1047

Bell Laboratories

subject: Explanation of Abnormal Conditions within the UNIX Operating System

date: March 17, 1975

from: T. M. Raleigh
MF-75-8234-28

ABSTRACT

Error messages printed by the UNIX operating system on the system console are discussed with their causes, implications and remedies. The messages are divided into categories according to their seriousness and their source from within the associated operating system source modules is catalogued. An indication of the relationship of various system parameters is given for several of the remedies as a guide to the dangers of haphazardly changing system parameters. A brief summary of user and system virtual address layout is given and trap handling is discussed as required to interpret UNIX PANIC TRAPS. An example of a PANIC TRAP is given and device errors printed by the recent implementation of error logging are also described.

Bell Laboratories

subject: Explanation of Abnormal Conditions within the UNIX Operating System

date: March 17, 1975

from: T. M. Raleigh
MF-75-8234-28

MEMORANDUM FOR FILE

I. Introduction

Error messages printed by the UNIX operating system on the system console are discussed with their causes, implications and remedies.

Messages from the UNIX operating system are roughly broken down into two categories, PANIC messages(which are followed by the machine halting) and WARNING messages from the system.

The two categories of messages printed by UNIX roughly correspond to situations where the system can continue running(WARNINGS) and where further operation could result in extensive file system damage(PANIC). For a number of PANIC situations there exist recovery strategies that would allow the system to continue operation at a degraded level, however, because there has as yet been no great demand for this kind of recovery on UNIX, because PANIC conditions do not occur frequently and do not usually violate the integrity of the system, and because of size limitations on UNIX(24K word virtual address space for the system) these strategies have not been implemented.

The two major categories are, for the purpose of this discussion, broken down into several subcategories for convenience.

II. PANIC Messages

A PANIC refers to the occurrence of a significant error within the system for which self diagnosis and healing are too involved and/or risky. Messages printed when these situations occur are called appropriately PANIC messages and are followed by the processor going into an idle state. Messages in this category are loosely divided into three subcategories. The first group can be expected in the normal operation of the system and will be designated as NORMAL PANICS. They are commonly seen at system startup time and are usually the result of improper specifications of the swap and root device and devices offline ,however, they also appear when file systems or swap area is filled. The second subcategory of PANICS should never be seen on a system. They are the result of the failure of some consistency check made by the system on essential table linkages. We will designate these PANIC situations as ABNORMAL PANICS for lack of a better name. They are usually the result of some improper modification to the system software, a wild store by new system software or possibly a bad core location. A third subcategory of PANICS are the result of traps occurring while executing in Kernel mode, that is while executing operating system code. The type of trap producing this PANIC is determined by using the console switches and requires some knowledge of how the operating system's Virtual Address space is laid out and how the Memory Management Unit operates. The method for making this determination is outlined below and explained in more detail in a succeeding memo.1

III. WARNING Messages

Warning messages constitute the second major category of messages printed by the operating system. In general, they are errors that fall into one of the following three categories,

1. Self healing in the sense that the system usually ignores them or discards the bad information and continues. This class consists of most device errors and errors in handling refreshed data(see BUSY I).

2. Errors that can be repaired by software without taking the machine down. This typically refers to problems in file systems that are not essential to the system (any file system except the root) and which may be logically dismounted and repaired.

3. Errors for which removal of the offending device only degrades the size of the system. In cases where devices or logical file systems are not essential to the operation of the system, they may be removed from the system without the need of halting the system. While the device cannot be taken off line and hardware serviced by repair personnel, their loss only entails some degradation in performance and/or size.

In the following descriptions, a particular format is followed. First, each description gives an indication of the function within the system that records the error and the source file in directory /usr/sys of the standard UNIX system where it may be found. An indication of the possible cause of the error is also given. (In the case of ABNORMAL PANICS the message should be taken as only a general indication of where the problem lies as bad software is probably involved and the trouble may not be localized.) Finally, the corrective action or a solution is outlined if the problem is easily solved. As a general rule, for most PANICS, a Post Mortem Dump should be taken(by starting the machine at the address of the dump routine - 44(octal) - see system namelist), the system should be rebooted single user and all file systems previously mounted should be checked for damage.

Caution: The dump routine is not very intelligent. It should write a copy of all of core on a pre-mounted magnetic tape with an EOF mark, however, any error is taken to mean that the end of core has been reached. This means that you must be sure the ring is in, the tape is ready, and the tape is clean and new. If the dump fails, you can try again, but some of the registers will be lost.

IV. NORMAL PANIC MESSAGES

The following list of panic messages can occur in the normal operation of UNIX. They occur on an infrequent basis, mostly at startup time, however, some appear when the load on the system or space available on the file systemss are not properly balanced.

4.1 PANIC IINIT

source - file ken/alloc.c, function - iinit()

This message is printed during the UNIX startup procedure if the root file system cannot be mounted. It is the result of a bad read on the superblock of the root file system. It can be caused by the root device being offline, by an unrecoverable read error of the superblock on the root device or by an improper specification of the major and minor device for the root device in file conf.c.

Solution - Correct file conf.c; turn root device on. Reboot.

4.2 PANIC OUT OF INODES

source - file ken/alloc.c, function - ialloc()

Attempting to allocate an inode on a file system for which there are no free inodes results in a "PANIC OUT OF INODES". This panic occurs whether the file system is the root file system or a mounted file system.

Solution - The remedy is to do housekeeping on the file system. This can be done by removing unneeded files on the file systems or by creating a larger file system with more inodes. To do this, dump the file system on tape, create a larger file system using /etc/mkfs and then restore the file system from tape. (see Section VIII of the UNIX Programmers Manual)

4.3 PANIC NO INODES

source - file ken/iget.s, function - iget()

This PANIC occurs when too many different files are open in the system resulting in the filling of the in-core inode table(INODE).

Solution - Post Mortem Dump. Either ignore it as a freak occurrence of an overloaded system or reboot and determine which processes were running and which inodes were open at the time of the crash. Files (inodes) are not closed or removed from the in-core inode table when a process is swapped so that inodes used by swapped processes are also in the process table. Determining which processes were running or swapped and which files were open requires use of the debugger on the dump and extensive knowledge of the system tables. If it is found that several processes (which probably run together as part of an application) are filling the in-core inode table then it should be determined whether they require all of the files be open at once. It should be remembered that files remain open across the creation of children so that unneeded files may remain open in the system long after the death of a parent process. Files can be closed by the parent before calling the child and the child may reopen them as needed. Also, mounted file systems require two entries in the in-core inode table.(see Fig.1) These entries are present as long as the file system is mounted. One inode table entry is for the root inode of the mounted file system and the other is for the inode on which the file system is mounted. Thus, the size of the in-core inode table is effectively decreased for each mounted file system(including the root). If it is determined that a larger number of open files is required, the size of the in-core inode table may be increased by changing the parameter NINODE in param.h. The size of the file table (FILE - the number of instances of an open file) should also be increased proportionately by changing NFILE in param.h The variable MAXIP holds the high water mark of the highest inode in the INODE table. It can be examined while the system is running to determine how close the system is to this condition.

