TED —

Here's a copy of our current UNIX documentation —
the organization of the document is as follows

A — Symbol Table (not done)

B — Memmory Map

B.1 — Core Memory Map

C — Routine Jumps Network (who calls who)

D —

E — Listings

E.0 — V_0

E.1 — V_1

⋮

F — General description of System

G — Description of data names

H — Routine descriptions

H.0 — V_0 Routine

H.1 — V_1 Routine

⋮

Sent a note thanking 4/3/72 couldn't reach by phone.

Hope it helps.

Raritan River

0-463-6600

Jim DeFelice
140 Centennial Ave.
Rm G155
R, R9-463-4004
From MH 165-dial tone-4004

ID — U0, 2 / allocate tty buffers

FUNCTION — Each DC-11 interface is assigned 140. bytes of buffer space, the first 140.-byte block beginning at location "buffer." Also for each interface a 4 word block of control and status type information is maintained. These 4-word blocks begin at location "tty", the fourth word in each block is a pointer to the beginning of the 140-byte buffer assigned to that device. This section of code loads these pointers into the proper places in the tty blocks. The results are shown in the diagrams on H.0 page

CALLING SEQUENCE —

ARGUMENTS —

INPUTS — ntty (number of DC11 interfaces)

OUTPUTS — (see diagrams H.0 page 3), r0, r1

ID_ u0;3 / allocate disk buffers

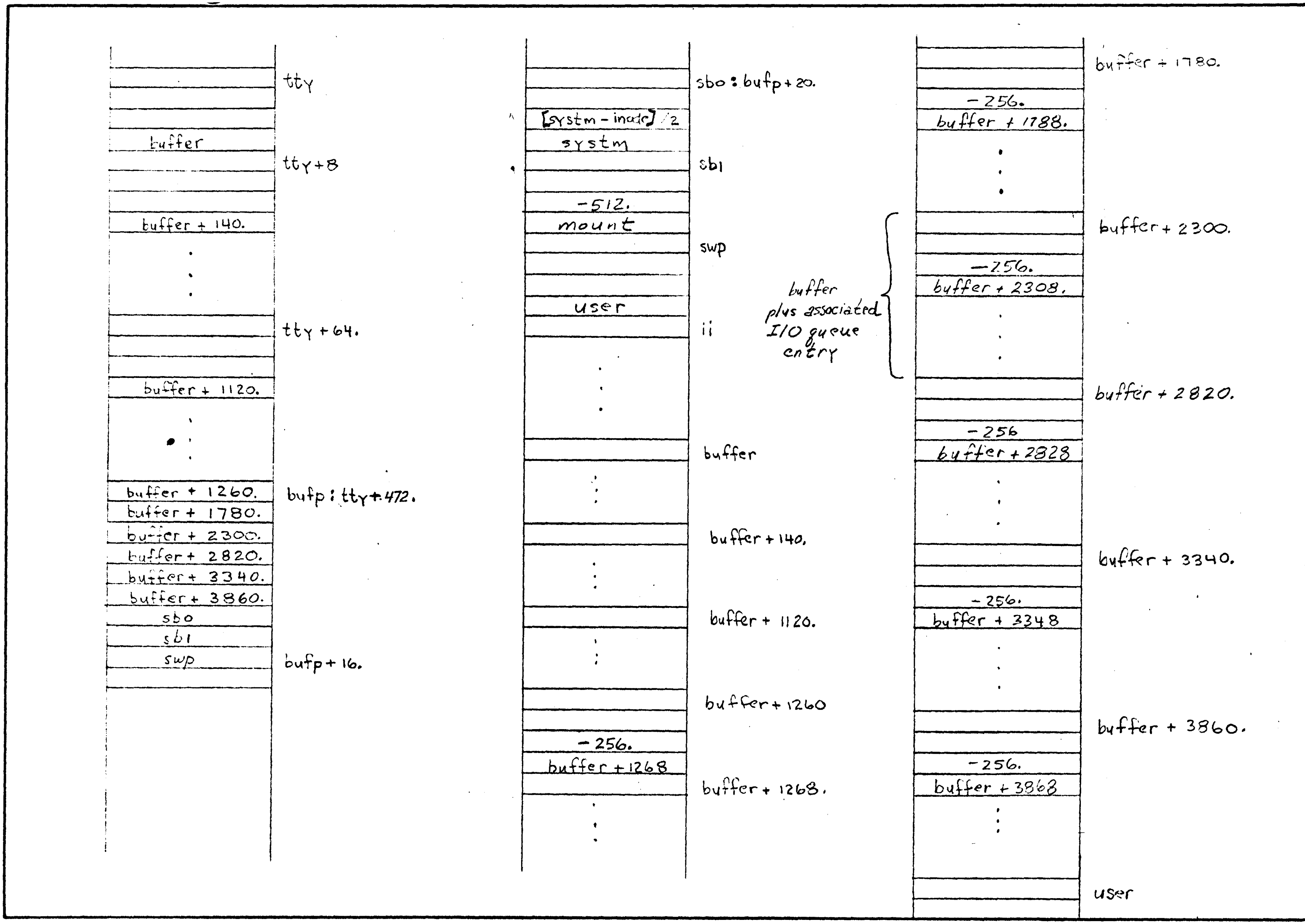
FUNCTION — Block I/O devices (drum, disk, dectape) use blocks of size 256. words. Thus for each of "nbuf" block I/O buffers 256. words must be assigned. In addition to the 256. words for data each block has four additional words which represent an I/O queue entry. Thus each block contains 260 words. These blocks begin at location "buffer + 1260.". This segment of code loads pointers to these 260 word blocks in consecutive locations starting at "bufp." Thus "bufp" contains pointers to I/O queue entries since the first four words in each block represent the I/O queue entry for the block. Three additional I/O queue entries located at locations "sbo", "sbi", and "swp" also exist and pointers to them are also loaded into "bufp". Finally, the last 2 words of an I/O queue entry contain a word count and a bus address, these locations are initialized. The results are shown in the diagrams on H.0 page 3

<u>CALLING SEQUENCE</u>

<u>ARGUMENTS</u>

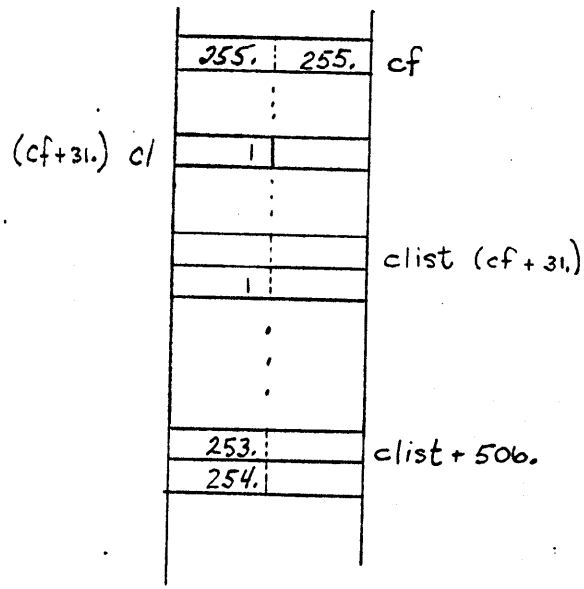
<u>INPUTS</u> r0 (points to first block I/O buffer)

<u>OUTPUTS</u> (see diarams H.0 page3), r1 (internal counter), r2 (internal pointer)


tty
buffer
tty+8
buffer + 140.
tty + 64.
buffer + 1120.
buffer + 1260.
buffer + 1780.
buffer + 2300.
buffer + 2820.
buffer + 3340.
buffer + 3860.
sbo
sb1
swp
bufp : tty+472.
bufp + 16.
sbo : bufp + 20.
[systm - inode] /2
systm
sb1
-512.
mount
swp
user
ii
buffer
buffer + 140.
buffer + 1120.
buffer + 1260
-256.
buffer + 1268
buffer + 1268.
buffer plus associated I/O queue entry
buffer + 1780.
-256.
buffer + 1788.
buffer + 2300.
-256.
buffer + 2308.
buffer + 2820.
-256
buffer + 2828
buffer + 3340.
-256.
buffer + 3348
buffer + 3860.
-256.
buffer + 3868
user

ID_ U0; 3 / free all character blocks

FUNCTION_ This segment of code initializes the cf, cl and clist blocks in core to the following state:

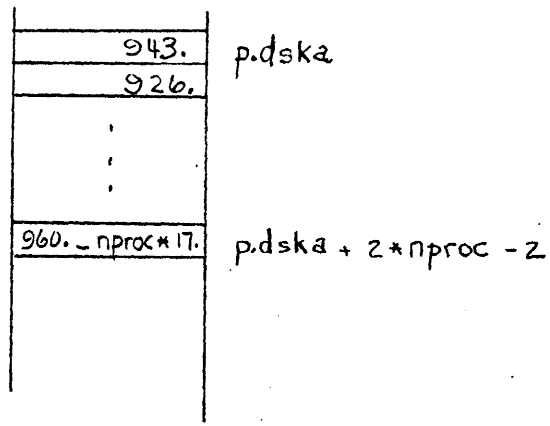


CALLING SEQUENCE_
ARGUMENTS_
INPUTS_
OUTPUTS_
CALLED BY_
CALLS_ PUT

ID- U0;3 /set up drum swap addresses

FUNCTION - The drum is divided into 1024. blocks of 256. words. The highest 64. blocks are set aside for storing UNIX itself. Processes swapped to and from core are stored on the drum. The area in core beginning at location p.dska contains a block number which is the number of the first block on the drum where the process is swapped to. There are 17 blocks on the drum assigned as swapping area for each process.

This segment of code initializes the p.dska area in core by supplying the block numbers for each of "nproc" processes. The results appear as follows:



CALLING SEQUENCE -
ARGUMENTS -
INPUTS -
OUTPUTS - p.dska - [p.dska + 2*nproc - 2], r1, r2

ID – uo;4 / free rest of drum

FUNCTION – This portion of code is executed during 'cold' boot. (See UNIX Programmers Manual - Boot Procedures (VII.)) It initializes the core image of the super block for the fixed head disk. Systm (which represents the no. of bytes in the free storage map) is set to 128. Systm +130. (which represents the number of bytes in the i-node map) is set to 64.. (See Sec F pp. 1, 2). Blocks 34., ... 687. on the drum are freed (the corresponding bits in the free storage map are set). These blocks are for user files.

CALLING SEQUENCE –

ARGUMENTS –

INPUTS – R1 Contains the number of the highest block to be freed.
(See inputs for 'free'; H.5 p. 2)

OUTPUTS – SYSTM, SYSTM+6, SYSTM+8, ..., SYSTM+85., SYSTM+130.
(See outputs for 'free'; H.5 p. 2)
R1 used internally

ID – u0;4 / zero i-list

FUNCTION – This portion of code is executed during 'cold' boot. (See UNIX Programmers Manual - Boot Procedures (VII)). It zeros blocks 1,...,33. on the drum. Block 1 is the 2nd block of the superblock for the drum. (Block 0 is the 1st block of the superblock. However since the in core image of the superblock (see UNIX Implementation Manual – p 3) is updated onto the RK03 whenever it is changed (can be changed by a call to 'free', updated by a call to 'sysret') it does not have to be zeroed.) Blocks 2,...,33. are used for i-nodes 1 thru 512 (See Sec F pp 1,3,4,5)

CALLING SEQUENCE –

ARGUMENTS –

INPUTS – R1 Contains the number of the highest block to be zeroed + 1.

(See inputs for 'clear' H.3 p.1)

OUTPUTS – Blocks 2,...,33. on disk are zeroed.

(see outputs for 'clear' H.3 p.1)

R1 used internally

ID UI-8 sysclose

FUNCTION "syclose", given a file descriptor in u.r0, closes the associated file. The file descriptor (index to the u.fp list) is put in R1 and "fclose" is called. (See "fclose" 11.2)

CALLING SEQUENCE: sys close

ARGUMENTS:

INPUTS (u.r0) - file descriptor

OUTPUTS See fclose outputs

ID U1-7 sys creat

FUNCTION: "syscreat" is called with two arguments; name and mode. u.namep points to the name of the file and the mode is put on the stack. "namei" is called to get the i-number of the file. If the file already exists, its mode and owner remain unchanged, but it is truncated to zero length If the file did not exist, an i-node is created with the new mode via "maknod" Whether or not the file already existed, it is open for writing The fsp table (See F page 8) is then seached for a free entry. When a free entry is found the proper data is placed in it (See outputs below), and the number of this entry is placed in the u.fp list. The index to the u.fp (also known as the file descriptor) is put in the users R0. For more information see SYSCREAT in the users manual.

CALLING SEQUENCE: sys creat ; name; mode

ARGUMENTS name - name of file to be created
mode - mode " " " " "

INPUTS R1 - i-number of file if found
sp - contains the mode argument
u.dirbuf - if file not found, contains i-number of new file
u.fp - list of file descriptors
fsp - table of open file entries

OUTPUTS if file not found - new i-node is created (See maknod)
R1 - contains i-number of new file
R3 - index into fsp table (file descriptor)
R2 index into u.fp list
in free fsp entry —— 1st word i-number of new file
2nd word device number
3rd word 0
4th word 0
u.fp list - entry number of new fsp entry
(u.r0) - index to u.fp list (file descriptor of new file

ID UI-1 sysent; unkni

FUNCTION; unkni or sysent is the system entry from various traps. The trap type is determined and an indirect jump is made to the appropriate system call handler.
If there is a trap inside the system a jump to panic is made All user registers are saved and u.sp points to the end of the users stack. The sys (trap) instruction is decoded to get the system code part. (See trap instruction in the PDP-11 handbook) and from this the indirect jump address is calculated. If a bad system call is made, i.e., the limits of the jump table are exceeded, "badsys" is called. If the call is legitimate control passes to the appropriate system routine

CALLING SEQUENCE through a trap caused by any sys call outside the system

ARGUMENTS arguments of the particular system call

INPUTS s.syst+2, R0, SP, R1, R2, R3, R4, R5, AC, MQ, SC

OUTPUTS clockp - contains $s.syst+2
u.r0 - points to the location of the users R0 on the stack
R0 - SC saved on the stack
u.sp - points to the end of the users stack

ID VI-3 sys exit

FUNCTION sysexit terminates a process. First each file that the process has opened is closed by "fclose. The process status is then set to unused. The p.ppid table is then searched to find children of the dying process. If any of the children are zombies (died but not waited for) they are set free. The p.pid table is then searched to find the dying process's parent. When the parent is found, it is checked to see if it is free or it is a zombie. If its one of these the dying process just dies. If its waiting for a child to die, it is notified that it doesn't have to wait anymore by setting its status from 2 to 1. (waiting to active). It is then awakened and put on the runq by "putlu". The dying process enters a zombie state in which it will never be run again but stays around until a "wait" is completed by its parent process. If the parent is not found, the process just dies. This means swap is called with u.uno = 0. What this does is that wswap is not called to write out the process and rswap reads a new process over the one that dies. i.e., the dying process is overwritten and destroyed.

CALLING SEQUENCE sys exit or conditional branch

ARGUMENTS;

INPUTS
- v.uno - the process number of the dying process
- p.pid - contains the name of the process (See F page 10)
- p.ppid - contains the name of the parent process
- p.stat - the status of the process

OUTPUTS
- u.intr - determines handling of interrupts - it is set to 0
- all open files of the process are closed
- the process is freed
- R3 - contains the dying process's name or number
- R4 - contains its parents name
- R2 - is used to scan the process tables
- children of the dying process are freed
- R1 & R5 are used to hold the parents process number × 2
- if the parent of this dying process is waiting it is set to active and the dying process is made a zombie and the parent is put on the runq.
- u.uno is cleared and the process is killed

10 UI-5 sysfork

FUNCTION: sysfork creates a new process. This process is referred to as the child process. This new processes core image is a copy of that of the caller of "sysfork". The only distinction is the return location and the fact that (u.ro) in the old process (parent) contains the process id (p.pid) of the new process (child). This id is used by "syswait." "sysfork" works in the following manner:

1) The process status table (p.stat) is searched to find a process number that is unused. If none are found an error occurs.
2) When one is found, it becomes the child process number and its status (p.stat) is set to active.
3) If the parent had a control tty, the interrupt character in that tty buffer is cleared.
4) The child process is put on the lowest priority run queue via "putlu".
5) A new process name is gotten from mpid. (actually its a unique number) and is put in the child's unique identifier; the process id (p.pid)
6) The process name of the parent is then obtained and placed in the unique identifier of the parent process of the child. (p.ppid) The parent process name is then put in (u.ro)
7) The child process is then written out on disk by "wswap", i.e., the parent process is copied onto disk and the child is born
8) The parent process number is then restored to u.uno
9) The child process name is put in (u.ro)
10) The PC on the stack sp+18 is incremented by 2 to create the return address for the parent process.
11) The u.fp list is then searched to see what files the parent has opened. For each file the parent has opened, the corresponding fsp entry must be updated to indicate that the child process also has opened the file. A branch to sysret is then made.

CALLING SEQUENCE from shell ?
ARGUMENTS
INPUTS
- p.stat status of a process active, dead, unused
- u.uno parent process number
- u.ttyp - pointer to parents process control tty buffer
- mpid - process name generator
- u.fp - list of index into the fsp table
- fsp - table of open files

UI-5 sysfork (continued)

OUTPUTS: p.stat - byte for child process is set to active
if control tty for parent exists buffer+6 is cleared
child process number is put on rung+4
p.pid - appropriate entry in this table contains the name of the child process
the child process is written out on drum with u.uno being the childs process number and (u.ro) containing the parents process name.
u.uno is restored to the parents process number
(u.ro) - contains the childs process name
sp+18 - gets 2 added to it to change the return address of the parent
fsp+6 - "number of processes that have opened this file" byte gets incremented in the particular fsp entry.

ID UI-8 sysmdate

FUNCTION: "sysmdate" is given a file name. It gets the i-node of this file into core. The user is checked to see if he is the owner or the super user. If he is neither an error occurs. "setimod" is then called to set the i-node modification byte and the modification time, but the moditation time is overwritten by whatever got put on the stack during a "systime" call. (See systime) These calls are restricted to the super user.

