LOGICON 2+2

MONITOR

SPECIFICATION

LOGICON INC.
1075 CAMINO DEL RIO, S.
SAN DIEGO, CALIFORNIA

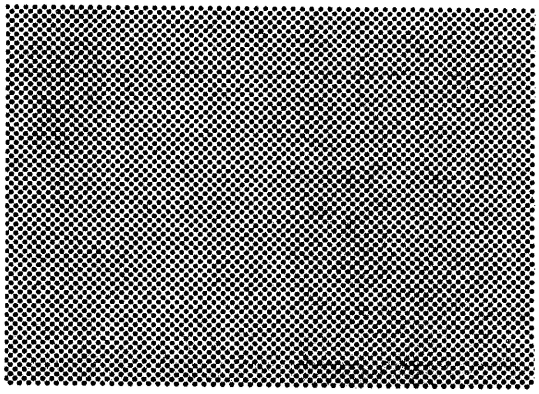
15 December 1970

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

LIST OF ILLUSTRATIONS

LIST OF TABLES



I...

Introduction

1.1 BACKGROUND

This document describes the LOGICON 2+2 MONITOR subsystem. In general, the MONITOR program plus the equipment configuration form a hardware-software environment in which processes may be run for many users at once in a time-sharing mode. Each user has access to a virtual computer, which executes one or more processes in his behalf free from undesired interaction with other users who may be using the physical machine at the same time.

The functions and general organization of the MONITOR in order to accomplish this are presented in the following sections.

1.2 GENERAL REQUIREMENTS

In order to give the appearance of servicing many users processes simultaneously, it is necessary to share physical resources. It is also necessary to protect the MONITOR from malfunctioning user processes, and user processes from each other.

1. All user processes are executed by one physical computer, which can only work on one process at a time. The execution time of this physical processor is time shared among the active user processes, giving "time slices" to each process in turn. The task of selecting the next process to receive processor time is called scheduling.

2. The main memory from which processor instructions and data are accessed during execution is limited in size; it cannot contain all program and data storage for all active processes at once. Hence when the time slice for a process ends, its memory and status must be saved in another storage medium, and the memory and status of the next process

to receive a time slice must be read into main memory. This procedure is called swapping; the storage medium that holds the status of processes that are active but not running is called swapping storage (physically a drum).

To avoid having to always swap all of memory, swapping is done in units of 512 words, each of which is called a page. To allow freedom of overlapping execution and swapping, each 512 word block of physical core memory is independently assigned a location in a user's virtual address space. The correspondence between a virtual address space and a physical memory is called a mapping. As a user's process is executed, the mapping from virtual to physical addresses is implemented in hardware by the Virtual Address Translator (VAT).

3. There may be many processes for different users in various stages of execution in the system at once. Many of these processes will include undebugged programs. They must not, due to programming errors, be allowed to interfere with each other or with the MONITOR program. The prevention of such inadvertant interference is called protection. Protection in the LOGICON 2+2 System has several aspects:

   a. Computer instructions involving input/output, mapping, or protection are privileged and may be executed only in system mode (the MONITOR runs in system mode; all user processes run in user mode). This protection feature is implemented in computer firmware.

   b. Memory access is controlled on a page basis. A given program may have no access, read access, write access, execute access, or any combination of these, to a given page of memory. This feature is implemented by the VAT and by firmware, and is used to prevent user processes from accessing each other's memory or MONITOR memory.

   c. Monitor functions are invoked on behalf of a user process via system calls. The use of system calls is restricted, by means of a capabilities access list belonging to each user, to certain users or certain processes, so that those system calls that could be misused will not fall into the wrong hands. System

programmers will be the only individuals with the capability of causing a system crash.

d. Files of data or programs are very strictly access protected, so that a given user's files may be accessed only by himself and those other users for whom he specifically allowed access.

1.3 USER VIRTUAL COMPUTER

The virtual machine which a user of the LOGICON 2+2 System may utilize has the following properties:

1. A processor capable of executing all non-privileged LOGICON 2+2 processor instructions, plus all system calls permitted by the user's capabilities access list.

2. Up to 128K of swapping storage:

   a. Shared memory – shared memory consists of system code, such as the Executive, Basic Compiler, etc., that is shared among all users. Only one copy of this code exists in swapping storage, and this copy is shared by all users wanting it. It is write protected and non-self modifying.

   b. Private memory – a user's private memory contains his own procedures and data. He controls access restrictions to this memory.

3. Up to eight independently scheduled and protected processes arranged in a tree or fork structure. The reason a user may wish to divide his total task into separate processes is protection: any process may be protected from other undebugged processes running under it in the tree structure. For example, a new BASIC program would be run as a separate process under the BASIC run time system until it was debugged.

4. Each process has a 32K address space and a 16K to 32K working set (depending on main memory size). A user controls the mapping of his 128K of swapping storage into 32K address spaces for each process, and controls the selection of a working set from the address space. The

working set is the memory that is swapped into main memory when the process is to be activated, or executed.

5. Each process has certain dedicated cells for interrupt and trap entrances in its address space as follows:

$Location_8$	Use
0	Normal Entry
1	Continue Entry (resume execution after return to Exec)
2	Interrupt 0
3-10	Interrupts 1 - 6
11	Memory Panic Interrupt
12	Command Abort Interrupt
13	Subsystem Abort Interrupt
14	Stack Overflow Interrupt
15	Stack Underflow Interrupt
16	Illegal Instruction Interrupt
17	Floating Point Overflow Interrupt
20	Floating Point Underflow Interrupt
21	Panic in Subsidiary Process or Panic Abort from Controlling Terminal
22-41	Interrupts 20_8 through 40_8.

NOTE: The interrupts a process may receive are software interrupts controlled by the MONITOR. All hardware interrupts are processed by the MONITOR.

6. There are three principle control tables associated with each process in the system. These tables contain data that links them to resident monitor control tables such as the schedule tables, user control tables, controlling device tables, etc.

A process is defined within the tables by the following elements:

a. Process Status Table (PST)

The PST is used to convey status and control information for creation and monitoring of a subprocess. There

is one PST for each process in the fork structure. This table resides within the creating process for a subprocess. This table communicates with the subprocess through system calls. The PST contains the following information:

- Content of registers: P, S, X, U, A, E, B, T, and L.
- Ephemeral and permanent memory counts.
- Pointer to capabilities list (access to system calls).
- Pointer to relabeling registers (memory control).
- Pointer to user working set bits.
- Interrupt enabled masks.
- File access control bits.
- Status of the process.

The table is set up initially by the creating process. It is updated whenever the creating process requests it for monitoring and control purposes.

b. User Context Block (UCB)

There is one UCB in system address space for each user. The UCB consists of one or two pages (normally one), and contains many tables. The UCB contains information pertinent to the relationship among all processes in the fork structure. It is initialized following login of the user and is modified as each new process is created or destroyed. Further modification can be made through system calls relating to any word in the UCB.

The UCB contains the following information:

For the User -

- Private Memory Table (PMT)
- Other user memory control information
- File control data
- Accounting data

For each process in the fork structure, a Process Environment Table (PET), including -

- PET index of upper fork (creating process)

- PET index of lower fork (subsidiary process)
- PET index of parallel fork (parallel process - created by same creating process).
- Interrupt enabled mask.
- Interrupt active mask.
- Access control bits for open files.
- Pointer to the drum address in PMT of the PCB for this process.
- Pointer to the scheduler queue entry assigned.

Process Status indicates the current condition of the process. It is used primarily by the controlling process to monitor the condition of a subprocess. There are five status conditions currently defined.

Value	Meaning
3	Blocked waiting for file I/O to complete
2	Blocked waiting for terminal I/O to complete
1	Running or ready to run
0	Deactivated waiting for external event
-1	Deactivated by parent
-2	Deactivated by memory panic (interrupt not armed)
-3	Deactivated by illegal instruction panic (interrupt not armed)
-4	Deactivated by ESCAPE (interrupt not armed)
-5	Deactivated by panic in subsidiary process (interrupt not armed)
-6	Destruction in progress

The PET indexes are indexes of UCB items which describe a process. This is the manner in which processes are able to maintain relationships to one another.

The interrupt enabled masks allow processes the ability to field specific interrupts during execution. One bit corresponds to a predefined interrupt (software).

When the interrupt occurs the monitor is able to detect from the mask which process or processes receive the interrupt.

The file access control bits relate to the open user files and provide protection for files among the processes.

The PCB pointer indexes the PMT entry containing the drum address of the PCB for this process.

The PMT contains the drum address and protection bits for all pages of all processes which the user has created. The relabeling registers in the PCB point to PMT entries. The relabeling registers in the PCB are accessed using the upper 6 bits of the 15-bit effective address of a memory location as an index. The relabeling registers are byte addressable. The PMT is word addressable.

Refer to Section V for a more detailed discussion of relabeling registers and the PMT.

c. Process Context Block (PCB)

The PCB contains status and control information for a process. Nine (9) words are the machine registers. These registers hold the register contents at dismissal or deactivation time. When a panic occurs, the registers may be read into the associated PST of the creating process by creating process action. Other information consists of the following:

- Memory use counts.
- Relabeling registers for system and user activations.
- Working set bits for system and user activations.
- Capabilities access bits.
- Memory access bits.
- Scheduler parameters.
- System stack storage.

Memory Use Counts are used to count the number of pages of ephemeral and permanent memory for this process. Acquisition and deletion of memory is affected by these counts. Ephemeral memory is memory that is automatically released when no process is using it.

The relabeling registers select certain PMT pages and map them into the address space of the process.

The relabeling registers, working set bits, and memory protection bits are all divided into 2 parts — system and user. The system part is further subdivided into one part used during system activations, and a part used during user activations. A system activation is one that occurs during performance of a system function, when no user code will be executed. This allows the same 'virtual' address space to be used by both system and user. The physical separation is controlled by the monitor through the PMT and associated ancillary tables.

The working set bits indicate which pages are swapped whenever the process is executed. Each bit corresponds to a relabeling register. Only the memory specified by the working set bits is loaded to reduce swapping time and memory space.

By manipulating the working set bits and relabeling registers, sharing memory and overlaying can be effected in an efficient manner.

The capabilities access bits refer to the system calls. Their use is to provide a convenient method for restricting the capabilities passed to a subsidiary process. If a process attempts to use a system call so restricted, it will be deactivated.

Memory access bits are indicators that control access to a page of memory. Access may be any combination of read, write, or execute. Indirect address references may be made if either read or execute accesses is allowed. Each page has 2 sets of access bits. One set is the maximum access for the page, the second is the currently allowed access. Whenever the page is swapped, the current access bits are loaded into the user map from which the hardware is able to detect violations and interrupt the processing.

The scheduling parameters provide linkage to the monitor schedule control table.

The process control tables are core resident only when one of a user's processes is running or is scheduled next. The UCB is resident when any of a user's

processes are scheduled or unning; the PST is resident only when a parent process is running; and the PCB is resident when the particular process it describes is scheduled or running.

7. Input/Output – Each user communicates with his processes via a terminal (his controlling terminal). He can attach or link to other terminals if they are not otherwise utilized.

   A user may set up a file from card input by submitting a deck with the proper control cards and the proper request form to the system operator. After the cards have been read into the file, he may access the file from an interactive or remote batch terminal.

   A user may set up a file to be printed on the high-speed line printer by entering the appropriate information in a file and requesting that the file be printed. The actual printing will be done by a system process when the printer is available.

   The drum and disk are completely invisible to the user – they are used for swapping and file storage respectively, and do not permit directly user-controlled input/output operations. Magnetic tape is used primarily for file backup, in which application it is also invisible to the user. Magnetic tape may also serve as an interface with other systems, but the mechanism is to have an operator-controlled procedure transfer data from magnetic tape to a system file. Therefore, magnetic tape is also invisible to most users.

   Batch terminals are like interactive terminals, except that the command language is punched on cards, and processing is non-interactive.

8. File System – A user may create, manipulate, control access to, and destroy named files of programs and data arranged in a hierarchical structure which he creates and controls. The file mechanism permits a user to save programs and data from one terminal session to another. He may bring files or parts of files into his address space and working set in order to examine, manipulate, or execute their contents. File access at the MONITOR level is on a page at a time basis.

1.4 OVERVIEW OF MONITOR FUNCTIONS

The Monitor occupies all of CP memory and from 12K to 16K of AP memory. It may not all be core resident. It performs the following functions:

1. System Bootstrap Start – initial program load, initialization, prestart equipment tests, and system startup.

2. Connect and Input Monitoring – monitoring all inactive input lines, testing for user activity, and placing him in communication with the EXECUTIVE.

3. Process Scheduling – Allocation of central processor time to user processes; control of swapping and process execution.

4. Swapping – writing all pages of current process that have been changed since last swap (dirty pages) back to drum; reading all required pages of next process that are not already in core into core; setting the hardware map for the new process.

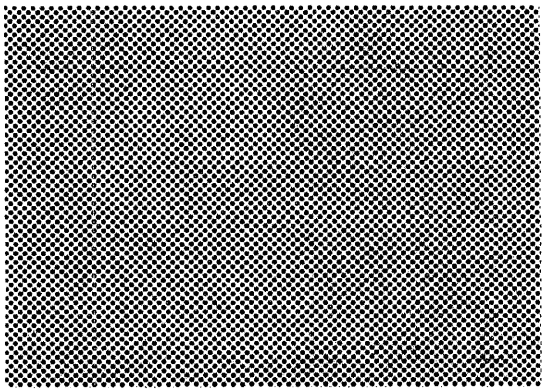
5. Resource Utilization Accounting – maintaining logs of connect time, processor time, storage, and other facilities employed by a user for account billing purposes; maintaining logs of usage frequency and execution time of system code for optimization purposes.

6. System Call Mechanization – responding to user requests made via system calls.

7. Input/Output Control – controlling all physical I/O, both on explicit user request via system calls, and as required by the swapping and file control functions.

8. File System Implementation – Implementing the basic file system including backup and recovery.

9. System Crash Recovery – Responding to all types of system crash conditions, including power down/power up sequences, equipment failures, and catastrophic system failures.



II . . .

System Startup

2.1 GENERAL

System startup involves a bootstrap program load, prestart equipment tests, system initialization, and entering normal system operation.

2.2 BOOTSTRAP PROGRAM LOAD

When power is turned on to a processor, it starts executing microcode at a fixed location (cell 0). It is necessary for the microcode to determine whether the power up sequence is the last half of a power down/power up cycle, (in which case it should trigger the software power up interrupt routine), or a cold start (in which case it should wait for an operator-initiated bootstrap load request). These cases will be distinguished by a time-delay relay on the computer master power switch: If power up was caused by activation of this switch, then the microcode will assume a cold start and will wait for a bootstrap start signal.

Bootstrap program load may be performed from drum, disk, or magnetic tape at operator option. A 512 word page from a reserved location on the selected device will be read into physical page 0 of AP memory, and an interrupt to the CP generated. The CP determines that this is a bootstrap start, and relays the interrupt to the AP, which must execute the program in page 0 to do the following:

1. Load additional programs in AP and CP memory as necessary.

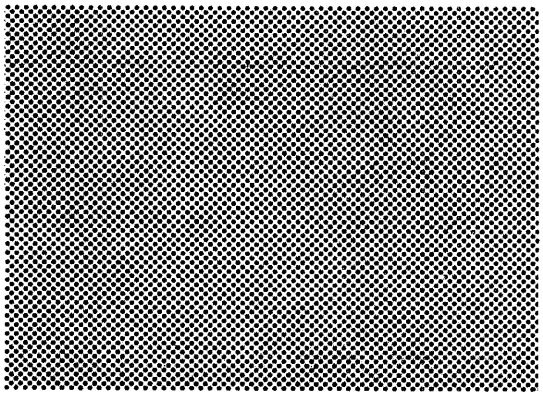
2. Inform the CP that programs are loaded and it can start execution.

3. Test the integrity of the hardware system, notifying the operator of any discrepancies.

4. Test the integrity of programs permanently on the drum, automatically reloading the drum if necessary.

5. Test the integrity of the file system, invoking the recovery sequence if necessary.

6. Load the normal monitor program in core, verify the load, and start normal monitor operation, waiting for the first user to log in.

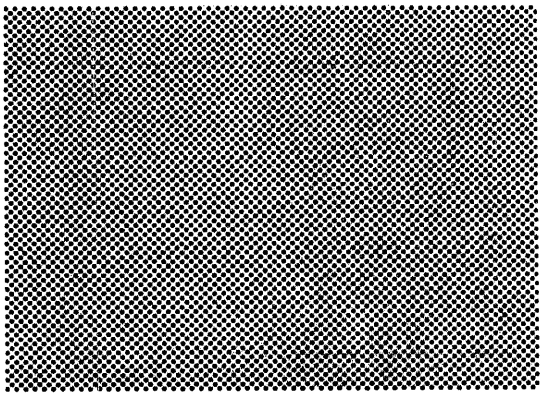


III...

User Activation

3.1 USER ACTIVATION

The Monitor periodically tests all terminals not attached to a user process for signs of user activity. Upon detecting a connect signal followed by a signature character, the Monitor notifies the LISTENER process which establishes the user in the system with the controlling terminal attached, initiates a single process (the User Executive) in the fork structure for that user, and places the user's terminal in communication with his EXECUTIVE.



IV...

Process Scheduling

4.1 GENERAL

The process scheduler (or simply the scheduler) has two primary functions:

1. To allocate certain system resources (mainly processor time) among the various competing processes in such a manner that each gets its "fair" share.

2. To allocate these system resources such that a "reasonable" balance is achieved between efficient system utilization and fast response to trivial requests.

Since the meanings of "fair" and "reasonable" as used above are not clearly defined, the Monitor must allow operational personnel to change resource allocation quickly and easily.

The mechanism for selecting the next process to run employs a set of scheduling tables and a set of queues. The scheduling tables provide an easily modifiable means of associating one of several different classes of service with each process in the system. The queues aid in keeping track of the status of the various processes.

Associated with each process known to the system is a small table called a Scheduler Queue Entry (SQE). The SQE's remain in core and contain enough identifying and control information to keep track of the process and its status when it is not running. Each of the queues is a linked list of SQE's. When the SQE associated with a given process is linked onto some queue, the process is said to be in that queue.

There are three classes of queues in the system: the 'inactive' queue, the 'blocked' queue, and the 'ready' queues. A process is placed in the 'inactive' queue at the explicit request of the process or one of its ancestors, as a result of some panic condition, as a result of an

action at the user's terminal. Once a process has been placed in this queue, some action external to the process is required to get it out again. The action can come from another process or from the user's terminal.

A process is placed in the 'blocked' queue when it requests some service (such as I/O) that the monitor cannot provide immediately. It remains in the 'blocked' queue until the 'activation condition' specified in the SQE is satisfied. At that time it is transferred to one of the 'ready' queues.

4.2 READY QUEUES

There are several 'ready' queues in the system. These are identified by number and each has an associated priority (the lower the number the higher the priority). In addition to the priorities there are other special meanings associated with some of the ready queues.

4.2.1 Queue #1

The first process in the queue is the next process to run. As soon as it is in core and ready to run, the Monitor dismisses the running process (unless it, too, came from queue 1) and turns on the new process. The process will be allowed to run until it exhausts its long quantum or is blocked or inactivated. It is anticipated that this queue will rarely be used.

4.2.2 Queue #2

If this queue is not empty, the Monitor dismisses the running process after a short time quantum (unless it came from queue 1 or 2) and selects the first process in this queue as the next to run. Frequent use of this queue is not anticipated.

4.2.3 Queue #3

The Monitor will try to service each entry in this queue within a specified time, T1, after it becomes ready. T1 will initially be set to 30 drum revolutions (approximately .522 seconds). It can be changed by operational personnel.

4.2.4 Queue #4

The Monitor will try to service each entry in this queue within a specified time, T2, after it becomes ready. T2 will initially be set to 60 drum revolutions (approximately 1.044 seconds). It can be changed by operational personnel.

4.2.5 Queue #5

The Monitor will try to service each entry in this queue within a specified time, T3, after it becomes ready. T3 will initially be set to 120 drum revolutions (approximately 2.088 seconds). It can be changed by operational personnel.

4.2.6 Queue #6 - Queue #n

The Monitor will service entries in these queues in order; first by queue, and then by order of entry into the queue.

A process is placed in a 'ready' queue when it is ready to make use of the AP. For example, a process 'blocked' for I/O would be placed in a 'ready' queue when the I/O action is completed. If the running process exhausts its time quantum without taking any action that causes it to be blocked or inactivated, it is still ready to run and is placed in the appropriate 'ready' queue. This is termed 'quantum overflow.'

The general rule is that the highest priority 'ready' process is the next to run. One notable exception to this is that no two processes for the same user will be in core running or about to run at the same time. (This convention is introduced to facilitate the releasing of memory and to ease certain other control problems.)

4.3 SCHEDULING TABLES

One of the requirements of the system is that it must be possible to assign different classes of service to different processes or users. For example, a user at a remote terminal obviously requires better response than a remote batch processing job. Scheduling tables are used to define these different classes of service.

Each process in the system has an associated scheduling table as defined by a field in the PCB. This table is generally not unique to the process. It is assumed that the system will have a few different classes of service (say half a dozen) and that all processes in a given

class will share the same scheduling table. This approach makes it possible to alter a class of service simply by changing the contents of a table. Only system operational personnel will have a sufficiently privileged status to do this.

Scheduling tables have the format shown in the following examples:

Row	Queue	Short Quantum	Requested Quantum	Long Quantum	Next Row On Overflow
1	3	2	3	10	2
2	4	2	4	10	3
3	6	2	5	20	3

Scheduling table for a user-written interactive process.

Row	Queue	Short Quantum	Requested Quantum	Long Quantum	Next Row On Overflow
1	8	2	20	33	1

Scheduling table for a remote batch process.

The row number is strictly an identifier and is not part of the table. If a table has more than one row, there can be an implicit relationship between the rows of the table and the reason for dismissing the process. In the first table this association might be:

Row 1 - Terminal input
Row 2 - Terminal output buffer full
Row 3 - All other except quantum overflow

The queue specified in the table is the 'ready' queue in which the process will be placed when it is ready to make use of the central processor. If the running process is dismissed for quantum overflow, it will be placed in the specified 'ready' queue immediately. If it is dismissed for taking some action resulting in a 'blocked' condition, it will be transferred from the 'blocked' queue to the specified 'ready' queue when the activation condition specified in the SQE is satisfied. As an example, if a process using the first scheduling table above were dismissed for terminal input, it would go into the 'blocked' queue until an input message was ready at which time it would be placed in 'ready' queue 3.

After a process in a 'ready' queue has been turned on, the general rule is that it is allowed to run for not less than a 'short quentum' and not more than a 'long quantum'. (In the table the quantum lengths are expressed in terms of drum revolutions - approximately 17.4 ms each.)

If the process to follow the running process comes from a higher priority 'ready' queue than the running process, the running process gets a 'short quantum'. If the next process comes from the same queue as the running process, the running process gets a 'requested quantum'. Otherwise the running process gets a 'long quantum'.

Exceptions to this general rule occur when queues 1, 3, 4, or 5 are involved. If the next process comes from queue 1 and the running process comes from any other queue, the running process may not get a full 'short quantum'. In this case the running process is placed back in the 'ready' queue from which it was scheduled.

If the running process and the next process both come from 'ready' queues 3, 4, or 5, a 'variable quantum' is allocated to the running process. The 'variable quantum' will not be shorter than a 'short quantum' nor longer than a 'long quantum', and will be varied in such a manner that the longest possible quantum is granted consistent with servicing all entries in these queues within the times specified for the queues. If the Monitor cannot service all the processes in these queues within the specified times, it will simply service them as fast as it can, granting a 'short quantum' to each process. As a trivial example of the use of variable quanta, consider the case of a running process that came from queue 3 and only one 'ready' process, also in queue 3.

If the running process has a 'requested quantum' of 2 and the next process has a 'requested quantum' of 3, the running process will get a quantum of 12 revolutions ($\frac{2}{2+3}$ x 30 revolutions) or a long quantum, whichever is less.

The last column of the scheduling table designates the row to be used if the running process is dismissed for quantum overflow.

4.4 EXAMPLE

As an aid to understanding the scheduler, let us follow a process from the time it is dismissed until it is allowed to run again. Assume that the process uses the second scheduling table shown in the example and that it is dismissed waiting for input from a file. At the time of

dismissal a Monitor routine selects the scheduling table designated by a field in the Process Context Block. The 'queue number' and the 'requested quantum' are copied from the scheduling table into the SQE (Scheduler Queue Entry). The anticipated size of the working set of memory pages and the activation condition are placed in the SQE and it is linked onto the 'blocked' queue. Part of the swapper is then called to schedule the writing of the process's memory to the drum, and the next process is turned on.

While other processes are running, the SQE's in the 'blocked' queue are checked periodically to determine whether the activation condition has been satisfied for any of them. When the activation condition for the process under consideration is satisfied (i. e., when an input is complete), the SQE is removed from the 'blocked' queue and placed in the appropriate 'ready' queue - queue 8. To lessen the overhead in the AP, most of this work is done by a CP routine.

As other processes ahead of it in the 'ready' queues are run and dismissed, the process we are considering finally gets to the point where it is the second process behind the running process. At this time a request is issued to read its PCB and UCB into core. Barring the use of queues 1 and 2 by processes that become 'ready' before this process is run, reading the PCB and UCB constitutes a commitment to run this process. After the PCB and UCB have been read in, a list of pages in the working set is constructed and placed in the 'swapper's unordered read request list'. When all of these read requests have been honored, a map is constructed for the process reflecting the current locations of the pages in the working set. This work will be done by a CP routine.

Once each drum revolution, the time used by the running process during its current quantum is checked to determine whether quantum overflow has occurred. This check is made using the 'short quantum', the 'long quantum', or the 'variable quantum' as appropriate. If the running process has overflowed its time quantum and the next process is in core and ready to run, the running process will be dismissed. If the next process has not been read in and it can be read in without using any of the memory used by the running process, the running process is allowed to run for another revolution. If reading of the next process is being blocked for lack of memory, the running process is dismissed.

When the process we are considering is in core with its map constructed and the preceding process has been dismissed, our process can become the running process. The process's map is loaded into the hardware

map, the hardware registers are set from the values stored in the PCB, and control of the AP is transferred to the process. It is allowed to run until its time quantum overflows or until it reaches a 'blocked' condition. At this point the cycle begins again.