4.4 NO CLOCK

source - file ken/main.c, function - main()

Printed at system startup time if neither the KW11-L or KW11-P clock is present on the machine.

Solution - Run DEC diagnostics. Reboot.

4.5 OUT OF SWAP SPACE

source - file ken/text.c, function - xswap()

When UNIX attempts to swap a process, and there is no space left in the swap area or the swap area is fragmented so badly that there isn´t a space large enough to fit the process, this message is printed.

Solution - The solution in all cases is to increase the size of the swap area. This is done by changing the parameter NSWAP (number of blocks in the swap area) in file conf.c. Conceivably, if the number of processes allowed in the system (NPROC - see param.h) is also increased significantly, increasing the size of the swap area might require an increase in the size(SMAPSIZE - see param.h) of the table managing the allocation of the swap area(swapmap).

4.6 PANIC SWAP ERROR

This message can be printed by two functions in the system.

1. source - file ken/slp.c, function - sched()

If an unrecoverable error occurs when attempting to read or write from the swap device, this message is printed.

2. source - file ken/text.c, function - xswap()

If an unrecoverable device error occurs on the swap device while attempting to write the original copy of the text of a shared program on the swap device, "PANIC SWAP ERROR" is also printed. Only if the error recovery procedure (on most devices retry 10 times) for the respective block device driver fails will this message appear. The message does however, appear frequently at startup time if either the swap device is offline or it has been specified improperly in conf.c.

Solution - If the swap device is offline, turn power on or disable write protect on the drive. The entries in the file conf.c should be checked carefully, particularly SWAPDEV, NSWAP and SWAPLO if the problem occurs at startup. SWAPDEV specifies the major and minor device number of the swap device. NSWAP specifies the number of blocks (512 byte) in the swap area and SWAPLO specifies the offset in blocks into the logical device area that the swap space begins. A peculiarity of the offset SWAPLO is that UNIX has built into it the notion that block 0 of any logical device cannot be allocated. It is used for a boot program. Therefore, offsets of zero for SWAPLO promptly produce "PANIC SWAP ERROR" at startup. Reboot.

4.7 PANIC OUT OF TEXT

source - file ken/text.c, function - xalloc()

Under UNIX, pure procedures have the text (instructions) and data loaded and managed separately (they are however in the same user virtual space - see Fig. 2). The data segment is managed by the process table entry(PROC) and the text segment is managed by a text(TEXT) table entry. When there are no more TEXT table entries available at the time of creation of a shareable process, "PANIC OUT OF TEXT" is printed. For non-shareable processes, the Process table manages both the text and data (as one unit) so there is no associated text table entry, but for shareable processes, it manages only the data segment. A pointer in the process table entry to a text entry is kept, therefore, when this PANIC message occurs, the shareable processes (running or swapped) should be examined (those for which p_ptext in the PROC table entry is nonzero).

Solution - Post Mortem Dump. The options are essentially the same as those of "PANIC NO INODES". The occurrence can be ignored or an analysis of how many text segments were in core could be made and it can be determined whether the number of text table entries (NTEXT - see param.h) should be increased. Reboot.

V. ABNORMAL PANIC MESSAGES

The following messages should never occur on a system. They are printed as the result of the failure of some basic consistency check and are the result of a serious bug or wild store in the operating system. UNIX is protected from user processes by the Memory Management Unit so that only operating system code can produce a wild store.

5.1 PANIC NO FS(no File System)

source - file ken/alloc.c, function - getfs()

Occurs when the system cannot find the in-core superblock for a particular device by searching the systems MOUNT table. There is one MOUNT table entry for each logical file system that is mounted. When a file system is mounted, one of the system buffers(BUF) is withdrawn from the pool of system buffers and is allocated to hold the superblock for that logical file system. The buffer remains allocated to that file system for the duration of time that it is mounted. The MOUNT table entry contains a pointer to this buffer (see Fig.1)

Solution - Post Mortem Dump. Software analysis using the debugger. Reboot.

5.2 PANIC NO IMT(No Mount Table Entry For Inode)

source - ken/iget.c, function - iget()

When a logical file system is mounted on a file(inode), both the inode for the file and the root inode of the logical file system (see Fig.1) are kept in the in-core inode table(INODE). The connection between the two is made by a bit (IMOUNT) set in the in-core inode table entry for the file, indicating that the MOUNT table should be consulted for the root inode mounted on this file (see Fig.1). When the system cannot find the mount table entry to make this connection, "PANIC NO IMT" is printed.

Solution - Post Mortem Dump. Software Analysis using debugger. Reboot.

5.3 NO PROCS(No Process Table Entries)

source - file ken/slp.c, function - newproc()

This message may be printed under two circumstances both involving the creation of a new process. First, if at startup time, the system cannot create the INIT process which spawns processes for each teletype, this message is printed. Second, if during the spawning of a new process (FORK) the system finds that it is out of room in the Process Table when it knows that there is room in the table this message is printed.

Solution - Post Mortem Dump. Software analysis using debugger. Reboot.

Note: This PANIC does not mean that there was an insufficient number of processes SYSGENED into a system.

The FORK system call limits the maximum number of processes in a system to the size of the Process Table (NPROC - see param.h). The FORK system call returns an error to the user process if the system is out of Process Table entries. If an installation requires the simultaneous existence of a large number of processes, NPROC may be changed. The number of shareable text segments allowed should probably also be increased proportionately. To complicate things, a larger number of processes may require an increase in the size of the swap area(NSWAP - see conf.c) and an increase in the number of open files(NFILE - see param.h). The increase in swap area size and number of processes may lead to greater fragmentation of the swap area which in turn may require an increase in the size of the swap map (SMAPSIZE in param.h). In general, the increase in the number of processes requires more adjustment and tuning than for increasing the number of open files in the system (see PANIC NO INODES).

5.4 PANIC UNLINK -- IGET

source - file ken/sys4.c, function - unlink()

If a system call to unlink a file is issued, UNIX must bring the inode for that file into core in order to remove it from the file system. If a directory entry exists for the file but the inode cannot be brought into core this message is printed. This should never occur.

Solution - Post Mortem Dump. Software Analysis using debugger. Reboot.

5.5 PANIC BLKDEV

source - dmr/bio.c, function - getblk()

This message is printed when a request is made for one of the system buffers to do I/ O to a bad logical device (bad major device number specification). Bad logical device numbers should have been rejected at higher levels of system software before reaching the I/O subsystem.

Solution - Post Mortem Dump. Software Analysis using debugger. Reboot.

VI. SYSTEM TRAP PANIC

The third class of PANIC messages are PANICS resulting from a trap occurring while the processor is in Kernel mode. UNIX is presently not equipped to recover from these situations so the processor halts after printing the following cryptic message,

```
KA6 = N1
APS = N2
```

The significance of these numbers is explained below, however, for a detailed description of interrupt and trap handling under UNIX see the succeeding memo. 1

6.1 PANIC TRAP

source - file ken/trap.c, function - trap()

When a "PANIC TRAP" occurs, the messages shown above are printed and the processor goes into a wait state. If the continue switch is pressed on the processor console and the processor is not hung, the message "PANIC TRAP" is printed. As mentioned previously, the PANIC TRAP occurs while the processor is executing UNIX(Kernel mode). The trap can result from an illegal memory reference by the operating system , the execution of an illegal instruction by the operating system ,etc. More likely, it can indicate hardware problems. The traps caught by UNIX and handled for user processes are the following.