CALLING SEQUENCE sysmdate; name

ARGUMENTS name - pointer to a file name

INPUTS: u.uid, users id
i.uid owners id
sp+4 } time set by super user
sp+2 }

OUTPUTS i.mtim } new modification time of the file
i.mtim+2 }

ID UI-9 sysgtty

FUNCTION "sysgtty" gets the status of the tty in question. It stores in the three words addressed by its argument the status of the typewriter whose file descriptor is in (u.r0)

CALLING SEQUENCE sysgtty ; arg

ARGUMENTS: arg - address of 3 word destination of status

INPUTS: R1 - tty block offset
R2 - destination of status data
rcsr+R1 - reader control status
tcsr+R1 printer control status register
tty+4+R1 - flag byte in tty block which contains the mode

OUTPUTS: (R2) - contains the reader control status
(R2)+2 " " printer " "
(R2)+4 " " mode

ID UI-8 sysmkdir

FUNCTION: "sysmkdir" creates an empty directory whose name is pointed to by arg 1. The mode of the directory is arg 2. The special entries "." and ".." are not present. Errors are indicated if the directory already exists or the user is not the super user.

CALLING SEQUENCE sysmkdir ; name; mode

ARGUMENTS name - points to the name of the directory

mode - mode of the directory

INPUTS u.uid - user id ; if its 0 the user is the super user

(sp) - contains the second argument "mode"

OUTPUTS - makes an i-node for the directory via "maknod"

- sets up the flag in the directory i-node

set user id on execution

executable

directory

I u.cdev, R1, u.cdir formener
I arg 1, arg 2

why is fsp table indexed 1, ..., 50
rather than 0, ..., 49 ?

~~a uses~~ R2, R3

I u.fp+0, u.fp+1, ..., u.fp+9

0 R2 = 0, ..., 9

I fsp+0, fsp+8, ..., fsp+ 49*8

0 the proper fsp + x? loaded in R1 (C#)

[fsp+2]*x = cdev
[fsp+4]*x = 0
[fsp+6]*x = 0

ϕ u.fp + $\frac{x}{8}$ = $\frac{x}{8}$ + 1

ϕ (u.R0) = $\frac{x}{8}$

R3 = $\frac{x}{8}$ + 1

ID UI-2 error

FUNCTION "error" merely sets the error bit of the processor status (c-bit) and then falls right into the sysret, sysrele return sequence.

CALLING SEQUENCE conditional branch to error

ARGUMENTS:

INPUTS:

OUTPUTS processor status - c bit is set (means error)

ID UI-3 badsys

FUNCTION: "badsys" is called either because the user executed an illegal trap type instruction or because a t-bit trap occured. (The t-bit is used to implement the quit function) "badsys" first turns on the bad system flag (u.bsys) and the calls "namei" with u.namep pointing to "core". The core image file is then opened for writing via "iopen". If the file is not found, and i-node whose mode is 17 is made by "maknod" and the i-number for that node is put in R1. Parameters to write out core are then set up and the core image is written out in the users directory. Then the users area of core are written out and the file closed. sysexit is entered to terminate the process.

CALLING SEQUENCE bhis badsys

ARGUMENTS —

INPUTS: R1 - i-number of core image files i-node
u.dirbuf contains i-number of new i-node made by "maknod"

OUTPUTS
u.bsys - turn on. its the users badsystem flag
u.base - holds address of "core" and user during writei calls
u.count - users byte count to write out
u.fofp - contains file offset
u.off - set to zero
R1 - has i-number of core image file

ID UI-9 gtty

FUNCTION "gtty" is called by "sysgtty" and "sysstty". It takes the first argument of the above calls and puts it in R2. This argument is either the source or destination of information about the tty in question. The file descriptor is put in R1 and the i-number of the file is obtained via "getf." The number of the tty is gotten by (the i-number - 14) If no tty with this number exist and error occurs. 8 x (i-number-14) is the tty block offset. This is outputed in R1

CALLING SEQUENCE: jsr r0, gtty

ARGUMENTS: —

INPUTS: (u.r0) - contains the file descriptor for the tty file
R1 - i-number of file

OUTPUTS R1 - tty block offset
R2 - source or destination of information

ID - u1; 7 error 2

FUNCTION - see 'error' routine

CALLING SEQUENCE - "

ARGUMENTS - "

INPUTS - "

OUTPUTS - "

ID_41;5 e[illegible]r'

FUNCTION — see 'error'
ARGUMENTS — "
CALLING SEQUENCE — "
INPUTS — "
OUTPUTS — "

ID UI-6 sysopen

FUNCTION "sysopen" opens a file in the following manner

1) The second argument in a sys open calls says whether to open the file to read (0) or write (≠0).
2) the i-node for the particular file is obtained via "namei"
3) The file is then opened by "iopen"
4) Next housekeeping is performed on the fsp table and the users open file list — u.fp.
    a) u.fp and fsp are scanned for the next available slot
    b) An entry for the file is created in the fsp table
    c) The number of this entry is put on the u.fp list.
    d) The index to the u.fp list (file descriptor) is pointed to by u.ro

ARGUMENTS name file name or path name
mode 0 - open for reading
1 - open for writing

INPUT R1 - contains an I-number (positive or negitive depending on whether an open for read or open for write is desired.

OUTPUT entry in fsp table and u.fp list
(u.ro) - index to u.fp list (the file descriptor) is put into R0's location on the stack
R2 - used as a counter through the u.fp list
R3 - used as a pointer to the begining of an fsp entry

CALLING SEQUENCE sys open; name; mode

ID UI-2 sysret

FUNCTION; sysret first checks to see if the process is about to be terminated (u.bsys). If it is sysexit is called. If not the following happens

1) The users stack pointer is restored

2) R1=0 and "iget" is called to see if the last mentioned i-node has been modified. If it has, it is written out.

3) If the super block has been modified, it is written out via "ppoke"

4) If the dismountable file system's super block has been modified it is written out to the specified device via "ppoke."

5) A check is made to see if the users time quantum (uquant) ran out during his execution. If so, "tswap" is called to give another user a chance to run.

6) sysret now goes into sysrele. (See sysrele for conclusion)

CALLING SEQUENCE; jump table or br sysret

ARGUMENTS —

INPUTS
u.bsys - users bad system flag
u.sp - users stack pointer
R1 - used internally - set to 0 for "iget" call
smod - set if super block has been modified
mmod - set if dismountable file systems super block has been modified
uquant - users time quantum

OUTPUTS
sp - points to users stack
smod - cleared if it was set
mmod - " " " "
sb0 - write bit is set during execution of sysret
sb1 - " " " " " " " "

ID – ui;5 sysret1

FUNCTION – see 'sysret'
CALLING SEQUENCE – "
ARGUMENTS – "
INPUTS – "
OUTPUTS – "

ID_ UI.7 sysret2

FUNCTION- see 'sysret' routine
CALLING SEQUENCE- "
ARGUMENTS- "
INPUTS- "
OUTPUTS- "

ID UI-2 sysrele

FUNCTION "sysrele" first calls tswap if the time quantum for a user is zero (See sysret). It then restores the users registers and turns off the system flag, It then checked to see if there is an interrupt from the user by calling "isintr". If there is the output gets flushed (See isintr) and interrupt action is taken by a branch to intract. If there is no interrupt from the user a rti is made.

CALLING SEQUENCE fall through a "bne" in sysret & ?

ARGUMENTS —

INPUTS stack
(s.chrgt+2) ?

OUTPUTS SC, MQ, AC, R5, R4, R3, R2, R1, R0 restored
systflag - turned off
clockp points to s.chrgt+2

ID U1-9 syssty

FUNCTION: "sysstty" gets the status and mode of the typewriter whose file descriptor is in (u.r0) First "gtty" is called to get the tty block and the source of the status information. "getc" is called until the input clist is flushed. The output character list is checked. If some characters are on it, the process is put to sleep and the input list is checked again. If there are no characters, the information in the source is put into the reader control status, printer control status registers and the tty's flag byte in the tty block.

CALLING SEQUENCE: sysstty ; arg

ARGUMENTS: arg - address of three consecutive words that contain the source of the status data

INPUTS: R1 - offset to tty block

R2 - points to the source of the status information, see arg above

R1+tty+3 - contains the cc offset

R3 - used to transfer the source information to the tty status registers and block

OUTPUTS: ps set to 5

rcsr+R1 - contains new reader control status

tcsr+R1 - contains new printer control status

tty+4+R1 - contains new mode in the flag byte of the tty block

ID U1-4 syswait

FUNCTION syswait waits for a process to die. It works in the following way

1) from the parent process number, the parents process name is found. The p.ppid table of parent names is then searched for this process name. If a match occurs R2 contains the childs process number. The child's status is checked to see if its a zombie, i.e. dead but not waited for, (p.stat=3) If it is, the child process is freed and its name is put in (u.ro). A return is then made via "sysret." If the child is not a zombie nothing happens and the search goes on through the p.ppid table until all processes are checked or a zombie is found.

2) If no zombies are found, a check is made to see if there are any children at all. If there are none and error return is made. If there are, the parents status is put to 2 (waiting for child to die), the parent is swapped out and a branch to syswait is made to wait on the next process.

CALLING SEQUENCE: ?

ARGUMENTS:

INPUTS: u.uno - parent process number (process number of process in core)
p.pid - table of names of processes
p.ppid - table of parents names of processes.
p.stat - contains status of process
0 - free or unused
1 - active
2 - waiting for process to die
3 - zombie

OUTPUTS: R2 - used as index to p.pid, p.ppid, p.stat, tables
R3 - used to kept track of the number of children
R1 - has parents process number
If zombie found - its status p.stat is freed (set to 0)
- its name is put in (u.ro)
if no zombies found - status of parent is set to 2 (waiting for child to die)
- parent is swapped out

ID U1-6 sysread

FUNCTION: sysread is given a buffer to read into and the number of characters to be read. It finds the file from the file descriptor located in *u.r0 (r0). This file descriptor is returned from a successful open call. (See sysopen H.1 page 1). The i-number of the file is obtained via "rw1" and the data is read into core via "readi."

ARGUMENTS: buffer - location of contiguous bytes where input will be placed.

nchars - number of bytes or characters to be read

INPUTS R1 - contains i-number of file to be read

OUTPUTS (u.r0) contains the number of bytes read

CALLING SEQUENCE sys read ; buffer; nchars

ID UI-6 syswrite

FUNCTION: syswrite is given a buffer to write, onto an output file and the number of characters to write. It finds the file from the file descriptor located in *u.r0 (r0). This file descriptor is returned from a sucessful open or creat call. (See sys open or sys creat). The i-number of the file is obtained via "rw1" and the buffer is written on the output file via "writei."

ARGUMENTS buffer - location of contiguous bytes to be written
nchars - number of characters to be written

INPUTS R1 - contains the i-number of the file to be written on.

OUTPUTS (u.r0) - contains the number of bytes written

CALLING SEQUENCE sys write; buffer; nchar

ID U2-9 anyi

FUNCTION: "anyi" is called if a file has been deleted while open, "anyi" checks to see if someone else has opened this file. It searches the fsp table for an i-number contained in R1. If that i-number is found (if someone else opened the file) the "file deleted" flag in the upper byte of the 4th word of the fsp entry is incremented. (See F page 8). In other words the deleted flag is passed on to the other entry of this file in the fsp table. Note: The same file may appear more than once in the fsp table.

If the i-number is not found in the fsp table (No one else has opened the file) the corresponding bit in the i-node map is cleared freeing that i-node and all blocks related to that i-node.

INPUTS R1 - contains an i-number
fsp - start of table containing open files
R2 - points to the i-number in an fsp entry

OUTPUTS - "deleted" flag set in fsp entry of another occurence of this file and R2 pts to 1st word of this fsp entry

if file not found - bit in i-node map is cleared (i-node is freed)
- all blocks related to i-node are freed
- all flags in i-node are cleared

CALLING SEQUENCE jsr r0, anyi

[illegible]: ARG extracts an argument for a routine whose call is of form:

SYS 'ROUTINE'; ARG1

or

SYS 'ROUTINE'; ARG1; ARG2

or

SYS 'ROUTINE'; ARG1; . . . ; ARGN (Sys[illegible])

CALLING SEQUENCE: JSR R0, ARG; 'ADDRESS'

ARGUMENTS: 'ADDRESS' - Address in which extracted argument is stored

INPUTS: U.SP+18. - Contains a pointer to one of ARG1, .., ARGN. This pointer's value is actually the value of the updated PC at the time the TRAP to SYSENT(UNKNI) is made to process the SYS instruction.

R0 - Contains the return address for the routine that called ARG. The data in the word pointed to by the return address is used as the address in which the extracted argument is stored.

OUTPUTS: 'ADDRESS' - Contains the extracted argument.

U.SP+18. - Is incremented by 2.

R1 - Contains the extracted argument.

R0 - Points to the next instruction to be executed in the calling routine.

CALLS:

CALLED BY: RW1, SYSEXIT, SYSILGINS, SYSMDATE, GTTY, SYSUNLINK, SYSFSTAT, SYSCHDIR, ARG2, SYSBREAK, SEEKTELL, SYSINTR, SYSQUIT, SYSUMOUNT

WORK REFERENCE: U2: 6

ID – u2; 7 arg2:

FUNCTION – Takes first arg in system call (pointer to name of file) and puts it in location u.namep ; Takes second arg and puts it in u.off and on top of the stack.

CALLING SEQUENCE – jsr r0, arg2

ARGUMENTS –

INPUTS – u.sp, r0

OUTPUTS – u.namep
u.off
u.off ↓ sp
r1

ID_ U2.4 error3

FUNCTION— see 'error' routine
CALLING SEQUENCE— "
ARGUMENTS— "
INPUTS— "
OUTPUTS— "

ID _ 42;1 error4

FUNCTION - see 'error' routine
CALLING SEQUENCE _ "
ARGUMENTS - "
INPUTS _ "
OUTPUTS - "

INPUTS — user parameters (see sec 6)

OUTPUTS — user parameters

ID-02;4 sysfstat

FUNCTION: "sysfstat" is identical to "sysstat" except that it operates on open files instead of files given by "name". It puts the buffer address on the stack gets the i-number and checks to see if the file is open for reading or writing. If the file is open for writing (i-number is negative) the i-number is set positive and a branch into sysstat is made.

CALLING SEQUENCE: sys fstat ; buf

ARGUMENT: buf - buffer address

INPUTS (u.r0) file descriptor

OUTPUTS - buffer is loaded with file information. See UNIX PROGRAMMERS manual under SYSSTAT (II) for format of the buffer

ID U2-9. sysgetuid

FUNCTION

sys getuid returns the real user ID of the current process. The real user ID identifies the person who is logged in, in contradistinction to the effective user ID, which determines his access permission at each moment. It is thus useful to programs which operate using the "set user ID" mode, to find out who invoked them.

CALLING SEQUENCE: sysgetuid

ARGUMENTS:

INPUTS: u.ruid - real users id

OUTPUTS (u.r0) - contains the real users id

ID U2-8 sysintr

FUNCTION "sysintr" sets the interrupt handling value. It puts the argument of its call in u.intr. "sysintr" then branches into the "sysquit" routine. u.tty is checked to see if a control tty exists. If one does the interrupt character in the tty buffer is cleared and sysret is called. If one does not exist sysret is just called.

CALLING SEQUENCE: sysintr; arg

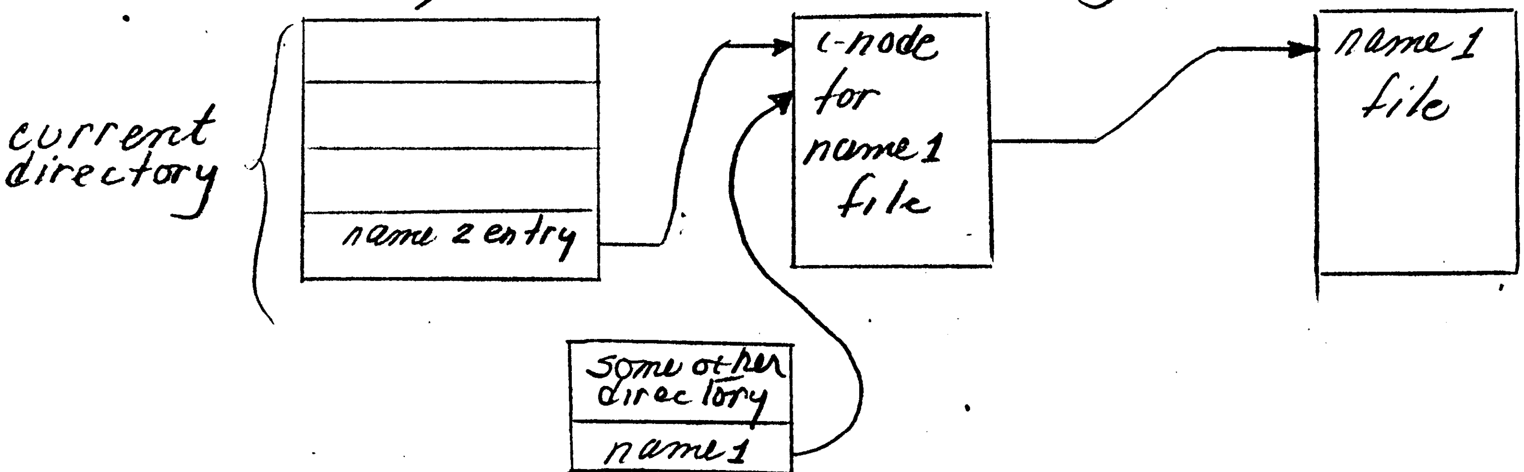
ARGUMENT arg - if 0, interrupts (ASCII DELETE) are ignored
- if 1, interrupts cause their normal result, i.e., force an exit.
- if arg is a location within the program, control is passed to that location when an interrupt occurs

INPUTS: u.tty - pointer to control tty buffer

OUTPUTS u.intr has value of arg
(R1)+6 (interrupt char in tty buffer) is cleared if a control tty exists

ID 02-1 syslink

FUNCTION: syslink is given two arguments name1 and name2. name 1 is a file that already exists. name2 is the name given to the entry that will go in the current directory. name2 will then be a link to the name1 file. The i-number in the name 2 entry of the current directory is the same i-number for the name1 file. At the end of a syslink call the following structure is constructed.