In order to prevent a user from getting more than his share of the processor, by running multiple processes, whenever a process is activated for a user all other ready processes for that user will be moved to the end of the queue.

The scheduler's goal in servicing 'ready' queues 3-5 is to service each process in these queues within the time specified for each queue. Actually, referring to these queues as three separate queues is somewhat inaccurate. Up until the time the SQE is placed in a 'ready' queue, the queue number is a convenient way of keeping track of the response time requirement for the process. When processes with ready queue numbers 3, 4, or 5 specified in the SQE become ready, they are merged into a single list according to the time each process should receive service.

The processes are serviced in order from this merged list, each receiving a variable length time quantum. A variable quantum has meaning only if the running process and the highest priority 'ready' process both come from queues 3-5. If the highest priority 'ready' process is in queue 1 or 2, the running process will usually get a 'short quantum' or less. If the highest priority ready process is in queue 6 or below, the running process will get a 'long quantum' or a 'requested quantum'. The variable quantum will be computed in such a way that each process gets as much time as possible within the service time constraints on all processes.

> NOTE: The operations of inserting entries into the merged queue and removing them from the queue occur relatively infrequently (i.e., at the rate of servicing of processes). This service rate will generally not exceed 1 process per 2 drum revolutions and will usually be lower than that. It should also be noted that this processing is done by a CP routine, not an AP routine.

4.5 SYSTEM CALLS

Four system calls will be provided for changing scheduling parameters. These are:

RSTC - Read Scheduling Table Contents

This call reads the contents of the specified scheduling table into the specified area in the process's memory.

SSTC - Set Scheduling Table Contents

This call checks the consistency of parameters in a table in the process's memory, and if valid, inserts the values in the specified scheduling table.

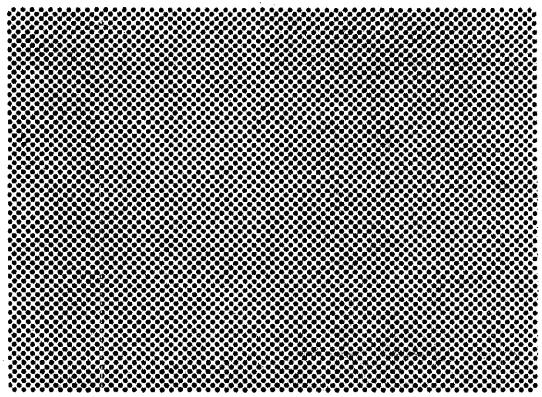
RRTV - Read Response Time Values

This call reads the values of T1, T2, and T3 into the U, A, and X registers.

SRTV - Set Response Time Values

This call sets the values of T1, T2, and T3 from the U, A, and X registers.

All of these calls are highly privileged and will be limited to use by operational personnel.



V...

Process Swapping and Memory Control

5.1 SWAPPING CONTROL

The function of the swapping control routines is to control the flow of information between core and the drum. Some of the objectives of these routines are as follows:

1. Efficient use should be made of the AP. Periods in which there are processes ready to run but none are in core and ready to be turned on are to be kept to a minimum. When possible, swapping should be overlapped with processing.

2. It should be possible to run processes of varying sizes up to and including all the core space available for process execution. Overlapped processing and swapping must automatically occur with small processes and must be partially or totally suspended with large processes.

3. It should be possible to respond rapidly (by human response standards) to a high priority process that becomes ready while another process is running. This implies that the processor must not be committed too far in advance.

It should be apparent that the processing objectives are competing and cannot all be satisfied at once. Hence, the swapping routines must attempt to achieve some "reasonable" balance between them. To the greatest extent possible, this balance should be alterable by changing system parameters.

The routines that make up the swapper can be separated into the following three broad categories.

1. Routines that must be synchronized with process execution. These routines:

   a. save the status of the dismissed process.

b. restore the status of the next process to run.

c. set the hardware map for the next process.

d. turn on the next process.

These routines are executed "in line" by the AP.

2. Routines that can be run in parallel and essentially asynchronously with process execution. These routines:

   a. return to the system the memory used by the process just dismissed.

   b. request the reading of the UCB, PCB and working set pages for the next few processes to run.

3. Routines that must be synchronized with drum revolutions. These routines:

   a. schedule drum reads and writes. Scheduling for the entire n + 1^{st} revolution is done during the n^{th} revolution.

   b. clean up after each drum revolution. Cleanup for the n - 1^{st} revolution is completed during the n^{th} revolution and before scheduling of the n + 1^{st} revolution. Cleanup consists primarily of checking for error free completion of all reads and writes.

   c. update system tables reflecting the current contents of the main memory - primarily the PICT and other related tables.

   These routines are described in detail in Section VII under Drum I/O.

5.1.1 Routines Synchronized with Process Execution

These are routines that must be executed "in line" by the AP in order to dismiss one process and turn on the next to run.

The action to be taken in dismissing a process is essentially the same whether or not the process is currently executing a system call: all that is required is to save the live registers, the location counter and the status register.

5.1.2 Asynchronous Routines

These routines are concerned with requesting pages that belong to processes that are to be run in the near future and returning pages for processes that have run recently but are not scheduled to run again soon. They are executed in the CP in parallel with AP processing. Execution of these routines must be coordinated with process execution but that coordination is sufficiently loose that the routines will be described as being asynchronous with process execution. Obviously pages for processes to run in the near future must be read before the process can run and pages for processes that have run must be returned or eventually the system will run out of core.

5.1.2.1 Returning Memory. When a process is dismissed for any reason the pages used by that process must be returned to the system. The action taken by the Monitor will depend on the status of the page being returned.

1. In all cases the use count in the applicable PICS entry is decremented. This removes the dismissed processes as a reason for holding the page in core. If the use count does not go to zero, the page must be held in core for some other reason and no further action is taken. If the use count goes to zero, there is no longer any reason to hold the page in core.

2. If the page does not have to be held in core, the page's DIRTY bit determines the status of the page and the action to be taken next. If the DIRTY bit is reset, the page has not been written into since it was last read from the drum and there is no reason to write it back. In this case the page is attached to the Clean Page List from which the core block can be reassigned if it is needed for some other function. Until the block is reassigned, the PICT will retain the identity of the page it contains. If the DIRTY bit is set, the page has been written into since it was last read from the drum. Hence, the drum copy is out of date and must be updated by writing the core copy into it. In this case the page is attached to the Dirty Page List from which it will eventually be scheduled for writing to the drum. The identity of the page is retained in the PICT; this is absolutely essential to prevent loss of information. If the page is requested again before the write has occurred the core

copy of the page must be supplied. This in contrast to the case involving the Clean Page List. In that case, the identity of the page is retained only to improve efficiency.

5.1.2.2 Requesting Pages from the Drum. When a process is to run soon, its UCB, PCB, and all the pages in its working set (WS) must be read into core. The drum addresses of the UCB and PCB are retained in system tables; the identity and drum addresses of the WS pages are contained in the UCB and PCB. This implies that reading in the pages of a process is a two step operation, first the control blocks and then the WS pages.

After the UCB and PCB for a given process have been read in, the pages making up the working set are determined as follows. The WS bits in the PCB are checked for ones. Each one means that the corresponding relabeling register (RR) points to a page in the working set. The 8-bit RR contents are used as an index into the PMT. The entry thus selected contains the drum address of the given page. A read request is then issued for that page.

A read request is made by placing an entry in the Swapper's Unordered Read Request List (SURRL). In addition to the drum address, each entry contains information that identifies the requested page as a control block or WS page and marks the first page pertaining to a new process. This information is needed by the drum scheduler.

After all the pages of the WS have been read in, a map is constructed for the process. This is the map that is loaded into the VAT when the process is turned on.

The number of processes that the system will try to read ahead is governed by the sizes of the working sets and a system parameter. No more advanced reading will occur if the memory is full or if the limit on the number of processes to be read in advance has been reached.

5.2 MEMORY CONTROL

The total LOGICON 2+2 System memory is divided among three types of devices: core, drum, and disk. The way in which the monitor utilizes these devices, moves pages of data around among them, and keeps track of where everything is, is discussed in this section.

5.2.1 Disk Storage

A disk storage page is either available or part of some file. Available pages are kept track of in a page availability table. Pages of files not currently open are kept track of in a hierarchical file structure, discussed under "file system." When a file is open its pages are kept track of by the Open File Index Table, discussed later in this section.

5.2.2 Drum Storage

A drum storage page must contain one of the following:

1. Nothing (page is available).
2. Private memory (memory private to one user), which includes his UCB and PCB's.
3. A page of an open file (a file page may also be part of one or more user's private memory as a result of an attach action).

Note that drum pages that contain private memory that is currently core resident may be outdated: if an executing process writes into a core page, then the corresponding drum page is outdated and must not be used again until the core page is copied into it.

A drum page may also be in an empty but unavailable state, waiting for a scheduled I/O operation. This state lasts less than two drum revolutions.

5.2.3 Core Storage

A core storage page may contain one of the following:

1. Nothing (page available).
2. Monitor program or data.
3. Page of an open file.
4. Private user program or data.
5. Nothing, awaiting a scheduled read.

Note that in order to get into core, a file page must be read into some user's private memory. The way in which core and drum pages are managed is discussed in the following pages.

5.2.4 Memory Control Tables

A diagram of the Monitor's principal memory control tables is given in figure 5-1.

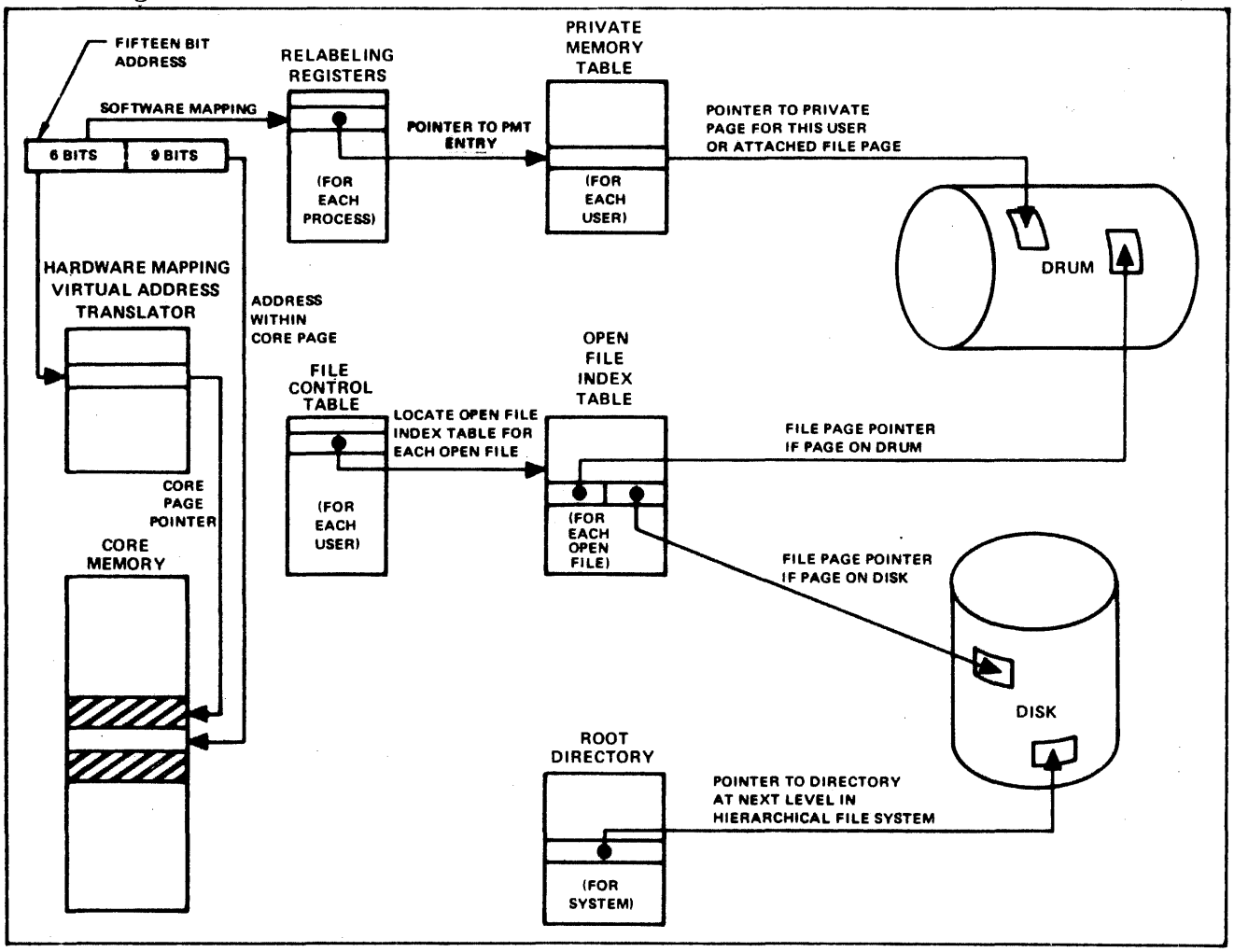


Figure 5-1. Memory Control System

The following discussion relates to figure 5-1.

1. All non-empty disk pages belong to files. They are located through a hierarchical file directory system that locates the file index block for each file, and a list of pointers in the file index blocks that locate individual file pages. The System Root Directory is at the head of the hierarchical directory structure for the entire system. It is normally drum resident.

2. Each user may have up to 32 open files. The user's File Control Table is drum resident and contains pointers to the Open File Index Table (OFIT) for each open file. The OFIT is drum resident and contains pointers, for each page in the file, to drum and/or disk locations of that page. In general, pages not yet accessed will be on the disk but not the drum; pages created since the file was opened will be on the drum but not the disk; and pages that have been accessed will be on both. When the file is closed all pages are written back to the disk.

   NOTE: One Open File Index Table may be pointed to by more than one user's File Control Table (i.e., files may be shared between users, and when they are, the same OFIT is used by both).

3. All non-empty drum pages are either user private memory, pointed to by that user's Private Memory Table; or pages of open files, pointed to by a File Control Table (one per user) or Open File Index Table (one per open file); or both.

4. A user selects pages of his PMT by placing pointers in a set of relabeling registers. There is one set of relabeling registers for each process in the user's fork structure. The relabeling register pointers not only select memory for a process out of PMT; they assign it locations in the virtual address space of the process.

   The i^{th} relabeling register ($0 \leq i \leq 63_{10}$) points (via PMT) to the drum page that occupies locations 512 i to 512 i + 511 in virtual address space.

5. In order to discuss the way in which drum pages are brought into core and accessed by a program, it is convenient to consider the sequence of events that transpires when an instruction within a process being executed references a core location not currently in the working set of the process:

   a. The 15-bit virtual address in the instruction is presented to the Virtual Address Translator (VAT) during the normal course of instruction execution. The VAT hardware determines that this address references a protected page, and causes a protection violation interrupt.

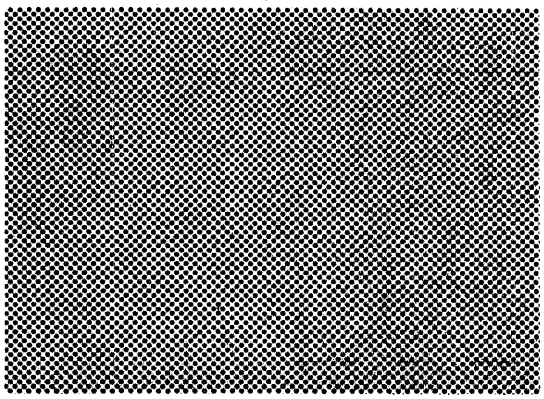
The monitor interrupt routine determines that the reference was to a page not in core, adds the desired page to the processes working set, and dismisses the process.

b. When the process is next scheduled to run, the swapper will find that the working set bit for the subject page is set; will read the relabeling register for that page, the PMT entry pointed to by the relabeling register, and will cause the drum page pointed to by the PMT entry to be read into an available core page.

   When the read is complete, the hardware map (VAT) is set so that the page number (high order 6 bits) of the virtual address will be mapped into the core page selected.

c. When the process is activated, the instruction that caused the previous dismissal is re-executed, and this time successfully accesses the newly added page of memory. The process continues execution, with the whole dismissal-activation cycle totally invisible except for the elapsed time. New pages may be acquired in this manner simply by referencing them.

The acquisition of memory for a user (loading PMT), selection of an address space for a process (loading relabeling registers), and selection of a working set or swapping set from the address space (setting working set bits) are controlled to a large extent by System Calls, discussed in Section X.



VI...

Resource Accounting

6.1 RESOURCE ACCOUNTING

There are two purposes for the accounting function:

1. Determination of the amount of resources a user is utilizing for purposes of billing his account.

2. Determination of system utilization and performance parameters, for purposes of design optimization and parameter adjustment.

With respect to user resource accounting, the Monitor will explicitly measure the following parameters for each user currently logged into the system:

1. Connect time (total elapsed time from login to logout).

2. Central processor time used (summation of all time slices granted to processes on behalf of this user).

3. Core memory pages x time used (page-seconds of memory used), computed by counting pages in the user working set at the start of each time slice, multiplying by the length of the time slice, and summing over all time slices.

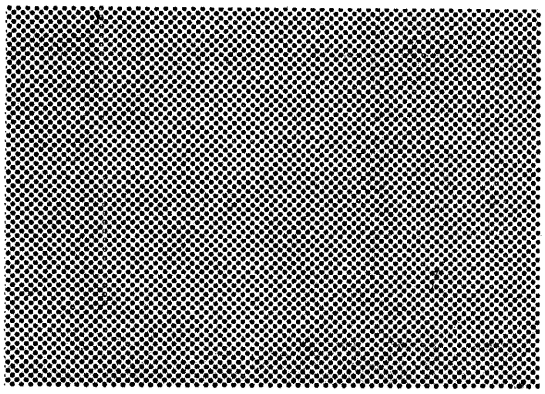
4. Page-seconds of drum storage used.

5. Number of process activations, and dismissals for various I/O functions.

When a user logs out, the Executive process for that user must obtain this running accounting data and add it into the previously accumulated data in the accounting data file.

It may subsequently be accessed by Executive Functions for billing purposes.

For performance monitoring, the Monitor will record frequency of usage and elapsed time for all system calls. This data is for operations personnel only.



VII...

Physical Input/ Output Control

7.1 GENERAL

Physical I/O includes communication with interactive terminals, batch terminals, drum, disk, and tape, as well as any peripherals that may be added to the system at a later date. Communication with interactive and batch terminals is controlled by system calls (these devices communicate directly with user processes). Communication with drum, disk, and tape is controlled by the swapping and file control portions of the Monitor (these devices are invisible to user processes, and interact with user processes only via the file and swapping systems).

7.2 DRUM I/O CONTROL

Drum I/O Control consists of a set of CP routines that schedule drum operations, cleanup internal tables when the operations are complete and update certain Monitor tables to reflect the current contents of core memory. It must be organized so as to minimize the time during which no process may be in execution: i.e., the dead time between time slices caused by the fact that the next process to run is not in core.

Most of the following discussion is built around the swapping and disk transfer part of drum I/O. The drum I/O system must also allow the drum to be initialized (i.e., to have a bootstrap and permanently resident system code written on it). This type of activity will not occur during normal system operation, and hence will not interact with the swapping mechanism. This requires the capability of writing a specific core page onto a specific drum page and marking that drum page as unavailable. This capability is provided by a system call, available only to system programmers.

7.2.1 Drum Scheduler

The Drum Scheduler schedules reads and writes for one revolution of the drum. It is synchronized with the drum and runs as a background task in the CP. It is entered once per drum revolution and when entered it schedules all reads and writes for the next revolution.

7.2.1.1 Drum Scheduler Data Base. The inputs, outputs and internal tables used by the drum scheduler are described in this section. The inputs are as follows:

1. The Swapper's Unordered Read Request List (SURRL). This is a list of UCB's, PCB's and working set pages for processes that are to be run soon. Read requests are placed in this list of the swapper in the order in which they are encountered. The term 'unordered' refers to the fact that the read requests are not ordered by drum sector address. Entries in the SURRL are marked such that it is possible to distinguish between working set pages and UCB/PCB pages. It is also possible to tell when the next page belongs to a different process than the current page.

2. The Dirty Page List (DPL). This is a set of n (n currently equals 32) sublists of dirty pages that should be written on the drum. Each sublist corresponds to one sector on the drum.

3. The Clean Page List (CPL). This is a list of clean core pages that may be overwritten if necessary. If these pages contain any information, the identity of that information is retained in the PICT until they are overwritten.

4. The Released Drum Page Table (RDPT). This table contains one bit for each page on the drum. A one in any bit position means that the corresponding page on the drum should be filled with zeros and its identification word set to indicate that the page now belongs to the system and not to any particular user. Once this has been done, the bit in the RDPT is set to zero and the page is marked as free in the Drum Page Bit Table.

5. The Disk's Unordered Read Request List (DURRL). This is a list of pages that are to be copied from the drum to the disk. This list contains only the next n pages to be copied.

The Drum Scheduler uses the following internal tables:

1. Swappers Ordered Read Request List (SORRL). This is a list of read requests ordered in accordance with the physical layout of the swapping device. In the case of the drum, the SORRL contains 32 entries, one for each drum sector. Each entry will be null or will contain the drum address of a page to be read. At any given time the SORRL will contain addresses of a UCB and a PCB or addresses of working set pages; it will not contain both.

   The UCB and PCB for the n^{th} process must be in core before the read requests for the working set can be issued, hence the read requests for the UCB and PCB for the n^{th} process cannot exist simultaneously with read requests for the working set for the n^{th} process. Main memory must be allocated for the working set for the n^{th} process before any can be allocated for the UCB and PCB for the $n+1^{st}$ process.

   This allocation is done at the latest possible time, namely at the time reads are being scheduled for the next drum revolution. To avoid the possibility of the UCB and PCB of the $n+1^{st}$ process tying up memory needed by the working set of the n^{th} process, all read requests for the working set will be taken from the SORRL before the UCB and PCB read requests are serviced.

2. Disk's Ordered Read Request List (DORRL). This is a short list of read requests taken from the DURRL and ordered according to drum sector address.

3. Scheduled Write List (SWL). This is a list of drum writes that have been scheduled but for which cleanup has not been performed. The actual writing may or may not have been completed, but until the cleanup is done the write is considered to be 'in progress'. This list will be used as a push down stack. Each time a write is scheduled, an entry containing the drum address and the core address is pushed into the stack. When a core page is needed for a read, an entry is popped out of the stack.

   At any given time, the SWL contains entries for at most 2 drum revolutions. During the cleanup cycle for the n^{th} revolution, any entries remaining in the SWL for that revolution represent clean pages. The entries are removed

from the SWL and corresponding entries are made in the Clean Page List (CPL).

The output of the Drum Scheduler is a list of Drum Control Blocks (DCB's). When completed, the list of DCB's for the $n+1^{st}$ revolution is linked to the end of the DCB list for the n^{th} revolution, thus providing the drum with a constant source of work. If it should turn out that there is nothing to do during a drum revolution, the DCB list for that revolution will contain at least one NO-OP with an interrupt to mark the end of the drum revolution.

7.2.1.2 Drum Scheduler Operation. The Drum Scheduler is entered once per drum revolution and schedules all reads and writes that are to be performed during the next revolution. At the time the drum scheduler is entered to schedule the $n+1^{st}$ revolution, cleanup for the $n-1^{st}$ revolution is complete and the n^{th} revolution is in progress. All entries in the Scheduled Write List (SWL) pertaining to the $n-1^{st}$ revolution have been removed.

Requests for service come from five sources:

1. Swapper's Unordered Read Request List (SURRL).
2. Disk's Unordered Read Request List (DURRL).
3. Dirty Page List (DPL).
4. Released Drum Page Table (RDPT).
5. Intra-line editor's read request list.

For convenience, entries in the SURRL and DURRL are ordered according to the drum sector they refer to and are placed in the SORRL (Swapper's Ordered Read Request List) and DORRL (Disk's Ordered Read Request List) respectively. The maximum number of entries in the DORRL is small (e.g., 3 or 4) and is an alterable parameter. The maximum number of entries in the SORRL is the number of sectors on the drum (currently 32). Furthermore, the entries in the SORRL at any given time all pertain to the same process. As an entry is transferred from a URRL to the corresponding ORRL the PICT (Pages In Core Table) is checked to see if the referenced page is already in core. If it is the page is marked as in use (by incrementing the use count in the PICS Table) and the entry is not placed in the ORRL.

The basic strategy employed is to attempt to prepare one DCB for each drum sector in the order that the sectors will pass under the read/write heads. For each sector the drum scheduler checks the RDPT, SORRL,

DORRL, and DPL, in that order looking for a request pertaining to that sector. The RDPT is checked to see if there is a drum page to be released in this sector. If there is one, a DCB is constructed that will write the system ID into the Drum Identification Word and write zeros into the data words of the drum page. This does not require a core block.

If there is a read request (an entry in the SORRL or DORRL) the drum scheduler also checks to see if there is an available core block to read the page into. If there is a read request and an available core block, a DCB will be constructed for the read. If either of these conditions does not hold, the DPL is checked for a dirty page to be written in the sector. If there is one, a DCB is constructed for the write and an entry is pushed into the SWL (Scheduled Write List). If none of the inputs yields work that can be done in a given sector, no DCB will be constructed for that sector. This operation continues until all sectors of the drum have been scheduled.

The SWL (Scheduled Write List) is utilized by the Drum Scheduler to reduce the number of core blocks that are tied up waiting for the drum to revolve. The basic notion is that once a page has been successfully written from a core block to the drum, that block can be reassigned to another page being read from the drum. Use of the Clean Page List as a tool in managing this reassignment proved unsatisfactory. A page cannot be entered in the CPL until the write has successfully completed. This occurs because many different entities take pages from the CPL. This implies that a page written during the $n\text{-}1^{st}$ revolution will not get into the CPL until the n^{th} revolution and at the very earliest will be reassigned to a page being read during the $n+1^{st}$ revolution. The purpose of the SWL is to make the page available for reassignment as soon as possible. It functions as follows:

1. When a write is scheduled, an entry is pushed into the SWL. This indicates that as of some future time this core block will be available for reassignment.

2. When a read is scheduled for a later drum sector (the sectors are scheduled in order), the SWL is checked to see if there is a core block available for reassignment (i.e., any entry in the SWL). If there is, that block is assigned to the read. This means that both a write and a read are now scheduled using the same core block. At this point the read is definitely a future event. The write may be a future, current or past event depending on how long the entry has been in the SWL.