Ø	Bus Error
1	Illegal Instruction
2	Breakpoint Trace
3	I/O Timeout
4	Power Fail
5	Emulator Trap
6	System Entry
7	Programmed Interrupt
8	Floating Point
9	Segmentation Violation

If any of these traps occur while executing operating system code, a PANIC TRAP occurs.

The PANIC TRAP message does not print the type of trap producing the PANIC. It is necessary for the operator to determine this from the processor console. Before detailing the algorithm for determining the type of trap, some background information on trap and interrupt handling and on UNIX processes will be summarized

1. Referring to Fig.3, the Kernel's Memory management Registers are shown. Kernel Instruction Space Address Register 6 (KISA6) is shown managing the U block area. Instruction Space Address Register 7 manages the high core device address area.

2. Context switching is accompanied by a change of KISA6 to point to the current process' U block(Fig.3). The scheduler is considered to be a process, albeit a locked in core process.

3. The U block is a 1Ø24 byte area and contains per process information (context switching, open files, current directory, id, etc.) in its lower core portion and contains the system stack growing downward from the high core area of the U block. The per process information is loaded at location 14ØØØØ(octal) in Kernel Virtual address space. There is no protection at present to prevent overwriting of the per process information by the stack(although the 11/45 contains a stack limit register).

4. The system stack resides in the U block of the curently running process. This means that if a user process (or the Scheduler) is interrupted to service a device, all stack usage (temporary variables, subroutine save area, etc.) to fulfill that request, occurs within the U block of the currently running process.

5. Devices send interrupts to the processor by raising their bus request line and the interrupt line on the

UNIBUS. When the processor grants the interrupt, the device sends the address of a new Processor Status Word (PSW) in the low core area of UNIX across the data lines. For traps, the mechanism is essentially the same except that traps are initiated by the processor and do not involve the UNIBUS. The software path that interrupt and traps take to get to their respective trap and interrupt handling routines is slightly different (Fig. 4). They both, however, have a stack frame of arguments constructed by the function CALL (in mch.s) so that entry into the trap and interrupt handling routines looks like a C function call.

6. The stack frame is built as shown in Fig.5. The steps in the process of building the stack frame are as follows,

a. At the time of the trap, the current Processor Status and Program Counter are pushed onto the stack in the order shown in Fig.5b and the new PS and PC are loaded from the proper low core vector.

b. A stack frame of arguments is built as shown in Fig.5c. The entries labeled RØ and R1 are registers rØ and r1 (the remaining registers r2-r5 are saved upon entry to the trap or interrupt handling routine by the function rsave). The PS entry is the current PS which was loaded from the low core vector for the trap. The Previous Mode field of the PS which was not set in the low core vector is set at the time the new PS is loaded and indicates the mode of the processor when the trap occurred. The SP entry is the value of the stack pointer from the previous mode of the processor. The DEV entry is the minor device number (or trap type) from the low core vector.

The information printed during a PANIC TRAP (i.e., KA6 = N1, APS = N2) relates directly to the location of the stack frame described above. The number N1 is the contents of Kernel Instruction Address Register 6(KISA6). This register is kept in 64 byte granularity so that the actual address is this address(N1) left shifted 6 bits(64 bytes). The number N2 is the address in virtual address space of the OLD PS entry on the stack frame. It must be remembered that this is a virtual address and only by following the address mapping algorithm of the Memory Management Unit can the physical address be determined. With the knowledge that KISA6 manages the addresssing of the U block and that the U block is loaded at virtual address 14ØØØØ(octal), the physical address may be found by taking the difference of the APS value and 14ØØØØ(octal) and adding it to the physical address that KISA6 points to,

```
     ADDR_OLD_PS = N1 x 2**6 + (N2 - 140000) (all octal)
```

If this address is examined on the console (on 11/45's when the display switches are set to CONSOLE PHYSICAL and DATA PATHS) the OLD PS may be displayed. The DEV entry may be found by examining the location 14 bytes(octal) lower in physical address space.

```
     ADDR_DEV = N1 x 2**6 + (N2 - 140000) - 14
```

The DEV entry will indicate what type of trap occurred to cause the PANIC(the minor device numbers of traps are assigned as indicated at the beginning of this section). In addition, since PANIC traps occur only while executing operating system code, the OLD PC entry will indicate what section of code was being executed when the trap occurred.

For the example shown in Fig.6, the following information would be printed on the system console,

```
                    APS = 140642
                    KA6 = 3402
```

From the KA6 message we know that the U block is in the area of location 340200. From the APS message we know that the OLD PS is at virtual address,

```
          ADDR_OLD_PS = 3402 x 2**6 + (140642 - 140000)
                      = 340200 + 642
                      = 341042
```

The DEV entry is at location,

```
          ADDR_DEV = 3402 x 2**6 + (140642 - 140000) - 14
                   = 340200 + 642 - 14
                   = 341026
```

The contents of this location is a one (1) and if we consult the list of traps at the beginning of this section we see that this PANIC was caused by the execution of an Illegal Instruction by the operating system. Consulting the NEW PS entry we see that the current and previous processor modes are Kernel. This verifies that the trap occurred while executing operating system code. If we examine the OLD PC entry and consult a namelist for the running system, we can find what function was being executed in the operating systen when the trap occurred (in most cases). Note that the value of the Previous Modes SP is equal to the address of the location preceding it (higher on the stack) in virtual space. This is a consequence of the fact that the previous and current mode are the same for PANIC TRAPS.

On 11/45's it is possible to work in virtual space from the processor console instead of in physical space. This is done by switching from CONSOLE PHYSICAL to KERNEL I space on the console switches. Addresses placed on the console switches for display are then mapped through the Kernel's Memory Management Unit.

VII. WARNING Messages

The following messages are printed on the system console as warnings. They constitute a class of errors which do not warrant halting the system, however, their meaning should be clearly understood as they indicate possible sources of trouble.

7.1 UNIX RELEASE N

When UNIX is booted into core, it prints the following message on the console teletype.

```
UNIX RELEASE 2
MEM = NNNNN
```

The release is the current major revision of UNIX. The number (NNNNN) printed after MEM is the size of user memory in hundreds of words (i.e. lsb = 100 words).

7.2 NO SPACE MAJOR MINOR

source - file ken/- alloc.c, function alloc()

Indicates that there are no more freeblocks on the logical device specified by MAJOR and MINOR (the major and minor device number).

Solution - Do housekeeping on the filesystem. Remove unnecessary files or expand the filesystem (see PANIC NO INODES).

7.3 BAD BLOCK MAJOR MINOR

source - ken/alloc.c, function - badblock()

Printed if during the deallocation of blocks constituting a file, a block number outside the range (within the ILIST area or beyond the end of the file system) of the file system is encountered.

