executive and User register sets are saved in an internal save area, the machine malfunction old PSW is loaded from reserved memory and stored in an internal save area. The OS does not use

the Power Restore Auto/Restart save area since multiple power fails would destroy the original state of the system. MMH then sets an internal flag to indicate a power restore sequence is in effect and loads an enabled wait PSW to wait for the power restore interrupt.

On power restore detect, the machine mulfunction handler enters ISU state, clears the display panel and loads the address of the TCB table. For each TCB, all active I/O is halted by passing the address of all leafs on the task's connected leaf chain to the time-out routine (see Section 7.6).

If the task is not dormant and it has power restore trap enabled, the power restore pending trap bit is set. Otherwise, the pause pending bit is set in the user TCB. A task in trap wait has the wait removed by a call to TMREMW. MMH then calls the power restore subroutines (entry point from the console DCB) to issue a message to the system console stating that a power restore has occurred and that all peripherals must be reset. This is necessary primarily due to the fact that on a true power fail/restore sequence the 2.5 and 10 Mbyte disc systems come up in a write protected state, making it impossible to retry any active disc I/O. When the operator has reset all necessary peripherals and typed GO, MMH reloads the register set from the internal save area, resets the power restore sequence in effect flag and reloads the machine malfunction old PSW from the internal save area.

MMH issues the power restore message from ISU state to insure that no interrupts from the timed out I/O can occur before the peripherals have been reset.

MMH must also cleanup the time of day and interval chains. For each entry on a chain, if the TQE indicates a wait is outstanding, TMREMW is called to remove the wait. The TQE is released by a call to RELESYS. When each item on both chains have been released, the pointer to each chain is zeroed in the SPT (0 SPT.TQHD, 0 SPT.IQHD).

4.8.2 Illegal Instruction Handler (IIH)

Entry to IIH is in NS state. IIH contains common processing for illegal instruction, illegal SVC call, invalid address passed in an SVC call and memory parity error. Each error causes control to be passed to a separate entry point which loads a pointer to the appropriate message and branches to the common interrupt processing.

4.8.3 Memory Fault Handler (MFH)

Upon occurrance of a memory fault interrupt, the Operating System clears the MAC status register by writing a 0 into it. The address of the memory fault error message is loaded, and MEMFAULT is entered to output the message, and pause the offending task.

4.8.4 Arithmetic Fault Handler (AFH)

Entry to the arithmetic fault handler is in NS state. If the error occurred in the system, the crash handler is entered. If the error occurred during user task execution, the pause pending bit is set in the status field of the user TCB unless the Arithmetic Fault Continue bit is set in the options field of the TCB. In either case, the interrupt is logged by loading a pointer to the arithmetic fault message and branching to the common interrupt processing in IIH.

4.9 SYSTEM INITIALIZATION

System initialization is performed by the routine SYSINIT. It is entered whenever the system is started at location X'60'. On entry, the status of the PSW is unknown, so the first operation performed by SYSINIT is to put the processor into an uninterruptable, privileged state. The display panel is cleared, the ISP table is set to ignore all interrupts, FBOT is reset to MTOP, all DCB's are reset to initial values, the EVT is refreshed, the timer queues are reset, the time and date are set to zero, the TCBs are reset, and SYSINIT puts the system TCB on the top of the ready chain and branches to the Command Processor initialization routine in the ET state.

CHAPTER 5

THE COMMAND PROCESSOR

The Command Processor is the highest priority task in OS/32 MT. It is an Executive Task (see Chapter 9), and it is the medium through which the operator communicates with the Operating System, and controls the system environment. Whenever possible, the Command Processor tries to perform functions by executing Supervisor Calls. The Command Processor is also responsible for control of the system console, memory partitions and the Command Substitution System (CSS).

5.1 COMMAND PROCESSOR INITIALIZATION (COMMAND)

After System Initialization has been performed, the Command Processor is entered. The mnemonic of the console device is obtained from the Initial Value Table (IVT) and the Device Mnemonic Table is searched for a matching mnemonic. When found, the device's read and write counts are forced to -1 and it's keys to X'FFFF', thereby making the device unavailable to any other task in the system (see Section 5.7, System Console Device). The OS identifier is now printed.

The system TCB has '.SYS' stored as its name, the background task has '.BG' stored as its name. The Command Processor calls EVQCON and connects the timer leaf to itself. The value of the "currently selected task" for console commands and for CSS commands is set to indicate no task selected. The RESET command is then exited to, to do initial partitioning of memory in the system.

5.2 COMMAND INPUT/PARSING (COMMANDR)

The Command Processor reads commands from the system console and also from the device/file indicated by the currently selected CSS level. The Command Processor checks the CSS level before issuing a read request for a new command line. If no CSS is in effect (CSS level = 0) then an I/O and wait is issued to the system console. If CSS is in effect, an I/O and proceed is issued to the system console, and an I/O and wait is issued to the CSS device/file.

When the CSS I/O is done, the currently selected task (for task related commands) is set to the value of the current CSS task, and the line read is processed. After its processing, the status of the proceed I/O to the console is checked. If it is finished, then the currently selected task is set to the current console task and that command line processed.

If the I/O and proceed is ongoing, and further CSS lines are to be read, then an I/O and wait is issued to the CSS file. If no further CSS lines are to be read, a "wait only" is performed on the console.

A single command or multiple commands may appear on a command line. The Command Processor executes all commands on a line until it hits an end of line indicator, or an invalid command. All commands on a line before the invalid command are processed, all commands following the invalid command are ignored.

5.2.1 Command Prompts

The Command Processor outputs to the system console an "*" to indicate to the operator that the Command Processor is ready to receive another input line. While the background task is active, if previous command input was from a CSS file, the processing of that file is suspended until the task goes to End-of-Job. During the interval that the background task is active, the console responds to command input.

5.2.2 Command Parsing

After the line is read, if it is a line read from a CSS file, it is expanded (see Section 5.5, CSS). All lines read are logged (see Section 5.4.3, Set Log). This is true unless one of the BUILD commands is in effect, (see Section 5.5.4, BUILD). After the logging, a scan is made of the command line for the first non-blank, non-terminator. Note that the Command Processor uses register 1 as the pointer to the current character being processed, and that this register is not used for anything else. When the first non-blank, non-terminator is found, it is compared against the Command Mnemonic Table (COMANTAB). If it is not found the command is assumed to be a CSS call (see Section 5.5, CSS). If the command turns out not to be a CSS call (file/device does not exist), a MNEMonic error has occurred.

A check is made to see if any IF statements have set the "skip" flag (see Section 5.5.3, IFs). If so, and if this statement is an IF, then the IF count is incremented, and a new command is searched for. If the statement is an $ENDC the IF count is decremented. If it is a $TERMJOB, it is also executed.

The normal path makes a "user journal entry" (X'8001'), and exits to the executor.

5.3 COMMAND ERROR HANDLING (CMDERROR)

When an error in syntax, or an invalid parameter, or a number of other parsing errors occur, the Command Processor enters the error handler. All entries are made via a BAL on UC, the next 4 bytes after the BAL contain the error mnemonic. An error message is constructed of the form:

```
XXXX-ERR    POS = XX.....
```

The position field attempts to display the last parameter parsed. It may not always be meaningful. The return code in the currently selected TCB is set to 255. If the JOB flag indicates a $JOB is in effect, the JOB "skip" flag is set to indicate all statements until a $TERMJOB are to be skipped. All CSS levels are closed down to the level of the $JOB.

If a load error, I/O error or SVC7 error is encountered while processing a command, additional information on the error is supplied. In these cases, the error message is:

```
XXXX-ERR   TYPE=XXXX   POS=XX.....
```

where TYPE indicates the error type (DU, NAME, BUFF, PRTY, etc.). If the load error or SVC7 error is an I/O error, then

```
XXXX-ERR   TYPE=IO   TYPE=XXXX   POS=XX.....
```

is displayed, where the second type field indicates the I/O error type.

5.4 COMMANDS

5.4.1 Task Related Commands

Certain commands pertain only to tasks. The task that these commands will be applied to is the currently selected task. The currently selected task is set by the TASK Command.

When a TASK command is entered, the Command Processor searches the "peer task" chain. If a task with a matching name is found, then it is set as the currently selected task. It is also set as the currently selected CSS task or console task, depending upon where the command was read from. If the task entered is '.BG', the currently selected task is set to the background.

From this point on any task related commands are applied to the currently selected task. When a task goes to end-of-job, a check is made to see if it is the currently selected task, CSS task or console task. If so, and if the task is not memory resident, that mode is set to indicate the absence of a selected task. A task related command given when there is no selected task causes a TASK-ERR message.

The task related commands in OS/32 MT R00 are: START, PAUSE, CONTINUE, CANCEL, ASSIGN, DISPLAY LU, CLOSE, OPTIONS, SET PRIORITY, DISPLAY PARAMETERS.

START - obtains the start location, moves in the starting 'options' above UTOP and calls TMSTART.

CANCEL - Calls CANEOJ, indicating the task to be cancelled.

PAUSE - Calls S21PAUSE to put the selected task in Console wait.

CONTINUE - calls TMREMW to remove the console wait and put selected task back on ready chain.

OPTIONS - used to set/reset the options bits in the user TCB options field (TCB.OPT).

SET PRIORITY - makes sure the specified priority is legal and not greater than the task's TET'ed maximum priority. Store the new priority as the task's priority (into TCB.PRI).

ASSIGN - used to assign a file/device to a user logical unit. It defaults access privileges to SRW, keys to 0000. Initially, the specified device/file is assigned to Command Processor LU3. If the task is a "user task" the Command Processor puts itself temporarily into UT state to perform the assign. After a successful assign, the Command Processor picks up from its TCB the amount of system space the assign took up and determines if this assignment would put the task over its maximum. If not, the Command Processor LU3 is copied to the appropriate LU table entry for the task. Command Processor LU3 is zeroed out.

CLOSE - Copies the user LU to Command Processor LU3, and puts a zero in the task's LU entry. It executes an SVC 7 to close LU3, obtains the amount of system space released, and deducts this from the task's total of used system space (TCB.USSP).

DISPLAY LUS - Displays a list of all user LUs that are currently assigned, and to what device/file they are assigned.

DISPLAY PARAMETERS - Displays various parameters associated with the task. Some of the parameters displayed are NAME, STATUS, TSW, UTOP, CTOP, etc.

5.4.2 Device/File Commands

The device/file related commands are: ALLOCATE, DELETE, MARK, RENAME, REPROTECT, DISPLAY FILES, and DISPLAY DEVICES.

ALLOCATE - Builds an SVC7 parameter block from data as defined in the input line. If type is chained, it defaults LRECL to 126 and BKSZ to 1. It then executes an SVC7.

DELETE - Executes an SVC7 with the File Descriptor specified.

MARK - This will mark a device on or off line. If a non-bulk device, the on-line bit is merely set/reset. If the device specified is a bulk device (disc) then:

> MARK ON: Reads the volume descriptor and moves the directory pointer to the DCB, and the volume ID to the VMT. If Protect is specified it sets DFLG.WPB, else it resets the bit.
>
> MARK OFF: Flushes the bit map buffer and directory buffer, and resets the presence bits. Clears the VMT entry name portion.

RENAME - Executes an SVC 7 to assign the specified file/device and an SVC 7 to rename it.

REPROTECT - Executes an SVC7 to assign the specified file/device and an SVC7 to reprotect it.

DISPLAY FILES - This displays the files on the specified volume. It gets the pack ID, finds which drive it is on and assigns the drive SRO. The extent of display is then checked for. In the syntax "-" means all, and hence, ABC.- means any files with name ABC and all extensions, or -.- means any file name and any extension.

DISPLAY DEVICES - Displays a list of all devices in the DMT, their physical addresses and their keys. It indicates whether the device is off-line. If the device is a disc, it displays the name of the volume currently mounted on that drive.

5.4.3 General Commands

The other commands are: BIAS, EXAMINE, MODIFY, RESET, SET LOG, VOLUME, SET TIME, DISPLAY TIME, DISPLAY MAP, SET PARTITION.

BIAS - read the BIAS value to be used by the EXAMINE and MODIFY commands and saves it.

EXAMINE - gets the starting location and adds the bias. It then checks for a '/' or ','. If a '/' is found, it gets the ending address and adds the bias. If a ',' is found, it gets the number of halfwords to be displayed, and computes an ending address from it. The contents of memory from the starting location through the ending location inclusive is displayed, 8 halfwords per line, to the console/log device.

MODIFY - This command obtains the starting address and adds the bias. Data is obtained from the command line as halfwords, and is stored in successive memory locations.

VOLUME - Sets up SPT.VOL for the default volume name.

SET LOG - gets the file/device and assigns it to LU2.

If no file/device is specified, LU2 (current log) is closed.

If the COPY option is specified, then the copy flag is set. This command causes all input lines to be logged to the device specified. If the COPY option is specified, all input/output messages will also appear on the system console. If COPY is not specified, system messages will not be logged to the console.

SET TIME - scans the input line and picks up the date in the form mm/dd/yy if SOPT.USB is reset, or in the form dd/mm/yy if SOPT.USB is set. It determines the validity of the date and if the year specified is a leap year, sets TM.FEB to 29, else it sets it to 28. The date is stored in the SPT. The time is obtained in the form hh:mm:ss, and is checked for validity. It is converted to seconds since midnight. If there are any items on the time-of-day queue, they are updated to reflect the change in hour only. The time is stored in SPT.TIME, TMFREQ is set to the value in SPT.FREQ and the line frequency clock is enabled.

DISPLAY TIME - The date and time are obtained via SVC 2,8 and SVC 2,9 and displayed.

DISPLAY MAP - The size of the Library segment and Task Common segments are displayed. For all partitions the partition number, name (if a task is loaded in the partition), starting address, size, status and priority (if a task is loaded in the partition) are displayed. The start and size of .SYS is displayed.

RESET - This command closes all background task logical units, and reconfigures the partitions as specified in the Initial Value Table (IVT). Maximum priority and maximum system space for the background are obtained, and stored in the background task TCB.

SET PARTITION - This command readjusts the partition boundaries of OS/32 MT R00. Any space needed to make a partition bigger is obtained from the background partition. Any space released when a partition is made smaller is given to the background partition. The Command is scanned to find the first partition specified, and the new size. The difference between existing partition size and requested partition size is computed.

If the partition is to be expanded, a check is made to see if there is enough space for the background to give to this partition. A routine is then called, specifying the partition and the amount it is to increase or decrease in size. That partition is expanded/contracted and its segmentation registers reset to the new values. All other foreground partitions, above the specified partition in memory are floated upward or downward, with their sizes unchanged, and have their segmentation registers reset to appropriate values. The background partition has its lower boundary floated up or down by the appropriate amount, and its segmentation registers reset.

If .RTL is specified, the size field of the command must be 0. This indicates that the RTL is to be deleted from the system. SPT.RTL is set to 0, and all partitions are floated downward. The background partition is increased in size the amount of space previously used for the RTL. The task common segment, if present, is also floated up.

If .TCOM is specified, a task common partition is to be created. If size specified is 0, then an existing task common is to be released. All partitions are floated up or down by the amount of memory being added/deleted from the Task Common. The background partition has its lower boundary floated up or down by the appropriate amount. If 0 was specified, SPT.TCMS is set to 0.

If .SYS is specified then the size of system space is to be changed. A check is made to see that the new size will not be smaller than the amount of system space currently in use. The amount of system space being given up or added is obtained. The upper bound of the background partition is moved up or down appropriately. No other partition is effected.

5.5 COMMAND SUBSTITUTION SYSTEM (CSS)

The Command Substitution System (CSS) is a means for the user to create catalogued but dynamically variable command input streams to perform a predetermined job. CSS consists of the preprocessor and CSS commands.

5.5.1 Calling CSS (CSSTEST)

Whenever a command is parsed, and it is determined that the command is not in the table of standard mnemonics, then it is assumed that a CSS call is being made (true only if CSS is in the system, as is this entire discussion). The mnemonic is treated as a file descriptor, and an attempt to assign the file/device is made. If the file/device does not exist, and the file descriptor did not have an extension, the extension of '.CSS' is appended, and an attempt is again made to assign it. If it does not exist, a MNEMonic error has occurred;

EXAMPLE: ABC is the command, an attempt is made to assign ABC (default system volume) and if it does not exist, ABC.CSS is assigned.