ARGUMENTS name 1 - file name to which link will be created
name 2 - name of entry in current directory that links to name1

INPUTS u.namep - points to the arguments above

OUTPUTS entry in the current directory with name name2
R1 = contains i-number of name1 on exit and
i-numbers of current directory intermittently during subr.
i.nlks - incremented by 1 to indicate another link added.
i.mod - set by call to setimod

sysexec call chain

```
sysexec u2;2 ── arg2* u2;7 ── arg* u2;6
             ── namei* u2;4 ── access* u5;3 ── iget u5;4 ── icalc* u5;4 ── dskrd* u8;3 ── bufaloc* u8;5
                                                                                      ── poke* u8;4
                            ── readi u6;1 ── dskr u6;2 ── iget u5;4 ── ...
                                                       ── mget* u5;1 ── alloc* u5;2
                                                                     ── setimod* u5;3
                                                                     ── clear u3;3 ── wslot* u8;3
                                                                                   ── dskwr u8;3 ── ppoke u8;4
                                                                     ── wslot* u8;3
                                                                     ── dskwr u8;3 ── ppoke u8;4 ── poke* u8;4
             ── iget u5;4 ── ...
             ── iopen u7;4 ── access* u5;3 ── ...
             ── copyz* u3;3
             ── readi u6;1 ── ...
             ── iclose u7;5 ── ...
             ── sysret3 u2;4 ── sysret u1;2 ── iget u5;4 ── ...
                                            ── ppoke u8;4 ── poke* u8;4
                                            ── tswap u3;1 ── putlu u3;3
                                            ── isintr u4;5 ── getc u7;2 ── get u7;3
                                            ── intract u1;4
```

ID – u2;4 sysret3

FUNCTION – see 'sysret' routine
CALLING SEQUENCE – "
ARGUMENTS – "
INPUTS – "
OUTPUTS – "

ID- 42;1 sysret4

FUNCTION- see 'sysret' routine
CALLING SEQUENCE- "
ARGUMENTS- "
INPUTS- "
OUTPUTS- "

ID_42;1 sysret9

FUNCTION- see 'sysret' routine
CALLING SEQUENCE - "
ARGUMENTS- "
INPUTS- "
OUTPUTS "

ID U2-8 sysseek

FUNCTION: sysseek changes the R/W pointer (3rd word in an fsp entry) of an open file whose file descriptor is in (u.r0.)

The file descriptor refers to a file open for reading or writing. The read (or write) pointer for the file is set as follows:

if <u>ptrname</u> is 0, the pointer is set to <u>offset</u>.

if <u>ptrname</u> is 1, the pointer is set to its current location plus <u>offset</u>.

if <u>ptrname</u> is 2, the pointer is set to the size of the file plus <u>offset</u>.

The error bit (c-bit) is set for an undefined file descriptor.

ARGUMENTS offset - number of bytes desired to move the R/W pointer by
ptrname - a switch indicated above

INPUTS u.base - } See seektell
u.count - }

OUTPUTS u.fofp - pts to the R/W pointer in the fsp entry
The R/W pointer is changed according to offset and ptrname

CALLING SEQUENCE sysseek ; offset; ptrname

ID U2-9 syssetuid

FUNCTION "syssetuid" sets the user id u.uid of the current process to the process id (u.r0). Both the effective user u.uid and the real user u.ruid are set to this. Only the super user and make this call.

CALLING SEQUENCE sys setuid

ARGUMENTS –

INPUTS (u.r0) . contains the process id
u.ruid – real user id
u.uid effective current user id

OUTPUTS u.ruid – set equal to the process id (u.r0)
u.uid – " " " " " " "

ID U2-4 sysstat

FUNCTION "sysstat" gets the status of a file. Its arguments are the name of the file and a buffer address. The buffer is 34. bytes long and information about the file is placed in it. sysstat calls "namei" to get the i-number of the file. Then "iget" is called to get the i-node in core. The buffer is then loaded and the results are given in the UNIX programmers manual SYS STAT (II).

CALLING SEQUENCE sys stat; name; buf

ARGUMENTS: name - points to the name of the file
buf - address of a 34 byte buffer

INPUTS: sp - contains the address of the buffer
R1 - i-number of file

OUTPUTS - buffer is loaded with file information

10- U2-7 sysstime

FUNCTION- "sysstime" sets the time. Only the super user can use this call.

CALLING SEQUENCE- sysstime

ARGUMENTS:

INPUTS- sp+2, sp+4 time system is to be set to

OUTPUTS- s.time, s.time+2 new time system is set to

ID U2-7 sys time

FUNCTION "systime" gets the time of the year. The present time is put on the stack

CALLING SEQUENCE sys time

ARGUMENTS:

INPUTS: s.time, s.time+2 - present time

OUTPUTS: sp+2, sp+4 - present time

ID U2-8 sys quit

FUNCTION: sysquit turns off the quit signal. It puts the argument of the call in u.quit. u.tty is checked to see if a control tty exists. If one does, the interrupt character in the tty buffer is cleared and sysret is called. If one does not exist, sysret is just called.

CALLING SEQUENCE sys quit; arg

ARGUMENT: arg; - if 0 this call disables quit signals from the typewriter (ASCII FS)
- if 1, quits are re-enabled and cause execution to cease and a core image to be produced.
- if an address in the program, a quit causes control to be sent to that location.

INPUTS u.tty - pointer to control tty buffer

OUTPUTS u.quit - has value of arg
(R1)+6 - (interrupt char in tty buffer) is cleared if a control tty exists

ID U2-1 sysunlink

FUNCTION: sysunlink removes the entry for the file pointed to by name from its directory. If this entry was the last link to the file, the contents of the file are freed and the file is destroyed. If, however, the file was open in any process, the actual destruction is delayed until it is closed, even though the directory entry has disappeared.

The error bit (c-bit) is set to indicate that the file does not exist or that its directory cannot be written. Write permission is not required on the file itself. It is also illegal to unlink a directory (except for the super-user).

ARGUMENTS name - name of directory entry to be removed

INPUTS u.namep - points to name

R1 - i-number associated with name

OUTPUTS i.nlks - number of links to file gets decremented

u.off - gets moved back 1 directory entry

imod - gets set by call to setimod

if name was last link contents of file freed & file destroyed

entry "name" in directory is free (its first word that usually contains and i-number is zeroed.

CALLING SEQUENCE sys link; name

ID U2-2 wdir

FUNCTION wdir - write a directory entry into the current directory whose i-number is in ii.

ARGUMENTS -

INPUTS:

ii - contains the current directories i-number

OUTPUTS an entry in the current directory
u.base points to u.dirbuf
u.count = 10,
R1 - contains the current directory's i-number

CALLING SEQUENCE jsr r0, wdir - in syslink
follows mkdir directly

ID U2-9 fclose

FUNCTION: Given the file descriptor (index to the u.fp list) "fclose" first gets the i-number of the file via "getf." If the i-node is active (i-number ≠ 0) the entry in the u.fp list is cleared. If all the processes that opened that file close it, then the fsp entry is freed and the file is closed. If not, a return is taken. If the file has been deleted while open (see "deleted flag" F page 8) "anyi" is called to see if anyone else has it open, i.e., see if it appears in another entry in the fsp table. (See "anyi" for details H.2 page 0) Upon return from "anyi" a check is made to see if the file is special

INPUTS R1 - contains the file descriptor (value = 0, 1, 2 ···· 9)

u.fp - list of entries in the fsp table

fsp - table of entries (4 words/entry) of open files See F page 8

OUTPUTS R1 - contains the same file descriptor it entered with

if all processes that open file close it, the fsp entry is freed and the file is closed

if "anyi" is called the outputs in "anyi" occur (H.2 p. 0)

- the "number of processes" byte in the fsp entry is decremented (see F page 8

- R2 contains i-number

ARGUMENTS —

CALLING SEQUENCE jsr r0, fclose

ID — u2;1 error 9

FUNCTION — see 'error' routine
CALLING SEQUENCE — "
ARGUMENTS — "
INPUTS — "
OUTPUTS — "

ID U2-3 "isdir"

FUNCTION: "isdir" checks to see if the i-node whose i-number is in R1, is a directory. If it is an error occurs, because "isdir" is called by syslink and sysunlink to make sure directories are not linked. If the user is the super user (u.uid = 0), "isdir" doesn't bother checking. The current i-node is not disturbed.

CALLING SEQUENCE – jsr r0, isdir

ARGUMENTS –

INPUTS. R1 – contains the i-number whose i-node is being checked

u.uid – user id

ii – current i-node number

i.flgs – flag in i-node (This is tested to see if the i-node is a directory i-node

OUTPUTS R1 – contains current i-number upon exit

current i-node back in core

10 U2-6 isown

FUNCTION: "isown" is given a file name. It finds the i-number of that file via "namei" then gets the i-node into core via "iget". It then tests to see if the user is the super user. If not, it checks to see if the user is the owner of the file. If he isn't an error occurs. If user is the owner "setimod" is called to indicate the i-node has been modified and the 2nd argument of the call is put in R2.

ARGUMENTS: —

INPUTS arguments of syschmod or syschown calls
u.uid - uid of user

OUTPUTS imod set to a 1
R2 - contains second argument of the system call

CALLING SEQUENCE jsr r0, isown

ID U2-7 maknod

FUNCTION: maknod creates an i-node and makes a directory entry for this i-node in the current directory. It gets the mode of the i-node in R1 the name is used in mkdir for the directory entry. (See mkdir H.2). The i-node is made in the following manner. First the allocate flag is set in the mode. A scan of i-nodes above 40 begins. The i-node map is checked to see if that i-node is active If it is the next i-node in the bit map is checked until a free one is found. If one is found a check is made to see if it is already allocated. If it is the search continues. It not the i-number is put in u.dirbuf and a directory entry is made via mkdir. Then the new i-node is fetched into core and its parameters are set (See outputs)

ARGUMENTS —

INPUTS
R1 - contains mode
ii - current i-number - should be of the current directory
mq, R2 - bit position & byte address in i-node map

OUTPUTS
u.dirbuf - contains i-number of free i-node
i.flgs - flag in new i-node
i.uid - filled with u.uid
i.nlks - 1 is put in the number of links
i.ctim - creation time
i.ctim+2 - modification time
imod - set via call to setimod

CALLING SEQUENCE jsr r0, mknod

10 U2-2 mkdir

FUNCTION "mkdir" makes a directory entry from the name pointed to by u.namep into the current directory.

It first clears the locations u.dirbuf+2 - u.dirbuf+10.

"mkdir" then moves a character at a time into u.dirbuf+2 - u.dirbuf+10, checking each time to see if the character is a "/." If it is an error occurs, because "/" should not appear in a directory name.

A pointer to an empty directory slot is then put in u.off. The current directory i-node is brought into core and an entry is written into the directory.

ARGUMENTS —

INPUTS R2, u.namep - points to a file name that is about to become a directory entry

R3 - points to u.dirbuf - locations

ii - current directory's i-number

OUTPUTS u.dirbuf+2 - u.dirbuf+10 - contains file name

u.off - points to entry to be filled in the current directory

u.base points to start of u.dirbuf

R1 - contains i-number of current directory

See wdir for others

ID U2-4 gett

FUNCTION: "getf" first checks to see that the user have not exceeded the maximum number of open files (10) If he has an error occurs. If not, the index into the fsp table is calculated from the u.fp list. u.fofp contains the address of the 3rd word in that fsp entry. (the file offset. See F page 8) cdev and r1 contain the device and I number of the file.

ARGUMENTS —

INPUTS R1 - contains index into u.fp list

OUTPUTS u.fofp - contains address of 3rd word in that fsp entry

cdev - contains files device number

R1 - contains files I-number

CALLING SEQUENCE: jsr r0, getf

ID U2-8 seektell

FUNCTION: seektell puts the arguments from a sys seek and systell call in u.base and u.count. It then gets the i-number of the file from the file descriptor in *u.r0 and by calling getf. The i-node is bought into core and then u.count is checked to see if it is a 0, 1 or 2.

If it is 0 - u.count stays the same
1 - u.count = offset (u.fofp)
2 - u.count = i.size size of file

ARGUMENTS —

INPUTS
u.base - puts offset from sysseek or systell call
u.count - put ptrname " " " " "
(u.r0) - contains file descriptor (index to u.fp list
i.size - size of file in bytes
(u.fofp) - points to 3rd. word of fsp entry

OUTPUTS - an i-node in core via "iget"
R1 - i-number of file in question
u.count - see function above

CALLING SEQUENCE: jsr r0, seektell

ID U2-7 sysbreak

FUNCTION: "sysbreak" sets the programs break point. It checks the current break point (u.break) to see if it is between "core" and the stack (sp). If it is so, it is made an even address (if it was odd) and the area between u.break and the stack is cleared. The new breakpoint is then put in u.break and control is passed to "sysret."

CALLING SEQUENCE: sysbreak ; addr

ARGUMENTS: addr - address of the new break point

INPUT: u.break - the current break point

OUTPUT: u.break - contains new break point
area between old u.break and stack is cleared if u.break is between "core" and the stack "sp."

ID - 42;4 namei:

FUNCTION - namei takes a file path name (address of string in u.namep) and searches the current directory or the root directory (if the first character in the string pointed to by u.namep is a "/") and returns the i-number for the file in R1.
namei operates in the following manner:

a file may be referenced in one of two ways, either relative to the users directory or relative to the rootdir directory; in the second case the file path name must begin with the char /. Whenever a / is encountered in a path name it indicates that the characters preceeding it represent the path name of a directory, and the file name following the / is stored in that directory.

Directories contain 10 byte entries, the first 2 bytes contain an inumber, the last 8 bytes a file name associated with the i-number.

namei scans the file path name until it reaches a </> or a <\0>, it reads the current directory until it finds a file name which matches the scanned portion of the file path name. When a match is found the i-number is taken from the matched directory entry. If namei has scanned to a <\0> then the i-number is that for the file specified by the file path name. If namei scanned to a </> then the i-number is that of the next directory in the path. namei scans the file path name until it reaches a </> or a <\0>, etc. If no file is found return to nofile:, otherwise normal:

ARGUMENTS -
INPUTS - u.namep (points to a file path name)
u.cdir (i-number of users directory)
u.cdev (device number on which user directory resides)
R1 - contains the i-number of the current directory (u.cdir)

OUTPUTS - r1 (i-number of file referenced by file path name)
- cdev
- r_2, r3, r4 (internal)
- u.dirp - points to the directory entry where a match occurs in the search for the file path name.
if no match u.dirp points to the end of the directory and
R1 = i-number of the current directory

CALLING SEQUENCE jsr r0, namei; nofile; normal;

ID U2-4 syschdir

FUNCTION: syschdir makes the directory specified in its argument the current working directory

ARGUMENTS name- address of the path name of a directory terminated by a 0 byte

INPUTS i.flgs - i-node flag

R1 - contains i-number

cdev - contains device number of i-node

OUTPUTS R1 -contains i-number

u.cdir - i-number of users current directory (same as R1)

u.cdev - device number of current directory

CALLING SEQUENCE syschdir ; name

ID U2-6 syschmod

FUNCTION "syschmod" changes the mode of a file.
It calls "isown" to get the files i-node into core and checks its owner against the user. A check then when made to see if the file is a directory. If it is, the "set user id" and "executable" modes are cleared because a directory cannot be executed. Then the remaining mode is set in the i.flgs.
If the file is not a directory i.flgs is set.

ARGUMENTS: name - name of file whose mode will be changed
mode - see syschmod UNIX programmers manual

INPUTS i.flgs - i-node flag
R2 - mode

OUTPUTS i.flgs - contains new mode

CALLING SEQUENCE sys chmod name; mod

ID U2-6 syschown

FUNCTION: "syschown" changes the owner of a file. "isown" is called to get the i-node of the name of the file. Then syschown checks to see if the "set user id on execution" flag is on. If it is, an error occurs, because one could create false files able to misuse other files. If not, the new owners id is put into the i-node.

ARGUMENTS name - file name
owner - id of new owner

INPUTS u.uid - user id
i.flgs - i-node flag

OUTPUTS i.uid - the i-node user id now contains id of new owner

CALLING SEQUENCE sys chown name; owner

ID 42;2 sysexec

FUNCTION sysexec initiates execution of a file whose path name is pointed to by "name" in the sysexec call. sysexec performs the following operations:

1. obtains i-number of file to be executed via "namei".

2. obtains i-node of file to be executed via "iget"

3. sets trap vectors to system routines.

4. loads arguments to be passed to executing file into highest locations of user's core.

5. puts pointers to arguments in locations immediately following arguments.

6. save number of arguments in next location.

7. initializes user's stack area so that all registers will be zeroed and the PS cleared and the PC set to $core when sysret restores registers and does an rti.

8. initializes u.r0 and u.sp

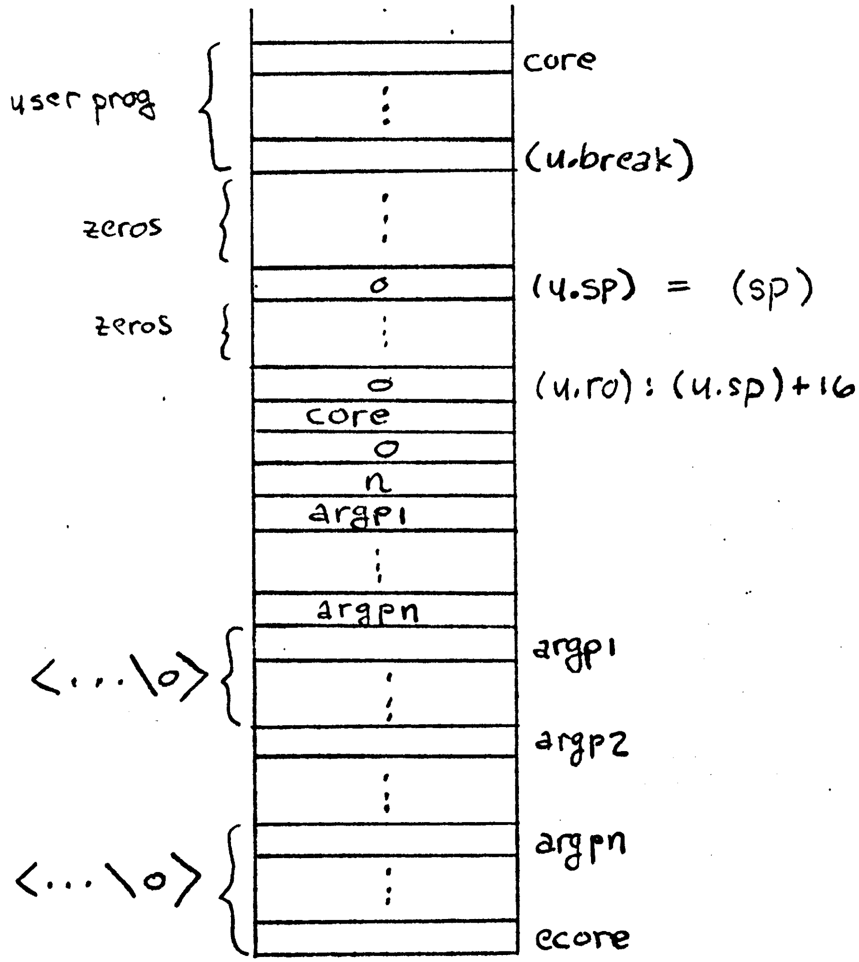
9. zeroes user's core down to u.r0.