Solution - The file containing the bad block number should be found and fixed. The CHECK command should shed some light on the identity of the file (inode) causing the problem. (File system problems, their variations and remedies are too involved to cover here. Consult section

VIII of the UNIX PROGRAMMERS MANUAL.) The gravity of the problem is dictated by the kind of file involved. If the file is a directory, the remedy is not as simple as if the file were a small file and may even require surgery on the inode itself (consult a guru). Needless to say, the healing process is more complicated if the file is on the root filesystem. Any missing blocks may be restored by using the CHECK -S option (salvage).

7.4 BUSY I

source - ken/alloc.c, function - ialloc()

The superblock for each mounted file system contains a list of free inodes(S_INODE). If a file is to be created, an inode number is chosen from this list. If this inode number corresponds to a file(inode) that is already allocated, "BUSY I" is printed. This message is only a warning from the system . A self healing process occurs whereby inodes are selected from the free inode list until a genuinely free inode is found. The BUSY I message will be printed for each inode that is not really free until a truly free inode is found, or until S_INODE is empty at which time the system will refill S_INODE with 100 free inode numbers. The message may be printed several times at startup if proper and liberal use of the SYNC command has not been made while running single user(i.e., when running single user the superblock, containing the ILIST for the file system, resides in core and is not rewritten onto the disk unless the SYNC command is invoked or the SYNC daemon (/etc/update) is running).

Solution - Be more liberal with SYNC. The mounted filesystems should be checked for integrity as a precaution. The ILIST is locked when an inode is allocated from the ILIST to prevent multiple use of the same inode so this message could indicate problems with the conventions used to lock the list (s_ilock) if system modifications have been made.

7.5 NO FILE

source - ken/fio.c, function falloc()

UNIX maintains a file table(u_ofile in the U block) for each process which contains one entry for each instance of file in use (open) by the process. At present, there is a limit of 15 open files per process in an attempt to keep down the total number of files (NINODE for INODE table) open and the total number of instances of files (NFILE for FILE table) open. When a process attempts to open a file and the FILE table is filled this message is printed. The user process requesting the opening of the file is returned an error.

Solution - If an application must have a process with more than 15 files open at once and cannot manage by closing unneeded files (like the standard input, standard output, controlling teletype, or other files) and reopening them as needed, then NOFILE (the total number of files allowed to be open by a process) can be increased. As with other changes in table sizes, this may have ramifications in othe parts of the system. For example, the in-core inode table (INODE) and file(FILE) table may need to have their sizes increased. Also, the increased size of the open file table (u_ofile[NOFILE] in the U block) decreases the space for the system stack and increases the probability of the system stack overwriting the per process information (U area). This problem occurred in an earlier release of UNIX and resulted in the increase in the size of the U block from 512 bytes to its current 1024 bytes.

7.6 DEVERR ON MAJOR/MINOR BN=BBB EO=EEEEEE

Error logging has recently been implemented for some of the block devices. Error information is printed in the above format with MAJOR and MINOR being the major and minor device number (in decimal) of the faulting device. BBB is the logical block number (in decimal) that the data is being transferred to or from on the device. Remember that the large block devices are segmented into logical areas by minor device and this block number can be considered to be an offset into that area. The six digit octal number EEEEEE is a printout of one of the error registers from the device. The significance of the code for each of the devices is as follows.

```
0/n - RK11
100000 - drive error - composite
040000 - overrun
020000 - write lock out violation
010000 - seek error
004000 - programming error
002000 - non-existent memory
001000 - data late
000400 - timing error
000200 - non-existent disk
000100 - non-existent cylinder
000040 - non-existent sector
000020 - unused
000010 - unused
000004 - unused
000002 - checksum error
```

```
ØØØØØ1 - write check error

1/n RPØ3
1ØØØØØ - write protect violation
Ø4ØØØØ - file unsafe violation
Ø2ØØØØ - non-existent cylinder
Ø1ØØØØ - non-existent track
ØØ4ØØØ - non-existent sector
ØØ2ØØØ - program error
ØØ1ØØØ - format error
ØØØ4ØØ - mode error
ØØØ2ØØ - longitudinal parity error
ØØØ1ØØ - word parity error
ØØØØ4Ø - checksum error
ØØØØ2Ø - timing error
ØØØØ1Ø - write check error
ØØØØØ4 - non-existent memory
ØØØØØ2 - end of pack
ØØØØØ1 - disk error

2/n RF11
1ØØØØØ - drive error - composite
Ø4ØØØØ - freeze
Ø2ØØØØ - write check error
Ø1ØØØØ - data parity error
ØØ4ØØØ - non-existent disk
ØØ2ØØØ - write lock out
ØØ1ØØØ - missed transfer
ØØØ4ØØ - disk clear
ØØØ2ØØ - control ready
ØØØ1ØØ - interrupt enable
ØØØØ4Ø - extend memory
ØØØØ2Ø - extend memory
ØØØØ1Ø - maintenance
ØØØØØ4 - function
ØØØØØ2 - function
ØØØØØ1 - go

3/n TM11 - no messages

4/n TC11
1ØØØØØ - end zone
Ø4ØØØØ - parity error
Ø2ØØØØ - mark track error
Ø1ØØØØ - illegal operation
ØØ4ØØØ - selection error
ØØ2ØØØ - block missed
ØØ1ØØØ - data missed
ØØØ4ØØ - non-existent memory
ØØØ2ØØ - tape is up to speed
ØØØ1ØØ - clock simulates timing
ØØØØ4Ø - maintenance mark track
ØØØØ2Ø - data track
```

000010	-	data track
000004	-	data track
000002	-	extended data
000001	-	extended data

5/n RP04

100000	-	data check error
040000	-	drive unsafe
020000	-	operation incomplete
010000	-	drive timing error
004000	-	write lock error
002000	-	invalid address error
001000	-	address overflow error
000400	-	header CRC error
000200	-	header compare error
000100	-	burst error (hard)
000040	-	write clock failed
000020	-	pack format error
000010	-	control bus (Massbus) parity error
000004	-	register modification refused
000002	-	illegal register
000001	-	illegal function

6/n RJS04/03

100000	-	data check
040000	-	drive unsafe
020000	-	operation incomplete
010000	-	drive timing error
004000	-	write lock error
002000	-	invalid address error
001000	-	address overflow error
000400	-	unused
000200	-	unused
000100	-	unused
000040	-	unused
000020	-	unused
000010	-	massbus parity error
000004	-	register modify refused
000002	-	illegal register
000001	-	illegal function

7/n TU16 - no messages

VIII. CONCLUSION

Error messages from the UNIX operating system have been briefly outlined along with their possible causes and remedies. For the majority of cases, UNIX installations need not worry about readjusting system parameters in the header file param.h to correct the occurrence of a PANIC or WARNING. Modification of parameters in param.h has only been indicated as a guide to those installations requiring it and to indicate how system parameters are interrelated. These parameters have been tuned for the presently distributed UNIX and should not be modified haphazardly without considerations of the implications of a change. In short, their modification is a last resort. It remains for the current measurement effort on UNIX to shed some light on the bottlenecks in UNIX and where changes will be most effective.

I would like to thank Ken Thompson for several discussions to clarify the meaning of some of the errors and Dennis Ritchie for his comments on the draft.