Since the Command Processor uses LUs 0-4 for its executors, CSS files are assigned starting at LU5 (level 1 = LU5, level 2 = LU6, etc.). The pointer to the current buffer is saved (in PTRSTACK) to be used by the preprocessor for parameter substitution. The address of a new buffer (for expansion) is also calculated.

5.5.2 Preprocessor/Expansion (PREPRO)

After each command line is read, it is sent to the preprocessor to be expanded. The preprocessor moves characters from the input buffer to the appropriate expansion buffer. When an "@" is encountered, CSS is alerted that parameter substitution is needed. The number of @'s are counted to determine how many levels back to go, and the parameter number is obtained. The address of the appropriate CSS call is obtained from PTRSTACK, and that call is scanned for the appropriate parameter. The parameter is moved into the expansion buffer, and then the moving of characters from the input buffer resumes.

EXAMPLE:

```
CSS call - ABC   PAR1,PAR2
FILE ABC - read -              *INPUT LINE @1
         - expands to -        *INPUT LINE PAR1

         - read -              *@1@2
         - expands to -        *PAR1PAR2

         - read -              *XX@1YY@2
         - expands to -        *XXPAR1YYPAR2
```

Whenever a request for substitution is made, and the parameter does not exist, a null is substituted.

```
EXAMPLE:  read            *@@1ABC
          expands to      *ABC
```

@0 is a special parameter. A call for substitution of @0 (or @@0, etc), will cause the file descriptor that called the appropriate CSS level to be substituted. Substitution is made of the file descriptor exactly as it appears in the call (without default system volume, or .CSS extension).

5.5.3 Additional Commands

Several additional commands are supplied to allow the user great flexibility in building CSS files and testing conditions. They are $COPY, $NOCOPY, $CLEAR, $EXIT, $JOB, $TERMJOB, $SKIP, $IFE, $IFNE, $IFG, $IFNG, $IFL, $IFNL, $IFX, $IFNX, $IFNULL, $IFNNULL, $ENDC.

$COPY and $NOCOPY - These commands turn on ($COPY) or off ($NOCOPY) the display of CSS command lines read from a CSS file. They will or will not be listed depending on whether $COPY or $NOCOPY is in effect. These 2 executors merely set/reset a flag (CSSLIST) used by MSGLOG to determine whether to print the line.

$CLEAR - This command terminates all CSS processing, closes all CSS LUs, and returns the input function to the console.

$JOB; $TERMJOB - These are used to delimit a given sequence of input as a unit. If a $JOB is in effect and any command error is detected, then all commands read are skipped until a $TERMJOB is read. $JOB merely saves the level number that the $JOB appears on. $TERMJOB resets the $JOB saved, resets any IFSKIP that is set (see below).

$SKIP - If $JOB is in effect, $SKIP causes all commands to be skipped until a $TERMJOB. It closes all CSS levels down to the level the $JOB was on.

$EXIT - This indicates that input from the current CSS level is done and that the current CSS LU should be closed. Input then begins from the next higher CSS level, or the console if there are no higher levels.

$IFE $IFNE, $IFG, $IFNG, $IFL, $IFNL - These commands pick up the value specified in the operand field of the command, and compare the current value of the return code to the value on the line. If the compare satisfies the condition specified, then the next statement is merely read. If the compare does not meet the condition then IFSKIP is set, and no statements are processed until a corresponding $ENDC is found. If IFSKIP is set, reading another $IF increments IFSKIP, each $ENDC read decrements it. When IFSKIP is 0, skipping is done.

$ENDC - Terminator of a $IF (as described above). Parsing one causes IFSKIP to be decremented.

$IFX, $IFNX - These two check to see if the file/device specified by the operand exists. If the condition specified is not met, IFSKIP is incremented. The fd is obtained, and an attempt is made to assign it. The result returned by SVC7 determines whether it exists. Success indicates it exists, but certain errors also indicate the file exists.

$IFNULL, $IFNNULL - These check to see if the parameter specified is null or not null. Since substitution has been performed, the line is scanned for the next non-blank; if it is a terminator, then the parameter was null.

5.5.4 Building CSS Files (BUILD, $BUILD)

BUILD and $BUILD are used to create a CSS file. BUILD copies input lines to the file/device specified, $BUILD performs substitution first. When a BUILD or $BUILD is encountered, an attempt is made to ALLOCATE and ASSIGN the file specified in the operand field. If it exists, it is assigned. The BUILD flag (BUILDFLG) is then set positive, for a BUILD, and negative for a $BUILD. Each time a line is read, if BUILDFLG is set, BUILDDSP is entered. If $BUILD is in effect, CSS expansion is done; if BUILD, no expansion is done. The line is now checked for the special terminator either $ENDB or ENDB, and if it is found, then the BUILDFLG is reset. If not, the line is then copied to the file/device and another line is read.

5.5.5 CSS Interaction with the Foreground and Background

A CSS file may be used to start a task running in the foreground. There is no restriction as to which task may be the current CSS task. Hence loads, assigns, starts, etc., may be performed from a CSS file, to get a real-time foreground system of tasks loaded and started. CSS may not be used to run a batch job consisting of multiple related steps in any partition other than the background. This is because CSS reads are keyed off of the background status. When the background task is active, reading from the CSS file is suspended until the background task goes inactive. This is not true of foreground. If the CSS file is being used to start a foreground task, it will continue to be read, even after the task has started.

5.6 LOAD COMMAND

This command is used to load tasks and Run Time Library segments. If loading a task, the taskid is obtained and checked for syntactic validity. The file/device to load from is assigned to LU 1. If the device/file mnemonic did not specify an extension, an attempt is first made to assign the device/file as specified. If it cannot be found, an attempt will be made to append an extension of 'TSK', and assign it again.

A check is made to see if a partition is specified. If so, the partition is checked to insure that it is currently vacant. The Command Processor puts itself into what appears to be RS state and calls the loader (SVC6.LOD1). The result is tested. If successful, the next command is executed. If not, an error message indicating the failure is logged.

If a load command specifies .BG, then the background partition is checked to see if it is dormant. If so, the load device/file is assigned and the loader is called, as above. A check is made to see that no partition is specified.

For a load command which specifies that a library segment is to be loaded, the load device/file is assigned. The amount of space available for the library segment is calculated, and a sample segmentation register, to be passed to the loader, is built. The loader is called. Upon return the actual size is obtained and a real segmentation register built and stored at SPT.RTLS. All partitions and task common are floated up or down by the difference in the size of the new and old libraries. The amount of space needed/released is added to or taken away from the background partition.

5.7 CONSOLE HANDLING

The system console device is handled by a special interrupt routine in OS/32 MT. The Command Processor always has the real console device assigned to its LU0. Tasks that try to assign a device that has the console bit set are instead assigned to the dummy device. When an I/O request is issued to this dummy device, the dummy driver is entered.

The dummy driver sets a flag to indicate to the Command Processor that I/O is being requested by a task to the console. The dummy driver also "times out" any read to the console being performed by the Command Processor, if no characters have been read.

When the Command Processor sees that I/O is being requested by a task (CMDPEND is non-zero), it performs the I/O for the task. It picks up the starting address, ending address, and function code from the dummy DCB (DCBCMD). If a read is requested, then TASKID is printed to indicate to the operator that a specific task requires input. The data is then read into the users buffer. If a write is requested, then HH:MM:SS TASKID: are written out, followed by the task's message. Any write is governed by the current value of logging. Hence, if log is set with no copy, the task's write goes only to the log device.

When the I/O has been performed the Command Processor adds an item to the System Queue for the dummy leaf (DMLV) in order to schedule the termination phase of the dummy driver. The termination phase merely calls IODONE.

5.8 THE BREAK KEY

The Break Key on the console has special meaning to OS/32 MT. It will cause any Command Processor initiated output (a display, EXAMINE, etc.) to be terminated. If a task write is in progress, it will be stopped, a "*" prompt printed and a command line will be read. After the command line has been read, the task write is retried. It may be interrupted as often as desired and will continue retrying until successful or until the task is cancelled.

If a task is in read mode (">" has been printed) and Break is depressed, a "*" is printed and a command line is read. As above, when the command is executed, the read is retried, until completion. If a task is cancelled an I/O is outstanding (a prompt has been interrupted by BREAK), I/O will be cancelled.

CHAPTER 6

FILE MANAGEMENT SYSTEM

6.1 FILE HANDLER

The routines in this package include all the logic needed to support the OS/32 MT file management system. The file handler is invoked by the SVC First Level Interrupt Handler (FLIH) any time a task issues an SVC 7 supervisor call. When an SVC 7 call is intercepted by FLIH, control is passed to the SVC 7 Second Level Interrupt Handler, SVC7. This routine then decodes each function specified by the SVC 7 parameter block and invokes the necessary executors. The SVC 7 executors contain routines to:

Allocate a new file
Assign a file or device to a logical unit
Change the access privileges of a file or device
Rename a file or device
Reprotect (change the protect keys of) a file or device
Close the assignment between a logical unit and a file or device
Delete a file
Checkpoint a file or device
Fetch the attributes associated with a file descriptor

More than one function can be performed by a single SVC 7 request. Each executor that completes successfully returns to SVC7 to determine if any other requests are still outstanding. When all functions have been processed, control is returned to the calling task via TMRSOUT. If any of the SVC 7 executors encounter an error, the appropriate error status is returned in the calling task's parameter block and control returns directly to the task via TMRSOUT. These executors make use of the following routines contained within the file handler:

A directory management package for maintaining information on all currently allocated files.

A bit map management package which provides a method for allocating and deleting files on direct-access volumes.

The file manager also contains SVC 1 intercept routines which intercept all I/O calls to a file.

6.2 VOLUME ORGANIZATION AND INITIALIZATION

Any direct-access volume to be used within an OS/32-MT environment must be formatted by the STANDALONE DISC TEST and FORMAT PROGRAM.

Since OS/32 handles file allocations in multiples of one sector, the arguments to this program must specify a DEFSEC of 1. Once a volume has been formatted using this procedure, it should not have to be formatted again unless a hardware failure occurs on the volume. After a disc is formatted, it must be INITIALIZED, using the OS/32 Disc Initializer. This utility will read-check each sector on the volume; any sector found to be defective will be marked as permanently allocated. A bit map and volume descriptor are also written on the volume.

A Volume Descriptor is shown in figure 6.1. The Volume Descriptor (VD) contains the volume name, a pointer to the bit map and first directory block, and a pointer and the size of an OS boot loadable image, if one exists.

Volume Name	Pointer to 1st Dir. Block	Pointer to OS Image	Size of OS	Pointer to bit map

FIGURE 6.1 Volume Descriptor

The size of the bit map is determined by the size of the volume; each complete bit map sector represents 2048 allocatable sectors on the volume. The final sector within the bit map represents between 1 - 2048 sectors. A sector is marked as allocated when the bit representing it is set; free when the bit is reset.

The Volume Descriptor is placed on CYLINDER 0, SECTOR 0; the bit map may be located anywhere on the volume, since it is pointed to by the VD.

6.3 DIRECTORY MANAGEMENT

A file directory is maintained as a chain of directory blocks, where each directory block contains the following fields (see Figure 6.2):

A chain field containing either a zero (indicating it is the last block in the chain) or the logical block address (sector) of the next block in the chain.

A volume that has just been INITIALIZEd contains no directory. The INITIALIZE logic sets the VD directory pointer (VD.FDP) to zero.

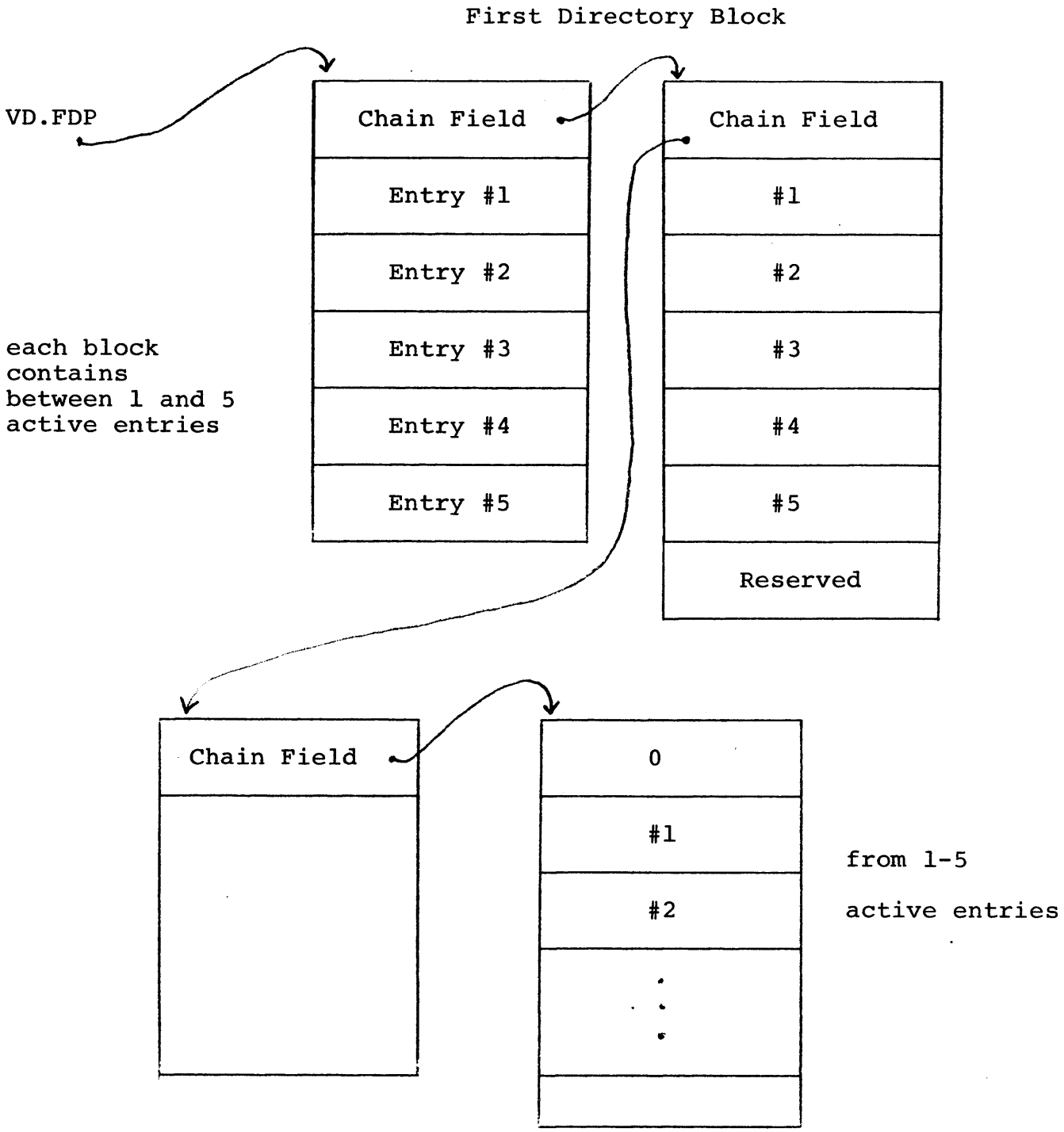


FIGURE 6.2 Directory Example

6.3.1 Directory Entry Creation and Deletion (ALLOD, RELED)

When the first file is allocated on a disc volume, a directory block is allocated. The first entry represents the new file and the remaining 4 entries are marked inactive and therefore available for additional new files. Subsequent allocations search the chain for the first unused directory entry and if none is found, a new directory block is allocated. The first directory block is always pointed to by the VD. If a file is deleted, its entry is marked inactive. If all entries in a directory block are marked inactive, the directory block is released and the chain relinked. If all the directory blocks are thus released, the VD.FDP field is again set to zero.

6.3.2 Directory Access (DIRLOOK, GETD, PUTD)

When a function is requested on a currently existing file, the directory block containing the directory entry (DIR) for the file must first be found via a call to DIRLOOK. The I/O routines used to read directory blocks into memory or to write out modified blocks are GETD and PUTD.

When a new file is allocated, and one or more directory blocks currently exist, the routine DIRLOOK searches each block until an inactive entry is found. If all entries are marked active, a new directory block is allocated as described above.

6.4 BIT MAP MANAGEMENT

OS/32 direct access files are allocated in multiples of one sector; the status (free or allocated) of each sector on the volume is maintained in the volume's bit map. When a volume is INITIALIZED, all non-defective sectors within the volume are marked as free by resetting the corresponding bit in the bit map. Then the VD and bit map are created; the sectors they occupy are marked as allocated by setting the appropriate bits. The INITIALIZE logic also provides a pointer from the VD to the bit map (VD.MAP).