3. The SWL itself does not guarantee that the write completes successfully before the read is begun. This function is handled within the DCB list. All write DCB's will have the "stop and interrupt on error" code set. This effectively guards against reading into the core block before the old contents have been successfully written to the drum. If a write error should occur, one or more revolutions will be lost in overcoming the problem. It is anticipated that write errors will be an extremely rare occurrence.

4. Entries will stay in the SWL for at most two revolutions. During the drum cycle cleanup for any given revolution, any SWL entries pertaining to that revolution represent clean pages assignable to any function and are transferred to the CPL. This is the motivation for implementing the list as a push down stack. It retains the oldest entries in the list when one must be removed thereby improving the probability that a "clean page" will result. Stated another way, it reduces the number of core blocks tied up waiting for the drum to revolve.

Entries in the PICT (Pages In Core Table) are altered by the Drum Scheduler to reflect the current state of core memory. Fields that may be modified are:

1. Drum address.
2. Read-in-progress (RIP) bit.
3. Write-in-progress (WIP) bit.
4. Use count.
5. Dirty (DTY) bit.
6. Link.

The changes these fields may undergo are described below:

1. When a dirty page is scheduled for writing to the drum, the DTY bit is reset and the WIP bit is set. The page is removed from the Dirty Page List.

2. When a page specified in a read request (SURRL or DURRL entry) is found in core, the use count is incremented. If the page was in the Clean Page List, Dirty Page List, or Scheduled Write List it is removed from the list.

3. When a read is scheduled into a clean page, the new drum address is placed in the PICT entry, the RIP bit is set and the Use Count is set to one. The Link is changed to reflect the new drum address.

4. When a read is scheduled into a page that also has a scheduled write, the new drum address is placed in the PICT entry, the RIP bit is set and the Use Count is set to one. The WIP bit is left set. The Link is changed to reflect the new drum address.

7.2.2 Drum Cycle Cleanup

Drum Cycle Cleanup routines perform two main functions. They verify that reads and writes have been completed successfully and update the PICT to reflect the new state of core. The cleanup for the n-1st revolution is completed early in the nth revolution.

The DCB's are processed one at a time in the order that the reads or writes were performed. Each DCB is processed as follows:

1. The status field in the DCB is checked to see that the operation completed correctly. Write errors will always cause an interrupt which will activate a retry mechanism, hence the cleanup routines will not see write errors. Read errors will be seen and will ultimately result in a retry.

2. When a read or a write completes successfully the core block address in the DCB is used as a pointer into the PICS and the RIP or WIP bit is reset.

3. When a drum page is successfully zeroed and returned to the system, the corresponding bit in the Drum Allocation Bit Table is set to reflect page's available status.

After all the DCB's have been processed, the SWL (Scheduled Write List) is checked to see if any entries pertain to the revolution being cleaned up. Those that do are removed from the SWL and corresponding entries are made in the Clean Page List.

7.3 DISK I/O CONTROL

It will quickly become evident to the reader that disk I/O has not yet been designed as thoroughly as drum I/O. The following general features are required:

1. It must handle from 1 to 8 2314 type disk drives (20 tracks/cylinder, 200 cylinders/drive, 6 or 7 sectors/track). The determination of whether 6 or 7 sectors will be written on a track requires a little experimentation with switching times, head alignment, etc. Seven sectors may be somewhat marginal.

2. It must use a "page available" table to tell where functioning empty pages are. Pages may be non-available because of a bad track or because data is written in them.

3. It must use a page assignment algorithm that distributes the pages of a file in an efficient manner. Note that when files are opened or closed, many pages of a single file may be scheduled for transfer to or from the drum at once.

4. It must synchronize its operations with the drum, noting that disk to drum or drum to disk transfers are often critical in system response time. A variable number of core pages will be made available for buffer storage during these transfers.

5. It must allow for read/write requests of specified disk pages to/from specified core pages, for disk initialization, file system backup, and file system recovery. This type of operation will not occur concurrently with normal system time-sharing operation. Such transfers will be controlled by system calls available only to system programmers.

6. It must order read/write requests to take advantage of the overlapped seek capability of different drives as well as the seek time and sector access latency characteristics of individual drives.

7. It must utilize the page identifier/integrity word features of the drum and disk controllers to provide maximum system reliability. This will certainly involve at least single re-read attempts on bad read operations.

7.4 MAGNETIC TAPE I/O CONTROL

Magnetic tape is used only for file backup and recovery, and for transfer of large amounts of data/programs to/from other systems. The magnetic tape I/O system need provide only a relatively conventional magnetic tape handler controlled by system calls, available only to system programmers. The file system will use this handler for its backup and recovery operations.

For intersystem communication, special routines will usually have to be written to handle the specific tape formats involved. The user will have to request system programmers to write this routine.

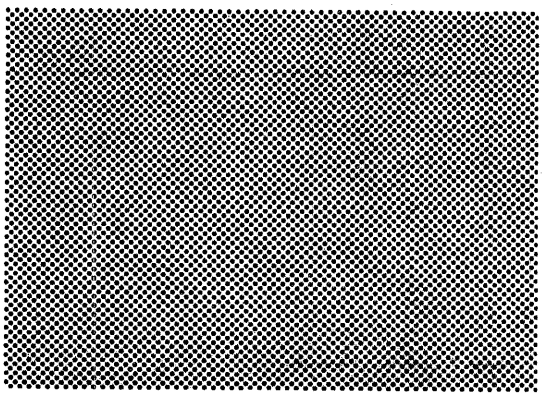
7.5 BATCH AND INTERACTIVE TERMINAL I/O CONTROL

Input/output with batch and interactive terminals is controlled by system calls. The Monitor does not use these devices for its own internal operations, except for messages to the computer operator. These will be handled through the system call mechanism also.

The system calls controlling terminal I/O are discussed in Section X.

7.6 TERMINAL I/O SIMULATION

A terminal input/output simulation capability is provided, via system calls, to assist in debugging processes that communicate with terminals, and to allow multiple subsystems to act as a single subsystem. This capability is also discussed in Section X.



VIII...

File System

8.1 GENERAL

The basic LOGICON 2+2 file system provides a hierarchical structure of named random access files, with access controls by user, user account, and type of access, and with up to 2800 pages (1.4 million words) per file.

Formatted files are not provided by the Monitor: file contents are addressable by file name and page number within the file. Formatted file capability will be supplied within the User Process.

The Monitor does include capabilities for a file backup and recovery system, by allowing a system process to dump changed files or all files on magnetic tape at operator-selected intervals, and restoring files from magnetic tape at operator request. Whenever a file is opened for writing, the date is recorded in the file index block, so that it is always possible to tell when the file was last (potentially) written into.

A user may create, modify or delete files only through the use of the file system. At the level of the file system, a file is formatless. All formatting is done by higher-level modules or by user-supplied programs, if desired. As far as a particular user is concerned, a file has one name, and that name is symbolic. The user may reference an element in the file by specifying the symbolic file name in an open file operation, and the linear index of the element within the file in a subsequent file access operation.

A directory is a special file that is maintained by the file system, and which contains a list of branches. To a user, a branch appears to be a file and is accessed in terms of its symbolic name, which is the user's file name. A branch name need be unique only within the directory in which it occurs. In reality, each branch is a pointer to a file (which may itself be a directory) which is stored in secondary storage.

Each branch contains a description of the way in which it may be used and of the way in which it is being used. This description includes information such as the actual physical address of the file, the time this file was created or last modified, and access control information for the branch (see below). Some of this information is unavailable to the user.

8.2 HIERARCHY OF THE FILE STRUCTURE

The hierarchical file structure is discussed here. For ease of understanding, the file structure may be thought of as a tree of files, some of which are directories. That is, with one exception, each file (e.g., each directory) finds itself directly pointed to by exactly one branch in exactly one directory. The exception is the root directory, or root, at the root of the tree. Although it is not explicitly pointed to from any directory, the root is implicitly pointed to by a fictitious branch which is known to the file system.

A file directly pointed to in some directory is immediately inferior to that directory (and the directory is immediately superior to the file). A file which is immediately inferior to a directory which is itself immediately inferior to a second directory is inferior to the second directory (and similarly the second directory is superior to the file). The root has level zero, and files immediately inferior to it have level one. By extension, inferiority (or superiority) is defined for any number of levels of separation via a chain of immediately inferior (superior) files.

In a tree hierarchy of this kind, it seems desirable that a user be able to work in one or a few directories, rather than having to move about continually. It is thus natural for the hierarchy to be so arranged that users with similar interests can share common files and yet have private files when desired. At any one time, a user is considered to be operating in some one directory, called his working directory. He may access a file effectively pointed to by an entry in his working directory simply by specifying the entry name. More than one user may have the same working directory at one time.

An example of a simple tree hierarchy is shown in figure 8-1. Nonterminal nodes, which are shown as circles, indicate files that are directories, while the lines downward from each such node indicate the entries (i.e., branches) in the directory corresponding to that node. The terminal nodes, which are shown as squares, indicate files other than directories. Letters indicate entry names, while numbers are

used for descriptive purposes only, to identify directories in the figure. For example, the letter "J" is the entry name of various entries in different directories in the figure, while the number "0" refers to the root.

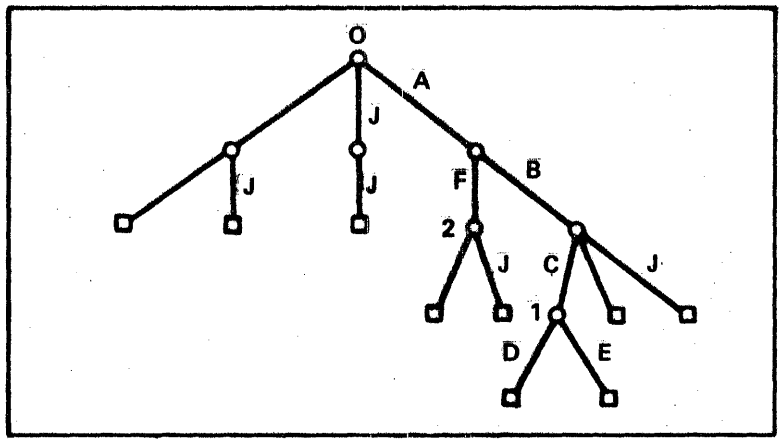


Figure 8-1. An Example of a Hierarchy

An entry name is meaningful only with respect to the directory in which it occurs, and may or may not be unique outside of that directory. For various reasons, it is desirable to have a symbolic name which does uniquely define an entry in the hierarchy as a whole. Such a name is obtained relative to the root, and is called the tree name. It consists of the chain of entry names required to reach the entry via a chain of branches from the root. For example, the tree name of the directory corresponding to the node marked 1 in Figure 8-1 is A:B:C, where a colon is used to separate entry names. (The two files with entry names D and E shown in this directory have tree names A:B:C:D and A:B:C:E, respectively.) In most cases, the user will not need to know the tree name of an entry.

Unless specifically stated otherwise, the tree name of a file is defined relative to the root. However, a file may also be named uniquely relative to an arbitrary directory, as follows. If a file X is inferior to a directory Y, the tree name of X relative to Y is the chain of entry names required to reach X from Y. If X is superior to Y, the tree name of X relative to Y consists of a chain of asterisks, one for each level of

immediate superiority. (Note that, since only the tree structure is being considered, each file other than the root has exactly one immediately superior file.) If the file is neither inferior nor superior to the directory, first find the directory Z with the maximum level which is superior to both X and Y. Then the tree name of X relative to Y consists of the tree name of Z relative to X (a chain of asterisks) followed by the tree name of Y relative to Z (a chain of entry names). For the example of Figure 8-1, consider the two directories marked 1 and 2. The tree name of 1 relative to 2 is :*:B:C, while the tree name of 2 relative to 1 is :*:*:F. An initial colon is used to indicate a name which is relative to the working directory.

In general, any file may be specified by a path name (which may in fact be a tree name, or an entry name) relative to the current working directory. A file may also be specified by a path name relative to the root. In the former case, the path name begins with a colon, in the latter case it does not.

A user may change the currently open directory by an open file system call, specifying the path name of the desired directory either relative to the currently open directory of the root directory.

8.3 ACCESS CONTROL

An initial log-in procedure is utilized in order to establish the identity of the user for accounting purposes.

The log-in procedure involves entering at least an account number and user name. One of the directories immediately subordinate to the root directory will be the account number directory. If a user during sign-on, correctly enters an account number listed in this directory, then the user name directory for that account (immediately subordinate to the account number directory) will be opened, and checked for the user name. If the user correctly enters a user name listed in this directory, then the main file directory for that user name (immediately subordinate to the user name directory) will be opened. Thus when a user completes login, his main file directory is already open. This technique results in users who do not wish to utilize the hierarchical structure of the file system having to do no extra work as a result of its presence.

> NOTE 1: When desired, one or more subaccount levels can be inserted in the hierarchy between the account number directory and the user name directory.

NOTE 2: By providing proper access controls, it is very easy to provide system public (listed in account directory) and account public (listed in user name file) files in this structure.

Each branch in each directory includes a list of users who may access it, and the type of access (read, write, execute) permitted. Due to the hierarchical structure of the access control mechanism, it does no good to permit a user access to a branch out of a directory, if he had no access to the directory in the first place.

8.4 BACKUP AND RECOVERY

The file backup and recovery system provides the following capabilities under operator control:

1. Changed File Dump. All files changed since the last back-up dump are dumped on magnetic tape.

2. Total File Dump. The entire file system is dumped on magnetic tape.

3. Recovery of Entire File System. All files are recovered by reading a total file dump and all change dumps taken since.

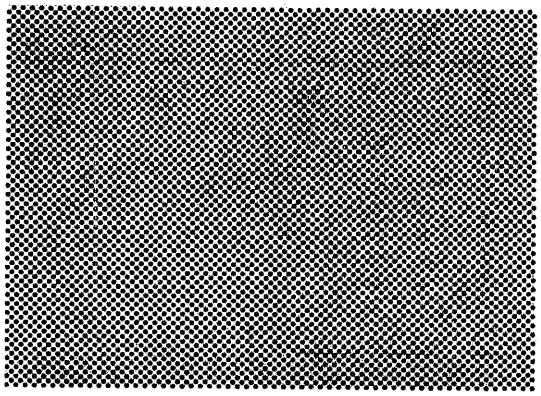
4. Specified File Dump. The specified file is dumped on magnetic tape.

5. Specified File Restore. The specified file is restored from magnetic tape.

Note that these things may not be done while a user has any of the concerned files open.

8.5 SYSTEM CALLS

The file system is manipulated by a user through the use of system calls, describedin Section X.



IX . . .
Crash Recovery

9.1 GENERAL

The Monitor will provide total or partial recovery capability from the following types of system crash situations:

1. Power down/Power up. Power down will be sensed by hardware and responded to by saving all registers in core storage and shutting down input-output in an orderly manner.

   On power up, the Monitor will automatically recover, losing no more than a character or two at each terminal, one line at the printer, and one card at the card reader. Appropriate messages to users and the operator will alert them to the situation to allow total recovery.

2. Non-Recoverable Read Error. On a non-recoverable read error from core or disk or drum, the system will notify the affected user (if only one) and continue. If many users are affected (shared code), all of them and the operator will be notified. If the Monitor is affected, the operator will be notified and the system shut down in an orderly fashion.

3. Malfunctioning Hardware. The Monitor will attempt to maintain system operation, working around the malfunctioning device, and notifying affected users and the operator.

4. Malfunctioning Monitor. If the Monitor detects that it is malfunctioning, via internal consistency checks, watchdog timer interrupt, etc., it will shut the system down and notify the operator.

9.2 RESTART PROVISIONS

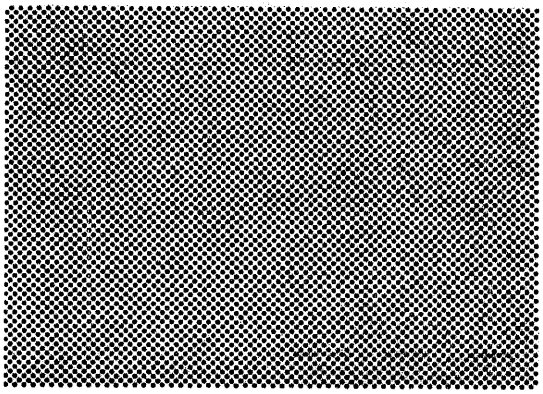
On restart, the Monitor will:

1. Check for good monitor load.
2. Check hardware operation.
3. Check the user and process structure for consistency.
4. Check the file system for consistency.

If all this is OK, the Monitor will try to resume operation. Otherwise, it notifies the operator of the problem and shuts down again.

9.3 PERFORMANCE MONITORING

The Monitor will tabulate all detected hardware and software malfunctions and inconsistencies, whether or not recovery was successful.



X...

System Calls

10.1 GENERAL

The system calls are effectively a set of macros that allow a user process to communicate with the Monitor program. The system calls divide naturally into the following groups:

1. I/O Commands.
2. File Commands.
3. Process Control Commands.

Each of these groups and the specific system calls within the group are discussed in the following paragraphs.

In general, when a System Call is executed by user code, the stack registers contents are saved, the stack registers reset to a system stack (in the PCB of the calling process), the processor registers are saved in the system stack, and a transfer made indirectly through the dedicated system call entry location. This technique of saving registers in a stack allows system calls to be interrupted and to call each other.

System calls are coded so that all parameters are passed through registers rather than following the call instruction in core or some other technique. When there are too many parameters to pass through registers, they are stored at some other location and an address pointer is passed through the registers.

The subset of the system calls that a user may use is defined by two sets of capabilities bits. One set defines those system calls that the users' EXECUTIVE PROCESS has access to, and may pass on to subsidiary processes. The second set defines the system calls that the user may access from his own programs. When a process is created, its parent may pass on any subset of its own capabilities bits. When a

process executes the Set Maximum Capabilities (SMC), its capabilities will be set to the EXECUTIVE capability set (if the calling process is the EXEC), or the user set (if the calling process is not the EXEC).

10.2 INPUT/OUTPUT SYSTEM CALLS

Input/Output System Calls divide into several groups for purposes of discussion.

1. Interactive Terminal and Batch Terminal I/O.
2. Magnetic Tape I/O.
3. Disk I/O.
4. Drum I/O.

Table 10-1 is a complete list of I/O System Calls. The "numbers" given in the table provide an index to detailed descriptions in Appendix A.

The control routines for magnetic tape, disk, and drum are very simple and require no discussion beyond the system call description itself. The Interactive Terminal I/O system requires discussion of echoing, intraline editing, terminal linking, simulated I/O, and attaching terminals to processes.

10.2.1 Echoing

The terminal input system will collect characters from all connected terminal devices and hold them until they are requested by some process, or until buffer overflow occurs. In addition, it automatically performs certain operations on the input characters stream:

1. ESCAPE codes (ASCII code 33) are detected, removed from the input character stream, and passed directly to the appropriate user process as an interrupt. A second escape code with no intervening characters will result in a different interrupt. (Refer to System Calls SEESC and writeup on Interrupt system.)

2. PANIC ABORT codes are detected, removed from the input stream, and passed back to the users Executive process as interrupts.

TABLE 10-1. I/O SYSTEM CALLS

Mnemonic	Name	Number (Ref. Appendix A)
MTC	Magnetic Tape Control	57
TCI	Terminal Character Input	58
TCO	Terminal Character Output	59
ISD	Input From Specified Device	60
OSD	Output To Specified Device	61
RNIC	Read Next Input Character	62
ADV	Attach Device	63
RDV	Release Device	64
TDV	Test Device Status	65
ECH	Set Echo Conditions	66
CDB	Clear Device Buffer	67
RDBZ	Read Device Buffer Size	68
SOT	Set Output Translation	69
RDT	Read Device Type	70
SDT	Set Device Type	71
STCI	Simulate Terminal Character Input	72
STCO	Simulate Terminal Character Output	73
SEESC	Set Escape	74
SIESC	Simulate Escape	75
DOBE	Dismiss Until Output Buffer Empty	76
SLEM	Set Line Editing Mode	92
LLEM	Leave Line Editing Mode	93
BEL	Build Edited Line	94
GSEL	Get Status of Edited Line	95
GEL	Get Edited Line	96
RLS	Read Link Status	98
SLS	Set Link Status	99
EDL	Enable or Disable Links	100

3. The remaining characters are translated by means of an ECHO TABLE to determine:

   a. What character or characters (note carriage-return/line feed) should be printed on the terminal as an echo (full duplex mode), and

   b. Whether or not the character is a break character (i.e., whether the user process should be activated.

Depending on the function being performed, different echoing modes and different break characters are required (see system call ECII, Appendix A).

10.2.2 Intraline Editing

The inclusion of intraline editing capabilities in MONITOR software results in extremely fast and powerful editing capabilities within a single line. It also allows process execution during building of the next edited line. These are five system calls associated with the intraline editing capability –

SLEM - Set line Editing Mode

LLEM - Leave Line Editing Mode

BEL - Start Building of Edited Line

GEL - Get Edited Line

GSEL - Get Status of Edited Line

The entire intraline editing system is discussed in Appendix A.

10.2.3 Terminal Linking

Linking of terminals allows the input characters stream from one terminal to feed more than one process (and have to be echoed on the controlling terminals for those processes), or for the output character stream from a process to feed more than one terminal.

The primary motivation for wanting to link terminals is to enable a systems programmer or customer service engineer to help a customer at a remote location who is having difficulty. To accomplish this, the system programmer needs to be able to see what is being typed in at the customer's terminal, what is being printed on the customer's terminal, and also has to be able to send messages to the customer and enter data and commands as though he were the customer.

A secondary motivation for linking terminals is to enable several "students" to observe at a remote terminal what an "instructor" is doing on his terminal. This could be useful in familiarizing new users with the system.

A third motivation for linking terminals is to simulate a heavy user load on the system when there are only a few terminals attached. By

setting things up properly, this can be accomplished by replicating the input from one terminal and making it appear as though it is coming from several terminals.

Three kinds of links are provided in the terminal tables. They are: 1) input link, 2) advise link, and 3) output link. They behave as follows:

1. Input link. An input link causes the input to be replicated on the terminal specified in the link. The link is a terminal number. Whenever the link is not null, each character coming from the terminal and being packed into the input buffer will also be packed into the input buffer specified in the link. If the linked teletype also has a link which is not null, the character will be packed into the third input buffer as well. Several terminals can be linked together in a chain or in a circle in this fashion. Echoes will be generated for each terminal separately in accordance with the conditions that hold at that particular time for that particular terminal.

2. Advise link. This is similar to input linking but differs in that the character coming from the terminal is not packed into the input buffer for the terminal sending the character. It is only packed into input buffer of the terminal specified in the link. Advise links will point in only one direction and will link only two terminals together.

3. Output link. An output link behaves in a manner analogous to an input link but it deals with characters being packed into the output buffer. Each character packed into the output buffer for the given terminal will also be packed into the output buffer of all other terminals in the chain or circle defined by the output links.

10.2.4 Attaching Terminals

Any user logged into the system is automatically attached to precisely one controlling terminal – the one he logged in on. He may in addition attach other terminals that are not otherwise in use, and thereby allow his processes to communicate with them. The system calls allow a user to determine if a given terminal is already attached, if not to attach it, while it is attached to communicate with it, and later to detach it.

10.2.5 Terminal I/O Simulation

The terminal I/O simulation system allows one process to supply simulated terminal input to another process, and to intercept the terminal output of that other process, for purposes of debugging and testing, as well as combining multiple subsystem programs into a single effective "user" subsystem.

The simulated input/output capability is implemented by bringing a simulated I/O page (SI/OP) into the system working set, having one process place input characters in the page and remove output characters from the page, and cause the process being tested to get its input from the page and place its output in the page.

There are two bits in the Process Context Block of a process that determine whether input/output will function normally or via the simulated I/O page:

1. Simulate Input (SI) bit – if this bit is set then terminal input system calls called by the process will get input from the SI/OP page.
2. Intercept Output (IO) Bit – if this bit is set then terminal output system calls called by the process will pack output characters into the SI/OP.

The SI/OP page is acquired the first time a process is created with the IO or SI bit set. One of the mask bits supplied to CSP specifies propagate SI and IO bits. The bits are initially set by CSP from the PST. Also any SYCL that sets the state of a subsidiary process can set or clear the bits.

A set of system calls analogous to the Terminal I/O calls are implemented to put characters in the SI/OP and get characters out of the SI/OP. These calls are for use by the controlling process: the controlled process uses normal Terminal I/O calls.

Activation of a process waiting for simulated input is handled by setting a bit in the first word of the SI/OP (one for each process) indicating that the process is waiting for input. Whenever a new break character is added to the input buffer the waiting bits are checked and the proper process is activated.

10.3 FILE SYSTEM CALLS

The file system calls allow a user to create and destroy directories and files in a hierarchical system, and to store and access information within the files so created. The file system is discussed in Section VIII. An understanding of the system calls pertaining to files requires the following additional defitions:

1. Opening a file – when a file is opened, its closed file index table is retrieved from the disk, expanded into an open file index table (with pointers to both drum and disk copies of individual file pages), and written on the drum. A pointer to the open file index table is placed in the user's file control table.

   Subsequent accesses to a page of the file require two page references (and hence two time slices), one to get the open file index table and one to get the file page.

2. Activating a file – when a file is activated, its OFIT is added to the working set of the user, so that subsequent file page accesses require only one drum or disk access (and hence only one time slice).

3. Attaching a page – when a page of a file is attached to a process, the file page becomes part of the address space of that process. Any changes made to that portion of the address space of the process are actually changing file contents. Similarly, any changes made to the file page by another process will be "seen" by the process that has the page attached. This is also the procedure for sharing code.

4. Getting and Putting file pages – when a GET action is performed a <u>copy</u> of a file page is brought into the address space of a process. Any changes made to that portion of the address space have no effect whatsoever on the file page.

   Also, any changes made to the file page by other processes, after the copy was made for the GET, have no effect on the copy. When and if the process wishes to change the file, it may perform a PUT operation, which takes a <u>copy</u> of a core page and stores it in the file.

5. Deferred I/O – all file attach, get, and put operations are handled as deferred I/O – the calling process will not be dismissed for I/O until it tries to access a page into which an attach or get has been initiated. Thus a process may issue a number of gets and attaches without dismissal.