10. reads in executable file from storage device into core starting at location "core".

11. sets u.break to point to end of user's code with data area appended.

12. calls "sysret" which returns control at location "core" via rti instruction.

The layout of core when sysexec calls sysret is :



CALLING SEQUENCE sys exec ; namep ; argp

ARGUMENTS namep (points to file path name of file to be executed)
argp (address of table of argument pointers)
argp1, ..., argpn (table of argument pointers)
argp1 : <...\0>, argp2 : <...\0>, ..., argpn : <...\0> (argument strings)

ID U3-3 clear

FUNCTION: "clear" zero's out a block (whose block number is in R1) on the current device (cdev). "clear" does this in the following manner:

1) 'wslot' is called, which obtains a free I/O buffer (See 'poke' H.8 page 5) via 'bufaloc'.

Bits 9 & 15 of the 1st word of the I/O queue entry are set to set up the buffer for writing

2) The buffer is zeroed and written out on the current device for the block (indicated by R1) via 'dskwr'.

ARGUMENTS: —

INPUTS: R1 - contains block number of block to be zeroed
cdev - current device number
R5 - points to data area of a free I/O buffer
See inputs for bufaloc, wslot, dskwr

OUTPUTS: a zeroed I/O buffer onto the current device
R5 points to last entry in the I/O buffer
R3 has 0 in it. It counts from 256 - 0. It is used as a word counter in the block.

CALLING SEQUENCE: jsr r0, clear

ID _ U3;3 copyz

FUNCTION _ clears core from arg1 to arg2

CALLING SEQUENCE _ jsr r0, copyz; arg1; arg2

ARGUMENTS _ arg1 - address of lowest location in core to be cleared
arg2 - address of highest location in core to be cleared

arg1 < arg2

INPUTS _ r0 - return address for the routine calling copyz. It is used to access arg1, then arg2 and, finally, set to the actual return address of the calling routine.

OUTPUTS _ r0 - points to the next instruction to be executed in the calling routine

ID - U3-3 idle

FUNCTION

"idle saves the present processor status word on the stack then clears the processor status word. CLOCKP is saved on the stack. It points to one of the clock cells in the super block. CLOCKP is then made to point to another set of clock cells specified as an argument in its call. When an interrupt occurs clockp and the processor status word are popped off the stack thus being reset to their values before the call took place

CALLING SEQUENCE jsr r0, idle

ARGUMENTS s.wait + 2

INPUTS

ps - process status

clockp clock pointer

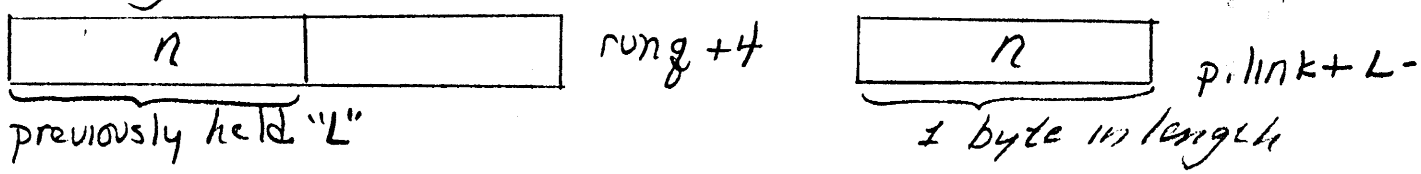
OUTPUTS

ps - restored to original value

clockp " " " "

ID U3-3 putlu

FUNCTION "putlu" is called with a process number in R1 and a pointer to the lowest priority q (rung+4) in R2. A link is created from the last process on the queue to the process in R1 by putting the process number in R1 into the last processes link. (The last processes number slot in p.link). The process number in R1 is then put in the last process position on the queue. If the last process on the queue was "L" and the process number in R1 was "n" then upon return from putlu the following would have occured:



ARGUMENTS —

INPUTS R1 user process number
R2 points to lowest priority Queue

OUTPUTS R3 process number of last process on the queue upon entering putlu
p.link-1+(R3) - process number in R1
rung+5 - process number in R1
R2 points to lowest priority queue

ID V3-1 swap

FUNCTION: swap is the routine that controls the swapping of processes in and out of core. It works in the following manner:

1) The processor priority is set to 6

2) The runq table is searched for the highest priority process. If none are found, idle is called to wait for an interrupt to put something on the queue. Upon returning after an interrupt, the queues are searched again.

3) The highest priority process number is put in R1. If it is the only process on that queue the queue entry is zeroed. If there are more processes on this queue the next one in line is put in the queue from p.link. (See F page 9)

4) The processor priority is set to 0

5) If the new process is the same as the process presently in core, nothing happens. If it isn't, the process presently in core is written out onto its corresponding disk block and the new process is read in. "wswap" writes out the old process. "rswap" reads in the new one. For more information see "wswap", "rswap", "unpack" and p17 of Implementation manual.

6) The new processes' stack pointer is restored. The address where this process left off before it was swapped out, is put in R0. So when "rts R0" is executed this new process will continue where it left off.

INPUTS
runq table - contains processes to be run. See F page 9
p.link - contains next process in line to be run. See F page 9
u.uno - process number of process in core
sstack - swap stack used as an internal stack for swapping

ARGUMENTS —

OUTPUTS
present process to its disk block
new process into core
uquant = 30. (Time quantum for a process)
u.pri points to highest priority run Q
R2 - points to the run queue
R1 - contains new process number
PS - processor status = 0
R0 - points to place in routine or process that called swap
all user parameters

ID U3-1 tswap

FUNCTION "tswap" is the time out swap. "Tswap is called when a user times out. The user is put on the low priority queue. This is done by making a link from the last user on the low priority queue to him via a call to "putlu." Then he is swapped out.

CALLING SEQUENCE – jsr r0, tswap

ARGUMENTS –

INPUTS u.uno - users process number

runq+4 - lowest priority queue

OUTPUTS R1 - users process number

R2 lowest priority queue address

ID U3-2 unpack

FUNCTION: "unpack" unpacks the users stack after swapping and puts the stack in its normal place. Immediately after a process is swapped in its stack is next to the program break. "unpack" move the stack to the end of core.

If u.break is less than "core" or greater than u.usp nothing happens. If u.break is in between these locations, the stack is moved from next to u.break to its normal location at the end of core.

CALLING SEQUENCE jsr r0, unpack

ARGUMENTS: —

INPUTS: u.break - users break point (end of users program)

OUTPUTS - stack gets moved if proper conditions stated above are met.

ID U3-2 rswap

FUNCTION: "rswap" reads a process, whose number is in R1, from disk into core. 2 * (the process number) is used as an index into p.break and p.dska. The word count in the p.break table is put in the 3rd word of the swp I/O queue entry. The disk address in the p.dska table is put in the second word. The first word of the swp I/O queue entry is set up to read. (bit 10 set to a 1) and "ppoke" is called to read the process into core.

ARGUMENTS: —

INPUTS
R1 - contains process number of process to be read in
p.break - table containing the negitive of the word count for the process
p.dska - table containing the disk address of the process
u.emt - determines handling of emts
u.ilgins - determines handling of illegal instructions

OUTPUTS
10 = (ilgins)
30 = (u.emt)
swp - bit 10 is set to indicate a read (bit 15=0 when reading is done)
swp+2 disk block address
swp+4 - negitive word count

CALLING SEQUENCE: jsr r0, rswap

ID U3-1 wswap

FUNCTION: "wswap" writes out the process that is in core onto its appropriate disk area. The process stack area is copied down to the top of the program area to speed up I/O. The word count is calculated and put in "swp+4." The disk address (block number) is put in "swp+2". "swp" is set up to write by setting bit 9 and "ppoke" is called to initiate the writing. The area from $user to the end of the stack is written out. The I/O queue entry "swp" looks like Fig 1 below just before the process is written out by ppoke.

	bit 9 among others is set	swp
	disk block address	swp+2
	neg. word count	swp+4
constant →	user (address to start writing from)	swp+6

When the writing is done, bit 15 of swp is cleared

ARGUMENTS INPUTS — u.break - points to end of program
u.usp - stack pointer at moment of swap
core - beginning of process program
ecore - end of core
user - start of user parameter area
u.uno - user process number
OUTPUTS p.dska - holds block number of process
swp I/O queue, (See above)
p.break - negitive word count of process
R1 - processes disk address
R2 - negitive word count

ID U4-1 clock

FUNCTION: "clock" handles the interrupt for the 60 cycle clock. It increments the time of day, increments the appropriate time category and decrements the users time quantum. It then searches through the toutt table and does the following:

1) If the processor priority is high (≥4) and the time in the toutt entry is not out (≠0), the time in the entry is decremented. If it turns 0 when decremented, it is incremented so that it will turn 0 next time when the priority might be low (See 2 below)

2) If the processor priority is low and (1) the user is not timed out or (2) we are presently inside the system and a toutt entry gets decremented to 0, the corresponding routine in the touts table is called. If the toutt entry was 0 before decrementing, nothing happens. If the user is timed out and we are outside the system, the users R0 is restored to him and "sysrele" is called to swap him out and bring in another process.

CALLING SEQUENCE   interrupt vector

ARGUMENTS   ——

INPUTS	LKS	- clock status register
	s.time+2	- time of day
	clockp	- points to one of the clock cells in the super block
	uquant	- users time quantum
	sysflg	- system flag -1 is outside system, 0 is inside
	toutt	- table of bytes. Each byte is a time count
	touts	- table of entry points of subroutines
OUTPUTS	s.time+2	- incremented
	clockp	- incremented
	uquant	decremented
	toutt	- entries decremented
	R0	- contains users R0 if conditions of (2) above are met

ID U4-4 isintr

FUNCTION "isintr" checks to see if an interrupt or quit from a tty belongs to the current user. If so, it won't skip on return; if not it will skip. When the interrupt does belong the output list in clist is erased via calls to getc
This prevents output coming out after the interrupt key is hit.

Case I Nothing happens except the return is skipped when

1) u.tty, the tty buffer pointer = 0
2) interrupt character in buffer = 0
3) interrupt char = "delete" and u.intr = 0
4) char = "FS" and u.quit = 0
5) no tty block is found that matches u.tty

Case II The return is not skipped and the output gets flushed if

1) interrupt character = "FS", u.quit ≠ 0 and the tty block in control is found
2) interrupt character = "delete" and u.intr ≠ 0 and the tty block in control is found.

CALLING SEQUENCE jsr r0, isintr

ARGUMENTS —

INPUTS
u.ttyp - pointer to buffer of tty in control of the current process
u.intr - determines handling of interrupts if 0 - nothing happens
u.quit " " quits " "
tty+6 - pointer to buffer of first tty block

OUTPUTS
Case I nothing except return is skipped
Case II Processor priority = 5
getc erases the output character list

ID- U4;3 PPTI- PAPERTAPE INPUT INTERRUPT ROUTINE

FUNCTION— PPTI DOES ONE OF FOLLOWING DEPENDENT ON VALUE OF "PPTIFLG"

1. IF "PPTIFLG" INDICATES FILE NOT OPEN (=0), NOTHING IS DONE

2. IF "PPTIFLG" INDICATES FILE JUST OPENED (=2), A CHECK IS MADE TO DETERMINE IF THE ERROR BIT IN PRS IS SET. IF IT IS "PPTITO" IS CALLED TO PLACE 10 IN THE TOUTT ENTRY FOR PPT INPUT. IF THE ERROR BIT IS NOT SET, "PPTIFLG" IS CHANGED TO INDICATE "NORMAL" OPERATION (SET TO 4) AND "WAKEUP" IS CALLED TO WAKEUP PROCESS IDENTIFIED IN WLIST FOR PPT INPUT. ALSO, THE CHARACTER IN THE PRB BUFFER IS PLACED IN CLIST IF THERE IS ROOM. IF THERE IS NO ROOM, THE CHARACTER IS LOST. FINALLY A CHECK IS MADE TO DETERMINE IF THE CHARACTER COUNT IN THE PPT INPUT AREA OF CLIST HAS < 50 CHARACTERS. IF IT DOES, THE READER ENABLE BIT IS SET.

3. IF "PPTIFLG" INDICATES FILE NORMAL (=4) THE PROCESS IN THE PPT INPUT ENTRY OF WLIST IS WOKEN UP (VIA WAKEUP). A CHECK IS THEN MADE TO DETERMINE IF THE ERROR BIT IN PRS IS SET. IF IT IS, THE "PPTIFLG" IS SET EQUAL TO 6. IF IT IS NOT THE CONTENTS OF PRB ARE PLACED IN THE CLIST VIA "PUTC". IF CLIST IS FULL, THE CHARACTER IS LOST. IN ADDITION IF THE CHARACTER COUNT FOR PPT INPUT IN THE CLIST IS LESS THAN 50, THE READER ENABLE BIT IS SET.

4. IF "PPTIFLG" INDICATES THE FILE IS NOT CLOSED (=6) THIS IS AN INDICATION THAT THE ERROR BIT WAS SET WHEN PPTIFLG EQUALLED FOUR AND THEREFORE NOTHING IS DONE.

CALLING SEQUENCE— PPTI IS THE PAPERTAPE INPUT INTERRUPT ROUTINE

INPUTS— PPTIFLG - FLAG WHICH INDICATES FUNCTION TO BE PERFORMED
PRS - PAPER TAPE READ STATUS BITS
CC+2 - CHARACTER COUNT FOR PPT INPUT IN CLIST
PRB - INPUT CHARACTER

OUTPUT— PPTIFLG—(SEE ABOVE)

ID_U4;4 PPTITO - PAPER TAPE INPUT TOUTS SUBROUTINE

FUNCTION — IF "PPTIFLG" INDICATES THE FILE HAS JUST BEEN OPENED (=2), "PPTITO"

1. PLACES 10 IN THE TOUTT ENTRY FOR PPT INPUT
2. CHECKS ERROR BIT IN PRS AND SETS READER ENABLE BIT IF ERROR BIT NOT SET.

FOR ALL OTHER VALUES "PPTIFLG" PPTITO DOES NOTHING.

CALLING SEQUENCE — JSR R0, PPTITO

INPUTS — PPTIFLG - VALUE OF THIS PARAMETER INDICATES TO PPTITO THE FUNCTION IT IS TO PERFORM
PRS — STATUS OF PPT READER

OUTPUTS — TOUTT+1 - CONTAINS TIC COUNT FOR PPT INPUT.
PRS — READ ENABLE BIT

ID - U4, 3 PPTO - PAPER TAPE OUTPUT INTERRUPT ROUTINE

FUNCTION - CALLS STARPPT TO OUTPUT NEXT CHARACTER IN CLIST PPT OUTPUT

CALLING SEQUENCE — INTERRUPT ROUTINE

INPUTS - SEE INPUTS FOR "STARPPT"

OUTPUTS — SEE OUTPUTS FOR "STARPPT"

ID 04-3 retisp

FUNCTION "retisp" pops the stack and restores the values of R0, R1, R2, R3 and clockp to what they were before the interrupt occured. retisp then executes and RTI and returns

CALLING SEQUENCE jmp retisp

ARGUMENTS —

INPUTS

OUTPUTS R0, R1, R2, R3, CLOCKP

ID U4-5 sleep

FUNCTION: sleep puts the process whose process number is in u.uno on the wait list (wlist) and swaps it out of core. It works in the following way:

1) A wait channel number is given as an argument to sleep. The process number occupying that channel is saved on the stack. The process number that is getting put to sleep (u.uno) is put in that wait channel

2) A call is made to "isintr to see if that user has any interrupts or quits. If he does a return to him via "sysret" is made. If he doesn't swap is called to swap out the process so it can sleep.

3) A check is made on the new user (the one who got swapped in) to see if he has any interrupts or quits. If not, a link is created to the old process number that first occupied the wait channel by a call to "putlu" a normal return is then made

CALLING SEQUENCE jsr r0, sleep; arg

ARGUMENTS arg - wait channel number

INPUTS u.uno - process number that gets put to sleep
wlist - wait channel list
runq+4 - lowest priority run Q

OUTPUTS sleeping process number onto wlist
sleeping process onto disk

ID – U4;5 STARPPT

FUNCTION – "STARPPT" CHECKS THE CHARACTER COUNT FOR PPT OUTPUT IN THE CLIST. IF IT IS ≤10, "STARPPT" USES "WAKEUP" TO WAKEUP PROCESS IDENTIFIED IN "WLIST" ENTRY FOR PPT OUTPUT. "STARPPT" THEN CHECKS THE READY BIT IN THE PUNCH STATUS WORD. IF IT IS SET, "STARPPT" USES GETC TO FETCH THE NEXT CHARACTER IN THE CLIST AND THEN PLACES IT IN PRB.

CALLING SEQUENCE – JSR r0, STARPPT

INPUTS – CC+3 – CHARACTER COUNT FOR PPT OUTPUT IN CLIST
PPS – CONTAINS READY BIT

OUTPUTS – See outputs for "GETC" and "WAKEUP"

PPB – PPT OUTPUT BUFFER

ID U4-1 setisp

FUNCTION: "setisp" stores R1, R2, R3 and clockp on the stack. Puts $s.syst+2 in clockp and returns via a jump without popping the stack

CALLING SEQUENCE jsr r0, setisp

ARGUMENTS

INPUTS R1, R2, R3, & clockp are saved on the stack

OUTPUTS clockp points to s.syst+2

ID u4;4 startty

FUNCTION startty prepares the system to output a character on the console tty.

It performs the following operations:

1_ SOME FOOLING WITH WAKEUP?

2_ tests console output status register ready bit, if bit is clear; return.

3 - if bit is set check timeout byte for console (toutt); if non zero; return.

4 - if toutt is zero, put char to be output in r1

5_ load character in console data buffer register

6_ If char = LF, make next char to be output a CR

7- If char = HT or CR, set time out to 15 clock cycles

ARGUMENTS

INPUTS ttyoch (character to be output), toutt

OUTPUTS tpb (loads a character in tty output data buffer register), r1 (character output), toutt

CALLING SEQUENCE jsr r0, startty

ID U4-3 ttyo

FUNCTION ttyo is the console typewriter output interrupt routine. It calls setisp to save registers during the interrupt then calls starttty to put the character in the tty output buffer and then restores the registers and returns from the interrupt.