6.4.1 File Allocation and Deletion (GETSECTR, RELEB, GETB, PUTB)

When a request is received by the bit map management routines to allocate a string of contiguous sectors, GETSECTR searches the bit map for a corresponding number of bits that are reset, thus indicating available sectors. Since allocations may span bit map sector boundaries, one or more calls to GETB may be required to read bit map sectors into memory. When enough available sectors have been found in this manner, GETSECTR then sets each bit in the bit map within this allocation. As bit map sectors are modified, they are written back to disc via PUTB.

When a file is deleted, the procedure is reversed by RELEB. Each bit representing the allocation is reset, indicating the sector is again available. GETB and PUTB may again be invoked, to read and write the bit map sectors.

6.5 SVC 7 SECOND LEVEL INTERRUPT HANDLER (SVC7)

The FLIH transfers control to the SVC 7 driver routine, SVC7, with two arguments, the address of the current TCB and the address of the calling task's SVC 7 parameter block.

SVC7 processes the function code specified by the parameter block from left to right. If the function code is initially zero, the call is a FETCH Attributes. Otherwise, the function code is saved in TCB.SYS, and each SVC 7 function specified within it is performed by branching to the appropriate executor. Each executor that completes successfully returns control to SVC7. As each function is performed, the bit representing it in TCB.SYS is reset, until each bit of TCB.SYS has been reset. Control then returns to the calling task via a branch to the Task Management routine TMRSOUT.

6.6 SVC 7 FUNCTION EXECUTORS

6.6.1 Allocate (ALLO)

The SVC 7 executor ALLO is called directly from SVC7 when the function code in the parameter block specifies an allocate operation.

The logic in ALLO proceeds as follows: The directory management routines are called to insure that the specified file descriptor is unique to that file, and establishes a directory entry for the file being allocated. For a contiguous type file, the complete file allocation size is established at allocation time; this is performed by the bit map management routines. Since a chain file is open-ended and has no predefined size, no allocations are performed on behalf of a chain file at allocation time. The necessary initial information is established in the directory for both file types. Control returns directly to SVC7 upon the successful completion of ALLO.

6.6.2 Assign (OPEN, OPEN.DEV, OPEN.CO, OPEN.CH)

The SVC 7 executor OPEN performs all common assign processing for direct and non-direct access devices. OPEN establishes the validity of the logical unit being assigned. If the OPEN function is being performed upon a non-direct access device,

OPEN transfers control to OPEN.DEV which completes any necessary validity checks, using the subroutines described in Section 6.6.10, and sets up the entry in the LU table to contain the DCB address and device attributes. If the device being opened is a direct access device, OPEN completes the assignment process itself and places the DCB address and device attributes in the LU table. Otherwise, control is transferred to OPEN.CO, if the file being assigned is a contiguous file, or OPEN.CH, if it is a chain file. In either case, the first event to occur is a call to the memory management routine, GETFCB, to allocate a file control block (FCB) within dynamic system space.

OPEN.CO and OPEN.CH also obtain current control information about the file from its directory entry and move this to the FCB.

OPEN.CH positions the file to the requested data block and allocates a new block for data via the bit map management routines, if the file is being opened with write privileges.

Upon successful completion, both OPEN.CO and OPEN.CH set up the entry in the LU table to contain the FCB address and a file attribute byte, which indicates the allowable data transfers to the file.

All OPEN processing successfully terminates by returning directly to SVC7.

6.6.3 Change Access Privileges (CAP)

The SVC 7 executor, CAP, performs the function of changing the access privileges associated with a given logical unit. The logical unit can be assigned to a file or device. CAP is a two pass operation.

On pass one, the routine insures that the new access privileges are legal, but makes no modifications to any control blocks. When pass one has completed successfully, the routine proceeds to pass two, this time making updates to all required control blocks to reflect the new access privileges. This requires modifying the write and read count fields in the DCB or FCB (DCB.WCNT, DCB.RCNT, FCB.WCNT, FCB.RCNT) to reflect the current access privileges. The current access privileges associated with a file are reflected in the WCNT and RCNT fields of the control block in the following manner:

```
WCNT/RCNT = 0  implies no task having write/read privileges
WCNT/RCNT = -1 implies one task having exclusive write/read
               privileges
WCNT/RCNT = +n implies n tasks sharing write/read privileges
```

6.6.4 Rename (RENAME)

RENAME is the SVC 7 executor that changes the name of a file or device. If the rename function is directed at a device, RENAME insures that the new name does not currently exist in the Device Mnemonic Table (DMT) and then replaces the device's previous name in the DMT with its new name. To RENAME a file, the procedure is similar except it is the directory that is checked for a duplicate name. The directory management routines are used to read the directory, search for a name match, and rewrite it with the new file name. RENAME returns to SVC7 upon successful completion.

6.6.5 Reprotect (REPRO)

The SVC 7 executor, REPRO, contains the logic to modify the protect keys associated with a given LU. The LU can be assigned to a file or device. The protect keys associated with a device are kept in its DCB (DCB.WKEY, DCB.RKEY); the protect keys associated with a file are kept in its directory entry (DIR.WKEY, DIR.RKEY). A file or device may be unconditionally protected (Keys = X'FF'), unconditionally unprotected (Keys = X'00') or conditionally protected with Write, Read Keys between X'01' and X'FE'. The logic in REPRO insures that the new protect keys are not in violation of the former protect keys, and updates the control block (DCB or Directory) with the new protect keys. Control returns to SVC7 for further SVC 7 processing.

6.6.6 Close (CLOSE)

The purpose of the CLOSE executor is to disconnect an open logical unit from a file or device. The logic of CLOSE insures that the given LU is currently assigned.

If the LU was assigned to a device, the read and write count fields in the DCB are modified as follows:

(1) if old WCNT,RCNT = -1, new WCNT,RCNT = 0
(previously one exclusive user)

(2) if old WCNT,RCNT = 0, new WCNT,RCNT = 0
(implies there were no users of this privilege)

(3) if old WCNT,RCNT = n, n 0, new WCNT,RCNT = n-1
(previously n shared users)

If the LU is assigned to a file the WCNT,RCNT fields in the directory and FCB are updated as specified above. A test is then made to determine if the FCB should be released or if

it is being shared. (Current file implementations preclude the possibility of sharing FCB's. The logic is included in CLOSE for future use). If the FCB is not being shared, its memory allocation is returned to system space by a call to the memory management routine, RELEFCB. The directory management routines are then used to update the directory with the information about the file which was in the FCB. CLOSE finishes by setting this LU's entry in the LU table to zero and exiting to SVC7.

6.6.7 Delete (DELETE)

The DELETE executor is used to delete contiguous and chain files; it has no meaning with regard to devices and will generate an error if an attempt is made to delete a device. The logic in delete requires dynamic system space; this is obtained by the memory management routine GETFCB. A file is deleted by releasing its allocated storage on the volume containing it via the bit map management routines and by relinquishing its directory entry via the directory management routines. Finally, the system space is returned via a call to the memory management routine RELEFCB. DELETE returns control to SVC7 on completion.

6.6.8 Checkpoint (CHECKPT)

The SVC 7 executor CHECKPT contains the logic to checkpoint to an LU; the LU may be assigned to a file or a device. If the checkpoint function is directed to a device, a subroutine is invoked to perform an SVC 1 I/O wait only operation on the device. To checkpoint a file, all current information about the file is moved from its FCB to its directory entry. The bit map and directory management routines are used to insure that the bit map and directory on the volume reflect the current file allocations. A chain file is also positioned to read random mode using the chain file reset routine, RESET.CH, described in 6.7.2. CHECKPT exits to SVC7.

6.6.9 Fetch Attributes (FETCH)

The purpose of the FETCH executor is to obtain the attributes associated with the file or device assigned to a given LU. The device/file attributes, device code, and name are moved from the DCB/FCB to the task's SVC 7 parameter block. FETCH returns directly to the calling task via the task management routine, TMRSOUT.

6.6.10 SVC 7 Integrity Checking Subroutines

This section briefly describes the integrity checking subroutines used by the SVC 7 executors:

(1) APCHECK - Verifies the legality of the requested access privileges; converts the requested access privilege to a numeric quantity, to be saved in the WCNT and RCNT field of the control block.

(2) LUCHECK - determines if a given LU is assigned, and picks up its LU entry from the LU table.

(3) DMTLOOK/VMTLOOK - searches the DMT/VMT for a given device/volume.

(4) FDCHECK - checks the syntax of a given file or volume name.

6.7 SVC 1 INTERCEPT ROUTINES

When the SVC 1 processor (SVC1) determines that an SVC 1 call is directed to a file, the file management SVC 1 intercept routines are entered to process the request.

6.7.1 Contiguous File Handler

The contiguous file handler package consists of the following two routines:

CONTIG - processes data transfer requests to a contiguous file
CMD.CO - processes command function requests to a contiguous file

6.7.1.1 Data Transfer for Contiguous Files (CONTIG)

The routine CONTIG is entered directly from SVC1; the length of the data transfer request is computed and the random address is obtained either from the FCB random address (for a random I/O request) or from the FCB current sector pointer (for a sequential I/O request). CONTIG copies the FCB information into the DCB and if the I/O request is a read or write, CONTIG exits by transferring control directly to the disc driver. If the I/O request is a test and set (both read and write bits set in the SVC 1 function code), CONTIG enters RSA state, moving the RS save area to a save area in the FCB via TMRSRSA, and then modifies the following fields in the TCB:

TCB.RPSW - the location field of the resume PSW is set up to contain a secondary entry point within CONTIG

TCB.RGPR - the general purpose register save area is set up to contain the current values of user register set.

The routine then obtains control of the directory leaf to ensure Test and Set as an indivisiable operation and then transfers control to the disc driver for the read portion of the test and set operation. By modifying the TCB.RPSW and TCB.RGPR fields as specified above, upon termination of the read the disc driver returns to CONTIG at its secondary entry point. CONTIG then processes the remainder of the test and set operation itself. If a write is to be performed (the first halfword of the buffer read contained a X'0000'), CONTIG does the write by issuing an SVC 1 WRITE, WAIT call. The Directory Leaf is released and control is returned to the calling task upon successful completion of CONTIG via the task management routine, TMRSAOUT. If CONTIG receives an EOM status following an I/O operation, EOM status is saved in the FCB and control is returned to the task by branching to IODONE2 to complete the request.

6.7.1.2 Command Requests to Contiguous Files (CMD.CO)

The command function intercept routine for contiguous files, CMD.CO, contains six command executors. Each executor and its function is briefly described below:

Rewind (CMD.REW) - Set current sector (FCB.CSEC) in the FCB to 0 and return via a branch to IODONE2.

Backspace Record (CMD.BSR) - Decrement FCB.CSEC by 1, enter RSA state and issue an SVC 1 read of new current sector to check for any I/O problems. Exit to TMRSAOUT.

Forward Space Record (CMD.FSR) - Increment FCB.CSEC by 1 and proceed as in CMD.BSR.

Write End of File (CMD.WEOF) - Increment FCB.CSEC by 1, enter RSA state and write a pseudo-file mark (X'1313') at that random address via an SVC 1 WRITE,WAIT call. Exit via TMRSAOUT.

Forward Space File (CMD.FSF) - Enter RSA state and issue SVC 1 read commands starting at FCB.CSEC, until a pseudo-file mark, X'1313' is found. Exit to TMRSAOUT.

Backward Space File (CMD.BSF) - Same as Forward Space File, except the X'1313' is searched for starting at FCB.CSEC and backing up one sector at a time. Exit to TMRSAOUT.

6.7.2 Chain File Handler (CHAIN, CMD.CH)

The chain file handler consists of the following two routines, CHAIN and CMD.CH, and various subroutines, described in 6.6.2.1. The purpose of each is:

CHAIN - process data transfer requests to a chain file

CMD.CH - process commands to a chain file

6.7.2.1 Chain File Handler Subroutines

The Chain File Handler requires various subroutines in order to process chain files. Each is briefly described below:

POSITN - position the chain file to a specific block and record beginning within that block. The current position of a chain file is indicated by the value of the FCB.CBLK.

GETCHL - move logical record from a system buffer to the task's buffer. If the logical record spans more than 1 physical block, a call is made to GETCHPR, to read the next block into a system buffer.

PUTCHL - move logical record from task's buffer to system buffer. If a logical record spans physical blocks, the block is written via a call to PUTCHP.

GETCHPR, GETCHPL - perform physical reads to a chain file to the right (entry point GETCHPR) or to the left (entry point GETCHPL). If the file is currently in sequential mode, double buffering is used; in random mode, single buffering is used.

PUTCHP - perform physical writes to a chain file. If the file is in sequential mode, the write logic uses double buffering; in random mode, single buffering is used. If the file is in write sequential mode, a new block of sectors is preallocated at this time, via a call to the bit management routine, GETSECTR. If an EOM status on the disc is returned from GETSECTR during PUTCHP processing, the file is returned to a known state by writing out the current buffer and backing up until the last logical record within the file ends in the current block.

CHDIR - establish the direction in which a chain file is to be processed (right or left).

RESET.CH - change the current state of a chain file. At any point in time, the contents of the FCB.FLGS field indicate the state of the chain file, where a state is defined as being one of the following:

Operation	Processing Mode
Read	Sequential
Read	Random
Write	Sequential
Write	Random

Therefore, there are sixteen possible state changes a file may undergo, where four of these are no-ops. RESET.CH performs whatever functions are required to change a file from one state to another.

6.7.2.2 Data Transfer for Chain Files (CHAIN)

The routine CHAIN is entered directly from SVC1 in RSA state. The routine determines the state the file should be processed in based upon the function code within the FCB. EOM status is generated if a Read at the end of the file or a Write beyond the end of the file is attempted. The current state of the file is established via a call to RESET.CH. Control is transferred directly to either GETCHL or PUTCHL, to perform the logical I/O operation.

6.7.2.3 Command Requests For Files (CMD.CH)

The command function intercept routine for chain files contains 5 executors to perform the 5 allowable commands to a chain file. These executors all make use of the chain file subroutines described in Section 6.7.2.1. The following is a brief description of each executor:

Rewind (CCH.REW) - The first block in the file (block 0) is positioned to by a call to POSITN, and the current logical record field in the FCB (FCB.CLRL) is set to zero.

Backspace Record (CCH.BSR) - The previous record in the file is positioned to by decrementing the FCB.CLRL field by 1 and then positioning to the block containing this record via a call to POSITN.

Forwardspace Record (CCH.FSR) - The logic of CCH.FSR is to increment the FCB.CLRL by 1 and proceed as CCH.BSR.

Forwardspace File (CCH.FSF) - The last block in the file is positioned to (FCB.NBLK -1) via POSITN.

Backward Space File (CCH.BSF) - identical to CCH.REW.

All the executors return to the calling task via TMRSAOUT since entry to CMD.CHN is in RSA state.

6.7.2.4 Error Recovery For Chain Files

I/O errors may occur during the processing of an SVC 1 Data Transfer or command request to a chain file. An End-of-Media status (X'90') is a software generated status that means an attempt to write to a chain file could not be satisfied because no more allocatable space exists on the direct-access volume. The file is then 'closed' in the sense that the last block in the file is written with a proper link field. The user can then continue to process the file in any way (i.e., Close, Delete, etc.) or, after making more direct-access space available on the volume, continue writing to the file.

Any other type of I/O error is caused by some hardware problem and may require user intervention to correct. If the user was updating an existing logical record within a file that has been closed or checkpointed, and an I/O error occurs, the file may be closed, the error corrected, and processing of that file may resume. If however the file was being processed in any manner and had not been previously closed or checkpointed, some link fields may not be set properly which causes the file to be unusable. The action taken by the user in this case should be to execute the Disc Integrity Checker Utility Program (program # 03-080) before continuing to process the file. Failure to do so may result in other files being inadvertently destroyed if the user attempts to process this file.

CHAPTER 7

DRIVER DESCRIPTION

7.1 DRIVERS

Each driver consists of three phases: Initialization, Interrupt Service and Termination (or Event Service). The Initiation phase runs as a reentrant subroutine (interrupts are enabled) of the task issuing the I/O request. In general, the initiation phase uses the information stored in the DCB by the SVC 1 executor to prepare the device dependent information required to execute the required function. After all processing has been done, the Initiation phase starts the physical I/O process by causing an interrupt on the device requested. The Initiation phase then enters the task manager which returns control to the calling task on an I/O and proceed call, or puts the calling task into I/O wait on an I/O and wait call.