   When a core page is slated to receive a file page or a copy of a file page, its access protection bits are set so that it cannot be accessed at all until the file page actually arrives. Any process attempting to access the page in the interim will be dismissed and blocked waiting for I/O.

Table 10-2 is a complete list of File System Calls. The "numbers" given in the table provide an index to detailed descriptions in Appendix A.

TABLE 10-2. FILE SYSTEM CALLS

Mnemonic	Name	Number (Ref. Appendix A)
CRFIL	Create File or Directory	102
OPFIL	Open File	103
RFILN	Read File Name	104
CFILA	Change File Access Protection	105
RFILA	Read File Access Bits	106
PFILA	Propagate File Access Bits	107
DFIL	Delete File	108
DCFIL	Delete Contents of File	109
CLFIL	Close File	110
CAFIL	Close All Files	111
APFIL	Attach Page of File	112
DPFIL	Detach Page of File	113
ACFIL	Activate File	114
DEFIL	Deactivate File	115
GPFIL	Get Page of File	116
PPFIL	Put Page of File	117

10.4 PROCESS CONTROL SYSTEM CALLS

The Process Control System Calls (Table 10-3) divide naturally into the following groups:

1. Process manipulation (creation, activation, destruction, etc.)
2. Memory acquisition, mapping, and working set selection
3. Miscellaneous

10.4.1 Processes and Process Manipulation System Calls

The basic, executing program unit in the system is called a process. A process is a related sequence of instruction executions executed by a user machine processor. A user machine processor consists of the physical processor operating in user mode and a system environment defined by the contents of several tables in the user and process context blocks (UCB and PCB). These tables delineate the augmented instruc-

tion set, accessible memory, and fraction of the physical processor to be allocated to this user machine processor. The state of these tables is dynamically variable as a result of instructions executed by other processes or changes in the running status of the system.

In addition to the system environment information, the context blocks also contain part of the state vector of the process, the remainder of which consists of all the memory allocated to the user machine processor. Every instruction executed for the process causes a change in the state of the process and whenever the process ceases to execute, that is whenever the physical processor is allocated to some other process, its last state is stored away in the state vector. The processor state information is stored away in the context blocks, and the process's memory is stored away on the drum or in unused core. Whenever the physical processor is reassigned to the process, the state vector is used to restore the process to the state it had when the physical processor was last assigned to it.

A user may have up to eight (8) active processes, one of which is always the Executive Process. They are related through a mechanism called the Fork Structure. This mechanism permits controlled relationships in a parental hierarchy among processes of the same user. Their relationships are upper, parallel, or subsidiary depending on the order of creation and the creating process.

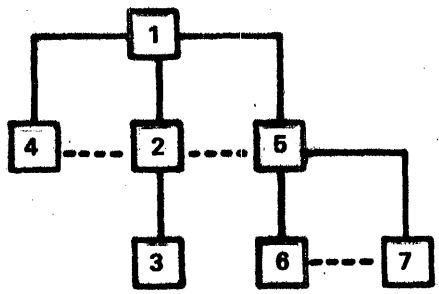
EXAMPLE:

Process 1 creates process 2, 4, and 5

Process 2 creates process 3

Process 5 creates process 6 and 7

The relationship in the fork structure is:



The dotted line indicates parallel processes.

TABLE 10-3. PROCESS CONTROL SYSTEM CALLS

Mnemonic	Name	Number (Ref. Appendix A)
CUEP	Create User Executive Process	1
CSP	Create Subsidiary Process	2
RSPS	Read Status of Subsidiary Process	3
SSPS	Set Status of Subsidiary Process	4
PCSP	Pre-Create Subsidiary Process	5
APSP	Activate Pre-created Process	6
DSP	Deactivate Subsidiary Process	7
DASP	Deactivate All Subsidiary Process	8
DOP	Deactivate Own Process	9
DCP	Deactivate Current Process Until Subsiding Process Causes Panic	10
SMC	Set Maximum Capabilities	11
REDC	Reduce Capabilities	12
MPND	Make Process Non-Dismissable	13
DIST	Dismiss for Specified Time	14
TSPS	Test Subsidiary Process Status	15
TASPS	Test All Subsidiary Process Status	16
DESP	Destroy Subsidiary Process	17
DEOP	Destroy Own Process	18
DEASP	Destroy All Subsidiary Processes	19
CPA	Cause Panic Abort	20
AMEM	Acquire New Memory	25
RMEM	Release Memory	26
RRRC	Read Relabeling Register Contents	27
SRRC	Set Relabeling Register Contents	28
RMA	Read Maximum Access Bits	29
CMAB	Copy Maximum Access Bits	30
RWS	Read Working Set	31
SWS	Set Working Set	32
AWS	Alter Working Set	33
SWSL	Set Working Set Limit	34
RSTC	Read Scheduling Table Contents	38
SSTC	Set Scheduling Table Contents	39
RRTV	Read Response Time Values	40
SRTV	Set Response Time Values	41
SNPA	Set Number of Processes Ahead	42
SNPD	Set Number of Pages for Disk	43

TABLE 10-3. PROCESS CONTROL SYSTEM CALLS (Continued)

Mnemonic	Name	Number (Ref. Appendix A)
CREV	Create Event	45
NOTE	Notify If Event Occurs	46
DELN	Delete Notify Request	47
CAUSE	Cause Event	48
UNCAUS	Uncause Event	49
RIS	Read Interrupt Status	50
SIS	Set Interrupt Status	51
INT	Cause Interrupt	52
CLAI	Clear Active Interrupt	53
CLRAI	Clear and Return from Active Interrupt	54
DELV	Delete Event	55

The status of a process is described by the terms, active and inactive.

Active – The process has been activated from an external source and is under Monitor control. There are two substates of the active condition.

1. Running or ready to run – a process is in this status when it is core resident and executing instructions independently, or ready to be swapped in and run.

2. Blocked – a process is blocked when it is waiting for a file or terminal I/O action.

Inactive – a process is inactive while it is deactivated awaiting activation. It is generally resident on the drum and is invoked only by another process or by an active user.

The process control tables hold all information required to describe and control the process. There are three (3) major tables of description.

1. The process status table (PST) – resides in the creating process. It holds status and control information for a subsidiary process.

2. The user context block (UCB) – resides in memory whenever any process in the fork structure is running. There is one UCB per user. There are up to eight (8) processes described in this block.

3. The process context block (PCB) – resides in memory only when the process that it describes is running. There is one PCB for each active process.

Whether or not the physical processor will be assigned or reassigned to a given process depends on the status of that process. If the status is "active," then the processor will be allocated to the process at some future time (if no action is taken that causes the status to change) independent of any action in the outside world or in some other process. If the status is "inactive," then some action outside of the process is required to return the process to the active status so that it may receive the services of the processor.

The process system calls are concerned with setting up the system environment of a process, defining and determining its state vector and manipulating its status. Create Subsidiary Process (CSP) creates a subsidiary process. Creating a process consists of creating a system environment and initial state vector for the process. Its initial status can be set either to active or inactive. Process B is called subsidiary to process A if B was created by A. A, furthermore, is called the controlling process of B. The subsidiary/controlling relationship among processes is kept track of by means of pointer chains in the context blocks linking processes into a fork structure.

A controlling process keeps track of its subsidiary processes by means of Process Status Tables (PST) in its own memory space. The PST contains space for storing the central registers and pointers to storage areas for information defining the system environment for a subsidiary process. When a process is created (CSP) the PST provides the information for setting the initial state of the process. Thereafter Set Status of Subsidiary Process (SSPS) can be used to change the state of an already existing process. SSPS also points to the PST for the process and gets its information for the new state from the PST. The information in the PST can be updated to give the present state of the subsidiary process by executing Read Subsidiary Process State (RSPS) which reads the present state vector and system environment information into the PST.

Deactivate Subsidiary Process (DSP) sets the status of a subsidiary process to inactive. The process to be deactivated is identified by its PET index in UCB. Deactivate All Subsidiary Processes (DASP)

deactivates all subsidiary processes. Deactivate Own Process (DOP) causes the process executing it to be deactivated. Deactivate All Processes (DAP) points to an entire fork structure and deactivates all the processes within it except the highest level controlling process. This system call is highly privileged and will exist in the user machine processor only for highly privileged processes such as the operator's EXEC.

When a process is created by CSP, the system environment given to the new process cannot exceed that of the creating process. This means that whatever instructions or memory were inaccessible to the old process are inaccessible to the new process. In some cases, however, it would be desirable for a process to act as the controlling process for a process with greater capability. For example, a user might wish to run a system accounting program in order to determine his accounting to date. This program would have to have access to the system accounting information, but we do not want the user program calling the accounting program to have access to the accounting information for obvious reasons. This feature can be implemented by means of the precreated process. This allows a highly privileged process to create a process but not cause it to be linked into the fork structure.

At some later time, another less privileged process can activate the precreated process by supplying the PET index in UCB of the precreated process, causing it to be linked into the fork structure at that point. The system keeps track of the creating process for each precreated process and allows that process to read or manipulate the precreated process's state. Pre Create Subsidiary Process (PCSP) is just like CSP for precreated processes, but it doesn't link the process into the fork structure. Activate Pre Created Subsidiary Process (APSC) activates the precreated process whose number is supplied to this system call.

Unlike matter and energy, processes can be both created and destroyed. When a process is destroyed, all references to it in the system are deleted as is all control storage such as its process context block. After a process has been destroyed it can be recreated if its state has previously been read into a PST and saved, and none of its memory has been released. Note that if a process is destroyed without first being deactivated and having its state read it cannot, in general, be restarted in the same state that it had just prior to being destroyed. Destroy Subsidiary Process (DESP) points to a PST and destroys the corresponding subsidiary process. Destroy Own Process (DEOP) destroys the process executing it. Destroy All Subsidiary Processes (DEASP) destroys all subsidiary processes. Destroy All Processes (DEAP)

destroys all the processes in a fork structure except the highest level controlling process and is highly privileged.

When a process is destroyed, its private memory (other than unshared ephemeral memory) is not released since other processes can access it. But since it was a link in the fork structure, its subsidiary processes can no longer be kept track of. Consequently, when a process is destroyed all subsidiary processes are also destroyed, as are their subsidiary processes, etc.

There are two other process system calls: Dismiss Process (DISP) causes a process to relinquish its physical processor. If that process remains active, it will automatically get the processor back again at some later time. Make Process Non Dismissable (MPND) is highly privileged and makes the process executing it non-dismissable or dismissable. A process that is non-dismissable cannot be dismissed for any reason until it is again made dismissable. This is necessary in some processes that may directly fiddle with system tables when dismissal of them would cause a disaster.

10.4.2 Memory and Memory System Calls

A hardware memory mapping system on the physical machine provides facilities for flexible memory protection and sharing. The time-sharing system has been designed to take full advantage of these facilities and allow protected sharing of memory among processes and users. A user can, furthermore, protect parts of the memory within a particular process from destruction by either external processes or the process itself. Besides being flexible, the memory system is efficient in minimizing duplication of storage and unnecessary swapping. This section is concerned with the structure of memory in the user machine and the system calls for manipulating parts of that structure.

There are three levels of virtual processor memory: the working set, the address space, and swapping storage (PMT). These are, respectively, subsets of each other; that is, the working set is a subset of the address space and the address space allows access to a subset of swapping storage. The address space of a virtual processor is determined by the contents of its relabeling registers (RR). A relabeling register for a given address is selected by the high-order six bits of the address. For example, location 12731_8 references the page given by relabeling register 12_8. The RR's are 8 bits wide and contain indices into the PMT. Each RR corresponds to a 512 word page of virtual memory and the PMT entry pointed to by that RR

gives the location in swapping storage of the actual page. The RR's in the virtual machine are treated just like hardware mapping registers in the real machine, in fact when a process is running, the hardware map is made to correspond to the RR.

Thus, when a process is given control of the processor, the actual locations in core of the memory block given by the PMT entries indicated by the RR are loaded into the hardware map. Thus, for example, suppose relabeling register 0 of a process contains 5. Then when the process references locations 000 through 777_8 it will reference the core image of the swapping storage page pointed to by PMT entry 5.

PMT stands for private memory table. The actual physical quantities in the PMT are drum addresses giving the locations of the blocks when they are not in physical core. When the blocks are in core, a system maintained table gives their current core locations so that the hardware map can be set up. The working set is just a set of active relabeling registers. There is a set of working set bits, one for each relabeling register, indicating which RR are in the current working set. When a process is scheduled to run, only its working set is brought into core.

Three types of private swapping storage (PMT) entries can be acquired by a user: permanent, ephemeral, and attached file pages. The only difference between these is that permanent memory must be explicitly removed from the PMT in order to be released, while ephemeral memory is released automatically when it is no longer in the address space of any process in the fork structure. When a process is created, part of the information supplied as its system environment specifies a maximum number of pages of each type of memory that this process is allowed to acquire. An attempt to acquire memory beyond these limits causes a memory panic.

Memory protection is handled by two types of memory protection bits: current access bits (CAB) and maximum access bits (MAB). There is one set of CAB for every relabeling register and one set of MAB for every PMT entry. Whenever a process acquires memory the MAB are set to the maximum access available and whenever it sets a relabeling register it specifies the CAB. When a process is created its MAB and CAB can be set by its controlling process. The MAB and CAB can also be set by the process system calls that change the processes system environment. Furthermore, a controlling process can copy the MAB from a subsidiary process to its own by executing CMAB (Copy Max Access Bits). Note that whenever a process modifies the protection bits of another process it <u>cannot</u> give away more access capability than

it already has. Also the CAB cannot specify greater access than the MAB of the PMT entry they correspond to.

The following system calls manipulate the relabeling registers, PMT, working set, and protection bits, making it possible to acquire, access, release, and protect memory:

- AMEM (acquire memory) assigns a new PMT entry to the process of the type specified as in input parameter (permanent or ephemeral) and returns the PMT index of the new memory. If the new memory thus acquired is ephemeral it must also be placed into a relabeling register immediately so AMEM requires a relabeling register number and CAB when acquiring ephemeral memory.

- RMEM (release memory) removes an entry from the PMT and clears all relabeling registers of all processes referencing this PMT entry. Write access to a page is required in order to release it.

- RRRC (Read Relabeling Register Contents) returns the CAB and PMT index from the specified relabeling register. There are two ways to set a relabeling register.

- SRRC (Set Relabeling Register Contents) sets the specified relabeling register to the given PMT index and CAB. The PMT index specified to SRRC must be non-empty.

Another way to set a relabeling register while simultaneously acquiring new memory is to reference an address corresponding to an empty relabeling register. This will cause a new PMT entry to be assigned; the relabeling register will be set to the value of that index and the CAB and MAB will be set to RWE. The type of memory thus acquired is either ephemeral or permanent depending on whether or not the process is an ephemeral memory process. A relabeling register set in this way is automatically put into the working set. The page acquired will contain all zeroes. The MAB for a PMT entry can be read by executing RMA (Read Max Access).

The working set of a process is specified by a bit mask with one bit for each relabeling register. If the corresponding bit for a page is set, that page is in the working set. The working set system calls read or manipulate this bit mask:

- AWS (Alter Working Set) sets or clears the specified bit in the WS mask.

- SWS (Set Working Set) specifies the entire working set by supplying the entire mask. SWS returns the new size of the working set and the current limit. If execution of SWS or AWS causes the limit to be exceeded a memory panic is generated.

- RSW (Read Working Set) reads the entire bit mask into the calling processes memory and also returns the working set size and limit.

- SWSL (Set Working Set Limit) is a highly privileged system call that sets the working set limit for the user issuing the call. Normally, only the EXEC will be allowed to execute SWSL. Attaching and detaching file pages also results in PMT entry changes.

10.4.3 Events and Interrupts

The event and interrupt system allows interprocess communication, both within a single fork structure and between users.

10.4.3.1 Software Interrupt System. Software interrupts are controlled via 2 sets of 32 bits in the UCB. They consist of an enable bit and an active bit for each software interrupt level. Interrupts are enabled or disabled by SIS. They are triggered by certain pre-set conditions (see below) or by INT.

When an interrupt is triggered the active bit for that interrupt is set. Whenever the swapper brings in a process it checks the enable bit of the highest level interrupt whose active bits is on. If that enable bit is on, the swapper clears it and causes the interrupt to occur by storing the registers from the PCB in the 6 cells pointed to by the entry to the interrupt routine which is pointed to by the interrupt address cell corresponding to that interrupt level. The swapper then starts execution at the entry+1 of the interrupt routine.

The 32 interrupt address entries are in locations 2 through 33 of the processes address space. The following 9 interrupts are reserved for the given functions. Levels 0-6 are used for interrupts yet undefined which may be assigned a higher priority.

LEVEL	INTERRUPT CAUSE
7	Memory panic in same process.
8	Command abort (first "escape" from terminal).
9	Subsystem abort (second "escape" from terminal).
10	Illegal instruction panic in same process.
11	Stack overflow.
12	Stack underflow.
13	Floating point overflow.
14	Floating point underflow.
15	Panic in subsidiary process, or Panic Abort from controlling terminal.

These interrupts can also be triggered by the INT instruction.

Return from an interrupt routine can be done by a CLAI followed (at any time) by an indirect jump through the cell that saved the L-register. If return is done this way, the process must explicitly restore its own registers before returning. An alternate way to return is the CLRAI system call which clears the active interrupt, reloads the registers and returns. Clearing the active interrupt is done by clearing the active bit and setting the enable bit for that interrupt.

Summary of active and enabled bits:

ACTIVE	ENABLED	MEANING
0	0	Interrupt not enabled.
1	0	Interrupt triggered and active (running).
0	1	Interrupt enabled and not triggered.
1	1	Interrupt triggered and not active.

A panic is the general result of an illegal action in a process. It may also be caused by a process deactivating itself. When any panic condition occurs, the process will be deactivated and the pannic interrupt will be taken in the parent process (if that interrupt is armed). If the parent process was in a deactivated state at the time the panic occurred, it will be activated and placed in a rather high priority queue. If the parent process does not have the "panic interrupt" armed, deactivate it

and try its parent, etc., until a process is found that can process the interrupt.

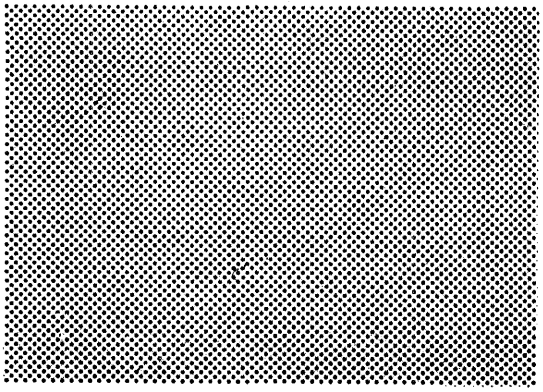
After the parent process receives the panic interrupt it should update the Process Status Table (PST) to determine what happened. If the parent has more than one subsidiary processes, it may have to update all the PST's in order to determine which one caused the panic interrupt.

10.4.3.2 Event Structure. A convenient inter-user event notification system is provided by the event structure and related system calls. The event structure uses the file system: all events defined in the system are catalogued in an Event Directory; each entry in this directory points to a Notify Directory, which is a list of users who wish to be notified when that event occurs.

A user may create or define an event (CREV), ask to be notified if a previously created event occurs (NOTE), delete a notification request (DELN), cause a previously created event to occur (CAUSE), turn off a caused event (UNCAUS), or delete an event definition (DELE). Note that an event may stay "on" for some period of time (CAUSE to UNCAUS). A user with a notify request overlapping any of this "on" time will be notified. The notification is given by an interrupt. The interrupt number associated with a notify request is given in the NOTE system call. Obviously the process must enable the appropriate interrupt or it will never see the notify.

10.4.4 Miscellaneous Calls

The miscellaneous calls include schedule control, swapper control, read clock and date, and read accounting data. These are all relatively self-explanatory.



Appendix A...

System Call Descriptions

SYSTEM CALL DESCRIPTION

MNEMONIC: CUEP **NUMBER:** 1

NAME: Create User Executive Process

PRIVILEGE LEVEL

☒ Listener ☐ Exec ☐ Subsystem ☐ User ☐ Other (Specify)

CAPABILITIES BITS:

Number: 44 **Meaning:** Enable system call

TIMING:

INPUT:

X: address of user directory entry
U: terminal number
A:
E:
Other:

OUTPUT: none
X:
U:
A:
E:
Other:

EXCEPTION CONDITION ACTION

☐ No exception possible ☒ Skip on no exception ☐ Exception causes panic condition

Description: user directory data is bad.

DISMISSAL CONDITIONS: Insufficient memory in core for creation of new UCB and PCB

FUNCTION:

Creates a new user on the system from the data in the user directory and assigns the specified terminal as that users controlling terminal. A new UCB is created as well as a PCB for the new exec.

SYSTEM CALL DESCRIPTION

MNEMONIC: CSP **NUMBER:** 2

NAME: Create Subsidiary Process

PRIVILEGE LEVEL

☐ Listener ☐ Exec ☐ Subsystem ☒ User ☐ Other (Specify)

CAPABILITIES BITS:

Number:	Meaning:
0	Allow system call to be executed
1	Allow setting scheduler table number

TIMING:

INPUT:

X: Address of PST
U:
A: Bit mask
E:
Other:

OUTPUT:

X: Exception condition code on exception return
U:
A: PET index of new process
E:
Other:

EXCEPTION CONDITION ACTION

☐ No exception possible ☒ Skip on no exception ☐ Exception causes panic condition

Description: Bad input data

DISMISSAL CONDITIONS: No memory available for new PCB

FUNCTION:

Creates a subsidiary process by creating a new PCB and a new PET entry in the UCB. The contents of the PCB are set from the PST.

PST

(B)
(T)
(L)
(S)
(X)
(U)
(A)
(E)
(P)
EMC | PMC
CLP
RLP
IMP
STATUS
WSP
SCTBL

INPUT TO CSP

STATUS = 3 ⇒ Blocked waiting for file I/O.
2 ⇒ Blocked waiting for terminal I/O.
1 ⇒ Running or ready to run.
0 ⇒ Deactivated, waiting for external event.
-1 ⇒ Deactivated by parent.
-2 ⇒ Deactivated by mem. panic (interrupt not armed).
-3 ⇒ Deactivated by inst. panic (interrupt not armed).
-4 ⇒ Deactivated by ESCAPE (interrupt not armed).
-5 ⇒ Deactivated by panic in subsidiary process (interrupt not armed).
-6 ⇒ Destruction in process.

EMC = ephemeral memory count
PMC = permanent memory count
CLP = capability list pointer. 0 ⇒ same as for current process.
RLP = relabeling reg. pointer. 0 ⇒ same as for current process.
IMP = interrupt mask pointer. 0 ⇒ same as for current process.
WSP = working set pointer. 0 ⇒ make WS 0 initially.
SCTBL = scheduler table number. 0 ⇒ same as for current process.

The bit mask supplied to CSP decodes as follows:

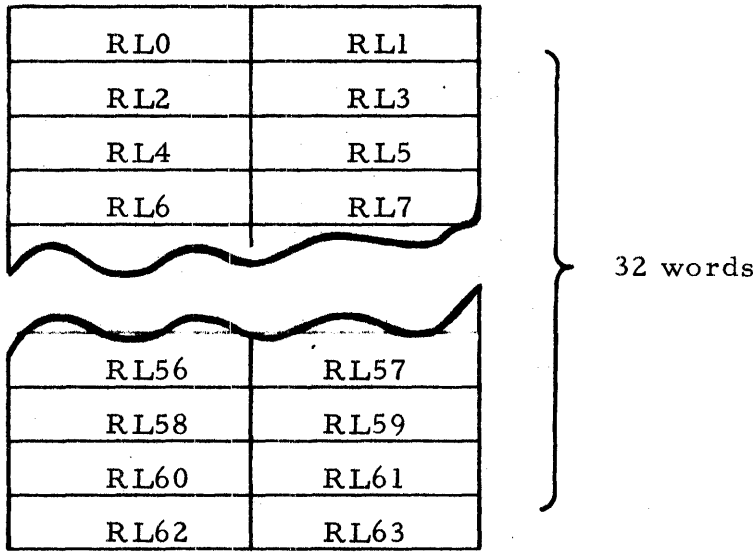
bit 0, 0 ⇒ propagate esc. assignment.
1, 1 ⇒ relabeling has partial format.
2, 0 ⇒ propagate simulated I/O assignment.
3, 1 ⇒ make process ephemeral memory.

CSP will set the status of the process to 1, 0, or -1 only.

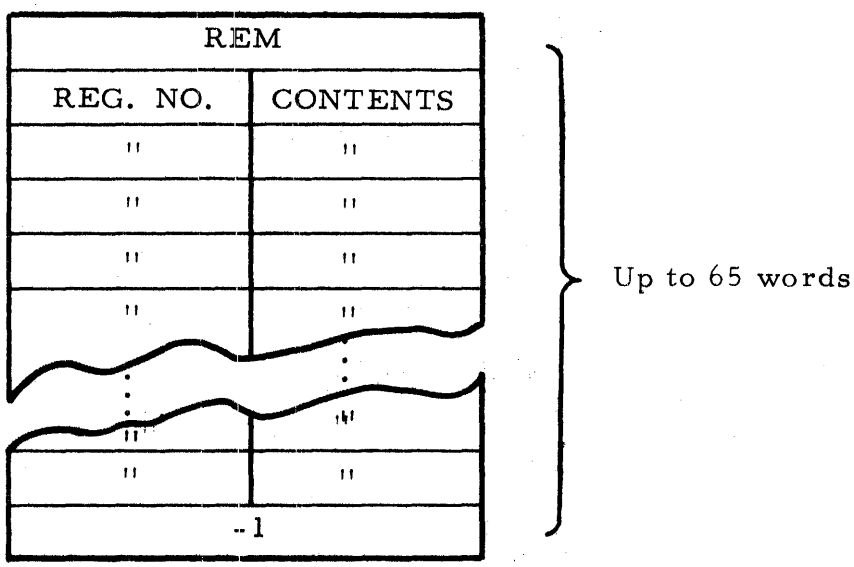
Open file access.

The relabeling formats are:

COMPLETE:



PARTIAL:



REM ≠ 0 ⇒ set remainder of relabeling from current fork (all not set by first part of table). If REM = 0, rest of relabeling is set to 0.