CALLING SEQUENCE interrupt routine called via trap

ARGUMENTS —

INPUTS character in ttyoch

OUTPUTS see starttty

ID U4-2 wakeall

FUNCTION "wakeall" wakes up all the processes on the wait list by making consecutive calls to wakeup going through all the wait channels. The processes are linked together on the lowest priority queue (runq+4). Used to notify the world when a quit or interrupt happens from a typewriter

CALLING SEQUENCE jsr r0, wakeall

ARGUMENTS —

INPUTS

OUTPUTS all sleeping processes are put on the lowest priority queue

ID U4-2 ttyi - console tty interrupt handler (process priority = 5)

FUNCTION: ttyi puts a character from the tty reader buffer (tkb) in R1 sets the enable bit of the tty status register, and strips the character to 7 bits. Depending on what the character is the following things may occur.

1) If the character is a letter (A-Z)
It is changed to lower case and put on the console tty input list in clist via "putc". It is then put on the tty output buffer via "starttty" (Thus echoed). If the number of characters on that clist (cc) exceeds 15 a call to "wakeup" is made to clear that list. If less than 15 nothing else happens.

2) If the character is a "}" or a "del":
If also, the console tty blocks buffer pointer is zero wakeall is called and all sleeping processes are put on the low priority queue.
If the console tty blocks buffer pointer ≠ 0 the char (} or del) is put in the 7th byte of the buffer and wakeall is called.

3) If the char is an "EOT" or "NL"
cc is not checked and wakeup is called

CALLING SEQUENCE

ARGUMENTS

INPUTS

tkb	tty reader buffer	
tks	" reader status register	
cc	number of characters on the character list	

OUTPUTS R1 is used to contain the character
ttyoch - has the character
see function for other outputs depending on what the character is
R2 - points to the console tty's buffer
(R2)+6 - the interrupt char byte is filled with the appropriate char, if above conditions met

ID UH-5 wakeup

FUNCTION: wakeup is called with two arguments: arg1 is one of the run queues and arg2 is a wait channel number. wakeup wakes the process sleeping in the specified wait channel by creating a link to it from the last user process on the run queue specified by arg1. This is done by a call to "putlu". If there is no process to wake up, (wait channel contains a 0) nothing happens.

CALLING SEQUENCE jsr r0,wakeup ; arg1; arg2

ARGUMENTS arg1 - points to one of the three run queues
arg2 - is the number of the wait channel of the process to be woken

INPUTS: wlist - wait channel table
u.pri users process priority (expressed as a pointer to one of the run queues)

OUTPUTS: if u.pri > arg1, then uquant = 0
wlist (R3) = 0, i.e., entry in wait channel = 0
R2 ~~is~~ used to point to one of the run queues
R3 - contains the number of the wait channel

ID _ U5;3 access:

FUNCTION _ reads in section of core beginning at location "inode" the i-node for file with i-number n. Checks whether user is owner and other stuff to be described.

ARGUMENTS _ argo (user, owner flagmask)

INPUTS _ r1 (i-number of file), u.uid, i.uid

OUTPUTS _ inode, r2 (internal)

CALLING SEQUENCE _ jsr r0, access; argo

ID – us;2 free

FUNCTION – Given a block number for a block structured I/O device, 'free' calculates the byte address and bit position of its associated bit in the free storage map of the in-core image of the superblock for the device (RF fixed head disk or mountable device super block). It then declares the specified block free by setting this bit. Then a flag is set to indicate that: 1) the super block for the RF-fixed head disk has been modified (smod = smod+1).

or

2) the super block for a mountable device has been modified (mmod = mmod+1).

CALLING SEQUENCE – jsr r0, free

ARGUMENTS –

INPUTS – byte mask table:

Mask for bit 1	2	1	mask for bit 0
" " 3	10	4	" " 2
" " 5	40	20	" " 4
" " 7	200	100	" " 6

R1: block number for a block structured device

cdev: current device; 0 = drum, non zero = mountable device

OUTPUTS – MOUNT – SYSTM + (R2) word in free storage map portion of the in core image of the superblock for a mountable device. If the device is mountable, the appropriate bit is set to free the block. If the device is not mountable, the bit remains unchanged.

SYSTM + 2 + (R2) same as above, but for drum with the superblock for the fixed head disk.

ID_u5; 2 free

OUTPUTS – MMOD is incremented if the superblock for the mountable device was modified

SMOD is incremented if the superblock for the drum was modified

R2 - saved on stack and restored on return
R3 - " "

ID — u5;2 alloc:

FUNCTION — "alloc" scans the free storage map of the super block of a specified device. When it finds a free block it saves the physical block number in r1, it then sets the corresponding bit in the free storage map and sets the super block modified byte (smod, mmod).

CALLING SEQUENCE — jsr r0, alloc

ARGUMENTS —

INPUTS — cdev (current device), r2, r3

OUTPUTS — r1 (physical block number of block assigned), smod, mmod, systm (drum superblock), mount (dismountable super block), r2 (internal), r3 (internal) stack (values of r2, r3 on input)

ID 45;4 icalc

FUNCTION icalc calculates the physical block number from the i-number of an i-node. it then reads in that block and calculates the byte offset in the block for the i-node with the particular i-number, then depending on whether the argument in the icalc call is a 0 or a 1 it reads the inode in the data buffer in core starting at location "inode" (argument = 0) or it will take the inode information currently stored at location "inode" and write it out on the device. (argument = 1)

the physical block number and byte offset for an inode is calculated as follows:

let N = i-number , PBN = physical block number , BO = byte offset

then PBN = (N + 31.) / 16

and BO = 32.*((N+31.) mod 16.) (see SEC F for general discussion of inodes)

ARGUMENTS arg – arg = 0 read inode
arg = 1 write inode

INPUTS inode: , r1 (inumber)

OUTPUTS inode: , r1 (internal), r5 (internal) , r3 (internal)

CALLING SEQUENCE jsr r0, icalc ; arg

ID — 4.5.4 iget

FUNCTION — "iget" get a new i-node whose i-number is in r1 and whose device is in cdev. If the new i-number and its device are the same as the current i-number and its device (r1=ii and cdev=idev) no action is taken. If they do not agree, "iget" checks to see if the current i-node has been changed (imod≠1) if it has been changed the current inode is written out to its device. Then if the current device is the drum, the new i-node i-number is checked to see if it is the i-number of the cross device file , if it is the current device becomes the mounted device and the i-number is set to 41. (thus the root directory for the mounted device is referenced).
Then the new inode is read into the "inode" block in core via "icalc".

ARGUMENTS

INPUTS   ii (current i-number) , rootdir
cdev (new i-node device)
idev (current i-node device)
imod (current i-node modified flag)
mnti (cross device file i-number)
r1 (i-number of new i-node)
mntd (mountable device number)

OUTPUTS   cdev, idev, imod, ii, r1

CALLING SEQUENCE   jsr r0, iget

ID US-5 itrunc

FUNCTION: "itrunc" gets and inode via iget. It increments through the i.dskp (list of contents or indirect blocks in the inode) table and frees the blocks specified there. If the file is small, the block numbers in the i.dskp list are freed. If the file is large, i.dskp contains pointers to indirect blocks. The block numbers in these indirect blocks are then freed and the indirect blocks are freed.

ARGUMENTS —

INPUTS
R1 - contains I-number for use by "iget."
i.dskp - pointer to "contents or indirect blocks" in an inode
i.flgs - contains flag for large file. see SEC F PAGE 5
i.size - size of file

OUTPUTS:
i.flags - "large file" flag is cleared
i.size - set to 0.
i.dskp - i.dskp+16 - the entire list is cleared
setimod - set to indicate i-node has been modified
R1 - contains I-number on return from this subr.
R3 - used in subroutine

Calling Sequence jsr r0, ITRUNC

ID US-3 imap

FUNCTION

imap finds the byte in core containing the allocation bit for an i-node whose number is in R1. This core area is a copy of the super block and happens to be the i-node map.
The byte address is calculated as follows

byte addr = addr of start of map + (i-number - 41) / 8

The bit position = (i-number - 41) mod 8

ARGUMENTS —

INPUTS R1 - contains I-number of i-node in question

OUTPUTS R2 - has byte address of byte with the allocation bit

MQ - has a mask to locate the bit position. a 1 is in the calculated bit position

R3 - used internally

CALLING SEQUENCE jsr r0, imap

ID – 4.5;3 setimod:

FUNCTION – sets byte at location "imod" to a 1, thus indicating that the i-node has been modified. Also puts the time of modification into the i-node.

CALLING SEQUENCE – jsr r0, setimod

ARGUMENTS –

INPUTS – s.time, s.time+2 (current time)

OUTPUTS – imod, i.mtim, i.mtim+2

ID U6-2 dskr

FUNCTION: "dskr" gets an inode into core via "iget"
It then sets u.count according to the following rules.
If the number of bytes left to read in a file is greater than the number of bytes he wants to read u.count is unchanged. If the number of bytes left to read in the file is less than u.count. u.count gets set to that number.
If the user offset u.fofp is greater than the file length, there is nothing left to read so dskr returns.
Once u.count is established a block address for the file is calculated via mget, the file is read into system buffers and the data is transferred to user buffers in core. If u.count is not 0 the process is repeated until u.count is 0. Processor status is then cleared

ARGUMENTS —

INPUTS: R1 - contains I-number
i.size - file size in bytes
u.count - byte count desired
u.fofp - offset in file telling how many bytes have been read

OUTPUTS data in user buffers in core.
R2 - internal register
PS = 0
R3 internal register

CALLING SEQUENCE jmp dskr

ID—u6;4 dskw

FUNCTION— 'dskw' writes user specified data into a file on the drum, as follows:

'dskw' obtains an i-node number from the stack. If the i-node currently residing in the i-node area of core has been modified, this i-node is written out onto the drum in its appropriate position in the i-list. In any event, the i-node specified in the stack by the caller is read into the i-node area of core. A file is composed of blocks. The caller can modify several blocks in several passes thru a single call to 'dskw'. The no. of the block to be modified next is calculated by 'dskw' from the file offset (relative to the start of the file in bytes) specified by the caller in (u.fofp). The caller specifies the number of bytes to be modified in u.count. If the number of bytes the user specifies plus the offset into the file is greater than the present size of the file in bytes, i.size, then the size of the file is increased to incorporate the data overflow by changing the file size field in the i-node for the file (which is currently in the i-node area of core). The time that this file size change occurs is also inserted into the i-node and the i-node modification flag (smod) is set. 'Dskw' then uses (u.fofp) to calcalate an offset (relative to the start of the block) which specifies the 1st location within the block at which the callers data is to be written. Note that the offset determines the maximum number of bytes of user data that can be written on the file during this pass thru dskw, 512.-file offset. If the number of data bytes the caller specifies is less than a block, the block is read from drum into a system buffer, then the appropriate bytes are overwritten. If the number of data bytes is less than a block, but exceeds 512.-file offset, only 512.-file offset bytes are over written. Succeeding passes thru 'dskw' are necessary to write out the rest of the data. After each pass, the modified file block (in the system buffer) is written out on drum. When all required blocks are written, counters and pointers are returned to the caller

CALLING SEQUENCE— jsr r0, dskw

ARGUMENTS—

INPUTS— sp - i-node number
(u.fofp) - file offset
u.count - no. of bytes of data the caller desires to write
i.size - size (in bytes) of file to be altered (This parameter appears in the i-node whose no. is in sp
see inputs for 'iget', 'setimod', 'mget', 'dskrd', 'wslot', 'sioreg'
r1 - pointr to callers data area
(r1), (r1)+1, ..., (r1)+[u.count-1] - the callers data
drum file

ID U6-4 cpass

FUNCTION: cpass gets the next character from the user into R1. A non-local return takes place (to the caller of "writei") when the users count (u.count) becomes zero.

ARGUMENTS —

INPUTS u.count - users character count

u.base - points to a users character buffer

OUTPUTS — if u.count ≠ 0

u.count gets decremented

R1 contains the next character

u.nread gets incremented

u.base - gets incremented to point to next character

— if u.count = 0

R0 - return address to program that called "writei"

R1 - i-number of file under consideration

CALLING SEQUENCE jsr r0,cpass

ID – u6;2 passc

FUNCTION – 'passc' moves a byte of information specified in the lower half of R1 to the byte address specified by (u.base). It then increments u.base to point to the next byte address, increments u.nread, the number of bytes passed, and decrements u.count, the number of bytes yet to be moved. If there are no more bytes to be moved, a "non-local return" to the caller of 'readi' (through which control was eventually passed to passc) is taken. The current i-number is popped off the stack into R1. If there are more bytes to be transferred, the processor status is cleared and control is returned to the caller.

CALLING SEQUENCE – jsr r0, passc

ARGUMENTS –

INPUTS – R1 – contains a data byte in the lower half

u.base – contains a pointer to the user area of core to which the data byte is to be transferred.

u.nread – the no. of bytes transferred

u.count – the no. of bytes to be read

(sp) – the non-local return address

(sp+2) – the value of R1 prior to calling 'passc'

OUTPUTS – (u.base) – 0, ..., (u.base) – [u.count – 1] contain the transferred information
u.base – points to the last byte transferred
u.nread – contains the number of bytes transferred + original value of u.nread
u.count – contains the number of bytes that still must be read
(sp) – if non-local return popped twice
ps – cleared

ID – 46;2 rerd

FUNCTION – see 'error' routine
CALLING SEQUENCE – "
ARGUMENTS – "
INPUTS – "
OUTPUTS – "

ID- u6;4 dskw

OUTPUTS — i.size — file size (may have been modified by 'dskw')
See outputs for 'i-get', 'setimod', 'mget'. 'dskrd', 'wslot', 'sioreg'
r1 — points to the location succeeding the last callerdata byte transferred
r2 — points to the location (in the system buffer) succeeding the last system buffer byte over written.
R3 - 0
u.count - 0
Modified drum file

ID – 46; 2 ret

FUNCTION – 'ret' is a special subroutine return, used by the following subroutines:

1. ret1
2. rppt
3. dskr
4. passc
5. dskw
6. bread
7. bwrite
8. rcvt

in place of the standard return. In addition to performing standard return functions, 'ret' pops the stack, and puts its value in R1. It also clears the program status word. 'ret' can be used simply to clear the program status word by entering via its 2nd entry point.

ARGUMENTS –

CALLING SEQUENCE – control should be passed to this routine by either a conditional or non conditional transfer to 'ret' (the 1st entry point), or to '1', the secondary entry point.

INPUTS – A. for primary entry : (SP)
B. for secondary entry : ______

OUTPUTS – A. for primary entry : R1, PS
B. for secondary entry : PS

ID U6;2 RPPT - READ PAPERTAPE

FUNCTION — RPPT USES "PPTIC" TO GET A CHARACTER IN PPT INPUT SECTION OF CLIST AND TO SET READER ENABLE BIT IN PRS. IF THE PPT INPUT SECTION IS EMPTY AND PPTIFLG = 6 (INDICATION THAT THE ERROR BIT WAS SET DURING "NORMAL OPERATION") RETURN IS MADE TO "RPPT" TO INSTRUCTION "BR RET" WHICH EVENTUALLY CAUSES A RETURN TO THE CALLER OF "READI." IF A CHARACTER IS AVAILABLE IN CLIST, RETURN IS MADE TO "RPPT" AT "JSR RO, PASSC."

UPON RETURN FROM "PPTIC", "RPPT" USES "PASSC" TO PLACE THE CHARACTER FETCHED BY PPTIC INTO THE USERS BUFFER AREA. IF THE NUMBER OF CHARACTERS THAT WERE SPECIFIED BY THE USER TO BE READ IN HAS BEEN READ IN, RETURN FROM PASSC IS MADE TO THE CALLER OF READI.

IT IS APPROPRIATE AT THIS POINT TO DESCRIBE HOW ALL THE PPT INPUT ROUTINES AND SUBROUTINES ARE TIED TOGETHER TO READ PPT. FIRST OF ALL THE PPT FILE MUST BE OPEN. TO DO THIS A "SYS OPEN" FOR READING OF THE PPT FILE IS USED. THIS ROUTINE OPENS THE FILE VIA "IOPEN" WHICH SETS THE "PPTIFLG" INDICATING FILE OPEN. IT ALSO SETS THE READER INTERRUPT ENABLE BIT IN THE PRS AND EMPTIES THE PPT INPUT PORTION OF CLIST

ONCE THE FILE IS OPEN, A "SYS READ" OF THE PPT FILE IS MADE. A POINTER TO THE LOCATION WHERE THE CHARACTERS ARE TO BE PLACED ALONG WITH THE NUMBER OF CHARACTERS TO BE READ ARE PASSED AS ARGUMENTS TO "SYSREAD". "SYSREAD" THEN USES "RW1" TO SET "U.COUNT" EQUAL TO THE NUMBER OF CHARACTERS TO BE READ AND "U.BASE" TO THE LOCATION WHERE THE CHARACTERS ARE TO BE PLACED. "READI" IS THEN CALLED WHICH JUMPS TO "RPPT" WHICH IS DESCRIBED ABOVE. IT SHOULD BE NOTED THAT WHEN "PPTIC" IS CALLED TO OBTAIN A CHARACTER FROM "CLIST", THE PROCESS WILL BE PUT TO SLEEP IF NO CHARACTERS ARE IN CLIST (WITH PPTIFLG ≠ 6) AND ALL CHARACTERS TO BE READ IN HAVE NOT BEEN READ. ALSO THE READER ENABLE BIT IS SET. UPON COMPLETION OF THE INPUT OF THE NEXT CHARACTER (READY BIT SET) THE PPT INPUT INTERRUPT ROUTINE (PPTI) IS STARTED WHICH USES "WAKEUP" TO WAKE UP THE PROCESS PREVIOUSLY PUT TO SLEEP.

CALLING SEQUENCE — JMP RPPT

INPUTS — SEE INPUTS FOR "PPTIC", "PASSC"

OUTPUTS — SEE OUTPUTS "PPTIC" AND "PASSC"

ID - u6;1 rtty

FUNCTION - Essentially, 'rtty' transfers characters from the console tty buffer into a user area of core, starting at byte address (u.base). If there are no characters in the console tty buffer, 'rtty' calls 'canon', which gets a line (120 characters) from the console tty clist and puts it in the console tty buffer. The caller specifies the number of characters to be transferred in u.count. If the number specified is greater than the number actually in the console tty buffer, a synthetic return is taken to the caller, after the characters in the buffer have been transferred. If the number specified is less than or equal to the number actually in the console tty buffer, a non-localized return to the caller of 'readi' (which is the routine via which control was actually transferred to 'rtty') is made when all the characters have been transferred to the users core area (via 'passc').