When an interrupt is detected from the device, the microcode causes control to pass to the Auto Driver Channel or to the Interrupt Service Phase. If the Auto Driver Channel is employed, end of buffer or error conditions cause control to be passed to the Interrupt Service phase of the driver specified. The Interrupt service phase of the drivers execute with all interrupts disabled(except for Machine Malfunction). This phase controls the actual I/O to the device, either in conjunction with the Auto Driver Channel or by I/O instructions. Error conditions cause status to be set in the DCB. On completion of the I/O, the Interrupt Service phase disables interrupts from the device and adds the address of the device's leaf (EVT entry) to the system queue.

When a PSW is loaded that has queue service interrupts enabled, the System Queue Service routine (SQS) is entered by the microcode. SQS removes the address of the device's leaf from the system queue and schedules the termination phase of the driver, as specified in the leaf. The scheduling of the termination phase is called an event. The Termination phase (or event service routine - ESR) of the driver executes as an asynchronous, reentrant, non-eventable subroutine of the task which requested the I/O. The termination phase is asynchronous because it is scheduled as the result of a queue service interrupt. If the calling task is executing (or about to execute) at the time the Termination phase of the driver is scheduled, the state of the task is saved in the TCB until the ESR is complete. The ESR executes with all interrupts enabled, so it is reentrant. Non-eventable means that if another queue service interrupt occurs for the calling task while the ESR is executing, the second ESR will be queued by the System Queue Service handler for scheduling when the first ESR completes.

The ESR performs post-processing on the I/O performed and either schedules another ISR and enters the Event Service Handler which passes control back to the calling task or schedules a queued ESR, or the ESR enters IODONE to complete the I/O request.

The Executive routine, IODONE, performs common post-processing for all drivers. It passes status and length of transfer from the DCB to the SVC 1 parameter block, calls Event Service Routines to disconnect the task from all EVT entries that were necessary to coordinate the I/O request, resets the ISP table entry for the device so that subsequent interrupts will not cause entry to the driver, removes the I/O wait condition from the task, if necessary, and enters the Event Service Handler to return control to the task or to schedule a queued ESR. IODONE also causes a task trap if a proceed I/O is being completed, and the task has I/O proceed traps enabled.

It is the responsibility of the Executive to schedule driver routines in the proper state - Initiation Routines in RS, Interrupt Service Routines in IS, and Event Service Routines in ES State. It is the responsibility of the Driver Initialization and Event Service Routines to enter and exit from NSU state via LPSW, LPSWR or EPSR instructions if necessary.

7.2 DRIVER CONTROL BLOCKS

7.2.1 Device Control Block (DCB)

All standard drivers make use of the device independent portion of the DCB (see Figure 11-2). The DCB is used to pass information between the executive and the drivers; it is also used by the drivers, SVC 1 and SVC 7 to control I/O requests. The use of the Event Service Handler allows a driver to assume exclusive access to a DCB for the duration of an I/O request to the device associated with that DCB. The following section describes each field in the DCB and its usage:

DCB.DMT - Address of Device Mnemonic Table entry for this DCB. Established by the Configuration Utility Program. Used by File Manager at assign time.

DCB.LEAF - Address of Event Coordination Table entry for device described by the DCB. Established by the Configuration Utility Program. Used by the SVC 1 executor to establish task connection to the required EVT entries before passing control the the driver.

DCB.INIT - Driver entry point for data transfer requests. Established at DCB assembly time by referencing the data transfer entry in the driver initialization routine. This entry point must have a name of the form INITxxxx where xxxx designates the driver. This address is used by the SVC 1 executor to enter the driver for data transfer requests.

DCB.FUNC - Driver entry point for command function requests. Established at DCB assembly time by referencing the command function entry in the driver initialization routine. This entry point must have a name of the form CMDxxxx where xxxx designates the driver. This address is used by the SVC 1 executor to enter the driver for command function requests.

DCB.TERM - Driver entry point for first Event Service Routine to be scheduled. Established at DCB assembly time by referencing the entry address of the desired Event Service Routine. This address is placed in the device EVT entry (leaf) at connection time (SVC 1 executor).

DCB.WCNT; DCB.RCNT - Read and Write count fields used by the File Manager to control access at assign time.

DCB.ATRB - Attributes of device. Used by File Manager to determine the attributes to associate with the device at assign time. The File Manager copies the attributes to the Logical Unit being assigned, possibly resetting the read or write bit. The Logical Unit is a field in the Task Control Block. The SVC 1 executor uses this copy of the attributes to determine the validity of an I/O request. The attribute bits are defined in Figure 7-2.

BIT	ATTRIBUTE
0	RESERVED
1	SUPPORTS READ
2	SUPPORTS WRITE
3	SUPPORTS BINARY
4	SUPPORTS WAIT I/O
5	SUPPORTS RANDOM
6	SUPPORTS UNCONDITIONAL PROCEED
7	SUPPORTS IMAGE
8	RESERVED
9	SUPPORTS REWIND
10	SUPPORTS BACKSPACE RECORD
11	SUPPORTS FORWARD SPACE RECORD
12	SUPPORTS WRITE FILEMARK
13	SUPPORTS FORWARD SPACE FILEMARK
14	SUPPORTS BACKSPACE FILEMARK
15	RESERVED

FIGURE 7-2 DCB Attribute Bit Definitions

DCB.RECL - This field defines the maximum length of a record for the device. Established at DCB assembly time. Used by the driver to truncate requests larger than maximum.

DCB.TOUT - Time-out constant. Established by Interrupt Service Routines to indicate desired treatment by the executive or the Timer Routines. The value of the timeout constant is defined as follows:

> -1 (X'FFFF') means the I/O request is in the process of normal termination by the driver. An ESR has been scheduled for the I/O request.
>
> 0 (X'0000') means the driver should abnormally terminate the I/O request. An ESR has been scheduled for the I/O request.
>
> 2^{15}-1 (X'7FFF') means the I/O request is not to be timed out by a timer interrupt.
>
> 1 through X'7FFE' means the timeout constant is to be decremented by 1 every second by the system clock until value is zero.

A driver's initiation phase must put the DCB on the driver timeout chain, and its termination phase remove it. This is accomplished via calls to TOCHON, and TOCHOFF respectively.

An ISR normally sets the timeout constant to the appropriate value for the device and request. After the timeout constant has been set to a positive value, all subsequent ISRs and ESRs check for timeout constant = 0. If the request has been timed out, the appropriate status is placed in the DCB, an ESR is scheduled if necessary, and the timeout constant is set to -1.

DCB.RTRY - Retry count. Established by driver if necessary. Used by standard drivers to control number of error retries.

DCB.FLGS - Flag bytes used to describe various characteristics of the device to the File Manager, the Executive and to drivers. The flag bits are defined in Figure 7-3.

Bit	Name	Meaning and Usage
0	Bulk device flag	Set at DCB assembly time, used by File Manager.
1	On-line flag	Set at DCB assembly time, Modified by Command Processor. Used by File Manager at assign time.
2	Directory Presence	Set by system initialization routine. Used by File Manager for directory processing on device. Bulk Device Flag must also be set.
3	Bit map Presence bit	Similar to Directory presence bit.
4	Check Pseudo File Mark flag	Set by the File Manager. designates to the disc driver to check for pseudo file mark.
5	Bit Map Modify flag	Set and modified by File Manager. Indicates Bit Map must be updated on device. Bulk device flag must also be set.
6	Console flag	Device is the console device. Set by System Initialization Routine.
7	Uncancellable flag	Set at DCB assembly time. Designates to Executive not to timeout I/O to this device on End of Task.
8	SVC 6 Connectable	Set at DCB assembly time. Indicates this device may only be used by SVC 6 connect, freeze, thaw, SINT, and break. Any device accessed by these calls must have this bit set.
9	Write Protect Bit	Set by Command Processor. Indicates the device is "write protected". All assigns are turned into SRO.
10-15	Reserved	Must be 0

FIGURE 7-3 DCB Flag Definitions

DCB.STAT - Status field. Set by driver to indicate status of I/O request. SVC 1 executor sets value to zero before passing control to driver. Executive routine, IODONE, copies this field to status field in SVC 1 parameter block on completion of I/O request.

DCB.DCOD - Device Code. Established at DCB assembly time. Must correspond to nnn in name of DCB module (DCBnnn).

DCB.WKEY; DCB.RKEY - File protect keys. Established by File Manager Reprotect function. Used by file manager to control access at assign time.

DCB.PBLK - Parameter Block Address. Established by SVC 1 executor. Contains the relocated physical address of the SVC 1 parameter block for current I/O request. Drivers generally do not use this address.

DCB.FC - Function Code. Established by SVC 1 executor. Used by driver initialization routine to determine nature of I/O request. Subsequent Wait only request may modify this field, therefore, ISRs and ESRs should not depend on contents.

DCB.LU - Logical Unit of current I/O request. Established by SVC 1 executor. Used by File Manager and SVC 1 executor. Must not be modified by driver.

DCB.DN - Device number. Established by Configuration Utility Program. Used by drivers to determine physical device to perform I/O request.

DCB.SADR; DCB.EADR - Data transfer start and end addresses. Established by SVC 1 executor for data transfer requests. Contains the relocated physical addresses. Used by drivers to define buffer for request.

DCB.RAND - Data transfer random address. Established by SVC 1 executor.

DCB.LLXF - Length of last transfer. Established by driver. This value is copied to length of last transfer field of SVC 1 parameter block by the Executive routine IODONE for data transfer requests.

DCB.TOCH - Time-out chain. This contains the address of the next DCB on the time-out chain, or 0 if this is the last device on the chain.

DCB.UPBK - Unrelocated parameter block address. Used by IODONE. This is passed to SV9.ATQ when an I/O proceed completes, so that it may be queued to tasks that have the I/O proceed queue bit enabled.

7.2.2 Channel Control Block (CCB)

The Channel Control Block is used to control Auto Driver Channel operations. The address of the CCB+1 (to make it odd) is placed in the ISP table entry for a device before any serviceable interrupts are generated. The CCB must reside in the first 64K of memory since the ISP table entry must contain a halfword entry. The following section described each field in the CCB. Refer to Figure 11-1.

CCB.CCW - Channel Command Word. Established and modified by the driver before enabling interrupts on the device. Used by Auto Driver Channel to control I/O request. The CCW bits are defined in Figure 7-5.

BITS	MEANING	
0-7	Status mask	
8	Execute Bit	
12	Buffer bit.	Zero value selects buffer 0, One value selects buffer 1 if Fast bit reset.
13	Write bit	When reset, indicates read operation.
14	Translate bit	Specifies translation if Fast bit reset.
15	Fast bit	Specifies no translate, no buffer switch, no redundance check.

FIGURE 7-5 CCW Bit Definitions

CCB.LB0 - Length of buffer 0. Used to specify length of data pointed to by buffer 0. Length is expressed as a negative number whose value is equal to start address minus end address. Thus, at any time, the length added to the ending address gives the next character to be processed.

CCB.EB0 - End Address of Buffer 0. Last character to be processed by Auto Driver Channel. Established by driver.

CCB.CW - Check Word. Used by Auto Driver Channel to accumulate redundancy check. Not used by standard drivers.

CCB.LB1 - Length of Buffer 1 (See CCB.LB0). Established by driver.

CCB.LB1 - End Address of buffer 1. Established by driver.

CCB.XLT - Translation table address. Specifies the translation table to be used by Auto Driver Channel when CCB.CCW flag bit 14 is set. Established at CCB assembly time by referencing the translation table address in the driver module.

CCB.SUBA - Subroutine address. Specifies an ISR entry point which is branched to by the Auto Driver Channel in the following cases:

Execute bit (CCB.CCW bit 8) is reset
End of Buffer Condition
Error condition detected

Since this is a halfword field, the ISR entry point must exist in the first 64K of memory. Established by the driver.

CCB.MISC - Miscellaneous field. Established and used by drivers to pass information between Initiation, Interrupt Service and Termination phases.

CCB.FLGS - Established and used by drivers to pass information between Initiation, Interrupt Service and Termination phases.

CCB.DCB - Address of DCB for device being controlled by CCB. Established at CCB assembly time by referencing the DCB name.

7.3 DRIVER INITIALIZATION ROUTINES

Each Driver Initiation Phase has two entry points: data transfer request (INIT) entry and command request (FUNC) entry. These entry points are named INITxxxx and CMDxxxx, where xxxx designates the driver. At driver assembly time, these entry point addresses are coded in each DCB the driver controls.

Driver Initialization Routines (DIR) execute in Reentrant System (RS) state, thus executing as reentrant sub-routines of the calling task. The user task's registers and resume PSW are stored in the TCB RS save area by the SVC 1 executor. The user task is connected to the Event Coordination Table entries corresponding to the peripherals required for the I/O request. This insures that no other I/O requests can be initiated to the device until the driver requests the task's disconnection. Register 13 of the user register set contains the address of the SVC 1 parameter block, function code, logical unit, physical start and end addresses and the random address as required by the function code in the DCB.

The DIR performs the preprocessing necessary to translate the device independent SVC 1 parameter block quantities into the device dependent information to be used by the ISR and ESR portions of the Driver or by the Auto Driver Channel (see 32 bit Series Reference Manual 29-365). After preprocessing, the DIR modifies the Interrupt Service Pointer Table entries for the devices required to point to the proper CCB or ISR, call TOCHON to put the DCB on the driver time-out chain, (if necessary), and then issues a Simulate Interrupt instruction on the device address. The Driver Initialization Routine then exits to the Task Management routine TMRSOUT. This routine returns control to the calling task following the SVC 1 if the call is for I/O and proceed or it places the calling task into I/O wait state if the call is for I/O and wait.

The DIR may determine that I/O to the device is not necessary due to an error condition or because of the nature of the request. In this case, no ISR will execute. In order to terminate the I/O request, the driver does one of two things:

1. Exits to executive routine IODONE at the alternate entry IODONE2.

2. Schedules an ESR by adding the address of the leaf contained in the DCB (DCB.LEAF) to the top of the system queue.

7.4 INTERRUPT SERVICE ROUTINES

Interrupt Service Routines (ISR) execute in the Interrupt Service (IS) state. They are entered as the result of an interrupt on a device involved in the I/O request. On entry, registers 0 and 1 of the executive register set contain the resume PSW for the program that was executing at the time the interrupt was serviced. Register 2 contains the device number of the interrupting device. In the case of drivers which employ the Auto Driver Channel, Register 4 contains address of the CCB which is controlling the Auto Driver Channel. ISRs may use Registers 2 through 7 of the executive register set.

In general, all I/O instructions (e.g., SS, RD, WB) are issued from ISRs. The Auto Driver Channel is used both to perform I/O requests through the appropriate Channel Command Word and to simply transfer control to an ISR, as would Interrupt Driven I/O but with the addition of the CCB pointer in register 4. An ISR always exits by loading the PSW in Registers 0 and 1. An ISR may place another ISR entry in the Interrupt Service Pointer Table or CCB to process the next interrupt. If the ISR detects that the I/O request is

complete or that some portion of the I/O request is complete (e.g., SEEK complete on a disc I/O request), it disarms the device to prevent further interrupts. The ISR then schedules the Event Service routine pointed to by the leaf (EVT entry) for the device by placing the address of the leaf on the top of the system queue with an ATL instruction. It is the responsibility of the driver to insure that the ESR address contained in the leaf is the proper address before adding the leaf address to the system queue. The address in the leaf is initially set by the SVC 1 executor to the value contained in the DCB (DCB.TERM). If the driver determines that some other ESR should be scheduled, it modifies the ESR address contained in the leaf by calling the Event Service Handler routine EVMOD with the address of the new ESR in Register 14 and the leaf address in Register 15.

If an ISR detects an error condition it sets the Status field of the DCB to the appropriate value (see respective driver program descriptions). If Auto Driver Channel translation is employed, the translation subroutines are ISRs. The ISR is responsible for setting up DCB.TOUT for any I/O operation on the device that he wishes timed.

7.5 EVENT SERVICE ROUTINES

Event Service Routines are scheduled in the Event Service (ES) state by the Task Manager, as a result of a System Queue Service interrupt. All interrupts are enabled and the user register set is used. On entry, the registers and PSW of the task which initiated the I/O request are saved in the Task Control Block, register 13 contains the address of the DCB and register 15 contains the address of the leaf corresponding to the device. ESRs can use registers 0 through 15 of the user register set.