SYSTEM CALL DESCRIPTION

MNEMONIC: RSPS **NUMBER:** 3

NAME: Read Status of Subsidiary Process

PRIVILEGE LEVEL

☐ Listener ☐ Exec ☐ Subsystem ☒ User ☐ Other (Specify)

CAPABILITIES BITS: none

Number: **Meaning:**

TIMING:

INPUT:

X: Address of PST (See CSP for PST format)
U: PET index of subsidiary process
A: Flag
E:
Other:

OUTPUT:

X:
U:
A:
E:
Other: Updated PST

EXCEPTION CONDITION ACTION

☐ No exception possible ☐ Skip on no exception ☑ Exception causes panic condition

Description: Illegal instruction panic on bad PET index

DISMISSAL CONDITIONS:

Dismissal will occur if the PCB of the subsidiary process is not in core.

FUNCTION:

Reads the state of the subsidiary process into the given PST. If (A) ≠0 on input, the relabeling (if read) is read into the partial format; that is the contents of the relabeling register specified in the upper byte of the table is read into the lower byte. In all cases where the PST contains a pointer, if that pointer is zero the corresponding part of the process state will not be read.

SYSTEM CALL DESCRIPTION

MNEMONIC: SSPS **NUMBER:** 4

NAME: Set Status of Subsidiary Process

PRIVILEGE LEVEL

☐ Listener ☐ Exec ☐ Subsystem ☑ User ☐ Other (Specify)

CAPABILITIES BITS: none

Number: 1 **Meaning:** Allow setting schedules table number.

TIMING:

INPUT:

X: PST address
U: PET index of subsidiary process
A: Bit mask (see CSP for description)
E:
Other:

OUTPUT:

X: Exception condition code on exception return
U:
A:
E:
Other:

EXCEPTION CONDITION ACTION

☐ No exception possible ☑ Skip on no exception ☐ Exception causes panic condition

Description: PET index on bad data in PST

DISMISSAL CONDITIONS:

PCB of subsidiary process not in core.

FUNCTION:

Sets the contents of the PCB from the PST. The status of the subsidiary process had not previously created and CSP was executed with the given PST.

SYSTEM CALL DESCRIPTION

MNEMONIC: PCSP **NUMBER:** 5

NAME: Pre-Create Subsidiary Process

PRIVILEGE LEVEL

☐ Listener ☐ Exec ☐ Subsystem ☒ User ☐ Other (Specify)

CAPABILITIES BITS:

Number:	Meaning:
4	Enable system call
5	Allow setting scheduler table number

TIMING:

INPUT:

X: Address of PST
U:
A: Bit mask (see CSP for Description)
E:
Other:

OUTPUT:

X:
U: PET index
A:
E:
Other:

EXCEPTION CONDITION ACTION

☐ No exception possible ☒ Skip on no exception ☐ Exception causes panic condition

Description: Bad input data.

DISMISSAL CONDITIONS: Dismissal if memory unavailable

FUNCTION:

Same as CSP but does not link process into fork structure or schedule it.

SYSTEM CALL DESCRIPTION

MNEMONIC: APSP **NUMBER:** 6

NAME: Activate Precreated Process

PRIVILEGE LEVEL

☐ Listener ☐ Exec ☐ Subsystem ☒ User ☐ Other (Specify)

CAPABILITIES BITS:

Number: 6 **Meaning:** Enable system call

TIMING:

INPUT:

X:
U: PET index
A:
E:
Other:

OUTPUT: none

X:
U:
A:
E:
Other:

EXCEPTION CONDITION ACTION

☐ No exception possible ☐ Skip on no exception ☒ Exception causes panic condition

Description:
Illegal instruction panic on bad PET index

DISMISSAL CONDITIONS:

none

FUNCTION:

Links the specified pre-created process into the fork structure and sets its status to active.

SYSTEM CALL DESCRIPTION

MNEMONIC: DSP **NUMBER:** 7

NAME: Deactivate Subsidiary Process

PRIVILEGE LEVEL

☐ Listener ☐ Exec ☐ Subsystem ☒ User ☐ Other (Specify)

CAPABILITIES BITS: none

Number: **Meaning:**

TIMING:

INPUT:

X:
U:
A: PET index of process to be deactivated
E:
Other:

OUTPUT:

X:
U:
A:
E:
Other:

EXCEPTION CONDITION ACTION

☐ No exception possible ☐ Skip on no exception ☒ Exception causes panic condition

Description: Illegal instruction panic on bad PET index.

DISMISSAL CONDITIONS: None

FUNCTION:

Deactivates the specified subsidiary process by removing it from the scheduler's active queue and setting its status word to inactive. The process will remain inactive until it receives an interrupt or its status is changed by its controlling process.

SYSTEM CALL DESCRIPTION

MNEMONIC: DSAP **NUMBER:** 8

NAME: Deactivate All Subsidiary Processes

PRIVILEGE LEVEL

☐ Listener ☐ Exec ☐ Subsystem ☒ User ☐ Other (Specify)

CAPABILITIES BITS: none

Number: **Meaning:**

TIMING:

INPUT: none

X:
U:
A:
E:
Other:

OUTPUT: none

X:
U:
A:
E:
Other:

EXCEPTION CONDITION ACTION

☒ No exception possible ☐ Skip on no exception ☐ Exception causes panic condition

Description:

DISMISSAL CONDITIONS: none

FUNCTION:

Deactivates all subsidiary processes by removing them from the scheduler's active queue and setting their status words to inactive. Each deactivated process will remain inactive until it receives an interrupt or its status is changed by its controlling process.

SYSTEM CALL DESCRIPTION

MNEMONIC: DOP **NUMBER:** 9

NAME: Deactivate Own Process

PRIVILEGE LEVEL

☐ Listener ☐ Exec ☐ Subsystem ☒ User ☐ Other (Specify)

CAPABILITIES BITS: none

Number: **Meaning:**

TIMING:

INPUT: none

X:
U:
A:
E:
Other:

OUTPUT: none

X:
U:
A:
E:
Other:

EXCEPTION CONDITION ACTION

☒ No exception possible ☐ Skip on no exception ☐ Exception causes panic condition

Description:

DISMISSAL CONDITIONS: none

FUNCTION:

Deactivates the process that executes the call by removing it from the schedulers active queue and setting its status word to inactive. The process will remain inactive until it receives an interrupt or its status is changed by its controlling process.

SYSTEM CALL DESCRIPTION

MNEMONIC: DCP **NUMBER:** 10

NAME: Deactivate Current Process Until Subsidiary Process Causes Panic

PRIVILEGE LEVEL

☐ Listener ☐ Exec ☐ Subsystem ☒ User ☐ Other (Specify)

CAPABILITIES BITS: none

Number: **Meaning:**

TIMING:

INPUT: none

X:
U:
A:
E:
Other:

OUTPUT: none

X:
U:
A:
E:
Other:

EXCEPTION CONDITION ACTION

☐ No exception possible ☐ Skip on no exception ☒ Exception causes panic condition

Description: Illegal instr. panic if process is non-dismissable.

DISMISSAL CONDITIONS: none

FUNCTION:

Arms the "panic in subsidiary process" interrupt and deactivates the current process without triggering the "panic in subsidiary process interrupt in its parent process.

SYSTEM CALL DESCRIPTION

MNEMONIC: SMC **NUMBER:** 11

NAME: Set Maximum Capabilities

PRIVILEGE LEVEL

☐ Listener ☐ Exec ☐ Subsystem ☒ User ☐ Other (Specify)

CAPABILITIES BITS:

Number: 41 **Meaning:** Enable system call

TIMING:

INPUT: none

X:
U:
A:
E:
Other:

OUTPUT: none

X:
U:
A:
E:
Other:

EXCEPTION CONDITION ACTION

☒ No exception possible ☐ Skip on no exception ☐ Exception causes panic condition

Description:

DISMISSAL CONDITIONS:

none

FUNCTION:

Set the capabilities bits in the PCB from the User Directory.
If this is the EXEC process, set from the EXEC capabilities bits.
If this is any other process, set from the user capabilities bits.

SYSTEM CALL DESCRIPTION

MNEMONIC: REDC **NUMBER:** 12

NAME: Reduce Capabilities

PRIVILEGE LEVEL

☐ Listener ☐ Exec ☐ Subsystem ☒ User ☐ Other (Specify)

CAPABILITIES BITS:

Number: 41 **Meaning:** Enable System Call

TIMING:

INPUT:

X: Capabilities Table Address
U:
A:
E:
Other:

OUTPUT: none

X:
U:
A:
E:
Other:

EXCEPTION CONDITION ACTION

☒ No exception possible ☐ Skip on no exception ☐ Exception causes panic condition

Description:

DISMISSAL CONDITIONS:

None

FUNCTION:

AND the table pointed to in the call with the current capabilities access bits.

SYSTEM CALL DESCRIPTION

MNEMONIC: MPND **NUMBER:** 13

NAME: Make Process Non-dimissable

PRIVILEGE LEVEL

☐ Listener ☒ Exec ☐ Subsystem ☐ User ☒ Other (Specify) System Programmers and system calls

CAPABILITIES BITS:

Number: 3 **Meaning:** Enable the system call

TIMING:

INPUT:

X:
U:
A: Flag
E:
Other:

OUTPUT:

X:
U:
A:
E:
Other:

EXCEPTION CONDITION ACTION

☒ No exception possible ☐ Skip on no exception ☐ Exception causes panic condition

Description:

DISMISSAL CONDITIONS: none

FUNCTION:

Set the non-desmissable status of the process according to the flag in A.

Flag: <0 = make process non-dismissable
≥ 0 = make process dismissable

SYSTEM CALL DESCRIPTION

MNEMONIC: DIST **NUMBER:** 14

NAME: Dismiss for specified time

PRIVILEGE LEVEL

☐ Listener ☐ Exec ☐ Subsystem ☒ User ☐ Other (Specify)

CAPABILITIES BITS: none

Number: **Meaning:**

TIMING:

INPUT:

X:
U:
A: Time in nsec.
E:
Other:

OUTPUT:

X:
U:
A:
E:
Other:

EXCEPTION CONDITION ACTION

☒ No exception possible ☐ Skip on no exception ☐ Exception causes panic condition

Description:

DISMISSAL CONDITIONS:

unconditional

FUNCTION:

Dismisses process as if time quantum had run out. Process will be placed back in ready queue when the given dismissal time is exceeded. Note that the given time is a lower bound on the actual dismissal time.

SYSTEM CALL DESCRIPTION

MNEMONIC: TSPS **NUMBER:** 15

NAME: Test Subsidiary Process Status

PRIVILEGE LEVEL

☐ Listener ☐ Exec ☐ Subsystem ☒ User ☐ Other (Specify)

CAPABILITIES BITS: none

Number: **Meaning:**

TIMING:

INPUT:

X:
U: PET index
A:
E:
Other:

OUTPUT:

X:
U:
A: Process status word
E:
Other:

EXCEPTION CONDITION ACTION

☐ No exception possible ☒ Skip on no exception ☐ Exception causes panic condition

Description: If specified process is non-existent or is not subsidiary to this one.

DISMISSAL CONDITIONS: none

FUNCTION:

Reads process status word (see CSP) into A.

SYSTEM CALL DESCRIPTION

MNEMONIC: TASPS **NUMBER:** 16

NAME: Test All Subsidiary Processes Status

PRIVILEGE LEVEL

☐ Listener ☐ Exec ☐ Subsystem ☑ User ☐ Other (Specify)

CAPABILITIES BITS: none

Number: **Meaning:**

TIMING:

INPUT:

X: Pointer to 7-word table
U:
A:
E:
Other:

OUTPUT:

X:
U:
A: Mask specifying subsidiary processes
E:
Other:

EXCEPTION CONDITION ACTION

☒ No exception possible ☐ Skip on no exception ☐ Exception causes panic condition

Description:

DISMISSAL CONDITIONS:

FUNCTION:

For every subsidiary process, this call reads the status word into the table and sets a bit in A. For subsidiary process with PET index equal to n, the status word will be stored in word n of the table and bit n of the A register will be set. All other bits in A will be cleared.

SYSTEM CALL DESCRIPTION

MNEMONIC: DESP **NUMBER:** 17

NAME: Destroy Subsidiary Process

PRIVILEGE LEVEL

☐ Listener ☐ Exec ☐ Subsystem ☒ User ☐ Other (Specify)

CAPABILITIES BITS: none

Number: **Meaning:**

TIMING:

INPUT:

X:
U: PET index of subsidiary process
A:
E:
Other:

OUTPUT: none

X:
U:
A:
E:
Other:

EXCEPTION CONDITION ACTION

☐ No exception possible ☐ Skip on no exception ☒ Exception causes panic condition

Description: Illegal instruction panic if PET index is illegal.

DISMISSAL CONDITIONS: none

FUNCTION:

This call destroys the specified process by releasing its PCB and removing it from the fork structure. It may be used either to destroy a normal subsidiary process or to destroy a precreated process linked into the fork structure immediately below the calling process. It may also be used by the creating process to destroy a precreated process. Whenever a process is destroyed, all of its subsidiary processes are also destroyed.

SYSTEM CALL DESCRIPTION

MNEMONIC: DEOP **NUMBER:** 18

NAME: Destroy Own Process

PRIVILEGE LEVEL

☐ Listener ☐ Exec ☐ Subsystem ☒ User ☐ Other (Specify)

CAPABILITIES BITS:

Number: 7 **Meaning:** Enable system call

TIMING:

INPUT: none

X:
U:
A:
E:
Other:

OUTPUT: none

X:
U:
A:
E:
Other:

EXCEPTION CONDITION ACTION

☒ No exception possible ☐ Skip on no exception ☐ Exception causes panic condition

Description:

DISMISSAL CONDITIONS: none

FUNCTION:

Destroys the process making the call along with all of its subsidiary processes. This triggers a panic in the parent process. When the Exec destroys itself (after user logout), all memory is released, files are closed, attached I/O devices are detached, etc.

SYSTEM CALL DESCRIPTION

MNEMONIC: DEASP **NUMBER:** 19

NAME: Destroy All Subsidiary Processes

PRIVILEGE LEVEL

☐ Listener ☐ Exec ☐ Subsystem ☒ User ☐ Other (Specify)

CAPABILITIES BITS: none

Number: **Meaning:**

TIMING:

INPUT: none

X:
U:
A:
E:
Other:

OUTPUT: none

X:
U:
A:
E:
Other:

EXCEPTION CONDITION ACTION

☒ No exception possible ☐ Skip on no exception ☐ Exception causes panic condition

Description:

DISMISSAL CONDITIONS:

FUNCTION:

Destroys all subsidiary processes. If no subsidiary process exist, this call is a NO-OP.

SYSTEM CALL DESCRIPTION

MNEMONIC: CPA **NUMBER:** 20

NAME: Cause Panic Abort

PRIVILEGE LEVEL

☐ Listener ☐ Exec ☐ Subsystem ☐ User ☒ Other (Specify) Priv. Erec

CAPABILITIES BITS:

Number: 2 **Meaning:** Enable system call

TIMING:

INPUT:

X:
U:
A: User number
E:
Other:

OUTPUT: None

X:
U:
A:
E:
Other:

EXCEPTION CONDITION ACTION

☐ No exception possible ☒ Skip on no exception ☐ Exception causes panic condition

Description: No skip if user number invalid

DISMISSAL CONDITIONS: None

FUNCTION: Causes a "panic abort" interrupt to be taken in the EXEC process of the designated user. (This will occur only if the interrupt is armed; the EXEC is expected to run with this interrupt armed all the time). This call will have the same effect as striking the "panic abort" key on the designated user's terminal. It can be used at any time including some times when the "panic abort" key might not be recognized, such as when the user is in a binary input mode.

SYSTEM CALL DESCRIPTION

MNEMONIC: AMEM **NUMBER:** 25

NAME: Acquire New Memory

PRIVILEGE LEVEL

☐ Listener ☐ Exec ☐ Subsystem ☒ User ☐ Other (Specify)

CAPABILITIES BITS:

Number: 9 **Meaning:** Enable system call

TIMING:

INPUT:

X: Relabeling Register Number (if request is for ephemoral memory)
U: Memory Type Flag in bit 15
A: Current Access (if request is for ephemoral memory)
E:
Other:

OUTPUT:

X:
U: PMT index of new memory page
A:
E:
Other:

EXCEPTION CONDITION ACTION

☐ No exception possible ☐ Skip on no exception ☒ Exception causes panic condition

Description: Illegal instruction if input parameters are bad. Memory panic of amount of memory allowed is exceeded.

DISMISSAL CONDITIONS: No physical memory available

FUNCTION:

Acquires a new page of memory. If the register is for ephemeral memory, the given relabeling register is loaded with the new PMT index. Otherwise, the PMT index is only returned in A.

Memory type flag:

0 → Permanent memory
1 → Ephemeral memory

SYSTEM CALL DESCRIPTION

MNEMONIC: RMEM **NUMBER:** 26

NAME: Release Memory

PRIVILEGE LEVEL

☐ Listener ☐ Exec ☐ Subsystem ☒ User ☐ Other (Specify)

CAPABILITIES BITS:

Number: 10 **Meaning:** Enable system call

TIMING:

INPUT:

X:
U: PMT index of page to be released
A:
E:
Other:

OUTPUT: none

X:
U:
A:
E:
Other:

EXCEPTION CONDITION ACTION

☐ No exception possible ☐ Skip on no exception ☒ Exception causes panic condition

Description: Illegal instruction if PMT index is invalid or if write access is not allowed to the given page.

DISMISSAL CONDITIONS: none

FUNCTION:

1. Removes the designated entry from the PMT.
2. Releases the drum page designated by the PMT entry.
3. Removes all references to the PMT entry from the Relabeling Registers of all process belonging to the user.
4. Sets the maximum access allowed to this PMT entry to zero (no access) for all process belonging to the user.

SYSTEM CALL DESCRIPTION

MNEMONIC: RRRC **NUMBER:** 27

NAME: Read Relabeling Register Contents

PRIVILEGE LEVEL

☐ Listener ☐ Exec ☐ Subsystem ☒ User ☐ Other (Specify)

CAPABILITIES BITS: none

Number: **Meaning:**

TIMING:

INPUT:

X: Relabeling Register Number
U:
A:
E:
Other:

OUTPUT:

X:
U: Contents of Relabeling Register
A: Current Access Bits
E:
Other:

EXCEPTION CONDITION ACTION

☐ No exception possible ☐ Skip on no exception ☒ Exception causes panic condition

Description: Illegal instruction if Relabeling Register number is invalid.

DISMISSAL CONDITIONS: none

FUNCTION:

Reads the contents (a PMT index) of the specified U. Bits 13-15 of the U register specify Read, Write, and Execute access respectively. A "1" in any of these positions indicates that the corresponding access is allowed.

SYSTEM CALL DESCRIPTION

MNEMONIC: SRRC **NUMBER:** 28

NAME: Set Relabeling Register Contents

PRIVILEGE LEVEL

☐ Listener ☐ Exec ☐ Subsystem ■ User ☐ Other (Specify)

CAPABILITIES BITS:

Number: 11 **Meaning:** Enable System Call

TIMING:

INPUT:

X: Relabeling Register Number New Relabeling Register Contents (a PMT index)
U:
A: New "current access" bits
E:
Other:

OUTPUT: none
X:
U:
A:
E:
Other:

EXCEPTION CONDITION ACTION

☐ No exception possible ☐ Skip on no exception ■ Exception causes panic condition

Description: Illegal instruction if Relabeling Register number invalid, Relabeling Register contents point to null PMT entry, or current access exceeds maximum access.

DISMISSAL CONDITIONS: none

FUNCTION:

Loads the PMT index specified in U into the Relabeling Register specified in X. Sets the current access bits to the values specified in bits 13-15 of A and resets the working set bit. Bits 13-15 of A specify Read, Write, and Execute access respectively. A "1" in any of these positions indicates that the corresponding access is allowed.

SYSTEM CALL DESCRIPTION

MNEMONIC: RMA **NUMBER:** 29

NAME: Read Maximum Access bits

PRIVILEGE LEVEL

☐ Listener ☐ Exec ☐ Subsystem ☒ User ☐ Other (Specify)

CAPABILITIES BITS: none

Number: **Meaning:**

TIMING:

INPUT:

X:
U: PMT index
A:
E:
Other:

OUTPUT:

X:
U:
A: Maximum Access Bits
E:
Other:

EXCEPTION CONDITION ACTION

☐ No exception possible ☒ Skip on no exception ☒ Exception causes panic condition

Description: Illegal instruction if PMT index is invalid. No skip if PMT entry is null.

DISMISSAL CONDITIONS: none

FUNCTION:

Reads the "Maximum Access" bits into bits 13-15 of A. These bits specify Read, Write, and Execute access respectively. A "1" in any of these positions indicates that the corresponding access in allowed. The maximum aceess bits specify the maximum access that the requesting process will be granted to the designated page.

SYSTEM CALL DESCRIPTION

MNEMONIC: CMAB **NUMBER:** 30

NAME: Copy Maximum Access Bits

PRIVILEGE LEVEL

☐ Listener ☐ Exec ☐ Subsystem ☒ User ☐ Other (Specify)

CAPABILITIES BITS: none

Number: **Meaning:**

TIMING:

INPUT:

X:
U: PET index of sbsidiary process
A:
E:
Other:

OUTPUT: none

X:
U:
A:
E:
Other:

EXCEPTION CONDITION ACTION

☐ No exception possible ☐ Skip on no exception ☒ Exception causes panic condition

Description: Illegal instruction if PET index is invalid

DISMISSAL CONDITIONS: PCB of the subsidiary process is not in core.

FUNCTION:

Logically OR's the maximum access bits of the designated subsidiary process into the maximum access bits of the requesting process.

SYSTEM CALL DESCRIPTION

MNEMONIC: RWS **NUMBER:** 31

NAME: Read Working Set

PRIVILEGE LEVEL

☐ Listener ☐ Exec ☐ Subsystem ☒ User ☐ Other (Specify)

CAPABILITIES BITS: none

Number: **Meaning:**

TIMING:

INPUT:

X: Pointer to 4-word table
U:
A:
E:
Other:

OUTPUT:

X:
U: Working set size limit
A: Size of current working set
E:
Other: Working set bits in given table.

EXCEPTION CONDITION ACTION

☒ No exception possible ☐ Skip on no exception ☐ Exception causes panic condition

Description:

DISMISSAL CONDITIONS:

none

FUNCTION:

Reads the working set bits for all pages in the process's virtual address space into the specified table. A "1" in any position in the table indicates that the corresponding page is in the working set.

SYSTEM CALL DESCRIPTION

MNEMONIC: SWS **NUMBER:** 32

NAME: Set Working Set

PRIVILEGE LEVEL

☐ Listener ☐ Exec ☐ Subsystem ☒ User ☐ Other (Specify)

CAPABILITIES BITS:

Number: 12 **Meaning:** Enable System Call

TIMING:

INPUT:

X: Pointer to a 4-word table of working set bits
U:
A:
E:
Other:

OUTPUT:

X:
U: Working set size limit
A: Size of new working set
E:
Other:

EXCEPTION CONDITION ACTION

☐ No exception possible ☐ Skip on no exception ☒ Exception causes panic condition

Description: Memory panic if the working set size limit is exceeded. Illegal instruction if a page to be put into the working set has an empty Relabeling Register.

DISMISSAL CONDITIONS:

New pages in the working set are not in core.

FUNCTION:

Sets the working set bits for all pages in the process's virtual address space from a 4-word table in core. A "1" in any position in the table indicates that the corresponding page is to be in the new working set.

SYSTEM CALL DESCRIPTION

MNEMONIC: AWS **NUMBER:** 33

NAME: Alter Working Set

PRIVILEGE LEVEL

☐ Listener ☐ Exec ☐ Subsystem ☒ User ☐ Other (Specify)

CAPABILITIES BITS:

Number: 12 **Meaning:** Enable system call

TIMING:

INPUT:

X: Relabeling Register Number
U:
A: Remove/Insert flag in bit 15
E:
Other:

OUTPUT: none

X:
U:
A:
E:
Other:

EXCEPTION CONDITION ACTION

☐ No exception possible ☐ Skip on no exception ☒ Exception causes panic condition

Description: Illegal instruction if Relabeling Register number is invalid or if a page with an empty Relabeling Register is to be put into the working set. Memory panic if working set size limit is exceeded.

DISMISSAL CONDITIONS:

Page added to the working set is not in core.

FUNCTION:

Sets or clears the working set bit for the given page of the process's virtual address space.

Remove/Insert flag:

0 → Remove the page from the working set
0 → Insert the page in the working set

SYSTEM CALL DESCRIPTION

MNEMONIC: SWSL **NUMBER:** 34

NAME: Set Working Set Limit

PRIVILEGE LEVEL

☐ Listener ☒ Exec ☐ Subsystem ☐ User ☐ Other (Specify)

CAPABILITIES BITS:

Number: 13 **Meaning:** Enable System Call

TIMING:

INPUT:

X:
U: New Working Set Size Limit
A:
E:
Other:

OUTPUT: none

X:
U:
A:
E:
Other:

EXCEPTION CONDITION ACTION

☐ No exception possible ☐ Skip on no exception ☒ Exception causes panic condition

Description: Illegal instruction if the new limit exceeds the largest possible limit.

DISMISSAL CONDITIONS: none

FUNCTION:

Sets the working set limit to the specified value. The working set limit is the maximum number of pages that the user may have in his working set at any one time; this includes both pages of the process's virtual address space and pages in the system's virtual address space that are being used for the requesting process.

SYSTEM CALL DESCRIPTION

MNEMONIC: RSTC **NUMBER:** 38

NAME: Read Scheduling Table Contents

PRIVILEGE LEVEL

☐ Listener ☐ Exec ☐ Subsystem ☐ User ☒ Other (Specify) System Managers E XE

CAPABILITIES BITS:

Number: 42 **Meaning:** Enable System Call

TIMING:

INPUT:

X: Pointer to table in care
U:
A: Scheduling Table Number
E:
Other:

OUTPUT:

X:
U:
A:
E:
Other:

EXCEPTION CONDITION ACTION

☒ No exception possible ☐ Skip on no exception ☐ Exception causes panic condition

Description:

DISMISSAL CONDITIONS: none

FUNCTION:

Read the contents of the specified scheduling table into the designated table in core. First word read in will specify the number of entries in the table.