T. M. Raleigh

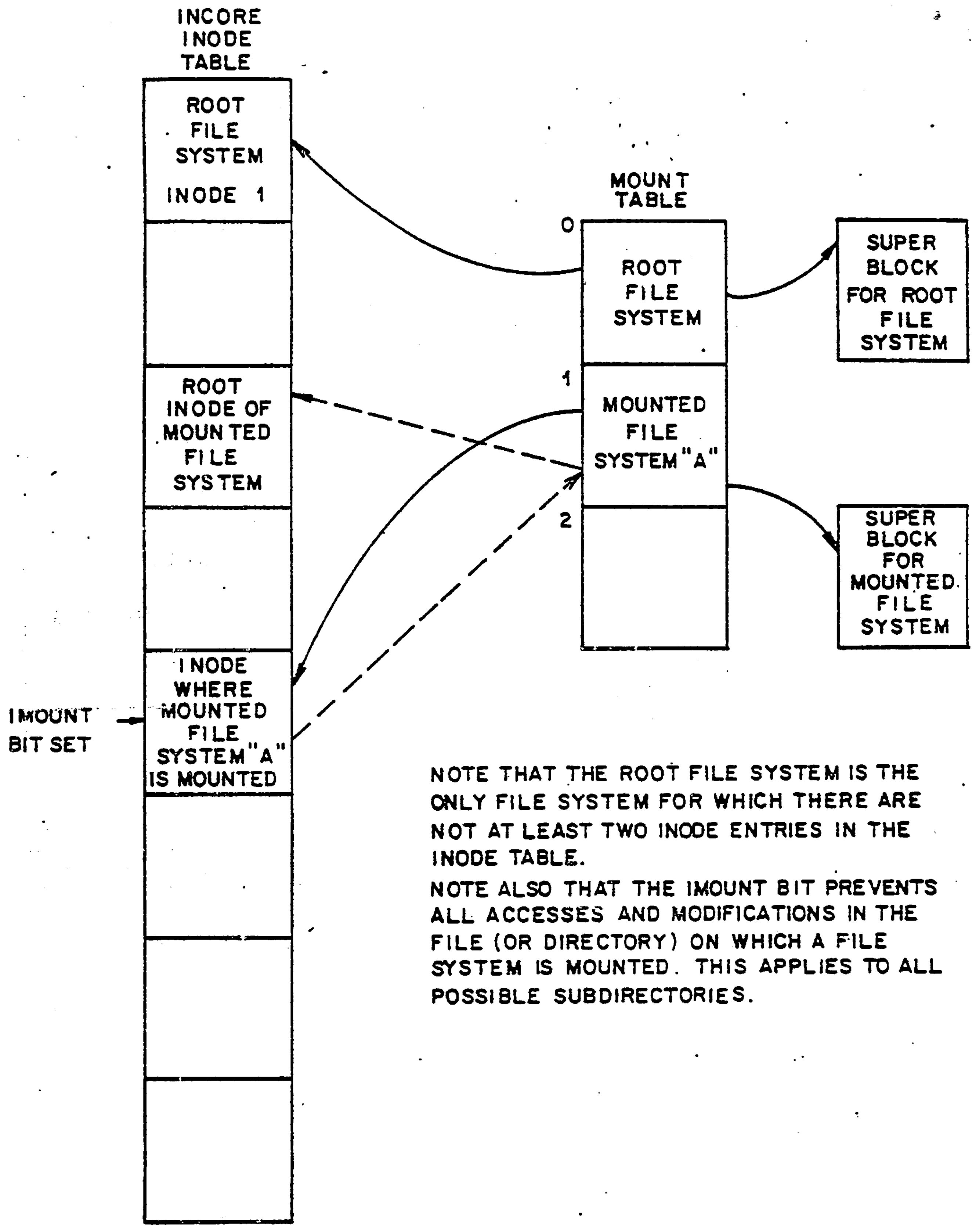
MH-8234-TMR T. M. Raleigh

Attachments

Copy to
G. L. Baldwin
B. A. Tague
Mercury Category UNSD

REFERENCES

1. T. M. Raleigh, "Trap and Interrupt Handling under UNIX", MF-75-8234-29, to be published.


INCORE INODE TABLE
ROOT FILE SYSTEM INODE 1
ROOT INODE OF MOUNTED FILE SYSTEM
INODE WHERE MOUNTED FILE SYSTEM "A" IS MOUNTED
IMOUNT BIT SET
MOUNT TABLE
0
ROOT FILE SYSTEM
1
MOUNTED FILE SYSTEM "A"
2
SUPER BLOCK FOR ROOT FILE SYSTEM
SUPER BLOCK FOR MOUNTED FILE SYSTEM

NOTE THAT THE ROOT FILE SYSTEM IS THE ONLY FILE SYSTEM FOR WHICH THERE ARE NOT AT LEAST TWO INODE ENTRIES IN THE INODE TABLE.

NOTE ALSO THAT THE IMOUNT BIT PREVENTS ALL ACCESSES AND MODIFICATIONS IN THE FILE (OR DIRECTORY) ON WHICH A FILE SYSTEM IS MOUNTED. THIS APPLIES TO ALL POSSIBLE SUBDIRECTORIES.