ESRs perform post-processing on the I/O request being terminated, such as calculating length of last transfer, or they process intermediate I/O events in the case where the request requires more than one I/O sequence to complete (e.g., a seek and then a data transfer is required to complete a DISC read). If additional ISRs are required, the ESR may modify the CCB to schedule a different ISR, or change the address in the leaf to schedule a different ESR on completion of the ISR. If further I/O must be initiated, the ESR causes an interrupt on the device and exits by branching to the Event Service Handler routine EVRTE (return from Event). If I/O request is complete, the ESR exits to the executive routine IODONE with DCB address in register 13 and leaf address in register 15. The ESR is responsible for removing a DCB from the timer chain, via a call to TOCHOFF.

7.6 DRIVER TIMEOUT

Each driver is responsible for putting a DCB on the timeout chain, and for removing it. To put a DCB on the chain, the initiation phase of the driver must call TOCHON. To remove the DCB, the termination phase must call TOCHOFF. When the DCB is added, the timeout constant (DCB.TOUT) should be set to -1. The first ISR has the responsibility for setting the timeout constant to the appropriate value. A value of -1 means the DCB has timed out, X'7FFF' means that the DCB is not to be timed out. Any other value is considered to be in interval (in seconds) after which the transfer is to be timed out.

7.7 HALT I/O ROUTINE (TIMEOUT)

At certain times it is necessary to cancel I/O requests that have already been started, such as in cancel processing. This is accomplished by a pseudo timeout facility in OS/32 MT. In order to halt I/O that is in progress, TIMEOUT is called from IS state with the address of the leaf corresponding to the device. TIMEOUT loads the DCB pointer from the leaf and returns if the pointer is zero (as in the case of the dummy leaf - see Section 5.7). If the DCB pointer is non-zero, TIMEOUT checks the timeout constant in the DCB. If it is zero or negative, an event service routine has already been scheduled for this request and the routine returns to the caller. If the timeout constant in the DCB is positive, TIMEOUT checks the value of the last entry in the system queue, since a power fail/restore sequence may have interrupted a driver ISR in between adding the leaf address to the system queue and setting the DCB timeout constant to -1. If the address of the leaf is not the last entry in the system queue, TIMEOUT adds it to the top of the queue, thus scheduling a termination routine for that request.

CHAPTER 8

SYSTEM FLOW EXAMPLES

8.1 System Start Up

Figure 8-1 illustrates system flow during initialization of the system, loading and starting a task. At location X'60' is a branch to the first location of the SPT which contains a branch to SYSINIT. SYSINIT initializes the ISP Table, the EVT, DCBs and TCBs. The system TCB is placed on the ready chain. All processing is performed with all interrupts disabled. The command processor is branched to in ET state. The command processor initializes all its internal flags and buffers and uses SVC 1 to display the OS ID on the system console. It then issues a write image SVC 1 to prompt with an * and issues an SVC 1 read and wait to the system console. The system task enters I/O wait state, placing the processor in a system wait state. When the load command is entered, the teletype driver ESR is scheduled by the task manager, the I/O is completed and the command processor resumes processing after the SVC 1. The command processor decodes the command and branches to the resident loader. The resident loader calls TMRSAIN to enter RSA state. It calls EVQCON to connect to the loader leaf. The loader reads the LIB via issuing SVC 1 calls. It now reads the task image into memory, calls TMRSAOUT to leave RSA state and EVDIS to release the loader leaf. Control is returned to the Command Processor. SVC 1 is used to write the prompt, followed by SVC 1 read and wait to the system console. The system task enters I/O wait state. The START command causes the task manager to schedule the teletype ESR, the I/O is completed and the command processor resumes execution following the SVC 1. The command is decoded and the command processor branches to the executive routine TMSTART with the start address. TMSTART constructs a start PSW in the dispatch PSW save area of the background TCB, calls TMCHN to put the TCB on the ready chain behind the system task and returns to the command processor. The command processor tests the background TCB to see if it is dormant and since it is not, no prompt is written to the console; an SVC 1 read and wait is issued thus leaving the background task at top of ready chain. The task manager dispatches the background task by loading the user register set from the dispatch register save area in the TCB and loading PSW from the TCB dispatch PSW save area.

8.2 I/O Request

Figure 8-2 illustrates system flow during an SVC 1 Write Image and wait request to the line printer. The task issues an SVC 1 write, image and wait. The First Level Interrupt Handler decodes the SVC and passes control to the SVC 1 processor. SVC1 checks the validity of the data in the parameter block and enters RS state, saving the user registers and resume PSW in the TCB. EVQCON is called to connect to the line printer leaf. On return the information in the parameter block is stored in the line printer DCB and SVC1 branches to the DIR.

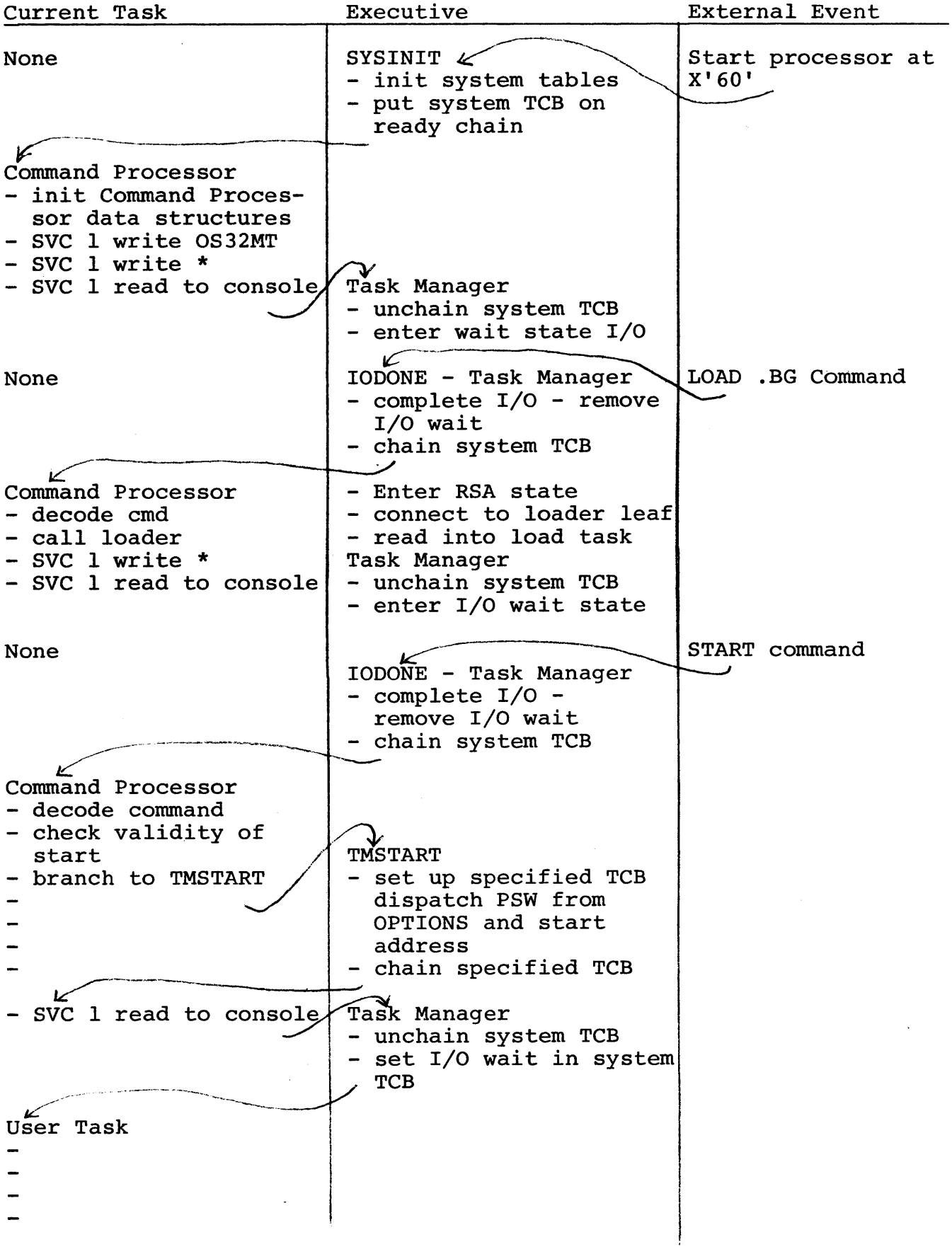


FIGURE 8-1 System Start Up

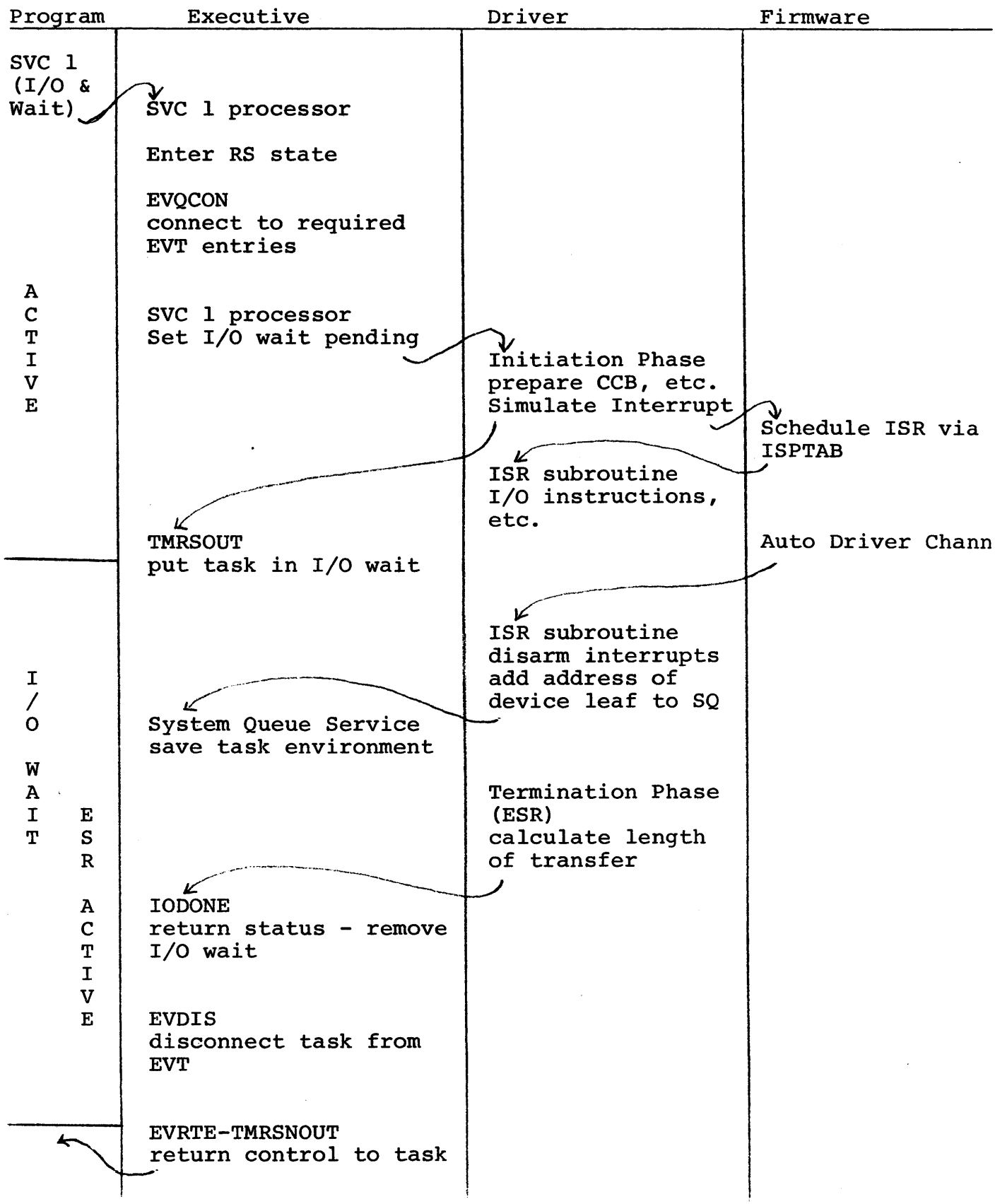


FIGURE 8-2 SVC 1 (I/O AND WAIT)

The Line Printer DIR sets up the CCB with the proper CCW, subroutine address and buffer information and then SINTs the device. The microcode transfers control in IS state to the ISR routine pointed to by the CCB. The ISR sets the timeout constant, checks the device status and enables interrupts on the device. It picks up the first character in the buffer and writes it to the device and returns control to the DIR following the SINT. The DIR exits to the task manager which puts the task into I/O wait by setting the I/O wait bit, unchaining the TCB and moving the user registers and resume PSW from the RS save area to the dispatch save area of the user TCB. The Auto Driver Channel completes the transfer and passes control to the ISR on buffer empty. The ISR disarms interrupts on the printer and adds the address of the line printer leaf to the system queue. The ISR exits by loading the PSW in registers 0 and 1 of register set 0. This PSW is the system wait PSW since no task was active. The Queue Service enable bit is set, so the microcode causes a queue service interrupt, passing control to SQS. SQS removes the address of the printer leaf from the system queue, queues the user task to the top of the EVT by calling EVPROP and branches to EVTDISP. EVTDISP saves the user task environment by moving the dispatch save area to the ES save area in the user TCB, chains the user TCB and schedules the ESR pointed to by the leaf. The ESR calculates length of transfer, stores it in the DCB and exits to IODONE. IODONE moves the status and length of transfer to the SVC 1 parameter block, disconnects from the leaf by calling EVDIS, resets the I/O wait bit in the TCB and exits to EVRTE to return from the event. EVRTE finds no queued events and exits to TMRSNOUT which returns control to the user task by loading the user registers and resume PSW from the TCB ES save area.

8.3 Log Message

Figure 8-3 illustrates system flow during a log message request. The background task issues an SVC 2 code 7. The SVC First Level Interrupt Handler decodes the SVC and passes control to the SVC 2 Second Level Interrupt Handler. SVC 2 SLIH enters the SVC 2 code 7 executor (SVC2.7) via TMRSIN. SVC2.7 checks the options and calls EVQCON to connect to the EVT leaf for the dummy device. On return SVC2.7 moves the text to an internal buffer, sets up the DCB for the dummy device to point to the internal buffer and branches to the dummy driver. The dummy driver sets the message pending flag in the command processor, times out the read outstanding to the system console by storing a carriage return in the command buffer and scheduling the teletype ESR. This causes the system task to be scheduled, suspending the background task inside the dummy driver. The command processor finds the command buffer empty and message pending set. The data in the DCB for the dummy device