CALLING SEQUENCE - [conditional or unconditional branch, or jmp] rtty

ARGUMENTS -

INPUTS - tty + 70. - contains pointer to the header of the console tty buffer

2(tty+70.) - 2ND word of console tty buffer header; contains a count of characters in the buffer

4(tty+70.) - contains a pointer to the next character in the buffer. Pointer values can include (tty+70.)+ ?, (tty+70.)+ ?, ..., (tty+70.) + ?

See inputs for 'canon', 'passc', 'ret1'

OUTPUTS - R1, R5 used internally by 'rtty', original values destroyed

R5 - points to header of console tty buffer

see outputs for 'canon', 'passc', 'ret1'

ID U6-1 READI

FUNCTION: "readi" reads from and i-node whose number is in R1. If the file in i-node is special a transfer is made to the appropriate routine. If not "dskr" is called and the file is read into user core. See "dskr" for details

ARGUMENTS ___

INPUTS
R1 - contains and I-number
u.count - byte count user desires
u.base - pts to user buffer
u.fofp pts to word with current file offset

OUTPUTS: u.nread - accumulates total bytes passed back
See "dskr" H.6 page 1

CALLING SEQUENCE: jsr r0, readi

ID_U6; 3 WPPT_ WRITE PAPERTAPE

FUNCTION _ WPPT USES "CPASS" TO GET A CHARACTER FROM THE USERS BUFFER AREA AND "PPTOC" TO OUTPUT THE CHARACTER ON THE PUNCH.

IT IS APPROPRIATE AT THIS POINT TO DESCRIBE HOW ALL THE PPT OUTPUT ROUTINES AND SUBROUTINES ARE TIED TOGETHER TO OUTPUT DATA ON THE PPT PUNCH. FIRST THE PPT FILE MUST BE OPEN. THIS IS DONE VIA A "SYS OPEN" FOR WRITING. THIS PLACES ENTRIES IN THE FSP TABLE AND THE USER'S FP AREA.

ONCE THE FILE IS OPEN A "SYS WRITE" OF THE PPT FILE IS MADE. A POINTER TO THE LOCATION WHERE THE CHARACTERS ARE STORED ALONG WITH THE NUMBER OF CHARACTERS TO BE PUNCHED ARE PASSED AS ARGUMENTS TO SYS WRITE. THEN USES "RW1" TO SET "U.COUNT" EQUAL TO THE NUMBER OF CHARACTERS TO BE PUNCHED AND "U.BASE" EQUAL TO THE LOCATION OF THE CHARACTERS. "WRITE1" IS THEN CALLED WHICH JUMPS TO "WPPT."

"WPPT" AS MENTIONED ABOVE USES "CPASS" TO GET A CHARACTER FROM THE USER'S BUFFER AREA. IF THE NUMBER OF CHARACTERS AS SPECIFIED IN "SYS WRITE" CALL HAS BEEN READ, CONTROL IS PASSED BACK TO "SYSWRITE". IF NOT "PPTOC" IS CALLED. "PPTOC" FIRST CHECKS TO SEE IF CHARACTER COUNT FOR PPT OUTPUT IN THE 'CLIST' IS ≥ 50. IF IT IS THE PROCESS IS PUT TO SLEEP. IF IT ISN'T THE CHARACTER IS PLACED IN THE 'CLIST' AND "STARPPT" IS CALLED.

"STARPPT" USES 'GETC' TO GET A CHARACTER FROM CLIST AND INSERTS IT INTO THE PPB IF THE READY BIT IS SET. IF IT ISN'T, CONTROL IS PASSED BACK TO "PPTOC".

UPON COMPLETION OF OUTPUT OF THE CHARACTER IN PPB (READY BIT SET) THE PAPER TAPE OUTPUT INTERRUPT ROUTINE (PPTO) IS STARTED VIA AN INTERRUPT. THIS ROUTINE CALLS "STARPPT" WHICH PERFORMS THE FOLLOWING FUNCTION ON AN INTERRUPT IN ADDITION TO THOSE DESCRIBED IN THE PREVIOUS PARAGRAPH, IT CHECKS TO SEE IF THE CHARACTER COUNT FOR PPT OUTPUT IS < 10. IF IT IS IT WILL WAKEUP THE PROCESS IN THE WLIST ENTRY FOR PPT OUTPUT.

AS SEEN FROM ABOVE A PROCESS PUTS ITSELF TO SLEEP WHEN IT HAS ≥ 50 CHARACTERS IN CLIST AND IS "WOKEUP" BY THE PAPER TAPE OUTPUT INTERRUPT ROUTINE (PPTO) WHEN THE COUNT BECOMES < 10.

CALLING SEQUENCE - JMP PPT

INPUTS — SEE INPUTS FOR "CPASS" AND "PPTOC"

OUTPUTS — SEE OUTPUTS FOR "CPASS" AND "PPTOC"

ID _ 46;5' sioreg

FUNCTION _ 1. calculates the first byte location (in the I/O buffer assigned to the caller) into which the callers data is to be written.

2. calculates the number of user data bytes to be transferred into this I/O buffer.

3. performs bookkeeping functions, supplying the caller with information pertinant to the data transfer.

CALLING SEQUENCE _ jsr r0, sioreg

ARGUMENTS _

INPUTS _ (u.fofp) - specifies the byte in a file (relative to the start of the file) at which the user wants to start writing data.

R5 - address of data area of I/O buffer assigned to the user

u.base - address of 1st byte of user data

u.count - number of bytes of data to be transferred from user data area to I/O buffer

u.nread - number of bytes of data written out on the file for this user previously

OUTPUTS _ (u.fofp) - specifies the byte immediately following the last byte of the file area in which the u.count bytes of user data is to be written

R1 - address of 1st byte of user data

u.base - specifies the byte immediately following the last byte of user data to be transferred to the I/O buffer

u.count - specifies the number of bytes of user data left to be transferred after the preceeding set is transferred.

u.nread - updated to include the count of to be transferred bytes

R2 - specifies the byte in the I/O buffer assigned to the caller at which the transfer of users' data is to start.

R3 - number of bytes of user data to be transferred to users I/O buffer

ID_U6;2 writei

FUNCTION_ 'writei' checks to see if there is any data to be written (on any device). If not, it does nothing more than return to the routine which called it. If there is data to be written, 'writei' saves the i-node number of the file to be written on the stack, so it can be used by the appropriate output routine. Then 'writei' checks to see if the output is to a special file (those files associated with i-nodes 1,...,40., or to a non-special file. Writes for non-special files are routed to the 'dskw' routine. Writes for special files are routed to appropriate routines, as follows:

Special File		Write Routine
ASR-33	: console tty	wtty
PC11	: paper tape punch	wppt
	core	wmem
RF11/RS11	: fixed head disk (drum)	wrfφ
RK03/RK11	: moveable head disk	wrkφ
TC11/TU56	: dectape unit 1	wtap
"	" 2	"
"	" 3	"
"	" 4	"
"	" 5	"
"	" 6	"
"	" 7	"
(any std. tty)	: tty unit 1	xmtt
"	" 2	"
"	" 3	"
"	" 4	"
"	" 5	"
"	" 6	"
"	" 7	"

CALLING SEQUENCE_ jsr r0, writei

ARGUMENTS_

INPUTS_ u.count contains a count of the number of bytes to be written
r1 contains the number of the i-node for the output file

OUTPUTS_ A: to the calling routine if return is made to it by 'writei'
u.nread - is cleared
B: to the write routine for non-special files
u.nread - is cleared
(sp) - contains the i-node number

c: to the write routine for special files

u.nread - cleared

(sp) - contains the inode number

R1 - contains the index into the special file routine jump table

ID— 46;3 WTTY:

FUNCTION — "WTTY" USES "CPASS" TO ~~ROBTAIN~~ THE NEXT CHARACTER IN THE USER BUFFER AREA. IF THE CHARACTER COUNT FOR CONSOLE TTY IS ≥20, THE PROCESS IS PUT TO SLEEP. IF NOT, IT THEN USES PUTC TO DETERMINE IF THERE IS AN ENTRY AVAILABLE IN "FREELIST" PORTION OF "CLIST." IF THERE IS, "PUTC" PLACES THE CHARACTER THERE AND ASSIGNS THE LOCATION TO THE CONSOLE TTY PORTION OF "CLIST". IF THERE IS NO PLACE AVAILABLE IN THE "FREELIST" PORTION OF "CLIST", THE PROCESS IS PUT TO "SLEEP". IF THERE WAS A VACANT LOCATION, "STARTTY" IS USED TO ATTEMPT TO OUTPUT THE CHARACTER ON THE TTY. UPON RETURN FROM "STARTTY", THE NEXT CHARACTER IS OBTAINED FROM THE USER BUFFER. IF THE BUFFER IS EMPTY, CONTROL IS PASSED VIA "CPASS" BACK TO "SYSWRITE". WHEN THE PROCESS IS AWOKEN BY "WAKEUP", IT AGAIN TRIES TO FIND A LOCATION AVAILABLE IN "FREELIST" AND THE CHARACTER COUNT FOR THE CONSOLE TTY <20 SO IT CAN OUTPUT THE CHARACTER.

CALLING SEQUENCE — JMP WTTY

ARGUMENTS—

INPUTS— cc+1 - CONTAINS CHARACTER COUNT FOR CONSOLE TTY OUTPUT.

See inputs for 'cpass', 'putc', 'startty', 'sleep'

OUTPUTS R1 (CHARACTER FROM USER BUFFER)

PS - processor priority set to 5

See outputs for 'cpass', 'putc', startty', 'sleep'

ID UT-1 canon

FUNCTION canon handles the erase, kill processing on the typewriter (console tty). R5 points to the start of the tty buffer. The argument following the call is where the characters are obtained. "canon" returns only when (1) a full line has been gathered (2) a new line has been received (3) ~~&~~ and EOT (004) has been received (4) 120 characters (the length of the buffer) have been received.

canon works in the following way

1) The address of the start of the characters is put in buffer+4 (4(R5))
2) buffer+2 (2(R5)) is cleared. This is the character count
3) a character is gotten off the queue. If it is a kill character (@) (100₈) a return to the beginning is made. Actually one starts over.
4) If the character is an erase (#), (43₈) the next char will overwrite the previous one and thereby erase it
5) If the character is an EOT (004) the byte pointer is reset to the first character and a return is made
6) If char is none of the above, it is put in the buffer where the character pointer tells it to go *4(R5)
7) The character count 2(R5) and the character pointer 4(R5) are then incremented
8) If the char is a newline (\n) the char pointer is reset and a return is made
9) If the buffer is full (byte count ≥ 120) the char pointer is reset and a return is made.
10) If the buffer isn't full, the next character off the queue is put through the above tests.

Note Canon should only be called when the number of already treated characters is zero, i.e., when the char count = 0; 2(R5) = 0

If the char count is ≠ 0 the character pointer, 4(R5) points to the first character not yet picked up

CALLING SEQUENCE jsr r0, canon, arg

ARGUMENT arg - where characters are to be obtained from

INPUTS R5 points to tty buffer address

10(R5) start of character buffer

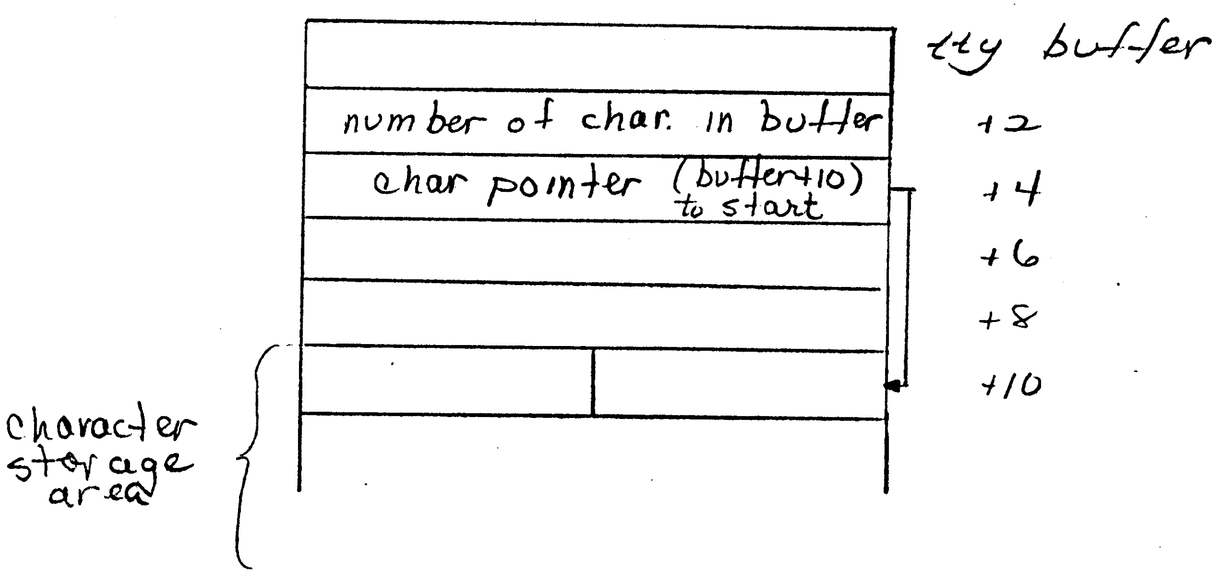
2(R5) character count

4(R5) points to next character position in data area

OUTPUTS a full buffer, or a full line

R1 pointers to buffer +10

4(R5) character pointer reset to start of data area buffer+10



ID U7-1 cesc

FUNCTION "cesc" is called by canon to check for an erase (#) or kill (@) character. R1 contains the character being tested If the character is not an erase or kill the return is skipped. If the char is an erase or kill the character count and character pointer are decremented. If the previous character was a "\" the # or @ are taken literally and the the return is not skipped

CALLING SEQUENCE jsr r0, cesc; arg

ARGUMENT .arg 100 - @ means kill the line
43 - # means erase last character

INPUTS R1 character to be tested
2(R5) character count
(R5)+4 - contains address of previous character

OUTPUTS skip return if test char is <u>not</u> erase or kill
if character was erase or kill
2(R5) - character count gets decremented
4(R5) character pointer " "

ID - U7; 7   CPPT - CLOSE PAPER TAPE FILE

FUNCTION — CPPT ASSIGNS ALL PPT INPUT LOCATIONS IN CLIST TO FREELIST, AND SETS "PPTIFLG" TO INDICATE FILE CLOSED (=0)

CALLING SEQUENCE — JMP CPPT

INPUTS — NONE

OUTPUTS — SEE OUTPUTS FOR "GETC"

PS - PROCESSOR PRIORITY SET TO 5
PPTIFLG - SET TO "0" TO INDICATE FILE CLOSED.

ID 07-6 ctty

FUNCTION: ctty closes the console tty. All it does is decrement the number of processes that have opened the console tty file. The first byte of the console tty buffer is the "number of processes that have opened this tty" byte. See F page 11. A return is made via "sret."

CALLING SEQUENCE jmp table in i-close

ARGUMENTS -

INPUTS: -

OUTPUTS R5 - points to console tty's buffer
(R5) - first byte of buffer gets decremented

ID- u7;8 erora

FUNCTION- see 'error' routine
CALLING SEQUENCE- "
ARGUMENTS- "
INPUTS- "
OUTPUTS- "

ID U7-1 ttych

FUNCTION ttych gets characters from the queue of characters inputted to the console tty. If there are none, sleep is called. ttych works in the following manner.

1) the processor priority is set to 5
2) a character is gotten off the queue via "getc" if the list is empty, sleep is called
3) if not the process status is cleared and a return is made

CALLING SEQUENCE jsr r0, *(r0) ttych was an argument in the call to "canon"

ARGUMENTS

INPUTS

OUTPUTS PS = 0
R1 - character on top of list
See getc H.7 page 2 for others

ID 47;3 get:

FUNCTION Removes the first clist entry from list identified by r1, makes the second entry the first. Puts the clist offset of entry removed from list in r2. return to "normal"
If the list identified by r1 is empty, r2 is returned equal to zero, and return made to "empty".
If the list has just one entry, the entry is removed and the first and last character pointers for the list are zeroed.

ARGUMENTS

INPUTS r1 (list identifier), cf+1(r1), cl+1(r1) (see sec G for general description of tty i/o handling)

OUTPUTS r2 (offset into clist of entry just removed from list r1), cf+1(r1), cl+1(r1), clist (r2)

CALLING SEQUENCE jsr r0, get; empty: ; normal:

ID U7-8 getspl

FUNCTION: getspl gets a device number from a special file name. "u.namep" points to the name. "namei" is called to get the i-number. i-number - 4 is the device number. If it is less than or equal to zero or if it is greater than 9 an error occurs. If not the device number is returned in R1.

CALLING SEQUENCE: jsr r0, getspl

ARGUMENTS: –

INPUTS u.namep - points to the name of the special file

OUTPUTS : R1 device number of the special file

ID U7-5 iclose

FUNCTION: "iclose" checks to see if the file, whose i-number is in R1, is special. If it is, a transfer is made to the appropriate routine. If it isn't a return is made.

ARGUMENTS —

INPUTS R1 - contains i-number of file being closed

OUTPUTS — If special file, R1 is put on the stack, ie., the i-number is put on the stack.

CALLING SEQUENCE jsr r0, iclose

ID 47;2 getc

FUNCTION getc removes the first clist entry from a list identified by arg, via call to get; decrements character count for list; puts the clist entry removed onto the free list; puts the character in the entry into r1 and takes "normal" return.

If list is empty take "empty" return.