SYSTEM CALL DESCRIPTION

MNEMONIC: SSTC **NUMBER:** 39

NAME: Set Scheduling Table Contents

PRIVILEGE LEVEL

☐ Listener ☐ Exec ☐ Subsystem ☐ User ☐ Other (Specify) System Manager's EXEC

CAPABILITIES BITS:

Number: 42 **Meaning:** Enable the system call

TIMING:

INPUT:

X: Pointer to table in core
U:
A: Sheduling table number
E:
Other:

OUTPUT:

X:
U:
A:
E:
Other:

EXCEPTION CONDITION ACTION

☐ No exception possible ☒ Skip on no exception ☐ Exception causes panic condition

Description: Values out of allowable range. Table not changed.

DISMISSAL CONDITIONS: none

FUNCTION:

Set the contents of the specified scheduling table from the table in core. The first word of the core table specifies the number of entries to set.

SYSTEM CALL DESCRIPTION

MNEMONIC: RRTV **NUMBER:** 40

NAME: Read Response Time Values

PRIVILEGE LEVEL

☐ Listener ☐ Exec ☐ Subsystem ☐ User ☒ Other (Specify) System Managers EXEC

CAPABILITIES BITS:

Number: 42 **Meaning:** Enable the System Call

TIMING:

INPUT:

X: T1 - response time associated with queue 3
U: T2 - " " " " " 4
A: T3 - " " " " " 5
E:
Other:

OUTPUT:

X:
U:
A:
E:
Other:

EXCEPTION CONDITION ACTION

☒ No exception possible ☐ Skip on no exception ☐ Exception causes panic condition

Description:

DISMISSAL CONDITIONS: none

FUNCTION:

Read the response time values (expressed as the number of drum revolutions) associated with ready queues 3, 4, and 5.

SYSTEM CALL DESCRIPTION

MNEMONIC: SRTV **NUMBER:** 41
NAME: Set Response Time Values

PRIVILEGE LEVEL

☐ Listener ☐ Exec ☐ Subsystem ☐ User ☒ Other (Specify) System Manager's EXEC

CAPABILITIES BITS:

Number: 42 **Meaning:** Enable the system call

TIMING:

INPUT:

X: T1 - response time associated with queue 3
U: T2 - " " " " " 4
A: T3 - " " " " " 5
E:
Other:

OUTPUT:

X:
U:
A:
E:
Other:

EXCEPTION CONDITION ACTION

☐ No exception possible ☒ Skip on no exception ☐ Exception causes panic condition

Description: Values out of allowable range. No change made.

DISMISSAL CONDITIONS: none

FUNCTION:

Set the response time values (expressed as the number of drum revolutions) associated with ready queues 3, 4, and 5.

SYSTEM CALL DESCRIPTION

MNEMONIC: SNPA **NUMBER:** 42

NAME: Set Number of Processes Ahead

PRIVILEGE LEVEL

☐ Listener ☐ Exec ☐ Subsystem ☐ User ☒ Other (Specify) System Manager's EXEC

CAPABILITIES BITS:

Number: 42 **Meaning:** Enable System Call

TIMING:

INPUT:

X:
U:
A: New value of parameter requested
E:
Other:

OUTPUT:

X:
U:
A: New value actually set
E:
Other:

EXCEPTION CONDITION ACTION

☐ No exception possible ☒ Skip on no exception ☐ Exception causes panic condition

Description: Couldn't set parameter to the requested value.

DISMISSAL CONDITIONS: none

FUNCTION:

Set the system parameter that determines the maximum number of processes that the swapper will try to read into core.

SYSTEM CALL DESCRIPTION

MNEMONIC: SNPD **NUMBER:** 43

NAME: Set Number of Pages for Disk

PRIVILEGE LEVEL

☐ Listener ☐ Exec ☐ Subsystem ☐ User ☒ Other (Specify) System Manager's EXEC

CAPABILITIES BITS:

Number: 42 **Meaning:** Enable system call

TIMING:

INPUT:

X:
U:
A: New value of parameter requested
E:
Other:

OUTPUT:

X:
U:
A: New value actually set
E:
Other:

EXCEPTION CONDITION ACTION

☐ No exception possible ☒ Skip on no exception ☐ Exception causes panic condition

Description: Couldn't set parameter to the requested value.

DISMISSAL CONDITIONS: none

FUNCTION: Set the system parameter that determines the maximum number of core pages that will be used to buffer information being transferred from the drum to the disk.

SYSTEM CALL DESCRIPTION

MNEMONIC: CREV **NUMBER:** 45

NAME: Create Event

PRIVILEGE LEVEL

☐ Listener ☐ Exec ☐ Subsystem ☒ User ☐ Other (Specify)

CAPABILITIES BITS:

Number: 51 **Meaning:** Enable System Call

TIMING:

INPUT:

X: Pointer to access information data
U:
A: Address of string pointer pair giving event name
E:
Other:

OUTPUT:

X:
U:
A:
E:
Other:

EXCEPTION CONDITION ACTION

☐ No exception possible ☒ Skip on no exception ☐ Exception causes panic condition

Description: Exception if event name already used or if event file full.

DISMISSAL CONDITIONS: Event directory and proper event list page not in core.

FUNCTION:

Create event with specified name. Uses file directory scheme, adds new event branch to event directory, pointing to empty notify directory for event. If no access stated, then anyone may cause (write access) or be notified (read access) of event.

SYSTEM CALL DESCRIPTION

MNEMONIC: Note **NUMBER:** 46

NAME: Notify If Event Occurs

PRIVILEGE LEVEL

☐ Listener ☐ Exec ☐ Subsystem ☒ User ☐ Other (Specify)

CAPABILITIES BITS:

Number: 52 **Meaning:** Enable System Call

TIMING:

INPUT:

X:
U: Interrupt Number
A: Address of string pointer pair giving event name
E:
Other:

OUTPUT:

X:
U:
A:
E:
Other:

EXCEPTION CONDITION ACTION

☐ No exception possible ☒ Skip on no exception ☐ Exception causes panic condition

Description: Exception if no such event or if interrupt is reserved or too big.

DISMISSAL CONDITIONS: Event directory and proper event list page not in core.

FUNCTION:

Place notify request in notify list for specified event. If event is on, give interrupt immediately, or else give interrupt when event is caused (each time it is caused).

SYSTEM CALL DESCRIPTION

MNEMONIC: DELN **NUMBER:** 47

NAME: Delete Notify Request

PRIVILEGE LEVEL

☐ Listener ☐ Exec ☐ Subsystem ☒ User ☐ Other (Specify)

CAPABILITIES BITS:

Number: 52 **Meaning:** Enable System Call

TIMING:

INPUT:

X:
U:
A: Address of string pointer pair giving event name.
E:
Other:

OUTPUT:

X:
U:
A:
E:
Other:

EXCEPTION CONDITION ACTION

☐ No exception possible ☒ Skip on no exception ☐ Exception causes panic condition

Description: No notify request in for specified event for this user.

DISMISSAL CONDITIONS: Event directory and proper event page list not in core.

FUNCTION:

Delete notify request.

SYSTEM CALL DESCRIPTION

MNEMONIC: CAUSE **NUMBER:** 48

NAME:

PRIVILEGE LEVEL

☐ Listener ☐ Exec ☐ Subsystem ☒ User ☐ Other (Specify)

CAPABILITIES BITS:

Number: 53 **Meaning:** Enable System Call

TIMING:

INPUT:

X:
U:
A: Address of string pointer pair giving event name
E:
Other:

OUTPUT:

X:
U:
A:
E:
Other:

EXCEPTION CONDITION ACTION

☐ No exception possible ☒ Skip on no exception ☐ Exception causes panic condition

Description: No such event

DISMISSAL CONDITIONS: Event directory and proper event page list not in core.

FUNCTION:

Turn on specified event. Notify all users currently in list. Also notify all users added to list until event is uncaused.

SYSTEM CALL DESCRIPTION

MNEMONIC: UNCAUS **NUMBER:** 49

NAME: Uncause Event

PRIVILEGE LEVEL

☐ Listener ☐ Exec ☐ Subsystem ☒ User ☐ Other (Specify)

CAPABILITIES BITS:

Number: 53 **Meaning:** Enable System Call

TIMING:

INPUT:

X:
U:
A: Address of string pointer pair giving event name
E:
Other:

OUTPUT:

X:
U:
A:
E:
Other:

EXCEPTION CONDITION ACTION

☐ No exception possible ☒ Skip on no exception ☐ Exception causes panic condition

Description: No such event is on

DISMISSAL CONDITIONS: Event directory and proper event list page not in core

FUNCTION:

Turn off specified event.

SYSTEM CALL DESCRIPTION

MNEMONIC: RIS **NUMBER:** 50

NAME: Read Interrupt status

PRIVILEGE LEVEL

☐ Listener ☐ Exec ☐ Subsystem ☒ User ☐ Other (Specify)

CAPABILITIES BITS: none

Number: **Meaning:**

TIMING:

INPUT: none

X:
U:
A:
E:
Other:

OUTPUT:

X:
U: Enabled interrupt mask
A:
E:
Other:

EXCEPTION CONDITION ACTION

☒ No exception possible ☐ Skip on no exception ☐ Exception causes panic condition

Description:

DISMISSAL CONDITIONS: none

FUNCTION:

Reads the current enabled/disabled status of the interrupts into the U and A registers (interrupts 0-15 correspond to U0-15, interrupts 16-31 correspond to A0-15). A "1" in any bit position indicates that the corresponding interrupt is enabled; a "0" indicates that it is disabled.

SYSTEM CALL DESCRIPTION

MNEMONIC: SIS **NUMBER:** 51

NAME: Set Interrupt Status

PRIVILEGE LEVEL

☐ Listener ☐ Exec ☐ Subsystem ☑ User ☐ Other (Specify)

CAPABILITIES BITS:

Number: 14 **Meaning:** Enable System Call

TIMING:

INPUT:

X:
U: } Enabled Interrupt Mask
A: }
E:
Other:

OUTPUT:

X:
U:
A:
E:
Other:

EXCEPTION CONDITION ACTION

☒ No exception possible ☐ Skip on no exception ☐ Exception causes panic condition

Description:

DISMISSAL CONDITIONS: none

FUNCTION:

Sets the current enabled/disabled status of the interrupts from the mask in the U and A registers (interrupts 0-15 correspond to U0-15, interrupts 16-31 correspond to A0-15). A "1" in any bit position indicates that it is to be disabled.

SYSTEM CALL DESCRIPTION

MNEMONIC: INT **NUMBER:** 52

NAME: Cause Interrupt

PRIVILEGE LEVEL

☐ Listener ☐ Exec ☐ Subsystem ☒ User ☐ Other (Specify)

CAPABILITIES BITS:

Number: **Meaning:**

TIMING:

INPUT:

X:
U: Process designator
A: Interrupt number
E:
Other:

OUTPUT:

X:
U: Number of interrupts triggered
A:
E:
Other:

EXCEPTION CONDITION ACTION

☐ No exception possible ☒ Skip on no exception ☒ Exception causes panic condition

Description: Illegal instruction if input parameters invalid. No skip if the interrupt could not be taken by some process.

DISMISSAL CONDITIONS: none

FUNCTION:

Causes the specified interrupt to occur in the specified process(es) if they have that interrupt inabled.

Process Designator: This word determines the process(es) in which the interrupt will be triggered. A scan is made of the fork structure looking for process that have the specified interrupt enabled. The scan is conducted according to the rules specified in bits 0-4 of this word.

(Continued)

Bit 0 = 1 → Scan all processes. This overrides all but bit 5, which remains operative
Bit 1 = 0 → Scan only the process whose PET index is contained in bit 13-15 of the word. This overrides bits 2-4.
Bit 1 = 1 → Trigger the interrupt in only the first N processes found by the scan that have the interrupt armed. The count is supplied in bits 13-15 of the word. Bits 2-5 are operative.
Bit 2 = 1 → Scan ancestor processes beginning with the process that created the process issuing the call.
Bit 3 = 1 → Scan subsidiary processes (i.e. processes created by the calling process) beginning with the first process created.
Bit 4 = 1 → Scan parallel processes (i.e. precess created by the same process beginning with the first process created.
Bit 5 = 1 → Do not trigger the armed interrupt(s) but return the skip/no skip indication and the count as if the interrupt(s) had been triggered.
Bits 13-15 = Count or PET index as specified above.

SYSTEM CALL DESCRIPTION

MNEMONIC: CLAI **NUMBER:** 53

NAME: Clear Active Interrupt

PRIVILEGE LEVEL

☐ Listener ☐ Exec ☐ Subsystem ☒ User ☐ Other (Specify)

CAPABILITIES BITS: none

Number: **Meaning:**

TIMING:

INPUT: none

X:
U:
A:
E:
Other:

OUTPUT: none

X:
U:
A:
E:
Other:

EXCEPTION CONDITION ACTION

☐ No exception possible ☐ Skip on no exception ☒ Exception causes panic condition

Description: Illegal instruction if no interrupt is currently active in the requesting process.

DISMISSAL CONDITIONS: none

FUNCTION:

Sets the state of the currently active interrupt to enabled and inactive (at the time the interrupt became active the state was set to disabled and active).

SYSTEM CALL DESCRIPTION

MNEMONIC: CLRAI **NUMBER:** 54

NAME: Clear and Return from Active Interrupt

PRIVILEGE LEVEL

☐ Listener ☐ Exec ☐ Subsystem ☒ User ☐ Other (Specify)

CAPABILITIES BITS:

Number: **Meaning:**

TIMING:

INPUT:

X: Pointer to location of saved registers
U:
A: Skip flag
E:
Other:

OUTPUT: none

X:
U:
A:
E:
Other:

EXCEPTION CONDITION ACTION

☐ No exception possible ☐ Skip on no exception ☒ Exception causes panic condition

Description: Illegal instruction if no interrupt is currently active in the requesting process.

DISMISSAL CONDITIONS: none.

FUNCTION:

Sets the state of the currently active interrupt to enabled and inactive (at the time the interrupt became active the state was set to disabled and active). Restores the registers (P, S, X, U, A and E) from the location given in X thereby returning the process to the state that existed when the interrupt occured.

Skip Flag: 0 ⇒ do not skip 1st instruction after return.
1 ⇒ skip 1st instruction after return.

SYSTEM CALL DESCRIPTION

MNEMONIC: DELV **NUMBER:** 55

NAME: Delete Event

PRIVILEGE LEVEL

☐ Listener ☐ Exec ☐ Subsystem ☒ User ☐ Other (Specify)

CAPABILITIES BITS:

Number: Meaning:

TIMING:

INPUT:

X: Pointer to access information
U:
A: Address of string pointer pair giving event name
E:
Other:

OUTPUT:

X:
U:
A:
E:
Other:

EXCEPTION CONDITION ACTION

☐ No exception possible ☒ Skip on no exception ☐ Exception causes panic condition

Description: Exception if access not allowed or no such event

DISMISSAL CONDITIONS: Event directory not in core.

FUNCTION: Delete specified event.

SYSTEM CALL DESCRIPTION

MNEMONIC: MTC **NUMBER:** 57

NAME: Magnetic Tape Control

PRIVILEGE LEVEL

☐ Listener ☒ Exec ☒ Subsystem ☐ User ☐ Other (Specify)

CAPABILITIES BITS:

Number: 54 **Meaning:** Enable System Call

TIMING:

INPUT:
X:
U:
A:
E:
Other:

OUTPUT:
X:
U:
A:
E:
Other:

Not Yet Defined

EXCEPTION CONDITION ACTION

☐ No exception possible ☒ Skip on no exception ☐ Exception causes panic condition

Description: Magnetic tape unit not attached

DISMISSAL CONDITIONS: Not defined

FUNCTION:

This call will essentially be a magnetic tape handler providing such functions as read, write, rewind, space over files, records, etc. Will be defined after magnetic tape controller hardware functions are defined.

SYSTEM CALL DESCRIPTION

MNEMONIC: TCI **NUMBER:** 58

NAME: Terminal Character Input

PRIVILEGE LEVEL

☐ Listener ☐ Exec ☐ Subsystem ☒ User ☐ Other (Specify)

CAPABILITIES BITS:

Number: 29 **Meaning:** Enable System Call

TIMING:

INPUT: None

X:
U:
A:
E:
Other:

OUTPUT:

X:
U:
A: Character from controlling terminal
E:
Other:

EXCEPTION CONDITION ACTION

☐ No exception possible ☐ Skip on no exception ☒ Exception causes panic condition

Description: Illegal instruction if a BEL system call is in progress.

DISMISSAL CONDITIONS: Terminal input buffer empty, or an echo is to be generated as the character is read in and the output buffer is full.

FUNCTION: Reads the next character from the controlling terminal input buffer into bits 8-15 of the A register and clears bits 0-7. The character is removed from the input buffer and the echo mode is set to generate the echo when a character is packed into the input buffer rather than when a character is read out of the buffer into some process.

SYSTEM CALL DESCRIPTION

MNEMONIC: TCO **NUMBER:** 59

NAME: Terminal Character Output

PRIVILEGE LEVEL

☐ Listener ☐ Exec ☐ Subsystem ☒ User ☐ Other (Specify)

CAPABILITIES BITS:

Number: 30 **Meaning:** Enable System Call

TIMING:

INPUT:

X:
U:
A: Character in bits 8-15
E:
Other:

OUTPUT: None

X:
U:
A:
E:
Other:

EXCEPTION CONDITION ACTION

☒ No exception possible ☐ Skip on no exception ☐ Exception causes panic condition

Description:

DISMISSAL CONDITIONS: Terminal Output buffer is full.

FUNCTION: Packs the character in A into the controlling terminal output buffer from which it will be transmitted to the terminal.

SYSTEM CALL DESCRIPTION

MNEMONIC: ISD **NUMBER:** 60

NAME: Input From Specified Device

PRIVILEGE LEVEL

☐ Listener ☐ Exec ☐ Subsystem ☒ User ☐ Other (Specify)

CAPABILITIES BITS:

Number: 31 **Meaning:** Enable System Call

TIMING:

INPUT:

X:
U: Device number
A:
E:
Other:

OUTPUT:

X:
U:
A: Character from specified device
E:
Other:

EXCEPTION CONDITION ACTION

☐ No exception possible ☐ Skip on no exception ☒ Exception causes panic condition

Description: Illegal instruction if the specified device is not attached

DISMISSAL CONDITIONS: Device input buffer is empty, or an echo is to be generated as the character is read in and the output buffer is full.

FUNCTION: Reads the next character from the input buffer of the specified device into bits 8-15 of the A register and clears bits 0-7. The character is removed from the input buffer and the echo mode is set to generate the echo when a character is packed into the input buffer rather than when a character is read out of the buffer into some process.

SYSTEM CALL DESCRIPTION

MNEMONIC: OSD **NUMBER:** 61

NAME: Output to Specified Device

PRIVILEGE LEVEL

☐ Listener ☐ Exec ☐ Subsystem ☒ User ☐ Other (Specify)

CAPABILITIES BITS:

Number: 32 **Meaning:** Enable System Call

TIMING:

INPUT:

X:
U: Device Number
A: Character in bits 8-15
E:
Other:

OUTPUT: None

X:
U:
A:
E:
Other:

EXCEPTION CONDITION ACTION

☐ No exception possible ☐ Skip on no exception ☒ Exception causes panic condition

Description: Illegal instruction if the specified device is not attached.

DISMISSAL CONDITIONS: Device output buffer is full.

FUNCTION: Packs the character in A into the output buffer of the specified device from which it will be transmitted to the device.

SYSTEM CALL DESCRIPTION

MNEMONIC: RNIC **NUMBER:** 62

NAME: Read Next Input Character

PRIVILEGE LEVEL

☐ Listener ☐ Exec ☐ Subsystem ☒ User ☐ Other (Specify)

CAPABILITIES BITS:

Number: 29 **Meaning:** Enable Systems Call if device number specifies the controlling terminal.

TIMING: 31 Enable the system call if the device number specified any other device.

INPUT:

X:
U: Device number
A:
E:
Other:

OUTPUT:

X:
U:
A: Character from specified device
E:
Other:

EXCEPTION CONDITION ACTION

☐ No exception possible ☒ Skip on no exception ☒ Exception causes panic condition

Description: Illegal instruction if device is not attached. No skip if the input buffer is empty.

DISMISSAL CONDITIONS: None

FUNCTION: If the input buffer of the specified device is not empty, read the next character into bits 8-15 of the A register and clear bits 0-7. The character is not removed from the input buffer.

If the input buffer is empty, do not change the contents of the A register and take the no-skip return.

SYSTEM CALL DESCRIPTION

MNEMONIC: ADV **NUMBER:** 63

NAME: Attach Device

PRIVILEGE LEVEL

☐ Listener ☐ Exec ☐ Subsystem ☒ User ☐ Other (Specify)

CAPABILITIES BITS:

Number:	Meaning:
38	Allow attaching of any device.
39	Allow attaching of interactive terminals only.

TIMING:

INPUT:

X:
U: Device Number
A:
E:
Other:

OUTPUT: None

X:
U:
A:
E:
Other:

EXCEPTION CONDITION ACTION

☐ No exception possible ☒ Skip on no exception ☒ Exception causes panic condition

Description: Illegal instruction if the device number is invalid. No skip if the device is already attached to another user.

DISMISSAL CONDITIONS: None

FUNCTION: Attaches the specified device to this user.

SYSTEM CALL DESCRIPTION

MNEMONIC: RDV **NUMBER:** 64

NAME: Release Device

PRIVILEGE LEVEL

☐ Listener ☐ Exec ☐ Subsystem ☒ User ☐ Other (Specify)

CAPABILITIES BITS:

Number: 40 **Meaning:** Enable System Call

TIMING:

INPUT:

X:
U: Device Number
A:
E:
Other:

OUTPUT: None

X:
U:
A:
E:
Other:

EXCEPTION CONDITION ACTION

☐ No exception possible ☐ Skip on no exception ☒ Exception causes panic condition

Description: Illegal instruction if the device is not attached to this user.

DISMISSAL CONDITIONS: None

FUNCTION: Detaches the specified device from this user and makes it available for other users.

SYSTEM CALL DESCRIPTION

MNEMONIC: TDV **NUMBER:** 65

NAME: Test Device Status

PRIVILEGE LEVEL

☐ Listener ☐ Exec ☐ Subsystem ☒ User ☐ Other (Specify)

CAPABILITIES BITS:

Number: 46 **Meaning:** Enable System Call

TIMING:

INPUT:

X:
U: Device Number
A:
E:
Other:

OUTPUT: None

X:
U:
A:
E:
Other:

EXCEPTION CONDITION ACTION

☐ No exception possible ☐ Skip on no exception ☒ Exception causes panic condition

Description: Illegal instruction if device number is invalid.

DISMISSAL CONDITIONS: None

FUNCTION: Skips if the specified device is free (i.e., not attached to any user).

SYSTEM CALL DESCRIPTION

MNEMONIC: ECH **NUMBER:** 66

NAME: Set Echo Conditions

PRIVILEGE LEVEL

☐ Listener ☐ Exec ☐ Subsystem ☒ User ☐ Other (Specify)

CAPABILITIES BITS:

Number: 37 Meaning: Enable System Call

TIMING:

INPUT:

X: Echo and Translation
U: Terminal Number
A: Break and Deferred Echo
E:
Other:

OUTPUT: NONE

X:
U:
A:
E:
Other:

EXCEPTION CONDITION ACTION

☐ No exception possible ☐ Skip on no exception ☒ Exception causes panic condition

Description: Illegal instruction if input parameters invalid or BEL call is in progress.

DISMISSAL CONDITIONS: NONE

FUNCTION: Sets four input conditions for the specified terminal. The conditions, the fields that specify them, and the values that can be set are as follows:

ECHO - Bits 0-7 of the X Register
0 - Normal text entry echoes
1 - No echo

TRANSLATION - Bits 8-15 of the X Register
0 - Device code to ASCII
1 - No translation

Function: (continued)

BREAK CHARACTERS - Bits 0-7 of the A Register

0 - All characters
1 - EOT character only
2 - All control characters ($1\text{-}37_8$ in the ASCII set)
3 - All except alphanumerics

DEFERRED ECHO CHARACTERS - Bits 8-15 of the A Register

0 - All characters
1 - None
2 - All control characters
3 - All except alphanumerics

NOTES:

1. If the translation designator is set to "1", the other three designators must also be set to "1".

2. Break characters are those that cause a process to run if it is waiting for input from the terminal.

3. Deferred echo characters cause the deferred echo mode to begin with the next character. If the deferred echo mode is not set, each character is echoed as it is received and packed into the input buffer. If the deferred echo mode is set, the character is not echoed until it is unpacked from the input buffer and read into some process.

SYSTEM CALL DESCRIPTION

MNEMONIC: CDB **NUMBER:** 67

NAME: Clear Device Buffer

PRIVILEGE LEVEL

☐ Listener ☐ Exec ☐ Subsystem ☒ User ☐ Other (Specify)

CAPABILITIES BITS:

Number: 40 **Meaning:** Enable System Call

TIMING:

INPUT:

X:
U: Device number
A: Buffer designators
E:
Other:

OUTPUT: None

X:
U:
A:
E:
Other:

EXCEPTION CONDITION ACTION

☐ No exception possible ☐ Skip on no exception ☒ Exception causes panic condition

Description: Illegal instruction if the device is not attached to this user.

DISMISSAL CONDITIONS: None

FUNCTION: Clears the specified buffers, resetting their states to empty. Sets the echo mode to generate an echo when a character is packed into the input buffer rather than when a character is read out of the buffer into some process (if the input buffer is specified). When an input buffer is cleared, all characters are removed from it whether or not the corresponding echo has been generated and placed in the output buffer. Thus some echoes could be transmitted to the device even though the corresponding characters will never be seen by any process as input from the device. When an output buffer is cleared, both program generated characters and input echoes that have already been packed into the output buffer are removed.

Buffer designators:
Bit 0 - Output buffer
Bit 1 - Input buffer
Bit 2 - BEL new line buffer (terminates BEL)

SYSTEM CALL DESCRIPTION

MNEMONIC: RDBZ **NUMBER:** 68

NAME: Read Device Buffer Size

PRIVILEGE LEVEL

☐ Listener ☐ Exec ☐ Subsystem ☒ User ☐ Other (Specify)

CAPABILITIES BITS: None

Number: **Meaning:**

TIMING:

INPUT:

X:
U: Device number
A:
E:
Other:

OUTPUT:

X:
U: Number of characters in output buffer
A: Number of characters in input buffer
E:
Other:

EXCEPTION CONDITION ACTION

☐ No exception possible ☐ Skip on no exception ☒ Exception causes panic condition

Description: Illegal instruction if the device is not attached to this user.