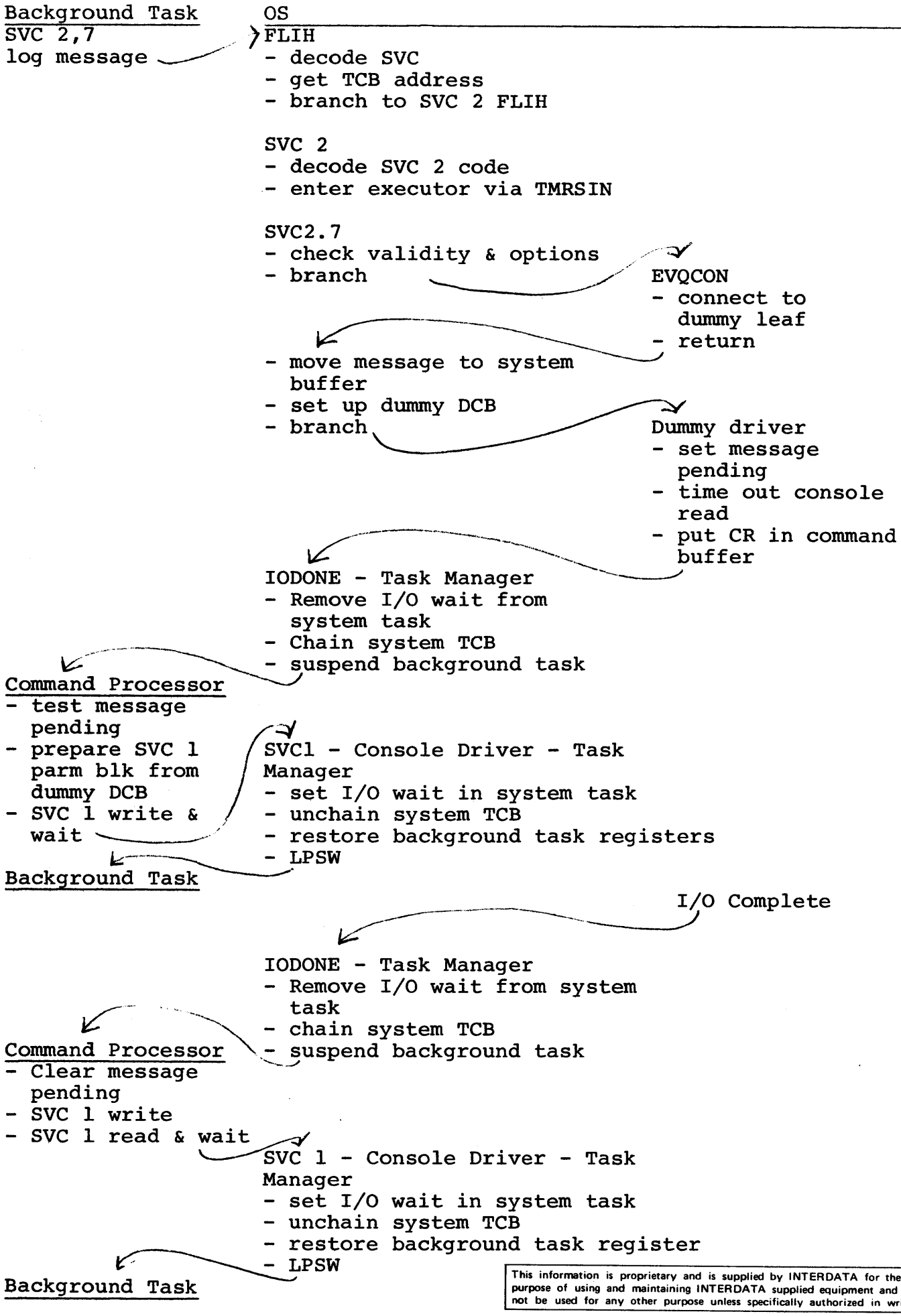


FIGURE 8-3 Log Message Request

is used to prepare a parameter block for the SVC 1 write and wait to the system log. The task manager puts the system task into I/O wait thus making the background task top of ready chain. The background task is dispatched at the exit from the dummy driver. The dummy driver exits to TMRSOUT which returns control to the user task following the SVC 2 code 7. When the message completes, the teletype ESR is scheduled for the system task, suspending the background task. The command processor clears message pending flag and issues an SVC 1 read and wait to the system console. The task manager puts the system task into I/O wait and returns control to the backbround task.

8.4 READ REQUEST TO CHAIN FILE

Figure 8-4 illustrates system flow during an SVC 1 read request to a chain file. On execution of the SVC 1, the First Level Interrupt Handler passes control to SVC1. SVC1 checks the validity of the request and since the LU entry points to a chain file FCB, SVC1 does not attempt to perform a connection since the leaf field in the FCB contains zeroes. SVC1 enters RSA state since the request is to an FCB with the buffered access method flag set. Entry to the file manager is made at CHAIN. CHAIN resets I/O wait pending in the task, determines that it is a read request and calls GETCHL. GETCHL moves the data from the current FCB buffer to the user task buffer. When the data in the current buffer is exhausted, GETCHL calls GETCHPR to refill the buffer. GETCHPR issues an SVC 1 read and proceed call for the next sector in the file to be read into the just exhausted buffer. This causes the SVC 1 processor to enter RS state, connect to the disc leaf and branch to the disc driver. The disc driver initiates the read and exits via TMRSOUT. Since CHAIN reset I/O wait pending, TMRSOUT returns control to the file manager in RSA state following the SVC 1 request. GETCHPR returns to GETCHL which completes the data move from the other buffer (now current). On completion, the file manager exits via TMRSAOUT which returns control to the user task following the SVC 1 read and wait.

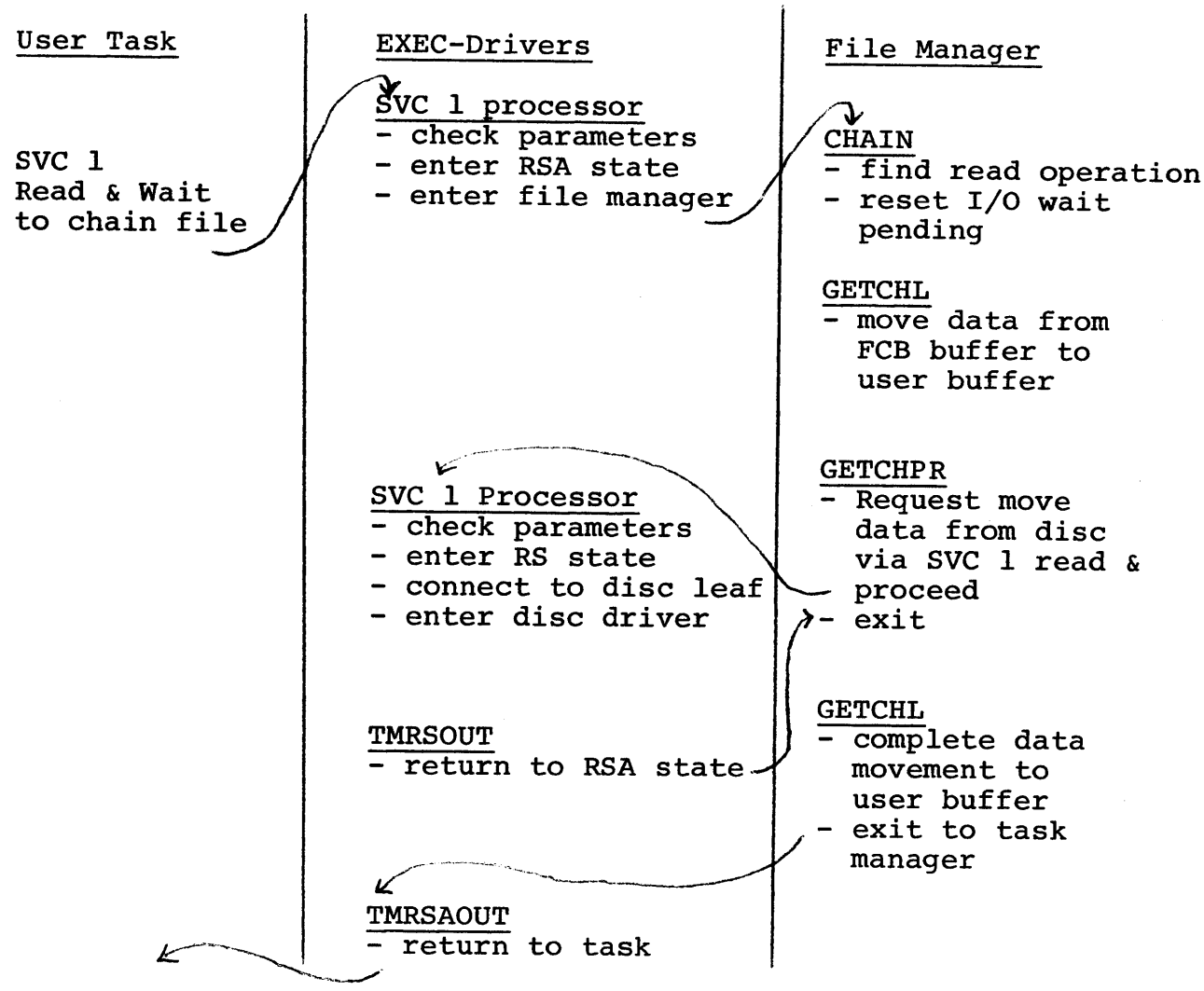


FIGURE 8-4 EXAMPLE OF READ REQUEST TO CHAIN FILE

CHAPTER 9

EXECUTIVE TASKS AND SYSTEM EXTENSIONS

9.1 INTRODUCTION

There are several ways of extending or modifying the capabilities of OS/32 MT. This chapter discusses the features designed into OS/32 MT to facilitate such extensions. The user may wish to incorporate the modification directly into the system by modifying one or more system modules or by adding a system module. For example, the user may support a non-standard peripheral device by writing a driver. On the other hand, the user may wish to support infrequently used extensions to the system by writing an executive task (E-task) which may be loaded and executed on demand.

9.2 EXECUTIVE TASKS

An Executive Task (E-Task) is written as a user task and executed in ET state by specifying OPTIONS ET when TETing the task. E-Tasks execute in a hardware and software privileged mode. All machine instructions are allowed and these additional capabilities are provided:

- All addresses are valid in SVC calls
- A disc device (rather than a file on the disc) may be assigned to the E-Task
- SVC 2 code 0 (Journal Entry) is valid
- REPROTECT (SVC 7) for a key of X'FF' and to non-bulk device is valid
- RENAME (SVC 7) for a key of X'FF' and to non-bulk device is valid

As a direct result of these added capabilities, E-Tasks must be designed and coded with extreme caution to prevent crashing the system. E-Tasks may not execute in halfword mode. The Operating System assumes E-Tasks are debugged tasks, and hence make no mistakes.

Access to system tables and control information is provided through the System Pointer Table (SPT). The address of the SPT is contained in the halfword at location X'62' in low memory. E-Tasks may use all SVCs. An example of a function which might require an E-Task is a disc utility program. The OS/32 MT Command Processor executes as an E-Task.

Special care must be taken when writing an E-Task. Since the task can be loaded into any partition, and furthermore, since it is run without the MAC enabled, the E-Task must be coded as a completely position independent. This means that the E-Task must:

(1) contain no RX3 instructions, other than immediate constant references
(2) may not assemble into any of his SVC parameter blocks an address; i.e.,

```
SVC 7 BLK   DB    X'80',7
            DAC   address
```

since addresses will not be relocated.
(3) E-tasks cannot reference the Task Common or RTL in a manner which would use TET to resolve references. Addressing E0000 will probably give erroneous data or memory parity error.

In order to find out where an E-Task has been loaded, it should perform the following instruction

```
LABEL    LA    BASE,LABEL
```

at the beginning or at some known offset in the task. The E-Task, then knowing where it is loaded, should dynamically set up its parameter blocks. For instance,

```
BASE     LA    U4,BASE
         LA    UE,BUFSTART-BASE(U4)
         LA    UF,BUFEND-BASE(U4)
         LA    U3,SVC1BLK-BASE(U4)
         STM   UE,SVC1.SAD(U3)
         SVC   1,0(REG3)
```

If no label references are made, that range farther than 16KB, then the CAL NORX3 option X can be used. In this case you can assemble in references, and CAL will turn references to these labels into RX2 (relative addressing) instructions.

9.3 SYSTEM EXTENSIONS

OS/32 MT may be extended or modified by incorporating changes into the source of one or more system modules or adding a system module and using the Configuration Utility Program (CUP) MODULE statement to include the modified or new module in the system (refer to the OS/32 MT Program Configuration Manual All system data structures should be referenced by copying the STRUC defining the data structure from the Parameters and Control Block Module at assembly time and using these field definitions in all instructions referencing the structure.

9.4 PATCHING

In making modifications to OS/32 MT, debugging usually entails making patches to new or existing code to avoid reassembling every time a bug is found. In order to insert a patch in OS/32 MT:

(1) locate the address of SPT.UBOT in the map of the system produced by the OS/32 Library Loader.

(2) Use the MODIFY command to increase the value of UBOT by an amount sufficient to contain the patch.

(3) Use the MODIFY command to insert the patch starting at the old value of UBOT.

(4) Use the MODIFY command or the console panel to insert a branch to the patch area.

CHAPTER 10

JOURNAL AND CRASH CODES

10.1 CRASH CODES

After a system crash, register 5 of register set 0 contains a pointer to the system journal and register 6 contains a pointer to the most recent entry to the journal. The following is a list of crash codes, their meanings and in some cases additional information concerning the cause of the crash. (Ex denotes register X of the executive register set (set 0); Ux denotes register X of user register set, Rx denotes register X of register set in use at time of crash).

CRASH CODE (HEX)	DESCRIPTION
1	Console device mnemonic not found in DMT
2	Unrecoverable error on system console
7	Invalid VMT during MARK processing
10	Invalid file descriptor during MARK processing
100	Arithmetic fault not in UT/ET state E9 contains address of current TCB. EE-EF contain PSW at time of fault.
101	Arithmetic fault not in user task. E9 contains current TCB ID, EE-EF contain PSW at time of fault.
102	Illegal instruction, illegal SVC or illegal address passed in SVC not in user task. E9 contains current TCB ID, ED contains pointer to 4 bytes before pointer to message, EE-EF contain PSW at time of fault. Contents of Executive registers are stored over the code starting at the first fullword following the crash code.
103	Illegal instruction, illegal SVC or illegal address passed in SVC-user task not in UT/ET state. E9 contains address of user TCB, ED-EF same as for 102.
104	Memory parity error during Auto Driver Channel operation.
105*	Attempt to pause system task.
106	Illegal SVC or illegal address passed in SVC with PSW not pointing after an SVC 1 instruction. EE-EF contain PSW at time of interrupt.
107*	Attempt to remove illegal TCB from ready chain. R9-TCB ID, R8-return address.
108*	Attempt to remove a wait condition from or chain an illegal TCB ID. R8-return address, R9-TCB ID.

CRASH CODE	DESCRIPTION
109*	Attempt to dispatch illegal TCB ID from top of ready chain. E9-TCB ID.
10A*	Attempt to dispatch ESR for illegal TCB ID. E9-TCB ID; EA-ES priority, EF-leaf address.
110*	Attempt to start illegal TCB ID. U9-TCB ID, UF-start location.
111*	Attempt to remove illegal wait bits from TCB. R8-return address, R9-TCB address, RD-wait bits.
112*	Attempt to put illegal TCB into RS state. E9-TCB ID; EA-EB-return PSW.
113*	Attempt to take illegal TCB out of RS state. U9-TCB ID.
115*	Attempt to suspend illegal TCB. E8-return address; E9-TCB ID.
118	TCB has ready chain bit set but is not on ready chain. E8-return address, E9-TCB ID.
119	Memory fault interrupt-hardware error. EE-EF-PSW at time of fault.
120*	Invalid size request to GETSYS. 03 = size.
132	Illegal SVC executed from within system code. UF contains SVC address.
142	Memory fault in SVC executed from within system code, i.e., buffer not on fullword boundary.
152	Parity error within system code. Locations X'20' - X'26' contain the PSW of the time of the parity error.
180	Clock ESR scheduled, but no flags set.
181*	Clock ESR scheduled, with PIC service flag set, but SPT.IQHD = 0.
182*	Clock ESR scheduled, with LFC service flag set, but SPT.TQHD = 0.
183*	Clock ESR in removing a wait, caused someone other than the system task to become the current task.
200*	System Queue Service interrupt-hardware error. EE-EF-PSW at time of fault.
201*	Invalid leaf address on system queue. ED-leaf address.
202*	Event for unconnected leaf. ED-leaf address.
205*	Attempt to disconnect or release leaf not connected to current task. U8-return address, U9-connected TCB ID; UF-leaf address.
206*	Release level 2 or greater than connection level for leaf. Same as for 205 with UE-release level.
207*	Attempt to connect to invalid leaf address. U8-return address; UD-DCB/FCB pointer, UF-leaf address.
208*	Attempt to modify a leaf not connected to current task. U8-return address, UE-new ESR address; UF-leaf address.