ARGUMENTS arg – list identifier

INPUTS r2 (clist offset from put)

OUTPUTS r1 (character on top of list), cc(arg), clist(r2)

CALLING SEQUENCE jsr r0, getc; arg; empty: ; normal:

 OPPT - OPEN PAPER TAPE FILE FOR READ OR WRITE

FUNCTION — OPPT PERFORMS THE FOLLOWING FUNCTIONS

1. SETS THE READER ENABLE BIT IN PRS
2. ASSIGNS ALL PPT INPUT LOCATIONS IN "CLIST" TO FREELIST
3. SETS "PPTIFLG" TO INDICATE FILE JUST OPEN (=2) AND PLACES 10 IN TOUTT ENTRY FOR PPT INPUT

CALLING SEQUENCE — JMP OPPT

INPUTS — PPTIFLG - USED TO DETERMINE IF FILE ALREADY OPEN

OUTPUTS — PPTIFLG - SET BY OPPT TO INDICATE FILE JUST OPEN
PS - PROCESSOR PRIORITY SET TO 5
PRS - CONTAINS READER ENABLE BIT
TOUTT+1 - CONTAINS COUNT FOR PPT INPUT
SEE OUTPUTS FOR "GETC"

ID — U7,2: PPTIC — PAPER TAPE INPUT CONTROL

FUNCTION — PPTIC PERFORMS THE FOLLOWING FUNCTIONS FOR PPT INPUT:

1. IF THE ERROR, BUSY AND DONE BITS ARE NOT SET IN THE PRS AND THE CHARACTER COUNT FOR PPT INPUT IN THE CLIST IS < 30, PPTIC SETS THE READER ENABLE BIT.

2. USES "GETC" TO GET CHARACTER FROM PAPER TAPE INPUT AREA OF CLIST. IF THIS AREA OF "CLIST" IS EMPTY, A CHECK IS MADE TO SEE IF 'PPTIFLG' IS SET EQUAL TO SIX (INDICATION THAT ERROR FLAG IN PRS IS SET DURING NORMAL OPERATION). IF IT IS, RETURN IS MADE TO THE CALLING ROUTINE WHICH IN TURN RETURNS TO ITS CALLING ROUTINE. IF "PPTIFLG" DOES NOT EQUAL SIX, THE PROCESS IS PUT TO SLEEP.

CALLING SEQUENCE — JSR, R0, PPTIC

INPUTS — CC+2 — CONTAINS CHARACTER COUNT FOR PPT INPUT
PRS — CONTAINS STATUS BITS FOR PPT READER
PPTIFLG — INDICATES CONDITION OF PPT FILE
[illegible]

OUTPUTS — PRS — CONTAINS READER ENABLE BIT

SEE OUTPUTS FOR "GETC" [illegible]"

PS — PROCESSOR PRIORITY SET TO 5 AND THEN TO 0

ID- U7;2 PPTOC - PAPER TAPE OUTPUT CONTROL

FUNCTION — "PPTOC" FIRST CHECKS TO SEE IF THE CHARACTER COUNT FOR PPT OUTPUT IN THE CLIST IS $\geq$ 50. IF IT IS, THE PROCESS IS PUT TO SLEEP. IF IT ISNT "PUTC" IS USED TO PLACE THE CHARACTER, WHICH IS IN R1, IN THE CLIST. IF THE CLIST IS FULL, THE PROCESS IS PUT TO SLEEP. IF THE CHARACTER IS PLACED IN CLIST, "STARPPT" IS CALLED TO OUTPUT THE NEXT ENTRY IN THE PPT OUTPUT SECTION OF CLIST.

CALLING SEQUENCE — JSR R0, PPTOC

INPUTS — CC+3 — CHARACTER COUNT FOR PPT INPUT IN CLIST

OUTPUTS — PS — PROCESSOR PRIORITY SET EQUAL TO FIVE.
SEE OUTPUTS FOR "STARPPT" AND "SLEEP" AND "PUTC"

see inputs for putc R1 - character

ID U7-4 iopen

FUNCTION: iopen opens the file whose i-number is in R1. If the file is to be opened for reading "access" is called and the the i-number is checked to see if the file is special. If it is special, a jump table of transfer addresses takes care of transferring control to the correct special file routine. If non special file a return is made.
If the file is to be opened for writing, "access" is called and a check is made to see if the file is a directory. If it is, an error occurs, because users cannot write into directories. Special files are handled in the same manner as above.

CALLING SEQUENCE jsr r0, iopen

ARGUMENTS —

INPUTS R1 - contains i-number of the file to be opened

OUTPUTS - files i-node is in core
R1 - if i-number was negitive upon entry it is positive on exit

ID 47;3 put

FUNCTION Takes a clist entry pointed to by r2, and makes it the last entry in the list identified by r1.
If this is the first entry in a currently empty list then the first char pointer in cf is also updated.

ARGUMENTS

INPUTS — r1 (list identifier)
r2 (clist offset)

OUTPUTS — cl+1(r1), clist-1(r2), cf+1(r1)

CALLING SEQUENCE — jsr r0, put

ID – u7;3 put

FUNCTION –

CALLING SEQUENCE – jsr r0, put

ARGUMENTS:

INPUTS: r2 – 2, 4, 6, ..., 510.
r1 – minus 1

CL – zero, or non zero

OUTPUTS: if CL = 0

CF = r2/2
CL = r2/2
CF+1 = 0 if r2/2 < 128.
CF+1 = 255. if r2/2 ≥ 128.

if CL ≠ 0

CL = r2/2
CLIST + [r2+1] = r2/2

ID U7-7 sysmount

FUNCTION: sysmount announces to the system that a removable file system has been mounted on a special file. The device number of the special file is obtained via a call to "getspl". It is put in the I/O queue entry for the dismountable file system (sb1) and the I/O queue entry is set up to read. (bit 10 is set), "ppoke" is then called to read the file system into core. This call is super user restricted.

CALLING SEQUENCE: sys mount ; special ; name

ARGUMENTS: special - pointer to name of special file (device)
name - pointer to the name of the root directory of the newly mounted file system. "name" should always be a directory

INPUTS: mnti - records i-number of unique cross file device
sp - contains the name of the file
sb1 - I/O queue entry for the dismountable file system

OUTPUTS: mnti - i-number of special file
mntd - device number of special file
sb1 - has device number in lower byte
cdev - has device number
file system is read into core via ppoke

ID _ 4.7;8 sysreta

FUNCTION_ see 'sysret' routine.
CALLING SEQUENCE_ "
ARGUMENTS_ "
INPUTS_ "
OUTPUTS_ "

ID U7-8 sys umount

FUNCTION: sysumount announces to this system that the special file, indicated as an argument, is no longer to contain a removable file system. "getspl" gets the device number of the special file. If no file system was mounted on that device an error occurs, mntd and mnti are cleared and control is passed to sysret

CALLING SEQUENCE: sysumount; special

ARGUMENTS special - special file to dismount (device)

INPUTS mntd - device number of mounted device
sb1 I/O queue entry for the dismountable file system

OUTPUTS mntd - zeroed
mnti - zeroed

ID 4.7;3 putc:

FUNCTION puts a character at the end of a list identified by the argument in the putc call.

In detail it takes a clist entry from the free list via call to "get". Appends the entry to the list identified by arg via call to "put". Then fills in the new entry with a character passed in r1.

ARGUMENTS arg – list identifier (see discussion in G on tty device I/O)

INPUTS r1 – charater from device buffer,

OUTPUTS r2 – clist offset where character stored, cc(arg), clist-1(r2)

CALLING SEQUENCE jsr r0, putc ; arg

ID U8-1 bread

FUNCTION: "bread" reads a block from a block structured device (RK, RF, tape). It operates in the following way:

1) If "cold" =1 (cold boot) the block specified in R1, is read into an I/O buffer via "preread." If its a warm boot (cold=0), the block in R1 and the next consecutive block are read into I/O buffers via "preread". The reason two blocks are read in is to speed up the overall reading process. On a cold boot however, only two I/O buffers are available so only one buffer is used.

2) The block number is always checked to see if the maximum block number allowed on the device has been exceeded. (see argument) If the block number does exceed the maximum, an error occurs.

3) "preread" is called again on the first block. Since the first block is already in an I/O buffer, all preread will do is reverse the priority (see bufaloc H.7?8 page 1) so that the first block is of higher priority than the second.

4) bit 14 of the first block's I/O buffer is set

5) bits 10 and 13 (the read bits) of this I/O buffer are now checked If they are set (reading is still in progress) and the device is disk or drum, or the device is tape and "uquant" ≠ 0, "idle" is called. If the device is tape and uquant = 0, "sleep" is called. If bits 10 and 13 are 0 (read done), bit 14 of the I/O buffer is cleared and the data is moved from the I/O buffer to the users area. "dioreg" does the bookkeeping on the transfer.

6) If u.count = 0 the reading is finished. If not a branch back to the start is taken and the above steps are repeated.

7) A return is taken to the routine that called "readi."

CALLING SEQUENCE jsr r0, bread; arg

ARGUMENTS arg - maximum block number allowed on device

INPUTS R2 points to the users data area; R3 has the byte count
(u.fofp) is the block number
cdev is the device
u.base - base of users data area
u.count - number of bytes to read in
R1 is used internally as the block number.
cold - 0 warm boot or 1 cold boot
R5 - points to the beginning of the I/O buffer or the data area
uquant time quantum allowed for each process

OUTPUTS - block or blocks of data into the users area starting at u.base
(u.fofp) points to next consecutive block to be read
R3 = 0 (used internally)

ID U8-2 bwrite

FUNCTION; "bwrite" writes on a block structured device. (RF, RK, tape) It operates in the following way.

1) The block number is placed in R1
2) If the block number exceeds the maximum allowable block number of the device an error occurs
3) (u.fofp) is incremented to point to the next block in sequence
4) "wslot is called to get an I/O buffer to write into
5) "dioreg" is called to set up the bookeeping for the transfer.
6) The data is then transferred from the users area to the I/O buffer,
7) "dskwr" is called to write it onto the device
8) If u.count ≠ 0 the procedure is repeated. If it is, a return to the routine that called "writei" is made

CALLING SEQUENCE; jsr r0, bwrite; arg.

ARGUMENTS; arg - is the maximum allowable block number for the device

INPUTS; (u.fofp) is the block number
cdev is the device
R1 is used internally to hold the blocknumber
R5 points to the I/O data buffer
R2 points to the users data area; initially its u.base
u.count - number of bytes user desires to write
R3 - has the byte count

OUTPUTS; - block or blocks of data onto the specified device
(u.fofp) is the next block to be written into
R3 = 0 (used internally)

ID U8-3 dioreg

FUNCTION: "dioreg" does the bookkeeping on block transfers of data. It first checks to see if there are more than 512 bytes to transfer. If so, it just takes 512. If not, it takes u.count.

ARGUMENTS: —

INPUTS u.count number of bytes user wants transferred

u.base - start of users data area

OUTPUTS R3 - used internally to hold the count

u.nread - updated by adding R3

u.base " " "

u.count " " subtracting "

R2 has value of u.base before it gets updated.

ID — u5;5 bufaloc:

FUNCTION — "bufaloc" scans the I/O buffers for block structured devices, looking for an active buffer (bits 9,..., 15 of the 1st word in the I/O queue entry for the buffer are set) which has already been assigned to the block number and device currently under consideration, or for a free buffer (bits 9,..., 15 not set) which has been previously assigned to this device and block number. If there is no such buffer, the vacant buffer with the highest core address is assigned. If no free buffer is found, "bufaloc" calls "idle". Eventually, a buffer is located. The routine "poke" which actually performs the I/O operations scans the "bufp" area of core from the highest to the lowest address. thus the priority of an I/O queue entry is established by where a pointer to the I/O queue entry appears in bufp.

The newly assigned buffer I/O queue entry pointer is placed in "bufp" thus making it the lowest priority I/O operation in the queue. the other entries in "bufp" are moved into higher addresses to accomodate the newly assigned buffer's I/O queue entry pointer at location bufp.

Once the buffer has been assigned the device number is put into the low half of word 1 of the corresponding I/O queue entry and the block number is put into word 2 of the I/O queue entry.

CALLING SEQUENCE — jsr r0, bufaloc

ARGUMENTS —

INPUTS — cdev, r1 (block number), {bufp+2*n-2, (bufp+2*n-2), (bufp+2*n-2)+2 : n=1,..., nbuf}

OUTPUTS — r5 (pointer to buffer assigned), bufp, ..., bufp+12, (bufp), (bufp)+2, ps

ID u8;3 dskrd

FUNCTION dskrd acquires an I/O buffer, puts in the proper I/O queue entries (via bufaloc) then reads a block (number specified in r1) into the acquired buffer. If the device is busy at the time dskrd is called, dskrd calls idle. Once the I/O operation is completed r5 is set to point to the first data word in the buffer.

ARGUMENTS

INPUTS

OUTPUTS r5 — pointer to first word in data block ; (r5) ; ps

CALLING SEQUENCE jsr r0, dskrd

ID u8;3 dskwr:

FUNCTION dskwr writes a block out on disk, via ppoke. the only thing dskwr does is set bit 15 in the first word of the I/O queue entry pointed to by "bufp". 'wslot' which must have been called previously has supplied all the information required in the I/O queue entry.

ARGUMENTS

INPUTS

OUTPUTS (bufp)

CALLING SEQUENCE jsr r0, dskwr

ID U8-7 drum

FUNCTION "drum" is the interrupt handling routine for the drum. drum is called after the transfer of data to or from the drum is complete, i.e., when the ready bit in the dcs (drum control register) is set. (See interface manual page 73-74). R1, R2, R3 and clock p are saved on the stack. (See setisp) calls "trapt" to check for stray interrupt or error. If neither, it clears bits 12 & 13 in 1st word of transaction buffer, checks for more disk buffers to read into or write.; then returns from interrupt by calling retisp

CALLING SEQUENCE called by interrupt vector at location 204 after data transmission has taken place, i.e, ready bit of dcs set.

INPUTS same as setisp, trapt and retisp

OUTPUTS " " " " "

CALLED BY interrupt vector

CALLS setisp, trapt

ID- 48;3 error 10

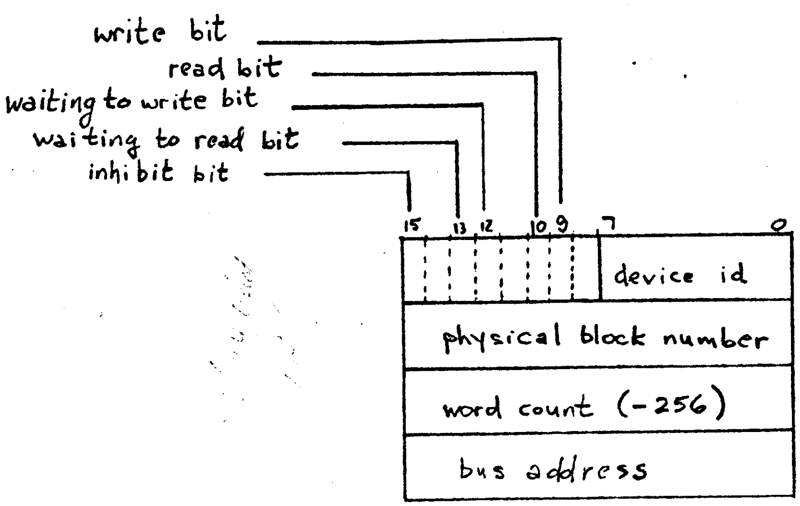
FUNCTION- see 'error' routine
CALLING SEQUENCE—
ARGUMENTS—
INPUTS—
OUTPUTS—

ID 48;4 poke

FUNCTION poke performs the basic I/O functions for all block structured devices. In order to understand the functioning of poke, the general handling of block structured I/O must be described.

I/O on block stuctured devices is handled via a collection of data buffers beginning at location "buffer" each buffer consists of a four word I/O queue entry followed by a 256 word data buffer.

An I/O queue entry has the following form:



byte 0 – device id codes are

0 = drum

1 = disk

other = dec tape

byte 1 – write bit – when set indicates write the data in the buffer out onto the device identified in byte 0.

read bit – when set indicates read data off of the indicated device into the data buffer

waiting to write bit – if set indicates that a write operation has been requested but not yet completed.

waiting to read bit – if set indicates that a read operation has been requested but not yet completed.

inhibit bit – when set will delay request for operation indicated by write bit or read bit until cleared.

byte 2-3 – physical block number (see sec G, discussion of file system)

byte 4-5 – word count – number of words in buffer; loaded into word count register for device

byte 6-7 – bus address – address of first word of data buffer.

In addition to the general I/O queue entries there are three special entries at locations sb0, sb1, and swp. These are the I/O queue entries for the super block for drum (sb0), the super block for the mounted device (sb1) and the core image being swapped in or out (swp) — these entries are initialized in the "allocate disk buffers" segment of code in u0.

An area in core starting at location "bufp" and extending nbuf + 3 words, contains pointers to the I/O queue entries. This table of pointers represents the priority of I/O requests, since poke scans these pointers starting at the highest address in "bufp", examining the control bits in byte 1 of each I/O queue entry pointed to by the bufp pointers. If either bit 9 or 10 is set and neither of bits 15, 13 or 12 is set then poke will attempt to honor the I/O request.

To honor an I/O request, poke checks "active" to see if the bit associated with the device is clear. If it is clear poke initiates the I/O operations by loading the appropriate device registers. In all I/O operations the interrupt is enabled and thus when completed an appropriate routine is called via the interrupt. When poke initiates a I/O operation it clears bit 9 or 10 and sets bit 11 or 12. The routine called upon completion of the I/O operation will clear bit 11 or 12 thus freeing that I/O queue entry "poke" calculates a physical disk address (which is loaded into register RKDA) from the physical block number in the following way:

let N = physical block number

then

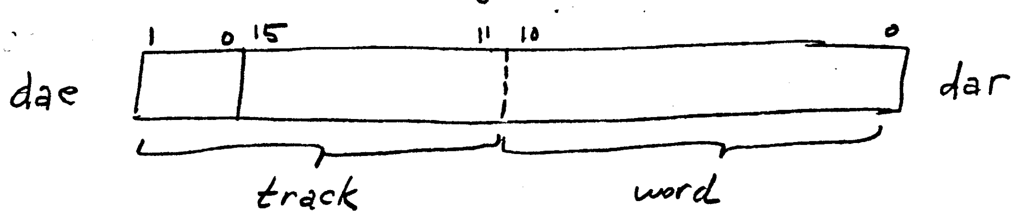
$$\text{sector number} = \text{remainder}\left(\frac{N}{12.}\right)$$

$$\text{surface} = \begin{cases} 0; & \text{quotient } \left(\frac{N}{12.}\right) \text{ even} \\ 1; & \text{quotient } \left(\frac{N}{12.}\right) \text{ odd} \end{cases}$$

$$\text{cylinder} = \text{quotient}\left[\left(\text{quotient}\left(\frac{N}{12.}\right)\right)/2\right]$$

"poke" calculates a physical disk address for the drum from the physical block number in the following way:

the drum address is given in the dae and dar registers.



the physical block number is essentially multiplied by 256 (by shifting the low order byte into the high order byte of the dar, and shifting the high order byte into the low order byte of the dae.