DISMISSAL CONDITIONS: None

FUNCTION: Reads the number of character remaining in the input and output buffers.

SYSTEM CALL DESCRIPTION

MNEMONIC: SOT **NUMBER:** 69

NAME: Set Output Translation

PRIVILEGE LEVEL

☐ Listener ☐ Exec ☐ Subsystem ☒ User ☐ Other (Specify)

CAPABILITIES BITS:

Number: 30 **Meaning:** Enable System Call

TIMING:

INPUT:

X:
U: Device number
A: Translation designator
E: 0 → ASCII to Device Code
Other: 1 → No Translation

OUTPUT: None

X:
U:
A:
E:
Other:

EXCEPTION CONDITION ACTION

☐ No exception possible ☐ Skip on no exception ☒ Exception causes panic condition

Description: Illegal instruction if the specified device is not attached to to this user or is of the wrong type.

DISMISSAL CONDITIONS: None

FUNCTION: Sets the translation mode that will be in effect for the specified device (usually a terminal). If no translation is called for, the low order bits of each 8-bit byte (is device dependent) will be transmitted to the terminal unchanged. If translation is called for, multiple blank characters will be expanded, line feeds will be insetted after carriage returns if necessary, idle characters will be inserted where necessary to allow for mechanical motion of the terminal, etc.

SYSTEM CALL DESCRIPTION

MNEMONIC: RDT **NUMBER:** 70

NAME: Read Device Type

PRIVILEGE LEVEL

☐ Listener ☐ Exec ☐ Subsystem ☒ User ☐ Other (Specify)

CAPABILITIES BITS: None

Number: **Meaning:**

TIMING:

INPUT:

X:
U: Device Number
A:
E:
Other:

OUTPUT:

X:
U:
A: Device Type
E:
Other:

EXCEPTION CONDITION ACTION

☐ No exception possible ☐ Skip on no exception ☒ Exception causes panic condition

Description: Illegal instruction if the device is not attached to this user.

DISMISSAL CONDITIONS: None

FUNCTION: Reads a device code that specifies the device type. Legal device codes include:

0 - Model 33 or 35 teletype in full duplex mode
1 - Model 33 or 35 teletype in half duplex mode
2 - Synerdata Beta
3 - IBM 2741 (Correspondence Code Set)
4 - Infoton Vista CRT terminal

SYSTEM CALL DESCRIPTION

MNEMONIC: SDT **NUMBER:** 71

NAME: Set Device Type

PRIVILEGE LEVEL

☐ Listener ☐ Exec ☐ Subsystem ☒ User ☐ Other (Specify)

CAPABILITIES BITS:

Number: 47 **Meaning:** Enable System Call

TIMING:

INPUT:

X:
U: Device number
A: Device type
E:
Other:

OUTPUT: None

X:
U:
A:
E:
Other:

EXCEPTION CONDITION ACTION

☐ No exception possible ☐ Skip on no exception ☒ Exception causes panic condition

Description: Illegal instruction if device not attached to this user.

DISMISSAL CONDITIONS: None

FUNCTION: Declares the specified device to be of a particular type. Legal device codes include:

0 - Model 33 or 35 teletype in full duplex mode
1 - Model 33 or 35 teletype in half duplex mode
2 - Synerdata Beta
3 - IBM 2741 (Correspondence Code Set)
4 - Infoton Vista CRT terminal

SYSTEM CALL DESCRIPTION

MNEMONIC: STCI **NUMBER:** 72

NAME: Simulate Terminal Character Input

PRIVILEGE LEVEL

☐ Listener ☐ Exec ☐ Subsystem ☒ User ☐ Other (Specify)

CAPABILITIES BITS:

Number: 33 **Meaning:** Enable System Call

TIMING:

INPUT:

X:
U: PET index of receiving process
A: Character in bits 8-15
E:
Other:

OUTPUT: None

X:
U:
A:
E:
Other:

EXCEPTION CONDITION ACTION

☐ No exception possible ☐ Skip on no exception ☒ Exception causes panic condition

Description: Designated process does not have its simulated input bit set.

DISMISSAL CONDITIONS: Simulated input buffer is full or buffer page is not in core.

FUNCTION: Places the character in the next available position in the simulated input buffer of the designated process. The process can then input the character with a TCI, ISD or GEL system call.

SYSTEM CALL DESCRIPTION

MNEMONIC: STCO **NUMBER:** 73

NAME: Simulate Terminal Character Output

PRIVILEGE LEVEL

☐ Listener ☐ Exec ☐ Subsystem ☒ User ☐ Other (Specify)

CAPABILITIES BITS:

Number: 34 **Meaning:** Enable System Call

TIMING:

INPUT:

X:
U: PET index
A:
E:
Other:

OUTPUT:

X:
U:
A: Character in Bits 8-15
E:
Other:

EXCEPTION CONDITION ACTION

☐ No exception possible ☐ Skip on no exception ☒ Exception causes panic condition

Description: Designated process does not have its simulated output bit set.

DISMISSAL CONDITIONS: Simulated output buffer empty

FUNCTION: Removes the next character from the simulated output buffer of the designated process.

SYSTEM CALL DESCRIPTION

MNEMONIC: SEESC **NUMBER:** 74

NAME: Set Escape

PRIVILEGE LEVEL

☐ Listener ☐ Exec ☐ Subsystem ☒ User ☐ Other (Specify)

CAPABILITIES BITS:

Number: 43 **Meaning:** Enable System Call

TIMING:

INPUT: None

X:
U:
A:
E:
Other:

OUTPUT: None

X:
U:
A:
E:
Other:

EXCEPTION CONDITION ACTION

☒ No exception possible ☐ Skip on no exception ☐ Exception causes panic condition

Description:

DISMISSAL CONDITIONS: None

FUNCTION: Sets to ESCAPE pointer to this process. A succeeding ESCAPE will cause the ESCAPE interrupt to be triggered in this process if it is armed; otherwise a panic will be generated and this process will be deactivated.

SYSTEM CALL DESCRIPTION

MNEMONIC: SIESC **NUMBER:** 75

NAME: Simulate Escape

PRIVILEGE LEVEL

☐ Listener ☐ Exec ☐ Subsystem ☒ User ☐ Other (Specify)

CAPABILITIES BITS:

Number: Meaning:

TIMING:

INPUT: None

X:
U:
A:
E:
Other:

OUTPUT: None

X:
U:
A:
E:
Other:

EXCEPTION CONDITION ACTION

☒ No exception possible ☐ Skip on no exception ☐ Exception causes panic condition

Description:

DISMISSAL CONDITIONS: None

FUNCTION: Produces the same result as depressing the ESCAPE key on the controlling terminal.

SYSTEM CALL DESCRIPTION

MNEMONIC: DOBE **NUMBER:** 76

NAME: Dismiss Until Output Buffer Empty

PRIVILEGE LEVEL

☐ Listener ☐ Exec ☐ Subsystem ☒ User ☐ Other (Specify)

CAPABILITIES BITS:

Number: Meaning:

TIMING:

INPUT: None

X:
U:
A:
E:
Other:

OUTPUT: None

X:
U:
A:
E:
Other:

EXCEPTION CONDITION ACTION

☒ No exception possible ☐ Skip on no exception ☐ Exception causes panic condition

Description:

DISMISSAL CONDITIONS: Output buffer of controlling terminal is not empty.

FUNCTION: Dismisses the process until the output buffer of the controlling terminal is empty. This call should be executed by all processes that produce terminal output before returning to the EXEC. This allows printing to complete.

SYSTEM CALL DESCRIPTION

MNEMONIC: SLEM **NUMBER:** 92

NAME: Set Line Editing Mode

PRIVILEGE LEVEL

☐ Listener ☐ Exec ☐ Subsystem ☒ User ☐ Other (Specify)

CAPABILITIES BITS:

Number: 40 **Meaning:** Enable System Call

TIMING:

INPUT:

X: Miscellaneous Information
U: Tab Stops
A: Delimiter Set
E:
Other:

OUTPUT: NONE

X:
U:
A:
E:
Other:

EXCEPTION CONDITION ACTION

☐ No exception possible ☐ Skip on no exception ☒ Exception causes panic condition

Description: Illegal instruction on bad input data.

DISMISSAL CONDITIONS:

FUNCTION:

This system call activates the intraline editing mode and changes (or initializes) certain parameters. The first SLEM issued will initialize the line editing apparatus. Subsequent SLEM's can be issued to change parameters. This call does not cause any lines to be edited. It will cause a page of memory to be allocated and placed in the system working set unless it has already been done as the result of a previous SLEM. If this causes the size of the working set to exceed the limit allowed, a normal "working set exceeded" interrupt or panic will occur.

A - Delimiter Set

0 = Carriage return only.

1 = Editor command set (Carriage Return, Line Feed).

More options may be defined if needed.

U - Tab Stops

0 = Don't change the current settings. If there are no current settings, use the default condition.

-1 = Reset to default settings.

>0 = Pointer to a 9-word table of tab stops. Each bit in the table represents a tab stop in the corresponding character position in the line.

Default tab stop positions are columns 10, 16, 35, 45, 55, and 65.

X - Miscellaneous Information

Bit 0: Multiple line deletion control.

0 = Multiple, successive line deletions are not allowed.

1 = Multiple, successive line deletions are allowed. A line deletion character issued when the new line is empty will result in a request to delete the previous line being passed back to the using process by the GSEL system call.

Bits 8-15: Maximum line length in characters. If 0, 256 will be used.

SYSTEM CALL DESCRIPTION

MNEMONIC: LLEM **NUMBER:** 93

NAME: Leave Line Editing Mode

PRIVILEGE LEVEL

☐ Listener ☐ Exec ☐ Subsystem ☒ User ☐ Other (Specify)

CAPABILITIES BITS:

Number: 40 **Meaning:** Enable System Call

TIMING:

INPUT:

X:
U:
A: Deactivate or release drum page
E: 0 = Deactivate
Other: 1 = Release

OUTPUT: NONE

X:
U:
A:
E:
Other:

EXCEPTION CONDITION ACTION

☐ No exception possible ☐ Skip on no exception ☒ Exception causes panic condition

Description: Illegal instruction on bad input parameter.

DISMISSAL CONDITIONS: NONE

FUNCTION: This system call deactivates or completely removes the line editing apparatus from the calling user. If the parameter in A is a 0, the line editing mode is only deactivated. The system page required for buffering and control information is removed from the working set but the page is still allocated on the drum. Parameters set by previous SLEM's will be retained. If the parameter in A is a 1, the system page required for buffering and control information is released. Parameters set by previous SLEM's will be lost. It will usually be better to release the buffer page than to deactivate it. If the page is deactivated (A = 0), a subsequent SLEM will almost always result in an extra process activation being required to get the buffer page from the drum. If the page is released, a subsequent SLEM will allocate a new buffer page. If there is a free page in core at the time, an extra process activation will not be required.

SYSTEM CALL DESCRIPTION

MNEMONIC: BEL **NUMBER:** 94

NAME: Build Edited Line

PRIVILEGE LEVEL

☐ Listener ☐ Exec ☐ Subsystem ☒ User ☐ Other (Specify)

CAPABILITIES BITS:

Number: 29 **Meaning:** Enable System Call

TIMING:

INPUT:

X:
U: New Line
A: Old Line
E:
Other:

OUTPUT: NONE

X:
U:
A:
E:
Other:

EXCEPTION CONDITION ACTION

☐ No exception possible ☐ Skip on no exception ☒ Exception causes panic condition

Description: Illegal instruction on bad input parameters or line editing mode not set.

DISMISSAL CONDITIONS: NONE

FUNCTION: Causes the line editing apparatus to begin building an edited line in the system buffer allocated by a previously executed SLEM call. The building operation continues in parallel with process execution.

Old Line:

0 = Current New Line
-1 = Null
-2 = Current Old Line
>0 = Pointer to a string pointer pair that points to the new line.

New Line:

0 = Null
-1 = Current New Line
>0 = Pointer to a string pointer pair

SYSTEM CALL DESCRIPTION

MNEMONIC: GSEL **NUMBER:** 95

NAME: Get Status Of Edited Line

PRIVILEGE LEVEL

☐ Listener ☐ Exec ☐ Subsystem ☒ User ☐ Other (Specify)

CAPABILITIES BITS:

Number: 29 Meaning: Enable System Call

TIMING:

INPUT:

X:
U:
A: Dismissal Control Flag
E:
Other:

OUTPUT:

X: Status
U: Length of compressed new line (0 if status ≠ 4)
A: Last character of new line (0 if status ≠ 4)
E:
Other:

EXCEPTION CONDITION ACTION

☐ No exception possible ☒ Skip on no exception ☒ Exception causes panic condition

Description: Skip if line is complete.

DISMISSAL CONDITIONS: Dismissal control flag = 0 and line not complete.

FUNCTION: Get the status of the edited line being built by a previous BEL call. The dismissal control flag provided as input to the call allows the process to proceed with some computation while the new line is being built or to dismiss until the new line is complete.

Dismissal Control Flag:

0 = Dismiss process until line is complete
1 = Do not dismiss process

Status:

1 = Delete previous line
2 = End of text
3 = Line not complete.
4 = Line complete.
5 = No BEL call has been issued.

SYSTEM CALL DESCRIPTION

MNEMONIC: GEL **NUMBER:** 96

NAME: Get Edited Line

PRIVILEGE LEVEL

☐ Listener ☐ Exec ☐ Subsystem ☒ User ☐ Other (Specify)

CAPABILITIES BITS:

Number: 29 **Meaning:** Enable System Call

TIMING:

INPUT:

X:

U: Count Indicator

A: Byte address of the beginning of the buffer

E:

Other:

OUTPUT:

X:

U:

A:

E:

Other: Edited, compressed line in the designated buffer.

EXCEPTION CONDITION ACTION

☐ No exception possible ☐ Skip on no exception ☒ Exception causes panic condition

Description: Illegal instruction if no BEL call has been issued or if count indicator is illegal.

DISMISSAL CONDITIONS: The input line is not complete.

FUNCTION: Copy the edited, compressed line into the buffer designated in the call.

Count Indicator:

0 = Don't supply a count as part of the line.

1 = Place the byte count of the edited compressed line (not including the byte count) in the first byte of the buffer. The line termination character is not placed in the buffer and is not counted.

SYSTEM CALL DESCRIPTION

MNEMONIC: RLS **NUMBER:** 98

NAME: Read Link Status

PRIVILEGE LEVEL

☐ Listener ☒ Exec ☐ Subsystem ☐ User ☐ Other (Specify)

CAPABILITIES BITS:

Number: **Meaning:**

TIMING:

INPUT: None

X:
U:
A:
E:
Other:

OUTPUT:

X: Advise Link
U: Input Link
A: Output Link
E:
Other:

} {
Bit 6: Accepted
Bit 7: Enabled
Bits 8-15: User number
}

EXCEPTION CONDITION ACTION

☒ No exception possible ☐ Skip on no exception ☐ Exception causes panic condition

Description:

DISMISSAL CONDITIONS: None

FUNCTION: Read the status of the 3 links from the terminal control table. The fields have the following meanings:

Accepted: Another user's terminal is linked to the given user for advice/input/output.

User number ≠ 0: Advice/input/output is currently being supplied to the designated user.

Enabled: Advice/input/output link will be accepted from another user.

SYSTEM CALL DESCRIPTION

MNEMONIC: SLS **NUMBER:** 99

NAME: Set Link Status

PRIVILEGE LEVEL

☐ Listener ☒ Exec ☐ Subsystem ☐ User ☐ Other (Specify)

CAPABILITIES BITS:

Number: 55 **Meaning:** Allow exception of the System Call

TIMING:

INPUT:

X:
U: Advise Link / Input Link
A: Output Link
E:
Other:

Bits 6-9: ignored
Bits 8-15: User number

OUTPUT: On exception only

X:
U: Bit 0 / Bit 0
A: Bit 0
E:
Other:

1 → user had already accepted link

EXCEPTION CONDITION ACTION

☐ No exception possible ☒ Skip on no exception ☐ Exception causes panic condition

Description: Attempting to establish a link that is not allowed because of some existing link.

DISMISSAL CONDITIONS: None

FUNCTION: If the user number is zero, break that link. If the user number is not zero, try to establish the link. If the user executing the call already has an active input or output link active, the new user will be placed at the end of the chain. If the designated user already has a link accepted, an exception condition occurs. Advise links cannot be chained.

If a link is broken in a longer chain, the deleted terminal is chained over and the rest of the chain continues to function, even if the broken link was the first in the chain.

SYSTEM CALL DESCRIPTION

MNEMONIC: EDL **NUMBER:** 100

NAME: Enable or Disable Links

PRIVILEGE LEVEL

☐ Listener ☒ Exec ☐ Subsystem ☐ User ☐ Other (Specify)

CAPABILITIES BITS:

Number: 56 **Meaning:** Enable system call

TIMING:

INPUT:

X:
U:
A: ←
E:
Other:

Bit 13: 1 => Enable advise link, 0 → disable advise link.
Bit 14: 1 => Enable input link, 0 → disable input link.
Bit 15: 1 => Enable output link, 0 → disable output link.

OUTPUT: NONE

X:
U:
A:
E:
Other:

EXCEPTION CONDITION ACTION

☒ No exception possible ☐ Skip on no exception ☐ Exception causes panic condition

Description:

DISMISSAL CONDITIONS:

FUNCTION:

Set the link enabled bits as indicated. If a link has been accepted, break that link. If a link has been accepted and the given user is in the middle of a chain, relink the chain with the given user removed.

SYSTEM CALL DESCRIPTION

MNEMONIC: CRFIL **NUMBER:** 102

NAME: Create File or Directory

PRIVILEGE LEVEL

☐ Listener ☐ Exec ☐ Subsystem ☒ User ☐ Other (Specify)

CAPABILITIES BITS:

Number:	Meaning:
16	Enable System Call
45	Enable Directory Creation

TIMING:

INPUT:

X: Address of string pointer pair specifying file name.
U: Pointer to a table of file access information or zero.
A: File type if (U) = 0
E:
Other:

OUTPUT:

X: Error code on exception return
U: FCT index
A:
E:
Other:

EXCEPTION CONDITION ACTION

☐ No exception possible ☒ Skip on no exception ☐ Exception causes panic condition

Description: File by this name already exists in the currently open directory.

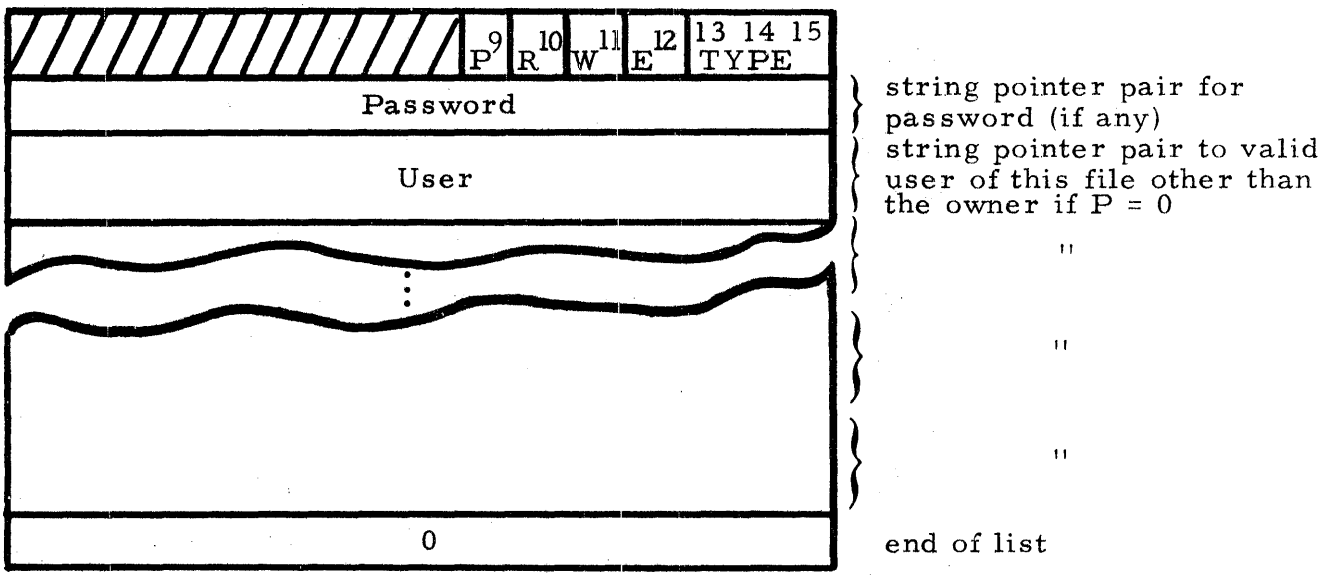
DISMISSAL CONDITIONS: Directory not in core or no free core page to use for CFIT.

FUNCTION:

Makes an entry for the specified file in the currently open file directory and creates a Closed File Index Table for the file or file directory being created. If (U)=0 access is defined as follows:

Private file with Read, Write, and Execute access.
No password required.
Type specified in A.

Access information block for file creation system call:



P: = 1 ⇒ public (anyone with access to this directory may access this file)

R, W, E: Read, Write, Execute protection bits for creating user.

TYPE: 0 ⇒ Binary
1 ⇒ Dump
2 ⇒ Go
3 ⇒ Symbolic
4 ⇒ Directory

The password is a character string that will be required whenever the file is opened. If the access code given to CRFIL is null then any password may open the file.

The list of valid users is meaningful only if P=0. The specified strings are the account number concatenated with the user name. The ASCII character "*" may be substituted for the file owner's account number.

SYSTEM CALL DESCRIPTION

MNEMONIC: OPFIL **NUMBER:** 103

NAME: Open File

PRIVILEGE LEVEL

☐ Listener ☐ Exec ☐ Subsystem ☒ User ☐ Other (Specify)

CAPABILITIES BITS:

	Number:	Meaning:
	17	Enable System Call
	23	Enable ACTIVE FILE bit
TIMING:	50	Enable DRUM FILE bit

INPUT:

X: Address of string pointer pair specifying file name
U: Address of string pointer pair specifying password
A: DRUM, ACTIVE, Read, Write, Execute bits in 11-15
E:
Other:

OUTPUT:

X: Error code on exception return
U: FCT index
A:
E:
Other:

EXCEPTION CONDITION ACTION

☐ No exception possible ☒ Skip on no exception ☐ Exception causes panic condition

Description: Unable to open file; reason in X.

DISMISSAL CONDITIONS: File directory (or directories) not in core, open file table not in core, OFIT not in core if ACTIVE specified, or OFIT not on drum if ACTIVE not specified.

FUNCTION: Open the file with the access specified by the Read, Write and Execute bits. If the file opened is a directory, it becomes the users "current directory". If the file name specifies that a data file is to be opened using some other directory, the "current directory" is not changed. If (X)=0 an "ephemeral" file will be created and opened. This file will not be entered in any file directory and will be destroyed when closed.

SYSTEM CALL DESCRIPTION

MNEMONIC: RFILN **NUMBER:** 104

NAME: Read File Name

PRIVILEGE LEVEL

☐ Listener ☐ Exec ☐ Subsystem ☒ User ☐ Other (Specify)

CAPABILITIES BITS: None

Number: 48 **Meaning:** Enable System Call

TIMING:

INPUT:

X: File Sequence Number
U:
A: Address of string pointer pair for file name
E:
Other:

OUTPUT:

X: Error code on exception return
U: Read, Write, Execute access for this user
A:
E:
Other: File name in string

EXCEPTION CONDITION ACTION

☐ No exception possible ☒ Skip on no exception ☐ Exception causes panic condition

Description: File sequence number too large or file name too long for string.

DISMISSAL CONDITIONS: File directory not in core

FUNCTION: Uses the file sequence numbers as an index into the "current file directory", reads the file name into the specified string storage area, and puts the access the user is allowed into U. Read, write, and execute access an specified by "1" bits in bits 13-15 respectively.

SYSTEM CALL DESCRIPTION

MNEMONIC: CFILA **NUMBER:** 105

NAME: Change File Access Protection

PRIVILEGE LEVEL

☐ Listener ☐ Exec ☐ Subsystem ☒ User ☐ Other (Specify)

CAPABILITIES BITS:

Number: 26 **Meaning:** Enable system call

TIMING:

INPUT:

X:
U: Pointer to a table of file access information
A: File Control table Entry Number
E:
Other:

OUTPUT: None

X:
U:
A:
E:
Other:

EXCEPTION CONDITION ACTION

☐ No exception possible ☐ Skip on no exception ☒ Exception causes panic condition

Description: Illegal instruction if input invalid.

DISMISSAL CONDITIONS: File directory not in core

FUNCTION: Changes the file access protection to that specified in the table. The table format is the same as that specified in the CRFIL system call.

SYSTEM CALL DESCRIPTION

MNEMONIC: RFILA **NUMBER:** 106

NAME: Read File Access Bits

PRIVILEGE LEVEL

☐ Listener ☐ Exec ☐ Subsystem ☒ User ☐ Other (Specify)

CAPABILITIES BITS: None

Number: Meaning:

TIMING:

INPUT:

X:
U:
A: File Control Table index
E:
Other:

OUTPUT:

X:
U: Access bits for this process (C
A:
E:
Other:

EXCEPTION CONDITION ACTION

☐ No exception possible ☐ Skip on no exception ☒ Exception causes panic condition

Description: Illegal instruction if FCT index invalid

DISMISSAL CONDITIONS:

FUNCTION: Reads bits into the A register that specify the access that this process has to the designated open file. Bits 12-15 specify Close, Read, Write, and Execute respectively.

SYSTEM CALL DESCRIPTION

MNEMONIC: PFILA NUMBER: 107

NAME: Propagate File Access Bits

PRIVILEGE LEVEL

☐ Listener ☐ Exec ☐ Subsystem ☒ User ☐ Other (Specify)

CAPABILITIES BITS:

Number: 27 Meaning: Enable System Call

TIMING:

INPUT:

X: PET index of subsidiary process
U: Access bits to be propagated (C,R,W,E in bits 12-15)
A: FCT index
E:
Other:

OUTPUT: None

X:
U:
A:
E:
Other:

EXCEPTION CONDITION ACTION

☐ No exception possible ☐ Skip on no exception ☒ Exception causes panic condition

Description: Illegal instruction if input parameters invalid.