CRASH CODE	DESCRIPTION
20A*	Leaf queued to system node with no task queued to leaf. EB-leaf address.
210	Entry to EVRTE not in ES state. U9-TCB ID; E0-E1-PSW at entry to EVRTE.
211	Task event count non-zero but all leaf occurrence counts zero. U9-TCB address.
212*	Leaf being disconnected has occurrence count greater than TCB event count. U8-return address; U9-TCB address; UF-leaf address.
307	Request for FCB of invalid size
308	Attempt to release FCB with FBOT=MTOP
30A	FCB not found during release attempt
30B	Invalid DCB link field during release FCB
30C	Invalid FCB chain found during release FCB
30D	Bit map or directory leaf added to system queue
30E	Invalid save attributes
30F	Attempt to close invalid file type

* denotes crash check present only if SGN.SAFE = 1

10.2 JOURNAL CODES

CODE	DESCRIPTION AND REGISTER CONTENTS
6x	Execution of SVC x. (FLIH) 12 - First word of SVC parameter block (if any) 13 - Address of parameter block 14 - SVC old PSW status 15 - SVC old PSW location (updated)
71	Task dispatched from suspended state or from NS state (TMRDISP) 12 - n.i. 13 - n.i. 14 - Status portion of PSW to be loaded 15 - Location counter of PSW to be loaded
72	Task exit from RS state. (TMRSOUT) 12 - address of TCB RS save area 13 - n.i. 14 - n.i. (if 15=0); status portion of exit PSW (15≠0) 15 - 0 means load PSW in TCB.RPSW; location of exit PSW if non-zero
73	Task entered ES state (TMRSNIN) 12 - n.i. 13 - DCB address 14 - n.i. 15 - leaf address
74	Task exit from ES state (TMRSNOUT) 12 - address of TCB ES save area 13 - n.i. 14 - n.i. 15 - 0
75	Task exit from RSA state (TMRSAOUT) 12 - address of alternate save area 13 - n.i. 14 - n.i. (15=0); status portion of exit PSW (15≠0) 15 - 0 means load PSW from save area; location of exit PSW if non-zero

CODE	DESCRIPTION AND REGISTER CONTENTS
76	Task entered RS state (TMRSIN) 12 - address of RS save area 13 - n.i. 14 - status portion of resume PSW 15 - location counter of resume PSW
77	Task entered RSA state (TMRSAIN) 12 - address of alternate save area 13 - n.i. 14 - status portion of resume PSW 15 - location counter of resume PSW
80*	Remove wait or Chain call (TMREMW) 12 - n.i. 13 - wait bits if Remove wait call; n.i. if chain call 14 - n.i. 15 - n.i.
91*	Illegal Instruction Interrupt (IIH) 12 - n.i. 13 - n.i. 14 - status portion of PSW at time of interrupt 15 - location counter of PSW at time of interrupt
92*	Arithmetic Fault Interrupt (AFH) 12 - n.i. 13 - n.i. 14 - status portion of PSW at time of interrupt 15 - location counter of PSW at time of interrupt
94	System Queue Service Interrupt (SQS) 12 - n.i. 13 - leaf address 14 - status portion of old PSW 15 - location counter of old PSW
95*	Connect to Leaf (EVQCON) 12 - 0 means QCON call; -1 means CON call 13 - DCB address 14 - ESR address 15 - leaf address

CODE	DESCRIPTION AND REGISTER CONTENTS
96	Disconnect from Leaf (EVDIS) 12 - n.i. 13 - DCB address 14 - n.i. 15 - leaf address
8001	Command Processor Command decoded (COMMANDR) 12 13 14 - Index of Command in command table 15
8002	Command Processor Dummy Driver Call (COMMANDR) 12 - n.i. 13 - n.i. 14 - n.i. 15 - n.i.
8xxx	User Journal Code (SVC 2 code 0)

* denotes journal code present only if SGN.SAFE = 1

n.i. means register contains no information or information meaningful only in context.

CHAPTER 11

DATA STRUCTURES

11.1 INTRODUCTION

This chapter presents the formats of system control blocks and table entry. Each field is identified by its name and a descriptive title. All control blocks and table entries are referenced in OS/32 MT by copying the CAL STRUC of the same name from the OS/32 MT Parameters and Control Block module. The full field identifier is of the form:

BBB.FFFF

where BBB is the control block name and FFFF is the field name. Most fields are self explanatory; those which are not are explained following the figure for that control block. Offsets are given in the form:

DD(HH)

where DD is the offset in decimal and HH is the offset in hexadecimal.

11.2 CHANNEL CONTROL BLOCK (CCB)

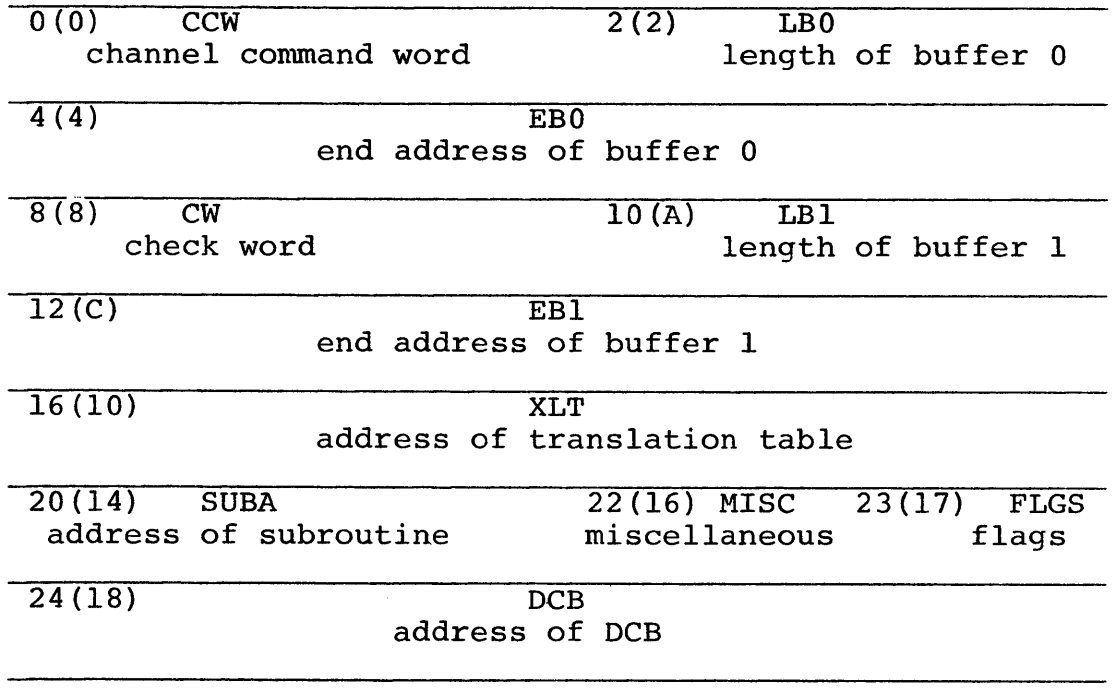


FIGURE 11-1 CHANNEL CONTROL BLOCK (CCB)

- Channel Command Word (CCW)

Bit	Flag Name	Meaning
0-7	CCWSTAT	This byte is AND'd with device status; if result is non-zero, control is passed to CCB subroutine.
8	CCWEX	Execute bit. If set Auto Driver performs operation specified by CCW; if reset, control is passed to CCB subroutine.
9-11		Reserved.
12	CCBBl	Buffer bit. If reset buffer 0 in use; if set buffer 1 in use (unless bit 15 also set).
13	CCBWR	Read/Write bit. Reset means read; set means write.
14	CCBTL	Translate bit. If set translation is performed using translation table pointed to by CCB.XLT.
15	CCWFST	Fast Bit. Set indicates fast mode - no translation, buffer 0, no buffer switch, no redundancy checking.

- Miscellaneous and Flags (MISC and FLGS)

These fields are used by drivers to pass and maintain information controlling the request from DIR to ISRs and ESRs. Sometimes referenced as a halfword field, sometimes as two byte fields.

11.3 DEVICE CONTROL BLOCK (DCB)

0(0)	DMT		
	address of DMT entry		
4(4)	LEAF		
	address of leaf		
8(8)	INIT		
	address of driver data xfer entry		
12(C)	FUNC		
	address of driver cmd function entry		
16(10)	TERM		
	address of driver termination routine		
20(14)	WCNT	22(16)	RCNT
	write count		read count
24(18)	ATRB	26(1A)	RECL
	attributes of device		record length
28(1C)	TOUT	30(1E)	RTRY
	time out constant		retry count
32(20)	FLGS	34(22)	
	flags halfword		reserved
36(24) STAT	37(25) DCOD	38(26) WKEY	39(27) RKEY
I/O status	device code	write key	read key
40(28)	PBLK		
	relocated SVC 1 parameter block address		
44(2C) FC	45(20) LU	46(2E)	DN
function code	logical unit		device number
48(30)	SADR		
	relocated SVC 1 start address		
52(34)	EADR		
	relocated SVC 1 end address		
56(38)	RAND		
	SVC 1 random address		
60(3C)	LLXF		
	length of last transfer		
64(40)	TOCH		
	time-out chain		
68(44)	UPBK		
	unrelocated parameter block address		

FIGURE 11-2 DEVICE CONTROL BLOCK (DCB)

The DCB is used by the I/O subsystem to identify characteristics of each device configured in the system and to serve as a work space for drivers during an I/O request. DCBs are pointed to by the Device Menmonic Table (DMT) entry for the device represented. DCBs are included in the system by the Configuration Utility Program (CUP) at object SYSGEN time.

- Attributes (ATRB)

This field is used at assign and SVC 1 time to check the validity of the request.

Bit	Meaning
0	reserved
1	supports read
2	supports write
3	supports binary formatted records
4	supports wait I/O
5	supports random requests
6	supports unconditional proceed
7	supports image
8	reserved
9	supports rewind
10	supports backspace record
11	supports forward space record
12	supports write file mark
13	supports forward space file mark
14	supports backspace file mark
15	supports device dependent command

- Time out constant (TOUT)

This field is used to control device time out and halt I/O functions.

Value	Meaning
X'001'-X'7FFF'	Device active for request
X'0000'	Device has been timed out (I/O halted)
X'FFFF'	ESR has been scheduled for this request

- Flags (FLGS)

Bit	Flag Name	Meaning
0	DFLG.BLM/B	Bulk device flags
1	DFLG.LNM/B	On-line flag. Set indicates device online.
2	DFLG.DRM/B	Directory presence flag. Set indicates valid directory record in memory for this device. Bulk device flag must also be set.
3	DFLG.MPM/B	Bit Map presence bit. Set indicates valid bit map record in memory for this device. Bulk device flag must also be set.

Bit	Flag Name	Meaning
4	DFLG.PFM/B	Set indicates moving head disc driver should check record for pseudo file mark.
5	DFLG.BMM/B	Bit map modify bit. Set indicates bit map record in memory has been modified and must be rewritten to disc. Bulk device flag must also be set.
6	DFLG.CNM/B	Console bit. Device is a console device.
7	DFLG.UCM/B	Uncancellable flag. Device not to be halted on cancel.
8	DFLG.S6B	SVC 6 Connectable Bit. Indicates that this device is useable only through SVC 6 ISA calls.
9	DFLG.WPB	Write-protect bit. Indicates this device is write protected.

- Device Code (DCB)

This field is used to identify the particular device. Value must be greater than X'0F'. A value of X'FF' indicates the null device.

11.4 DIRECTORY ENTRY (DIR)

0(0)	FNM File Name		
8(8)	EXT Extension		11(B) VERS Version
12(C)	FLBA First Logical Block Address		
16(10)	LLBA Last Logical Block Address		
20(14) WKEY Write Key	21(15) RKEY Read Key	22(16) LRCL Logical Record Length	
24(18)	DATE Creation Date/Time		
28(1C)	LUSE Last Used Date/Time		
32(20) WCNT Write Count		32(22) RCNT Read Count	
36(24) ATRB Attributes	32(25) BKSZ Blocksize	38(26) FLRO First Logical Record Offset	
40(28)	CSEC Current Sector/# of Logical Records		

FIGURE 11-3 DIRECTORY ENTRY (DIR)

Each directory record contains up to five directory entries. Version (VERS), Creation date (DATE), date last used (LUSE), are unused in OS/32 MT.

- Current Sector/# of Logical Records (CSEC)

This field contains a pointer to the current relative sector for a contiguous file; number of logical records in a chain file.

11.5 DEVICE MNEMONIC TABLE (DMT)

0(0)	DM Device Mnemonic
4(4)	DCB Address of DCB

FIGURE 11-4 DEVICE MNEMONIC TABLE (DMT)

The DMT consists of 1 entry for each device configured in the system. The table is terminated by a doubleword of zeroes. The DMT is pointed to by the SPT. There is no structure for the DMT.

11.6 EVT LEAF (EVL)

0(0)	CORD coordination-address of parent		
4(4) CPRI connection priority	5(5) FLGS flag byte	6(6) QPRI highest queued priority	7(7) QTCB 1st TCB ID in queue
8(8) DSCN descendent number		10(A) OCNT occurrence count	
12(C)	PREV previous leaf in connected chain		
16(10)	NEXT next leaf in connected chain		
20(14)	DCB connected DCB		
24(18)	ESR entry point of ESR		
28(1C) CLEV connection level	29(1D) TSIZ tree size	reserved	31(1E) CTCB connected TCB

FIGURE 11-5 EVT LEAF (EVL)

- Flags (FLG)

Bit Offsets are from the halfword boundary - EVL.CPRI.

Bit	Flag Name	Meaning
8	EVF.LEFM/B	Leaf bit. EVT entry is a leaf.
9	EVF.ASSM/B	Assert bit. Task is asserting reconnection to upper nodes of leaf.
10	EVF.PENM/B	Pending flag. Leaf has evented.
11	EVF.DUMM/B	Non-eventing flag. This leaf doesn't represent a physical device, thus it should never appear on the system queue.

- Tree Size (TSIZ)

This is the number of entries in the path up to the system node including the leaf. For example, TSIZ = 1 for a TTY leaf; TSIZ = 3 for a disc leaf (disc leaf, disc controller node, selch node). When connection level (CLEV) = tree size (TSIZ) the task is connected to all required entries for this path. EVL is contained in the STRUC named EVT.

11.7 EVT NODE (EVN)

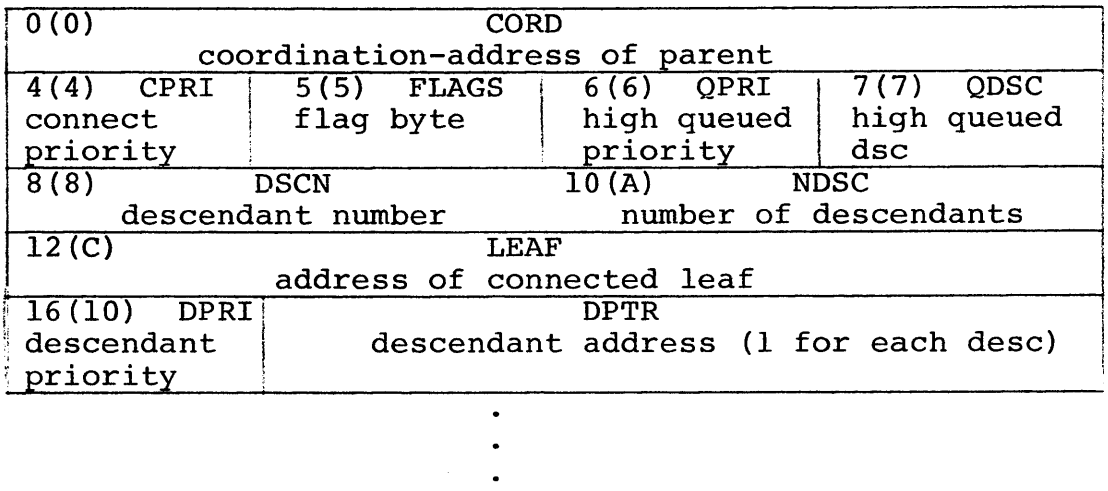


FIGURE 11-6 EVT NODE (EVN)

The bit definitions of the flags field (EVN.FLGS) is identical to those for the EVT leaf (EVL). The leaf bit (bit 8) or the pending bit (bit 10) should never be set in a node. The length of a node is variable since the descendant pointer list occupies 4 bytes for each direct descendant.