FIG. 1 MOUNTING A FILE SYSTEM

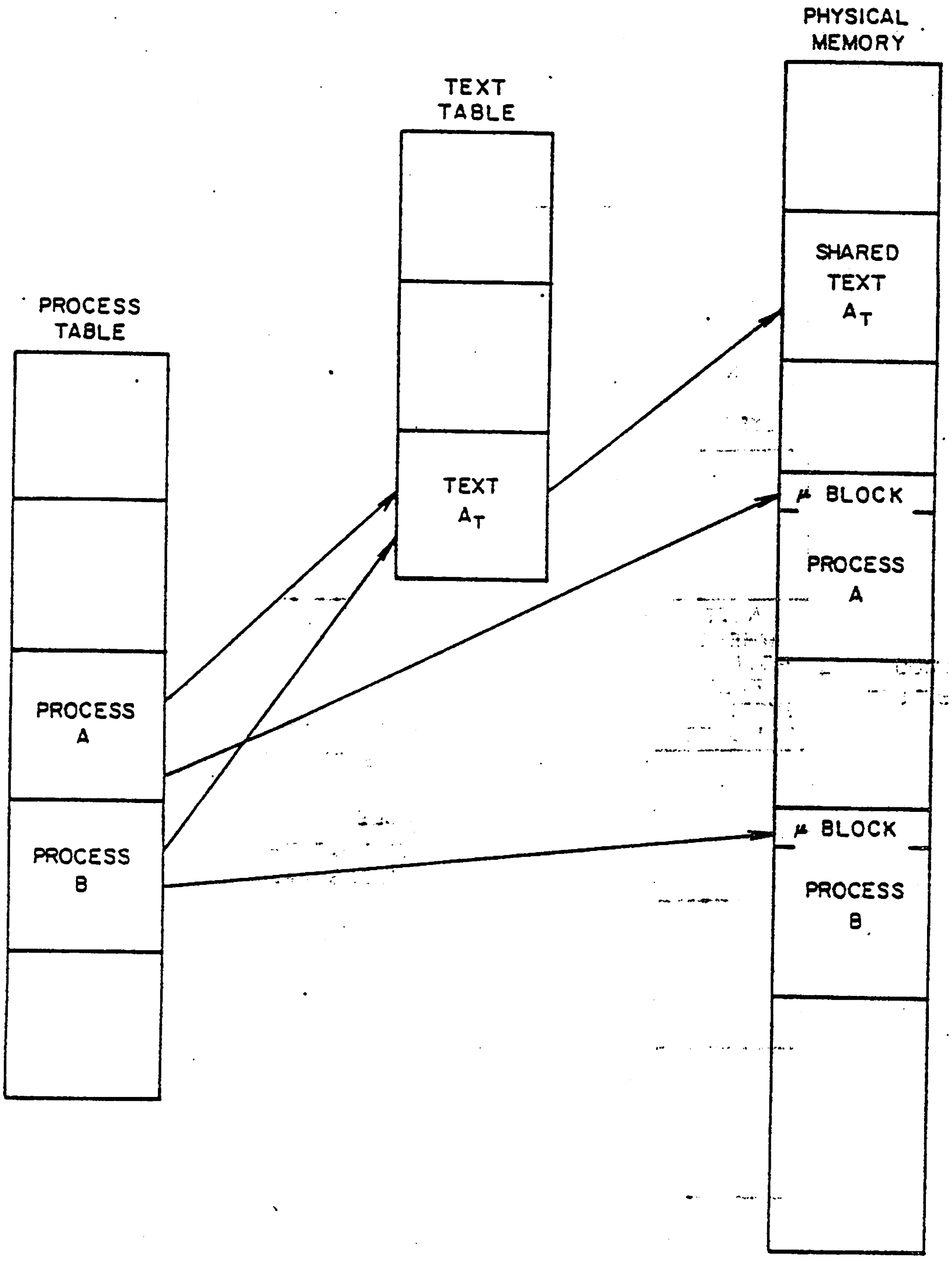
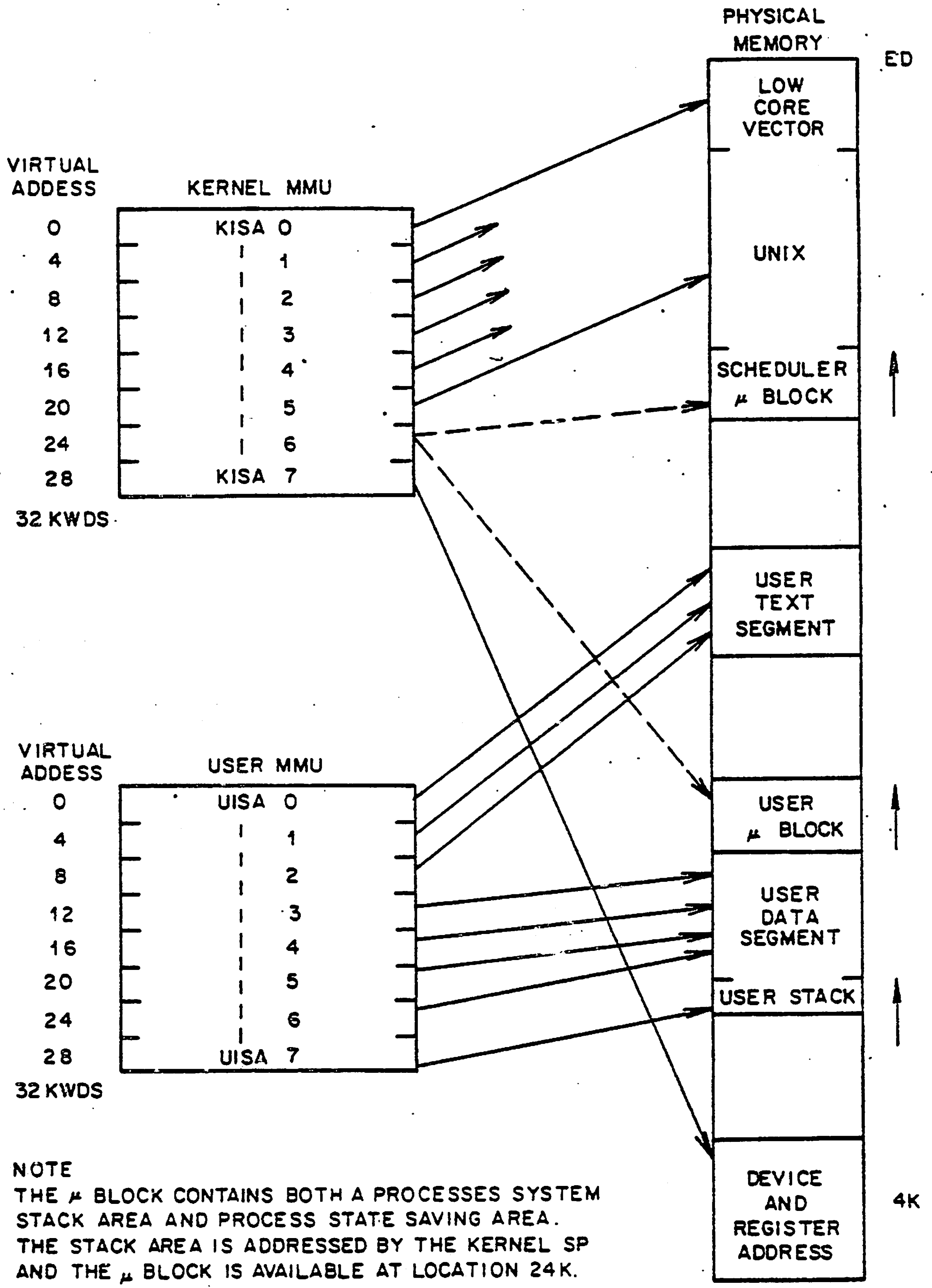


FIG. 2 PROCESS TABLE FOR SHARED PROCESSES



NOTE
THE μ BLOCK CONTAINS BOTH A PROCESSES SYSTEM STACK AREA AND PROCESS STATE SAVING AREA. THE STACK AREA IS ADDRESSED BY THE KERNEL SP AND THE μ BLOCK IS AVAILABLE AT LOCATION 24K.