DISMISSAL CONDITIONS: None

FUNCTION: Selectively propagates access bits to an open file from this process to the designated subsidiary process. Access bits to be propagated are specified by bits 12-15 of the U register; they designate Close, Read, Write, and Execute access respectively. Bits 12-15 of U are ANDed with the corresponding bits from this process's PET and the result is placed in the PET of the subsidiary process.

SYSTEM CALL DESCRIPTION

MNEMONIC: DFIL **NUMBER:** 108

NAME: Delete File

PRIVILEGE LEVEL

☐ Listener ☐ Exec ☐ Subsystem ☒ User ☐ Other (Specify)

CAPABILITIES BITS:

Number: 20 Meaning: Enable System Call

TIMING:

INPUT:

X: Address of string pointer pair specifying file name
U: Address of string pointer pair specifying password
A:
E:
Other:

OUTPUT:

X: error code on exception return
U:
A:
E:
Other:

EXCEPTION CONDITION ACTION

☐ No exception possible ☒ Skip on no exception ☐ Exception causes panic condition

Description: File can't be deleted; reason in X.

DISMISSAL CONDITIONS: File directory not in core

FUNCTION: Deletes the specified file by removing it from the directory and releasing all its data and index pages. Write access to the file is required. If the designated file is a directory it must be empty.

SYSTEM CALL DESCRIPTION

MNEMONIC: DCFIL **NUMBER:** 109

NAME: Delete Contents of File

PRIVILEGE LEVEL

☐ Listener ☐ Exec ☐ Subsystem ☒ User ☐ Other (Specify)

CAPABILITIES BITS:

Number: 19 **Meaning:** Enable System Call

TIMING:

INPUT:

X:
U: Address of string pointer pair specifying password
A: FCT index
E:
Other:

OUTPUT:

X: Error code on exception return
U:
A:
E:
Other:

EXCEPTION CONDITION ACTION

☐ No exception possible ☒ Skip on no exception ☐ Exception causes panic condition

Description: File contents can't be deleted; reason on X

DISMISSAL CONDITIONS: File index table not in core

FUNCTION: Deletes all data pages for the file, resetting it to the state it had when first created. Write access to the file is required. This call may not specify a file directory.

SYSTEM CALL DESCRIPTION

MNEMONIC: CLFIL **NUMBER:** 110

NAME: Close File

PRIVILEGE LEVEL

☐ Listener ☐ Exec ☐ Subsystem ☒ User ☐ Other (Specify)

CAPABILITIES BITS:

Number: 18 **Meaning:** Enable System Call

TIMING:

INPUT:

X:
U:
A: FCT index
E:
Other:

OUTPUT: None

X:
U:
A:
E:
Other:

EXCEPTION CONDITION ACTION

☐ No exception possible ☐ Skip on no exception ☒ Exception causes panic condition

Description: Illegal instruction if FCT index invalid.

DISMISSAL CONDITIONS: Open File Index Table, directory and open file table not all in core.

FUNCTION: Closes the specified file and updates information about it in a file directory and the open file table. If the file is ephemeral, it is automatically deleted.

SYSTEM CALL DESCRIPTION

MNEMONIC: CAFIL NUMBER: 111

NAME: Close All Files

PRIVILEGE LEVEL

☐ Listener ☐ Exec ☐ Subsystem ☒ User ☐ Other (Specify)

CAPABILITIES BITS:

Number: 28 Meaning: Enable System Call

TIMING:

INPUT: None

X:
U:
A:
E:
Other:

OUTPUT: None

X:
U:
A:
E:
Other:

EXCEPTION CONDITION ACTION

☒ No exception possible ☐ Skip on no exception ☐ Exception causes panic condition

Description:

DISMISSAL CONDITIONS: Open File Index Tables, directories, and open file table not all in core.

FUNCTION: Creates the same effect as executing a CLFIL for all open files except the currently open file directory. If no files are open this call acts as a NO - OP.

SYSTEM CALL DESCRIPTION

MNEMONIC: APFIL **NUMBER:** 112

NAME: Attach Page of File

PRIVILEGE LEVEL

☐ Listener ☐ Exec ☐ Subsystem ☒ User ☐ Other (Specify)

CAPABILITIES BITS:

Number: 21 **Meaning:** Enable System Call
49 Allow drum sector specification

TIMING:

INPUT:

X: File page number
U: Core page number
A: FCT index in bits 11-15; bit 0 = 1 means specific drum sector requested
E: Drum sector requested if $U_0 = 1$
Other:

OUTPUT:

X:
U:
A:
E: Drum sector granted if one was requested
Other:

EXCEPTION CONDITION ACTION

☐ No exception possible ☐ Skip on no exception ☒ Exception causes panic condition

Description: Illegal instruction if input parameters invalid. Memory panic if core page does not have write access.

DISMISSAL CONDITIONS: OFIT not in core when call issued or file page not in core when first referenced.

FUNCTION: Attaches the specified file page to the specified core page. Protection on the core page is set to correspond to the file protection bits. Original information in the given core page (if any) is lost. Write access to the core page is required. If the designated core page already has a file page attached to it, that attachment is broken. This is unlike detaching the old file page first in that all processes that referred to this PMT entry still refer to it (and consequently now refer to the newly attached page).

SYSTEM CALL DESCRIPTION

MNEMONIC: DPFIL NUMBER: 113

NAME: Detach Page of File

PRIVILEGE LEVEL

☐ Listener ☐ Exec ☐ Subsystem ☒ User ☐ Other (Specify)

CAPABILITIES BITS:

Number: 22 Meaning: Enable System Call

TIMING:

INPUT:

X:
U: Core page number
A:
E:
Other:

OUTPUT: None

X:
U:
A:
E:
Other:

EXCEPTION CONDITION ACTION

☐ No exception possible ☐ Skip on no exception ☒ Exception causes panic condition

Description: Illegal instruction if input parameter invalid.

DISMISSAL CONDITIONS: None

FUNCTION: Detaches the specified page from the process. The PMT entry previously occupied by the pointer to the attached page is released. All relabeling registers pointing to the released PMT entry (in all processes) are set to null.

SYSTEM CALL DESCRIPTION

MNEMONIC: ACFIL **NUMBER:** 114

NAME: Activate File

PRIVILEGE LEVEL

☐ Listener ☐ Exec ☐ Subsystem ☒ User ☐ Other (Specify)

CAPABILITIES BITS:

Number: 23 **Meaning:** Enable System Call

TIMING:

INPUT:

X:
U:
A: FCT index
E:
Other:

OUTPUT: None

X:
U:
A:
E:
Other:

EXCEPTION CONDITION ACTION

☐ No exception possible ☐ Skip on no exception ☒ Exception causes panic condition

Description: Illegal instruction if input parameter invalid. Memory panic if working set exceeded.

DISMISSAL CONDITIONS: Open File Index Table not in core.

FUNCTION: Puts the OFIT for the specified file in the system working set for the calling process. Subsequent access to the file will then normally require only one dismissal instead of two.

SYSTEM CALL DESCRIPTION

MNEMONIC: DEFIL NUMBER: 115

NAME: Deactivate File

PRIVILEGE LEVEL

☐ Listener ☐ Exec ☐ Subsystem ☒ User ☐ Other (Specify)

CAPABILITIES BITS:

Number: 23 Meaning: Enable System Call

TIMING:

INPUT:

X:
U:
A: FCT index
E:
Other:

OUTPUT: None

X:
U:
A:
E:
Other:

EXCEPTION CONDITION ACTION

☐ No exception possible ☐ Skip on no exception ☒ Exception causes panic condition

Description: Illegal instruction if input parameter invalid.

DISMISSAL CONDITIONS: None

FUNCTION: Removes the OFIT for the specified file from the system working set for the calling process. Subsequent access to the file will then normally require at least two dismissals instead of only one. If the specified file is already inactive, this call acts as a NO-OP.

SYSTEM CALL DESCRIPTION

MNEMONIC: GPFIL NUMBER: 116

NAME: Get Page of File

PRIVILEGE LEVEL

☐ Listener ☐ Exec ☐ Subsystem ☒ User ☐ Other (Specify)

CAPABILITIES BITS:

Number: 24 Meaning: Enable System Call

TIMING:

INPUT:

X: File page number
U: Core page number
A: FCT index
E:
Other:

OUTPUT: None

X:
U:
A:
E:
Other:

EXCEPTION CONDITION ACTION

☐ No exception possible ☐ Skip on no exception ☒ Exception causes panic condition

Description: Illegal instruction if input parameter invalid. Memory panic core page doesn't have write access.

DISMISSAL CONDITIONS: OFIT not in core when call issued or file page not read into core page when it is next referenced.

FUNCTION: Reads a copy of the specified file page into the specified core page. Read access to the file and write access to the core page are required.

SYSTEM CALL DESCRIPTION

MNEMONIC: PPFIL **NUMBER:** 117

NAME: Put Page of File

PRIVILEGE LEVEL

☐ Listener ☐ Exec ☐ Subsystem ☒ User ☐ Other (Specify)

CAPABILITIES BITS:

Number: 25 **Meaning:** Enable System Call

TIMING:

INPUT:

X: File Page Number
U: Core Page Number
A: FCT index
E:
Other:

OUTPUT: None

X:
U:
A:
E:
Other:

EXCEPTION CONDITION ACTION

☐ No exception possible ☐ Skip on no exception ☒ Exception causes panic condition

Description: Illegal instruction if input parameter invalid or file is write protected.

DISMISSAL CONDITIONS: OFIT is not in core when call is issued or clean core page is not available before page is next referenced.

FUNCTION: Writes a copy of the specified core page into the specified page of the file. Read access to the core page and write access to the file are required.

SYSTEM CALL DESCRIPTION

MNEMONIC: RRTC **NUMBER:** 125

NAME: Read Real Time Clock

PRIVILEGE LEVEL

☐ Listener ☐ Exec ☐ Subsystem ☒ User ☐ Other (Specify)

CAPABILITIES BITS: None

Number: **Meaning:**

TIMING:

INPUT: None

X:
U:
A:
E:
Other:

OUTPUT:

X: Date
U: Real time clock value
A:
E:
Other:

EXCEPTION CONDITION ACTION

☒ No exception possible ☐ Skip on no exception ☐ Exception causes panic condition

Description:

DISMISSAL CONDITIONS: None

FUNCTION: Returns the current date and value of the real time clock in the X, U and A registers. The date is expressed as:

Bits 0-6 = Year--1970
Bits 7-10 = Month
Bits 11-15 = Day

The real time clock expresses the time of day in milliseconds beginning at midnight. The high order bits of the time of day are in the U register.

SYSTEM CALL DESCRIPTION

MNEMONIC: SRTC **NUMBER:** 126

NAME: Set Real Time Clock

PRIVILEGE LEVEL

☐ Listener ☐ Exec ☐ Subsystem ☐ User ☒ Other (Specify) Operator's Exec

CAPABILITIES BITS:

Number: 8 **Meaning:** Enable System Call

TIMING:

INPUT:

X: Date
U: Real Time Clock Value
A:
E:
Other:

OUTPUT: None

X:
U:
A:
E:
Other:

EXCEPTION CONDITION ACTION

☐ No exception possible ☒ Skip on no exception ☐ Exception causes panic condition

Description: New value is inconsistent with old value.

DISMISSAL CONDITIONS: None

FUNCTION: Sets the date and real time clock from the X, U and A registers. The date is expressed as:

Bits 0-6 = Year -- 1970
Bits 7-10 = Month
Bits 11-15 = Day

The real time clock expresses the time of day in milliseconds beginning at midnight. The high order bits of the time of day are in the U register.

SYSTEM CALL DESCRIPTION

MNEMONIC: RAD **NUMBER:** 127

NAME: Read Accounting Data

PRIVILEGE LEVEL

☐ Listener ☐ Exec ☐ Subsystem ☒ User ☐ Other (Specify)

CAPABILITIES BITS:

Number: Meaning:

TIMING:

INPUT:

X: Address of beginning of table
U:
A: Selection bits
E:
Other:

OUTPUT:

X:
U:
A:
E:
Other: Accounting data placed in table

EXCEPTION CONDITION ACTION

☐ No exception possible ☐ Skip on no exception ☒ Exception causes panic condition

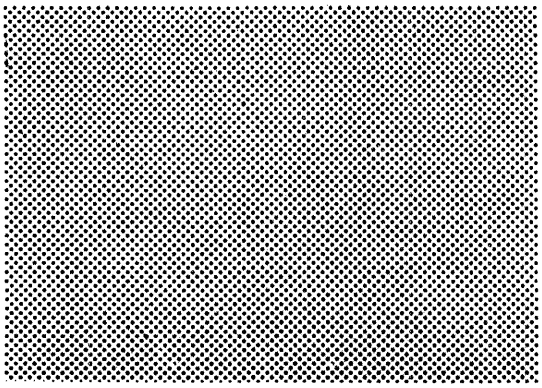
Description: Illegal instruction if selection bits are invalid.

DISMISSAL CONDITIONS: None

FUNCTION: Copies the selected items of accounting data into consecutive locations in the specified table. The selection bits are as follows:

0 - Connect time (seconds) — 1 word
1 - CP time (seconds) - 1 word
2 - Memory use (page . seconds) - 2 words
3 - Drum use (page . seconds) - 2 words
4 - Disk reads and writes (count) - 2 words
5 - Characters input from terminal (count) - 2 words
6 - Characters output to terminal (count) - 2 words

Accounting is from the time the user logs on and is for all processes belonging to the user.



Appendix B . . .

Intraline Editing

1.1 INTRODUCTION

An INTRALINE EDITING capability is provided for most LOGICON 2+2 Terminal input. This capability presents a uniform interface to user processes or subsystems and another uniform interface to users of similar terminals. Users of dissimilar terminals (e.g., IBM 2741 versus Teletype) will observe different interface conventions. The interface described on the following pages is for Teletype-like terminals.

Intraline editing on Teletype-like terminals depends on the use of the control characters (produced on a teletype by depressing the CONTROL key in conjunction with an alphabetic key) to specify the manipulation of text strings, which in general do not contain control characters. These control characters are written with a superscript ^{c}, as A^{c}, B^{c}, e etc. An example of an editing control character is K^{c}, which means "delete the preceding character." Thus the character string:

$$THFK^{c}E$$

typed at a terminal would result in an edited string:

$$THE$$

and in a printout of:

$$THF\leftarrow E$$

The intraline editing capability is called by a subsystem or a user process. Certain characteristics of the editing process are determined by the calling subsystem: only those features determined by the intraline editing capability itself are described herein.

In general, the calling subsystem will request an edited line from the intraline editor, which then collects terminal characters through the next line terminating character, and returns the entire line to the calling subsystem.

The intraline editing capability will be described in three parts:

- User Interface.
- Subsystem Interface.
- Program Design Approach.

2.1 USER INTERFACE

2.1.1 Definition of a Line

A line of text is a string of characters terminated by an end of line character. In most languages, carriage return is recognized as the end of line character and a line of text always ends with a carriage return. A line may contain no more than 256 characters, including the end of line character. It is possible to define a maximum line length of less than 256 characters, if desired.

2.1.2 The "Old Line"

Many of the editing control characters produce interaction between a "new line," or the line being entered, and an "old line." This is useful when a number of lines of similar nature are to be produced, or when a previously entered line requires correction. In general, during the text entry process, the "old line" is the line just entered, and the "new line" is the line currently being typed. Certain subsystems have commands that result in a different definition of "old line."

During entry of the first line of text there is no "old line," and certain control characters are illegal.

2.1.3 The Old Line Character Position

In general the editing process consists of combining characters typed at the terminal with characters selected from the old line. The control characters specify how the combination is to be effected. In order to know the effect of an editing action, it is necessary to understand where the "next character" is in the old line. The following actions affect the position of the next character in the old line:

- When a new line is started, the "next character" of the old line is the first character. (Assuming there is an old line)

- When a character is typed at the terminal, it is appended to the new line, and next character position in the old line is advanced.

  Example:

```
                NC
                ↓
Old Line    ABCDEFG
New Line    GFE
```

  A "D" is now typed in the new line:

```
                NC
                ↓
Old Line    ABCDEFG
New Line    GFED
```

- Many editing control characters result in either copying or skipping characters in the old line. In such cases, the next character position is advanced over all characters copied or skipped, to the character following the last one copied or skipped.

- It is possible, using appropriate control characters, to "insert" characters in the new line without affecting the next character position in the old line.

- Some control characters cause "backspacing," or deletion of characters in the new line. The old line next character position is affected in the following manner in this event:

  1. The number of characters in the new line that are to be deleted, not counting any "inserted" characters, is determined.
  2. The old line next character position is backed up this number of non-skipped characters.

  Example:

```
                          Next Character
                Skipped   ↓
                ┌──┐
Old Line    ABCGHIEF
New Line    ABQDE
               ‾‾
            Inserted
```

A backspace of 3 characters is performed on the new line. The situation is then:

```
                Next Character
                      ↓
Old Line        ABCGHIEF
New Line        AB
```

2.1.4 End of Text

Sometimes it is necessary to recognize the end of a group of lines. A special control character provides such an end of text signal. It must be entered at the end of a line. If an end of text character is entered without an end of line character immediately preceding it, end of line is assumed.

2.1.5 Assignment of Control Character Meanings

There is no way to provide meaningful mnemonic names to a large set of control characters to aid the operator's memory. For this reason, a plastic overlay that furnishes mnemonic names to editing control characters is planned for 2+2 terminals. A possible layout of such an overlay is shown in Figure B-1.

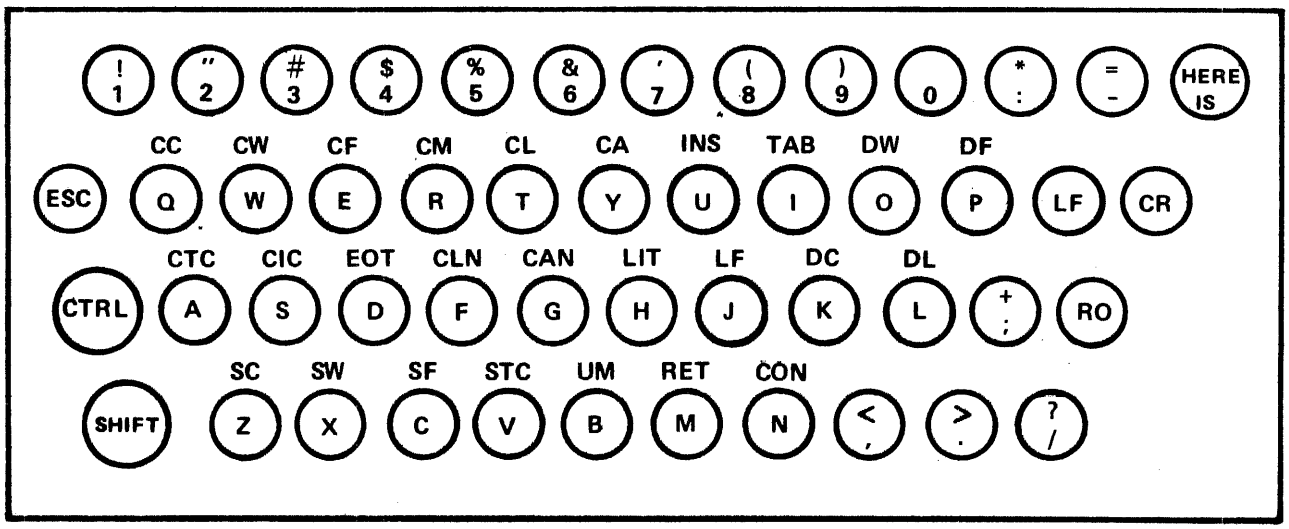


Figure B-1. Possible Intraline Editor Overlay

2.1.6 Summary of Intraline Editor Characters

The editing control characters are defined in the following table (Table B-1) in terms of their mnemonic (CC, CW, etc.) names.

TABLE B-1. SUMMARY OF CONTROL CHARACTERS

Mnemonic	Symbol Printed	Function	Error Signal
DC	←	Delete Character:	
	*	Deletes preceding character in new line.	Bell if no characters in new line.
DW	\	Delete Word:	
		Deletes preceding word in the new line. Any trailing blanks, and all consecutive non-blanks back to, but not including the next blank or the beginning of the new line, are deleted.	Bell if no non-blanks in new line. All blanks will be deleted anyway.
DF		Delete Field:	
		Delete characters in new line back to previous tab stop or to beginning of line.	Bell if no characters in new line.
DL	↑	Delete Line:	
	*	Deletes any text in new line. If there is no text in new line, and using program allows multiple line deletions, passes a "delete previous line" signal back to currently active subsystem. If no text in new line and using program does not allow multiple line deletions, take no action.	Bell if it cannot delete previous line, and no text is in new line.
CC		Copy Character:	
		Copies next character in old line and appends to new line; prints character copied. If copied character is line termination character, new line is ended.	Bell if no old line.

*These are not standard ASCII characters. On some terminals these will print as _ and ^ respectively.

TABLE B-1. SUMMARY OF CONTROL CHARACTERS (Continued)

Mnemonic	Symbol Printed	Function	Error Signal
CM and a digit 1-9		Copy Multiple Characters: Copies the indicated number of characters from the old line and appends them to new line; prints characters copied. If a line termination character is copied, new line is ended.	Bell if no old line.
CW		Copy Word: Copies characters from the old line up to but not including the first blank after the next non-blank, appends characters copied to new line and prints them. This function will not copy the end of line character unless it is the first non-blank encountered. If it is copied, the new line is ended.	Bell if no old line.
CF		Copy Field: Copies characters from old line and appends them to new line until next tab stop is reached in new line. Prints characters copied. If this results in copying a line termination character, the new line is terminated.	Bell if no old line.
CL		Copy Line: Copies rest of old line up to, but not including the line termination character, appending characters to new line. Prints characters	Bell if no old line.

TABLE B-1. SUMMARY OF CONTROL CHARACTERS (Continued)

Mnemonic	Symbol Printed	Function	Error Signal
CL (Cont)		copied. Since termination character was not copied, new line entry process continues.	
CA		Copy All: Copies rest of old line and appends to new line; prints characters copied. Since line termination character in old line is copied, this always ends the new line.	Bell if no old line.
CTC and a character		Copy to Character: Copies the old line up to, but not including the next occurrence of the character typed after it, printing the characters copied, and appending them to the new line.	Bell if no old line or if there are no more occurrences of next character in old line.
CIC and a character		Copy Including Character: Copies and prints the old line up to and including the character typed after it, appending characters to the new line. If a line terminating character is copied, terminates the new line.	Bell if no old line or if no more occurrences of next character in old line.
CLN		Copy Line, No Print: Copies the rest of the old line up to, but not including the line termination character, appending characters to new line. Does not print.	Bell if no old line.

TABLE B-1. SUMMARY OF CONTROL CHARACTERS (Continued)

Mnemonic	Symbol Printed	Function	Error Signal
CAN		Copy All, No Print: Copies the rest of the old line including the line termination character, appending characters to new line. Does not print. Ends new line:	Bell if no old line.
SC	%	Skip Character: Skips next character in old line, appends nothing to new line.	Bell if no old line.
SW	%	Skip Word: Skips characters in old line up to, but not including the first blank after the next non-blank. Will not skip the line termination character unless it is the first non-blank encountered. Prints % for each character skipped.	Bell if no old line.
SF	%	Skip Field: Skips characters in old line up to the next tab stop. Prints % for each character skipped.	Bell if no old line.
STC and a character	%	Skip to Character: Spaces over characters from old line up to, but not including the next occurrence of the character typed after it. Types % for each character so spaced over. Appends nothing to new line.	Bell if no old line or if no more occurrences of next character in old line.
CON		Continue Edit: Considers the new line with carriage return appended as an	None

TABLE B-1. SUMMARY OF CONTROL CHARACTERS (Continued)

Mnemonic	Symbol Printed	Function	Error Signal
CON (Cont)		old line and initiates an intraline edit with the new old line and an empty new line. Does not return a line to the calling subsystem.	
INS text INS	< >	Insert: Appends text to new line. Next character position in old line does not change as INS's and characters between INS's are typed. First INS prints as <, second as >. Any line terminating action, skip control, or any copy from previous line action automatically supplies the second INS. More than one pair of INS's may be used in a single line.	None
LIT and a character		Literal: Used before control characters to inhibit their usual functions. LIT followed by any other character has the following effect: The next character position in the old line is advanced one space, unless INS is in effect, and the character following the LIT is appended to the new line and printed if it has a normal echo (which most control characters do not).	None
UM (Mapped onto B^C)	!	Universal Match: Appended to new line just as a normal non-control character would be; no intraline editing	None from intraline editor. Many subsystems will

TABLE B-1. SUMMARY OF CONTROL CHARACTERS (Continued)

Mnemonic	Symbol Printed	Function	Error Signal
UM (Cont)		consequences. Used by EDITOR subsystem as a "universal match" character in string comparisons. For example, ABB^{C} means AB followed by any character.	treat B^{C} as an illegal character.
TAB		Append blanks to new line until next tab stop is reached. Print blanks appended (i.e., move carriage appropriate number of spaces). Move old line next character position to the same tab stop selected for new line.	None
RET	Carriage Return-Line Feed	Carriage Return: Spaces to beginning of next line on printed output. Carriage return character is appended to new line, which is then terminated.	None
LF	Carriage Return-Line Feed	Line Feed: Spaces to beginning of next line on printed output without ending line. Line feed character is appended to new line, and next character position in old line is advanced just as for any other character.	None
EOT		End of Text: End of text signal. Ends current group of lines for whatever line entry action was in process.	None

2.1.7 Error Response

In general, when a control character results in an error condition, no effect occurs on either the old or the new line.

For example, CTC X results in no action if there are no more occurrences of X in the old line.

2.1.8 Tab Stops

Tab stops are set at columns 10, 16, 35, 45, 55, and 65 unless changed by the calling subsystem.

LOGICON, INC.

	LOS ANGELES	SAN DIEGO	WASHINGTON, D.C.
[illegible]	255 West Fifth Street	1075 Camino Del Rio South	6231 Leesburg Pike
[illegible]	San Pedro, California 90731	San Diego, California 92110	Falls Church, Virginia
[illegible]	Phone: (213) 831-0611	Phone: (714) 291-4240	Phone: (703) 534-7087