ARGUMENTS —

INPUTS — bufp, ..., bufp+22 , deverr ; active
(bufp), ..., (bufp)+6 ; (bufp+2), ..., (bufp+22)+6 (I/O queue entries)

OUTPUTS — sets bits 12 & 13 on I/O queue entries where I/O operation is initiated, active, rkap, rfap, dae, dar, wc, cma, dcs, rkcs, rkwc, rkba, rkda

CALLING SEQUENCE — jsr r0, poke

ID U8-3 preread

FUNCTION "preread" is called by "bread" to read in a disk block on device "cdev." The block number is in R1. "preread" gets a free I/O buffer via "bufaloc." It sets bit 10 of the first word of the I/O buffer and then reads the specified block into the I/O buffer via "poke." If the I/O buffer already contains the specified block bit 10 is not set and the call to "poke" is skipped. The processor status is then cleared.

CALLING SEQUENCE jsr r0, preread

ARGUMENTS:

INPUTS R1 - block number to read

R5 - points to first word of I/O buffer

OUTPUTS - specified block into an I/O buffer

PS = 0

R5 points to first word of the I/O buffer

ID U8-1 rtap

FUNCTION: "rtap" is the read routine for dec tape. The device number is (i-number/2) -4. The i-number is in R1 upon entry. "bread" is called to read the proper block or blocks

CALLING SEQUENCE from jump table in readi

ARGUMENTS: -

INPUTS R1 is the i-number of the special file

OUTPUTS cdev is the device number
See outputs for "bread"

ID U8-6 tape

FUNCTION "tape" handles the dec tape interrupts. "setisp" is first called to save registers and the clockp. The state of the dectape (tcstate) i.e., reading, writing, idle etc is put in R3. "trapt" is then called to check for data transmission errors. If none occur control passes to the appropriate dec tape routine depending on what the state is. Control is passed by putting R3 in the PC. If an error occurs a jump to "taper" is made.

CALLING SEQUENCE: Interrupt vector

ARGUMENTS:

INPUTS tcstate - the state of the dec tape (read, write etc)

OUTPUTS - control passes to appropriate dec tape routine
PC - set to address of above routine
R3 is used to hold the address of above routine

ID U8-2 tst deve

FUNCTION ; "tstdeve" checks to see whether a permanent error has occured on special file I/O. (It only works for tape however). If there is an error, the error is cleared and the user is notified.

CALLING SEQUENCE jsr r0, tstdeve

ARGUMENTS —

INPUTS cdev - the device in question
(R1) + Deverr) the device's in question error indicator

OUTPUTS R1 = cdev = the device number
If no error, nothing else happens
If error, (R1) + deverr gets cleared and user notified via error 10.

ID U8-8 trapt

FUNCTION: "trapt" is part of the drum, disk, or dec tape interrupt handler. The ready bit of the device control register is checked. If the ready bit is not set the device is still active so a return through "retisp" is made. It then checks to see if a stray interrupt has occured. If not "trapt" checks to see if an error in the data transmission has occured. If so, the return is skipped. If not, the return is not skipped. The return is via a jmp.

CALLING SEQUENCE: jsr r0, trapt ; dv ; buf ; act
br normal
br error

ARGUMENTS dv - device control status register (for dec tape it is the command register)
buf - contains address of disk buffer being read into or written)
act - tested against the bits in "active" to see if the device was busy

INPUTS: active - contains bits that tell which devices are busy

OUTPUTS R1 - points to the disk buffer
R2 points to the device control and status register or command register depending on the argument

ID – U8;3 wslot:

FUNCTION – wslot calls "bufaloc" and obtains, as a result, a pointer to the I/O queue of an I/O buffer for a block structured device. "Bufaloc" has inserted into this I/O queue the device number and block number which "wslot" passes from its caller to "bufaloc".

It then checks the first word of the I/O queue entry. If bits 10 and/or 13 (read bit, waiting to read bit – see H.8 p.5) are set, "wslot" calls "idle".

When "idle" returns, or if bits 10 and/or 13 are not set, "wslot" sets bits 9 and 15 of the first word of the I/O queue entry (write bit, inhibit bit), sets the processor priority to zero, and sets up a pointer to the first data word in the I/O buffer associated with the I/O queue.

CALLING SEQUENCE – jsr r0, wslot

ARGUMENTS –

INPUTS – See inputs for "bufaloc" – H.8 p.1

OUTPUTS – (bufp) – bits 9 and 15 are set, the remainder of the word is left unchanged
PS = 0
R5 – points to first data word in I/O buffer

See outputs for "bufaloc" – H.8 p1. Note that outputs given above take precedence over outputs from "bufaloc"

ID – U9;6 RCVCH – RECEIVE CHARACTER

FUNCTION – "RCVCH" USES "GETC" TO READ A CHARACTER FROM THE TTY'S READ SECTION OF THE CLIST. IT IS EMPTY, THE PROCESS IS PUT TO SLEEP. WHEN THE PROCESS IS AWAKEN, RCVCH AGAIN TRIES TO OBTAIN A CHARACTER FROM CLIST.

CALLING SEQUENCE JSR R0,RCVCH

INPUTS R2 – CONTAINS 8X TTY NO.

RCSR + 8XTTYN – CARRIER DETECT & CLEAR DATA TERM BITS

SEE INPUTS FOR "GETC" AND "SLEEP"

OUTPUTS PS – SET PROCESSOR STATUS TO 5

SEE OUTPUTS FOR "SLEEP" AND "GETC"

ID_ U9;6 RCVT - READ TTY

FUNCTION "RCVT" PLACES TTY CHARACTERS IN THE USER BUFFER AREA. IF THE "RAW" FLAG IN THE TTY AREA IS SET A CHARACTER IS OBTAINED FROM THE TTY'S INPUT AREA OF CLIST. IF THE FLAG IS NOT SET, "CANON" IS USED TO PROCESS A LINE OF TTY CHARACTERS AND PLACE THEM IN THE USER'S BUFFER AREA.

CALLING SEQUENCE JMP RCVT

INPUTS r1 - CONTAINS 2XTTY NO.
RCSR+8XTTYNO - CARRIER DETECT AND CLEAR DATA TERM BITS
TTY+8XTTYNO+6 - POINTER TO TTY BUFFER
TTY+8XTTYNO+4 - RAW DATA FLAG
SEE INPUTS FOR 'CANON', 'PASSC', GETC + RCVCH

OUTPUTS PS - SET PROCESSOR PRIORITY TO 5
SEE "CANNON" "PASSC", "GETC", "RCVCH" AND "SLEEP" OUTPUTS.

ID_U9;3 STARXMT

FUNCTION — STARXMT DOES THE FOLLOWING:

1. CHECKS TO SEE IF THE OUTPUT CHARACTER COUNT FOR THE TTY IN CLIST IS ≤ 10. IF IT IS, "STARMXT" USES "WAKEUP" TO WAKEUP THE PROCESS IDENTIFIED IN THE "WLIST" ENTRY FOR THE TTY OUTPUT CHANNEL.

2. CHECKS TO SEE IF THE TOUTT ENTRY FOR THE TTY OUTPUT IS EQUAL TO ZERO. IF IT IS NOT, CONTROL IS PASSED BACK TO THE CALLING ROUTINE.

3. CHECKS TO SEE IF THE READY BIT IN THE TTY'S TSCR REGISTER IS SET. IF IT IS NOT, CONTROL IS PASSED BACK TO CALLING ROUTINE

4. CHECKS 3RD BYTE OF TTY'S "TTY" AREA (CONTAINS CHARACTER LEFT OVER AFTER LF.) FOR A NULL CHARACTER. IF THE BYTE CONTAINS A NON NULL ENTRY, THE ENTRY IS USED AS THE NEXT CHARACTER TO BE OUTPUT. IF THE ENTRY IS NUL, THE NEXT CHARACTER TO BE OUTPUT IS OBTAINED FROM THE CLIST VIA "GETC"

5. ADDS 200_8 TO ASCII CODE OF CHARACTER TO BE OUTPUT IF DIGIT 2 (FAR LEFT DIGIT) OF ENTRY IN "PARTAB" TABLE FOR CHARACTER IS A "2".

6. CHECKS TTY'S RCSR BUFFER TO DETERMINE IF CARRIER IS PREENT. IF IT IS NOT THE CHARACTER IS "DROPPED" AND A NEW CHARACTER IS OBTAINED BY RETURNING TO THE BEGINNING OF THE SUBROUTINE. IF THE CARRIER IS PRESENT A CHECK IS MADE TO DETERMINE IF THE CHARACTER TO BE OUTPUT IS "HT". IF IT IS A CHECK IS MADE TO SEE IF THE "TAB TO SPACE" FLAG (BIT 1 OF 5TH BYTE IN "TTY" AREA) IS SET. IF IT IS THE CHARACTER TO BE OUTPUT IS CHANGED TO A SPACE (ASCII 40_8)

7. PLACES CHARACTER TO BE OUTPUT IN TTY'S "TCBR" BUFFER. "STARXMT" THEN DOES ONE OF THE FOLLOWING DEPENDENT ON THE CHARACTER TO BE OUTPUT (DIGITS 0 AND 1 OF THE CHARACTERS "PARTAB" ENTRY ARE USED AS OFFSETS INTO JUMP TABLE)

   a. FOR ASCII CODES 40-176, INCREMENTS COLUMN POINTER WHICH IS IN BYTE 2 OF TTY AREA

   b. FOR ASCII CODES 0-7, 16-37 AND 177, DOES NOTHING

   c. FOR ASCII 010 (BS), DECREMENTS COLUMN POINTER

   d. FOR ASCII 012 (LF), CHECKS FOR SETTING OF CR FLAG (BIT 4 OF 4TH BYTE IN "TTY" AREA). IF IT IS ASCII 015 (CR) IS PLACED IN BYTE 3 OF "TTY AREA (CHARACTER LEFT OVER AFTER LINE FEED). "STARXMT" THEN DETERMINES VALUE FOR THE TTY'S OUTPUT ENTRY IN THE TOUT TABLE. THIS VALUE IS DEPENDENT ON WHETHER "LF" IS TO BE OUTPUT OR

ID_U9; 3 STARXMT (CONT)

BOTH "LF" AND "CR".

e. FOR ASCII 011 (HT), DOES SOME FOOLING AROUND WITH COLUMN COUNT AND 3RD BYTE OF "TTY" AREA (CHARACTER LEFT OVER AFTER LF) DEPENDENT ON VALUE OF "TAB TO SPACE" FLAG IN 5TH BYTE OF "TTY" AREA. IT THEN DETERMINES VALUE FOR THE TTY'S OUTPUT ENTRY IN THE TOUT TABLE.

f. FOR ASCII 013 (VT), DETERMINES VALUE FOR THE TTY'S OUTPUT ENTRY IN TOUT TABLE.

G. FOR ASCII 015 (CR), DETERMINES VALUE FOR THE TTY'S OUTPUT ENTRY IN TOUT TABLE AND SETS COLUMN POINTER = 0.

CALLING SEQUENCE — JSR R0, STARXMT

INPUTS — (SP) - CONTAINS 8 X TTY NUMBER

TTY+3+8XTTY NUMBER - CONTAINS OFFSET IN CC, CF, AND CL LISTS FOR TTY

CC+(TTY+3+8XTTYNUMBER)+1 — CONTAINS CHARACTER COUNT FOR TTY OUTPUT IN CLIST

TTY+1+8XTTY NUMBER - CONTAINS COLUMN POINTER FOR TTY

TTY+2+8XTTY NUMBER - CONTAINS CHARACTER LEFT OVER AFTER LF FOR TTY

TTY+4+8XTTY NUMBER - CONTAINS FLAGS FOR TTY

SEE OUTPUTS FROM 'GETC'

RCSR+8XTTY NUMBER - CONTAINS CARRIER PRESENT FLAG FOR TTY

TCSR+8XTTY NUMBER - CONTAINS READY FLAG FOR TTY

OUTPUTS — SEE INPUTS TO 'GETC'

CC+(TTY+3+8XTTYNUMBER)
TTY+1+8X TTY NUMBER
TTY+2+8XTTY NUMBER
(SEE INPUTS ABOVE)

TCBR+8XTTY NUMBER - CONTAINS CHARACTER TO BE OUTPUT ON TTY

TOUT+3+TTY NUMBER - CONTAINS TOUT ENTRY FOR TTY

ID– U9; XMTT

FUNCTION – "XMTT" USES "CPASS" TO OBTAIN THE NEXT CHARACTER IN THE USER'S BUFFER AREA. IF THE CHARACTER COUNT FOR THE TTY (IDENTIFIED BY i-NODE NUMBER OF TTY'S SPECIAL FILE IN STACK) IS ≥ 50, THE PROCESS IS PUT TO SLEEP. IF NOT, "XMTT" USES "PUTC" TO DETERMINE IF THERE IS AN ENTRY AVAILABLE IN "FREELIST" PORTION OF "CLIST". IF THERE IS, "PUTC" PLACES THE CHARACTER THERE AND ASSIGNS THE LOCATION TO THE TTY PORTION OF "CLIST". IF THERE IS NO LOCATION AVAILABLE IN "FREELIST" PORTION OF "CLIST", THE PROCESS IS PUT TO SLEEP. IF THERE IS A VACANT LOCATION, "STARXMT" IS USED TO ATTEMPT TO OUTPUT THE CHARACTER ON THE TTY. UPON RETURN FROM "STARXMT", THE NEXT CHARACTER IS OBTAINED FROM THE USER'S BUFFER AREA. IF THE BUFFER IS EMPTY, CONTROL IS PASSED BACK TO THE CALLING ROUTINE VIA "CPASS". WHEN THE PROCESS IS AWOKEN BY "AWAKE," IT TRIES AGAIN TO FIND A LOCATION AVAILABLE IN FREELIST AND A CHARACTER COUNT FOR THE TTY OUTPUT < 50 SO IT CAN OUTPUT CHARACTERS.

CALLING SEQUENCE – JMP XMTT

INPUTS – SEE INPUTS FOR "CPASS"

(SP) – CONTAINS i NUMBER OF TTY'S SPECIAL FILE

r1 – CONTAINS CHARACTER TO BE PLACED IN CLIST UPON RETURN FROM 'CPASS'

OUTPUTS – SEE INPUTS FOR "STARXMT" AND "PUTC"

PROCESSOR PRIORITY SET TO FIVE

10 U-6 rw1

FUNCTION: rw1 is called by sysread and syswrite. It puts the buffer pointer (buffer) into u.base and the number of characters (nchars) into u.count. It then finds the i-number of the file to be read by getting the file descriptor in *u.r0 and calling "getf." The i-number is returned in R1

ARGUMENTS —

INPUTS
buffer - buffer pointer
nchar - number of characters
(u.r0) - file descriptor

OUTPUTS
u.base - buffer pointer
u.count - number of characters
R1 - contains the i-number of the file to be read

CALLING SEQUENCE jsr r0, rw1

ID UI-4 intract

FUNCTION: "intract" checks to see if the process owns a quit or interrupt from the typewriter. If it owns a quit, the quit flag is cleared and the T bit (trace trap) of the processor status is set. If the interrupt character is a "del" (177_8), u.intr is checked to see if it is equal to the address "core." If it is, control is transferred to "core." If not, sysexit is taken.

CALLING SEQUENCE br intract

ARGUMENTS —

INPUTS (sp) - contains the instruction R0 is pointing to
u.tty - pointer to buffer of tty in control of the process
(R1)+6 - interrupt character in the control tty's buffer
u.intr - determines handling of interrupts (See. sys intr in the UNIX programmer's manual)

OUTPUTS clock pointer is poped. ?

—If the interrupt char is a quit character
(R1)+6, the interrupt character in the control tty's buffer, is cleared
u.quit is cleared
T bit of PS is set

If the interrupt char is a "del" (interrupt)
(R1)+6 is cleared
control is transferred to "core" if (u.intr)= core

ID— 45;1 mget;

FUNCTION— mget takes the byte number of a byte to be read/written in a file and obtains the physical block number of the block in which it occurs. The file offset for the byte (ie the byte number) is passed by passing a pointer to the offset in u.fofp. The block number for the byte is returned in r1.

Along the way several things can happen:

1. The file is small (less than 8*256. words) and the byte number extends beyond the current size of the file but does not exceed 8*512. In this case mget assigns a new block from the free area of the file device and updates the i-node for the file by adding the physical block number of the new block. And modifying the free storage map.
2. The file is small and the byte number exceeds 8x512. In the case the status of the file changes from small to large. mget sets the large file bit in i.flgs of the i-node. Next an indirect block is assigne to the file. the block pointers in i-node are moved intr the new indirect block and a pointer to the indirect block is put in the inode. Next a new data block is assigned via the large file handling logic, described below.
3. The file is large and the byte number exceeds the current size of the file, but does not exceed the capacity of the highest , indirect block. mget assigns a new file block and adds a newentry to the indirect block.

4. The file is large and the byte number exceeds the current size of the file, and also exceeds the limit of the highest indirect block anew indirect block is assigned from free storage and a pointer to it put in the i-node. Then a new file block is assigned and a pointer to it stored in the new indirect block.

See File Structure write up in the UNIX programmer's manual.

CALLING SEQUENCE— jsr r0,mget

ARGUMENTS —
INPUTS — u.fofp (file offset pointer), inode, u.off (file offset)
OUTPUTS — r1 (physical block number), r2 (internal), r3 (internal), r5 (internal)

10- U7-3 otty

FUNCTION- "otty" opens the console tty for reading or writing The interrupt enable bits are set in the TKS and the TPS. If the console is the first tty opened in this process assign its buffer address to u.ttyp return through "sret."

CALLING SEQUENCE- [conditional or unconditional branch, or jmp] otty

ARGUMENTS-

INPUTS- see inputs for 'sret'

u.ttyp - points to the buffer header for the process control typewriter

(tty+70.) - lower byte of 1st word of header contains the no. of processes that opened the buffer

tty+70. - contains pointer to the header of the console tty buffer

OUTPUTS- u.ttyp - points to the console tty buffer header if it was the 1st tty opened by the process. otherwise points to ?

R5 - points to header of console tty buffer

(R5) - lower byte (no. of processes that opened the buffer) incremented by one.

tks - reader status register interrupt enable bit set, rest of bits zeroed.

tps - punch status register "

see outputs for 'sret'