11.8 FILE CONTROL BLOCK (FCB)

0(0)	VMT address of VMT entry						
4(4)	LEAF address of leaf						
8(8)	INIT address of driver data xfer entry						
12(C)	FUNC address of driver cmd-function entry						
16(10)	TERM address of driver term entry						
20(14)	WCNT write count			RCNT read count			
24(18)	ATRB attributes of file			RECL record length			
28(1C) OFF directory offset	29(1D) BKSZ file block size		30(1E) reserved				
32(20)	FLGS flag halfword		34(22) reserved				
36(24) STAT DCB I/O status	37(25) DCOD device code		38(26) WKEY write key		39(27) RKEY read key		
40(28)	PBLK relocated SVC 1 parameter block address						
44(2C) FC function code	45(2D) LU logical unit		46(2E) PA physical address				
48(30)	SADR relocated SVC 1 start address						
52(34)	EADR relocated SVC 1 end address						
56(38)	RAND SVC 1 random address						
60(3C)	LLXF length of last transfer						
64(40)	DS 8 reserved for DCB compatibility						
72(48)	NAME file name						
80(50)	EXT file extension				83(53) VERS reserved		
84(54)	DIR address of directory block						
88(58)	DCB address of DCB						

FIGURE 11-7 FILE CONTROL BLOCK (FCB)

90(5C)	FLBA first logical block address
96(50)	LLBA last logical block address
100(64)	CSEC current sector logical block address
104(68)	RPSW PSW save area
112(70)	RGPR general register save area
176(B0)	ASP pointer to next RSA save area

180(B4)	BAPB			
	parameter block of buffer A			
200(C8)	FCB			
	FCB Linkage Field			
204(CC)	SLU			
	saved LU entry			
208(D0)	BAPT			
	buffer for contiguous files/buffer A parm block ptr. for chained files			
212(D4)	BBPT			
	address of buffer B parm block			
216(D8)	FCB.RSAS			
	save area			
228(E4)	PBUF			
	previous buffer			
232(E8)	NBLK			
	number blocks in file			
236(EC)	CBLK			
	current block number			
240(F0)	NLR			
	number logical records in file			
244(F4)	CLRL			
	current logical record number			
248(F8)	COFF	250(FA)	219(FB)	CBUF
	offset of current blk	reserved		current buffer
252(FC)	BBPB			
	parameter block of buffer B			

272(110) BUFA

BUFB

Chained Files

Buffers

The first portion of the FCB is defined as the DCB with the exception of a VMT pointer instead of a DMT pointer and different flag definitions. The value of the device code identifies this control block as an FCB rather than a DCB.

- Flags (FLGS)

Bit	Flag Name	Meaning
0	FFLG.BAM/B	Buffered access method flag. Set indicates a buffered access file.
1	FFLG.USM/B	Set implies FCB in use.
2	FFLG.OPM/B	Operation flag. Set implies write.
3	FFLG.ABM/B	Active buffer bit. Set implies buffer active (chain files only)
4	FFLG.BMM/B	Current buffer flag (chain files only)
5	FFLG.MOM/B	Mode flag. Set indicates random (chain files only)
6	FFLG.DIM/B	Direction flag. Set implies left (chain files only)

- Device Code (DCOD)

Device code must be X'00' (contiguous file) or X'01' (chain file).

11.9 INITIAL VALUE TABLE (IVT)

0(0)	CSL Console Device Mnemonic
4(4)	MXBX 1 byte = .BG MAXPRI other 3 = .bg maximum system space
8(8)	.TCMS Intertask common size
12(C)	.SYSS Initial size of system space
16(10)	PART Initial size of part 1
20(14)	Initial size of part 2
	.
	.
	.

IVT is pointed to by SPT.IVT

11.10 SYSTEM POINTER TABLE (SPT)

0(0)	INIT branch to SYSINIT		
		6(6)	CRSH system crash code
8(8)	FLV address of first leaf		
12(C)	LLV address of last leaf		
16(10)	MLBL message log buffer length	18(12)	CTSP ctop expand quantity
20(14)	CSLV number of CSS levels		
24(18)	CSBF size of CSS buffer + 2		
28(1C)	CHBK maximum chain file block size	30(1E)	ISPT Top of ISP + MAC
32(20)	CTOP top of TQE's - 2		
36(24)	UTOP lst byte in system space		
40(28)	UBOT lst byte above OS/32 MT		
44(2C)	FBOT lower bound of FCB's		
48(30)	MTOP lst byte above configured memory		
52(34)	OSID system ID = OS32MT RR RR = release level		
60(3C)	IVT address of initial value table		
64(40)	TTAB address of TCB table		
68(44) CTCB current TCB ID	69(45) NTCB max TCB ID + 1	70(46)	SPT.60 timer constant
72(48)	DMT address of DMT		
76(4C)	VMT address of VMT		

FIGURE 11-9 SYSTEM POINTER TABLE (SPT)

80(50)	SVOL name of default volume		
84(54)	SNOD address of system node		
88(58)	JRNL address of system journal		
92(5C)	FREQ line frequency #2	94(5E)	PIC address of PIC
96(60)	LFC address of line freq. clock	98(62)	reserved
100(64)	SOPT system options		
104(68)	MNTH month	108(6A)	DAY day
110(6C)	YEAR year	112(6E)	reserved
114(70)	TIME time in seconds since midnight		
118(74)	DTHD address of head of device timeout queue		
122(78)	TQHD address of head of time of day queue		
126(7C)	IQHD address of head of interval queue		
130(80)	RTLS RTL segment register		
134(84)	TCMS Task Common segment register		
138(88)	RTLN RTL name		
150(94)	TCMN Task Common name		
162(A0)	PSV Task Manager PSW save		
170(A8)	RSV Task Manager register save		
174(AC)	TSV Task Manager TCB address save		
178(B0)	AFSV Task Manager Save Area REG A-F		

0(0) ID TCB ID #	1(1) PRI TCB PRIORITY	2(2) DPRI DISPATCH PRIO.	3(3) NLU # LOGICAL UNIT	
4(4) MPRI max. prio.	5(5) NPRI # non-evnt lvs cnct	Reserved		
8(8) OPT options halfword		10(A) STAT status halfword		
12(C) WAIT wait condition halfword		14(E) EVC event occurrence count		
16(10) PTCB prev. tcb on ready	17(11) NTCB next tcb on ready	18(12) PCWT prev. tcb in cnct wt	19(13) NCWT next tcb in cnct wt	
20(14) SLOC default starting address				
24(18) NAME Task Name				
32(20) CTOP				
36(24) UTOP				
40(28) UBOT				
44(2C) MXSP maximum system space				
48(30) USSP				
52(34) SEG				
116(74) LST				
180(B4) SYS system word				
184(B8) SYS 1 system word				
188(BC) USER user TCB field				
196(C4) PTCH peer task chain				
200(C8) ASV alternate save area pointer				
204(CC) LEAF leaf ptr. during connect wait				
208(D0) CLC connect leaf chain				

212(D4) RC return code	Reserved
216(D8)	CTSW current task status word
220(DC)	DPSW dispatch save PSW
228(E4)	DGPR dispatch save registers
292(124)	RPSW RS Save PSW
300(12C)	RGPR RS save registers
364(16C)	EPSW ES save PSW
372(174)	EGPR ES save registers
436(1B4)	FMLU file manager dummy LU
440(1B8)	LTAB Logical Unit Table

- Options (OPT)

Bit	Flag Name	Meaning
0	TOPT.ETM/B	Set means E-task, reset means user task.
1	TOPT.ACM/B	Set means continue on arithmetic fault; reset means pause.
2	TOPT.FPM/B	Set means task uses floating point registers.
3	TOPT.MRM/B	Set means task is memory resident.
4	TOPT.TCM/B	Set means task uses task common.
5	TOPT.LBM/B	Set means task uses RTL.
6	TOPT.SGM/B	Set means if task is background and issues an SVC 6, it will be ignored. Reset means treat SVC 6 from the background as illegal.
7-15	reserved	must be 0

- Status (STAT)

Bit	Flag Name	Meaning
0	TSTT.ESM/B	Set implies valid data in TCB ES-Save area. Task is non-eventable. Task is in ES state.
1	TSTT.RSM/B	Set implies valid data in TCB RS-save area of alternate save area. Task may be in RS, RSA or ES state depending on other status bits.
2	TSTT.RPM/B	Pause pending. Set means task to be put into console wait on dispatch into UT/ET state.
3	TSTT.RCM/B	Set implies task on ready chain.
4	TSTT.ASM/B	Set means valid data in save area pointed to by TCB.ASV. TSTT.RSM/B must also be set.
5	TSTT.IPM/B	I/O wait pending. Task to be put into I/O wait on exit from RS or RS state.
6	TSTT.SYM/B	Set means TCB is the system task.
7	TSTT.CPM/B	Cancel pending. Task is in SVC 3 processing.
8	TSTT.APM/B	Set means ABTERM is pending. Task will go to EOJ whem ready to return to UT/ET state.
9	TSTT.PWM/B	Set means task was in system when power fail/restart took place.
10	TSTT.FVM/B	Set means task floating point registers are saved; reset implies task's floating point registers are current (meaningful only if TOPT.FPB = 1).
11-15	reserved	must be 0

- Wait (WAIT)

Bit	Flag Name	Meaning
0	TWT.IOM/B	I/O wait
1	TWT.CWM/B	Connection wait. Task on an EVT queue.
2	TWT.CNM/B	Console wait. Task paused.
3	TWT.LWM/B	Load wait. No task has been loaded or task in SVC 5 processing.
4	TWT.DMM/B	Dormant. Task loaded but not started or task has gone to EOT.
5	TWT.TWM/B	Set means Task in trap wait.
8	TWT.TMM/B	Set means task in time/interval wait
9-15	reserved	must be 0

11.12 VOLUME MNEMONIC TABLE (VMT)

0(0)	VM Volume Mnemonic
4(4)	DMT address of corresponding DMT entry

FIGURE 11-11 VOLUME MNEMONIC TABLE (VMT)

There is one entry in the VMT for each disc device configured in the system. When the disc is marked online the volume name is read from the volume directory and placed in the VMT entry. The VMT is terminated with a doubleword of zeroes. There is no structure for the VMT.

11.13 VOLUME DESCRIPTOR (VD)

0(0)	VOL Volume Name
4(4)	ATRB Volume Attributes
8(8)	FDP First Directory Block Pointer
12(C)	OSP Pointer to OS Image
16(10)	OSS Size of OS Image
20(14)	MAP Pointer to Bit Map

FIGURE 11-12 VOLUME DESCRIPTOR (VD)

The volume descriptor is written onto sector 0 of a disc pack by the INITIALIZE command. Volume attributes field is not used in OS/32 MT.

11.14 TASK LOADER INFORMATION BLOCK

0(0)	1(1)	2(2)	3(3)
SEG TYPE "1"	NO OF LIBS	MAX LU RQD	RESERVED
4(4)	5(5)	6(6)	7(7)
MAX PRIORITY	INIT PRIORITY	RESERVED	NO OF TCOM's
8(8)	9(9)	10(A)	
NO OF RTL's	RESERVED	OPTIONS HALFWORD	
12(C)			
SIZE OF IMAGE AS NO. OF 256-BYTE BLOCKS			
16(10)			
20(14)			
MAX SYSTEM SPACE AVAILABLE			
24(18)			
INITIAL TSW			
32(20)			
44(2C)			
RESERVED FOR FUTURE USE			
60(3C)			
INITIAL CAPABILITY MATRIX			
80(50)			
TIME/DATE			
100(64)			
CTOP			
104(68)			
UTOP			
108(6C) IF RTL REQUIRED: 11 char RTL seg. name, 1 byte seg. reg used (=14)			

11.15 RTL LOADER INFORMATION BLOCK

0(0) seg type #3	1(1) no. of LIB's	2(2)
4(4)		
8(8)		
12(C)	Size of Image as No. of 256-Byte Blocks	
16(10)	No. of entry symbols in RTL	
20(14)		
24(18)		
32(20)	11 char seg name	
44(2C)		
60(3C)		
80(50)	TIME/DATE	
100(64)		
104(68)		
108(6C)	List of item, one per entry: "8 char symbol, 4 byte offset"	

11.16 OVERLAY LOADER INFORMATION BLOCK

0(0) seg type "5"	1(1) no. of LIB's	2(2)
4(4)		
8(8)		
12(C) size of Image in No. of 256-Byte Blocks		
16(10) Start Address of Overlay Area		
20(14)		
24(18)		
32(20) 8 Char Overlay Name		
44(2C)		
60(3C)		
80(50) TIME/DATE		
100(64)		
104(68)		
108(6C)		

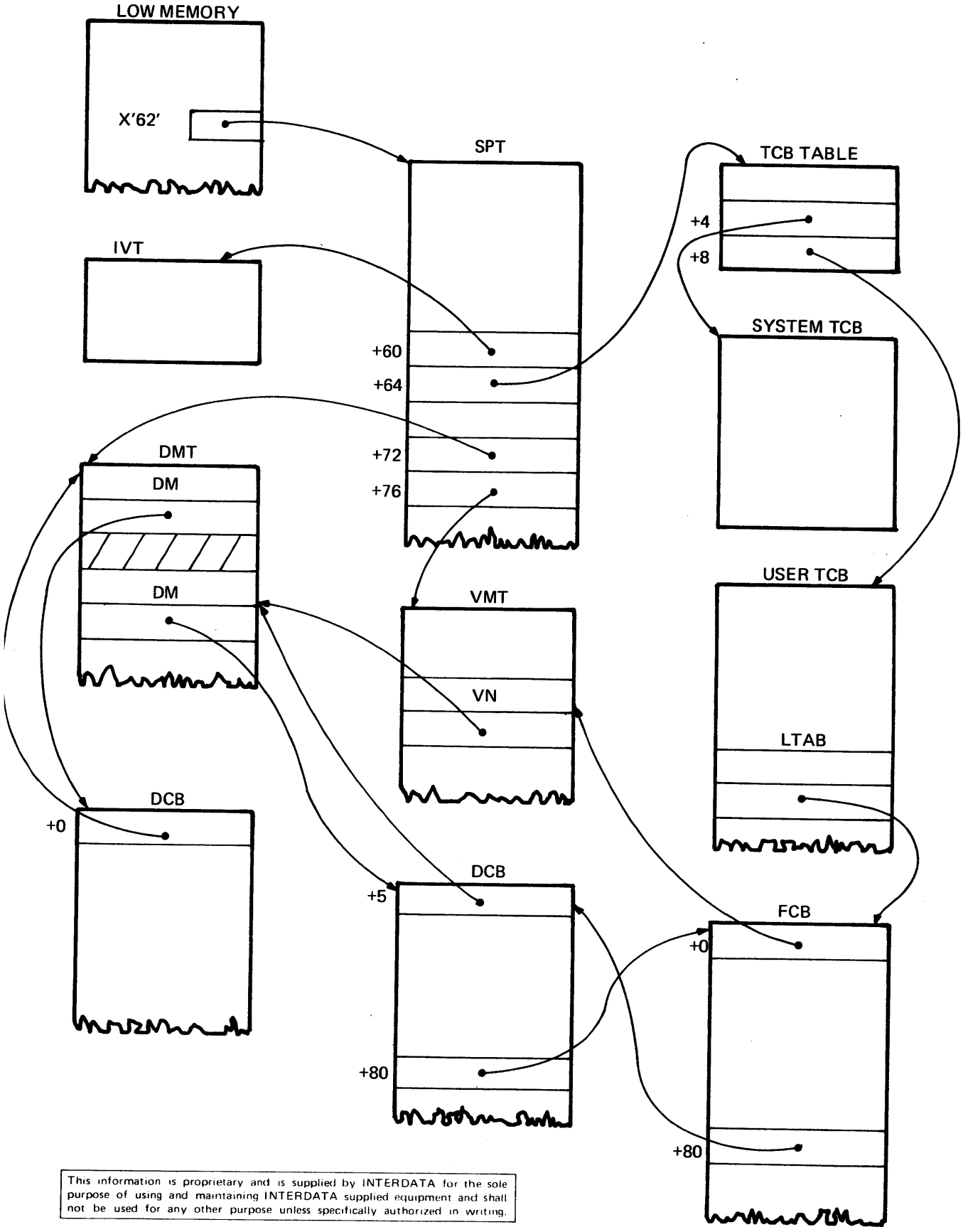


Figure 11-13. System Data Structure Relationships

CUT ALONG LINE

PUBLICATION COMMENT FORM

Please use this postage-paid form to make any comments, suggestions, criticisms, etc. concerning this publication.

From _____ Date _____

Title _____ Publication Title _____

Company _____ Publication Number _____

Address _____

_____

_____

FOLD FOLD

Check the appropriate item.

Error (Page No. ___, Drawing No. _____)
Addition (Page No. ___, Drawing No. _____)
Other (Page No. ___, Drawing No. _____)

Explanation:

FOLD FOLD

Fold and Staple
No postage necessary if mailed in U. S. A.

STAPLE STAPLE

FOLD FOLD

FIRST CLASS
PERMIT No. 22
OCEANPORT, N.J.

BUSINESS REPLY MAIL

NO POSTAGE NECESSARY IF MAILED IN U. S. A.

POSTAGE WILL BE PAID BY:

INTERDATA®

Subsidiary of PERKIN-ELMER
Oceanport New Jersey 07757, U.S.A.

TECH PUBLICATIONS DEPT. MS 53

FOLD FOLD

STAPLE STAPLE