FIG. 3 SYSTEM AND USER VIRTUAL ADDRESS SPACE

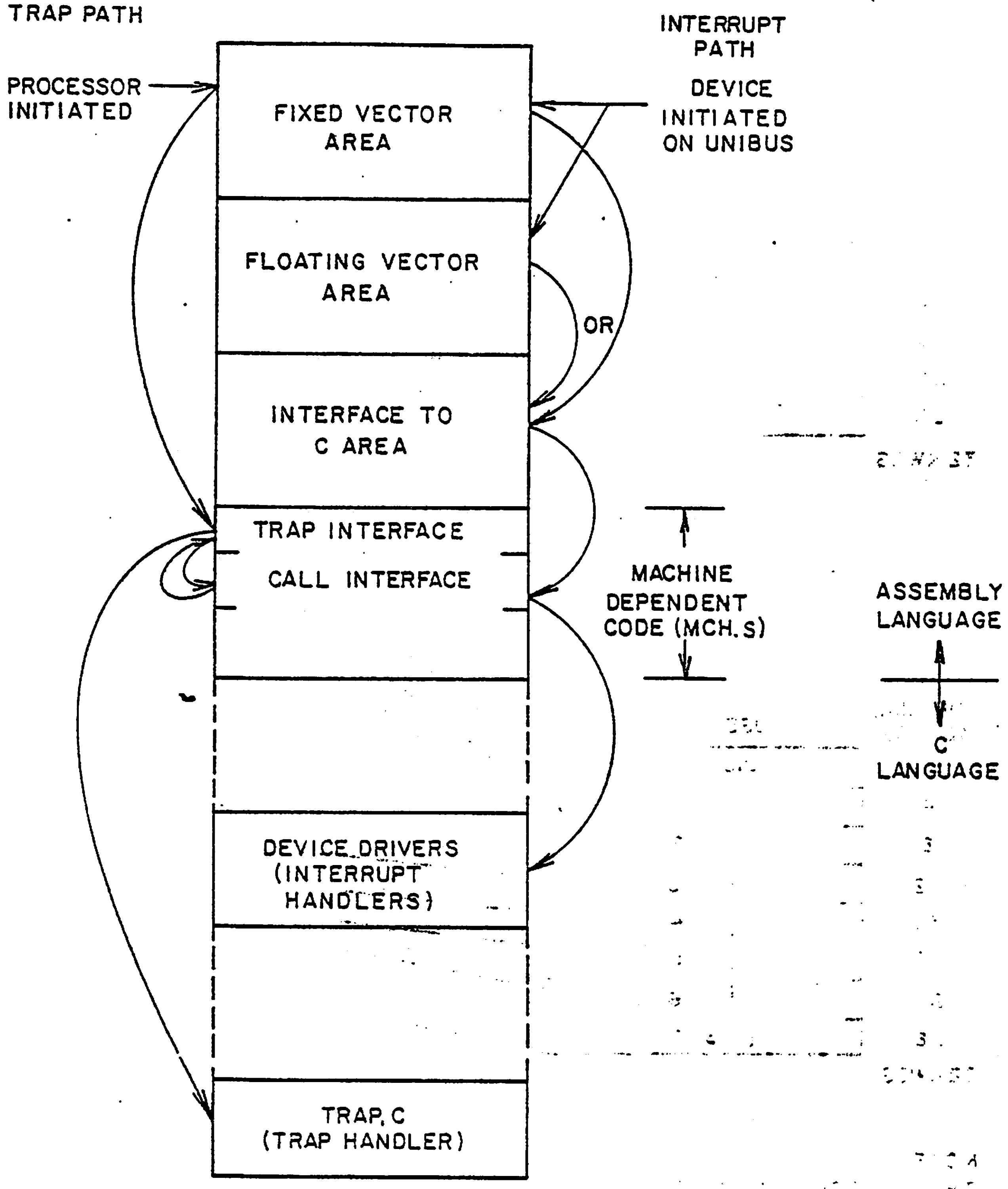
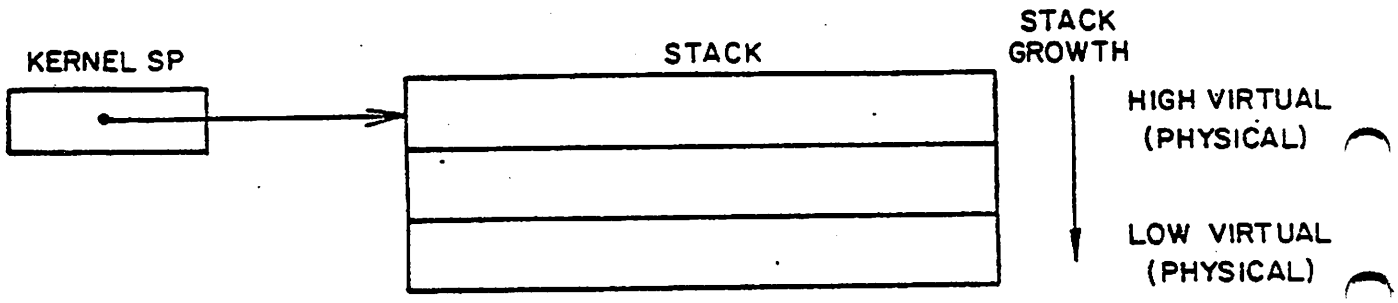
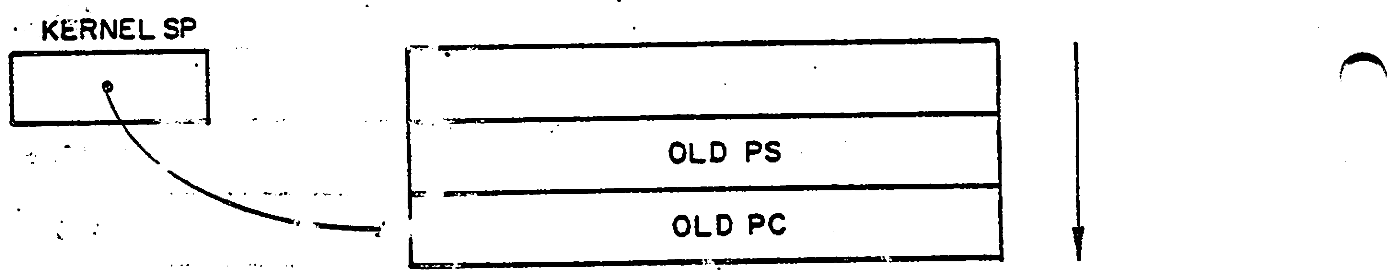


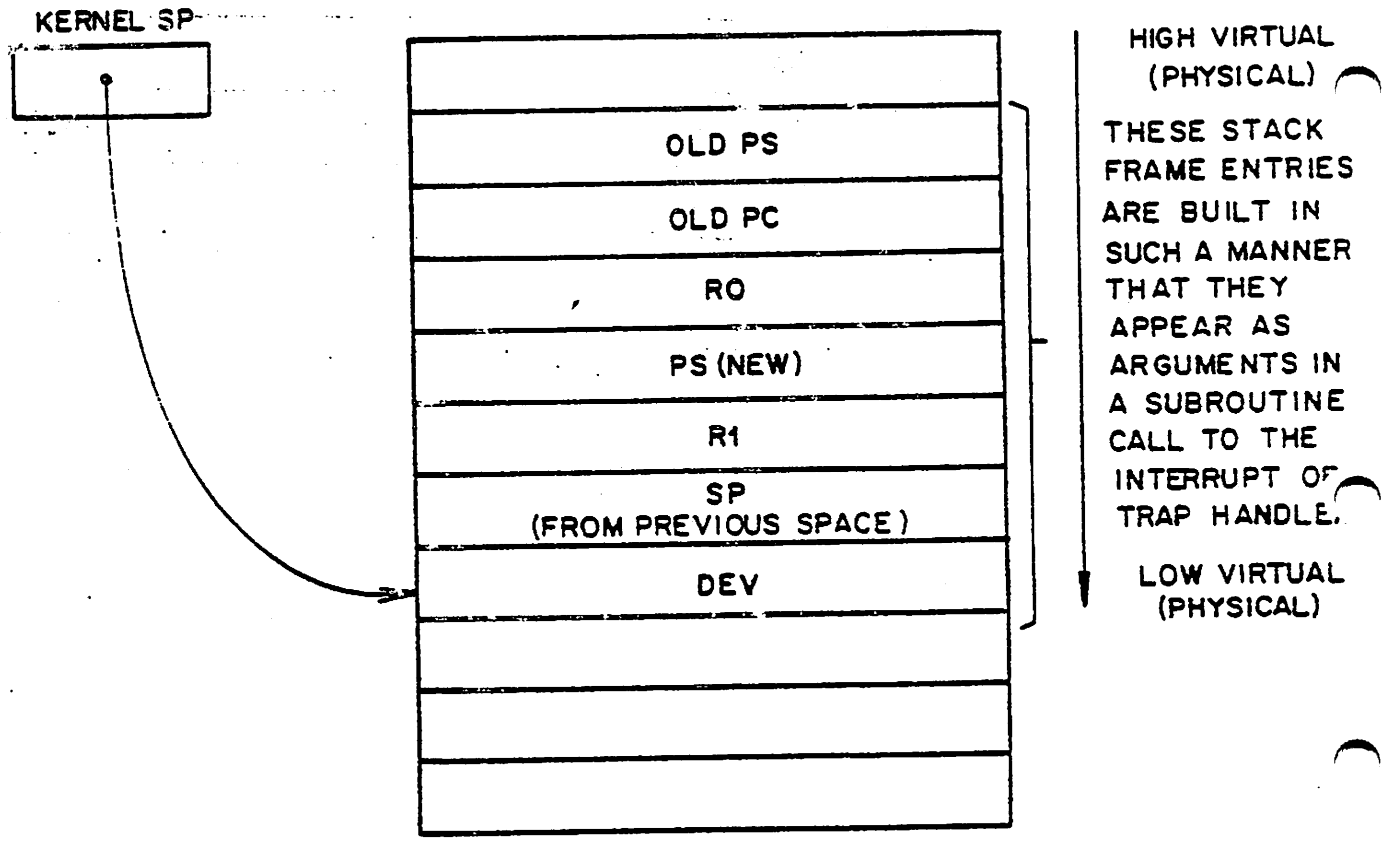
FIG. 4 TRAP AND INTERRUPT PATHWAY



(A) KERNEL STACK BEFORE TRAP OR INTERRUPT



(B) KERNEL STACK AFTER TRAP OR INTERRUPT


KERNEL SP
HIGH VIRTUAL (PHYSICAL)
OLD PS
OLD PC
RO
PS (NEW)
R1
SP (FROM PREVIOUS SPACE)
DEV
THESE STACK FRAME ENTRIES ARE BUILT IN SUCH A MANNER THAT THEY APPEAR AS ARGUMENTS IN A SUBROUTINE CALL TO THE INTERRUPT OF TRAP HANDLE.
LOW VIRTUAL (PHYSICAL)

(C) STACK FRAME BUILT FOR TRAP AND INTERRUPT HANDLING

FIG. 5

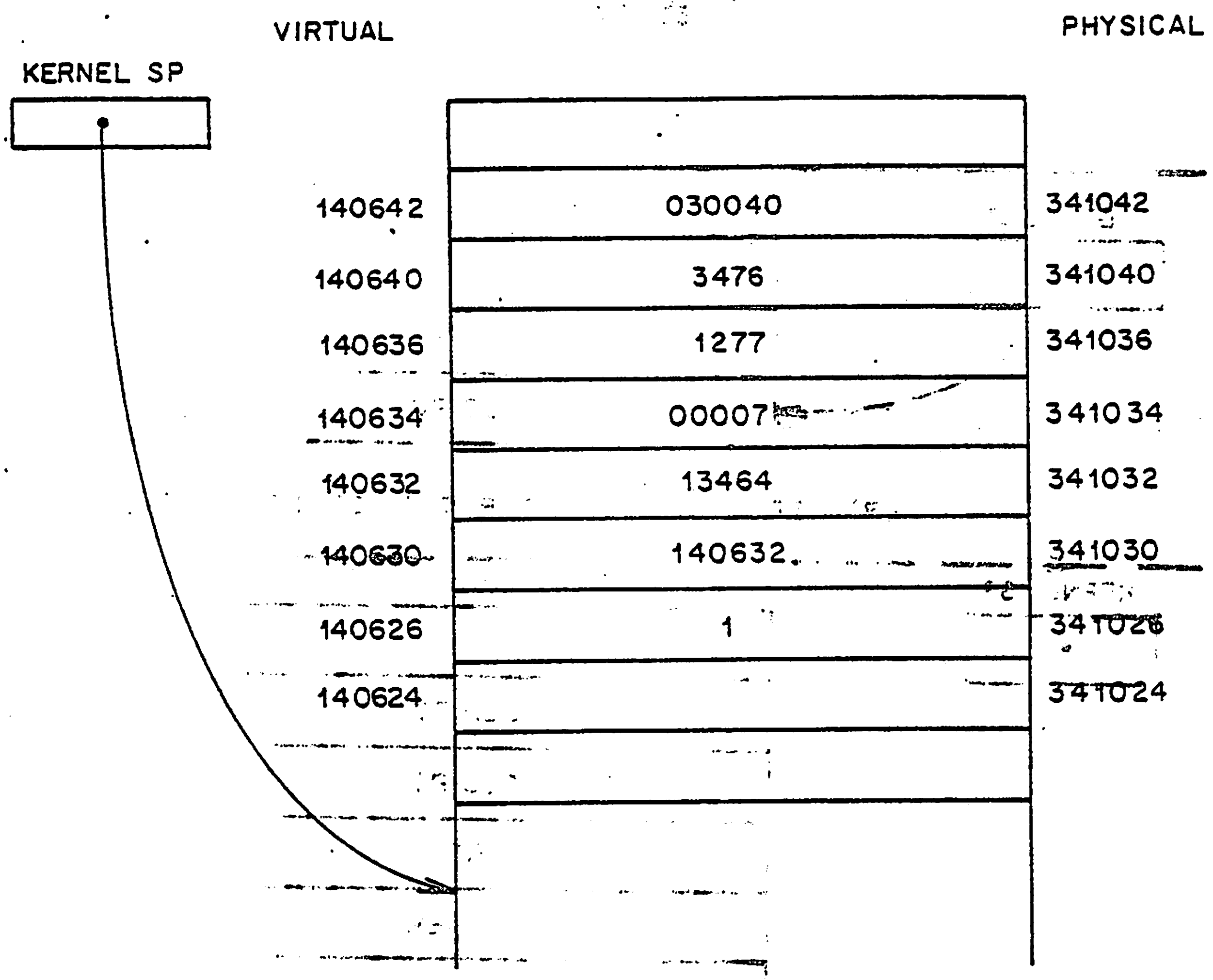


FIG. 6 A TRAP FROM WITHIN THE OPERATING SYSTEM FOR AN ILLEGAL INSTRUCTION.