Bell Laboratories

subject: Study of UNIX

date: September 14, 1972

from: T. R. Bashkow

Messrs.	W. S. Bartlett	Messrs.	J. J. Ludwig
	D. P. Clayton		J. F. Maranzano
	D. H. Copp	Mrs.	G. Pettit
Mmes.	G. J. Hansen	Messrs.	J. E. Ritacco
	J. Hintz		B. A. Tague
Mr.	L. J. Kelly		D. W. Vogel
Miss	R. L. Klein	Mrs.	L. S. Wright

On Tuesday, September 19, at 9:30 a.m. in Room 2A-418 at Murray Hill, I will give a talk on my study of the UNIX operating system. The emphasis will be on the structure, functional components, and internal operation of the system.

MH-8234-TRB-mbh

T. R. Bashkow

T R Bashkow

Copy to
Mr. G. L. Baldwin

Subject: Preliminary Release of
UNIX Implementation Document

Date: 6/20/72

The contents of this document are incomplete and subject to rapid change both in subject matter and organization. The purpose of this release is to make the information it contains available to persons who have an immediate and pressing need. The sections that are included here contain the following information:

Section	Contents
E.0 - E.10	Commented listing of UNIX operating system
E.11	Commented listing of UNIX shell
E.12	Commented listing of UNIX initialization program
F	System Overview
G	Data Base Item Descriptions
H.0 - H.9	Verbal descriptions of UNIX routines.

The verbal descriptions in sections H.0 - H.9 correspond to the listings in E.0 - E.9. However, the routines are listed in alphebetical order in the H sections, rather than in the order they appear in the listings.

J. DeFelice

modifications to UNIX to accomodate the T4002A graphic console

uo Page 1 add

```
gks = 177 --- / graphic input status
gkb = 177 --- / graphic input buffer
gps = 177 --- / graphic output status
gpb = 177 --- / graphic output buffer
```

uo Page 2 add somewhere

```
dspi; 240 / graphic input interrupt vector
```

uo Page 3 add at end of "set up time out routines"

```
mov $ wakdsp, (r0) / time out subroutine for display
```

uo Page 4 add at end of device directory

```
23.
<dsp\0\0\0\0> / T4002A
```

u7 Page 4,5 add to end of iopen list

```
odsp / T4002A
```

add program odsp below

```
odsp: / open T4002A for reading or writing
        mov     $100,*$gks / set interrupt enable on input
        mov     $14,r1 / put "np" in r1 (erase, home)
        jsr     r0,chout / output the char
        mov     $21,r1 / put "dc1" in r1 (turn on joystock)
        jsr     r0,chout / output char
        mov     $37,r1 / put "us" in r1 (alpha mode)
        jsr     r0,chout / output char
        br     sret
```

/Note: a graphic block and buffer like the tty's are not used. May need them when more than 1 disply is added.

u6 Page 1 add at end of readi list

```
rdsp / T4002A
```

add the routine rdsp

```
rdsp: / read from the graphics terminal
        mov     $240,*$ps / set ps to 5
        jsr     r0,getc; 22 / take char off clist and put it in r1
                br 1f / list is empty, go to sleep
        clr     *$ps / clear ps
        jsr     r0,passc / move char to user core
        br      rdsp / get next char
1:
```

```
        mov     r5,-(sp) / save r5
        jsr     r0,sleep; 22 / put input process to sleep
        mov     (sp)+,r5 / restore r5
        br      rdsp / try again
```

add somewhere dspi

```
dspi: / graphic display input interrupt routine
        jsr     r0,setisp / save r1, r2, r3
        mov     *$gkb,r1 / put char in r1
        inc     *$,gks / set reader enable bit
        bic     $!177,r1 / strip char to 7 bits
        jsr     r0,putc; 22 / put char on the clist
                br 1f / if full return
/Note:  char is not echoed and quit
/       (fs) and interrupt (del) char are
/       not processed
        cmp     r1,$4 / char = eot
        beq     1f
        cmp     r1,$12 / char = lf
        beq     1f
        cmpb    cc+22,$15 / are there less than 15 char on the clist?
        blo     retisp / yes, return

1:   jsr r0,wakeup;runq; 22 / wakeup the process that's inputting
     br    retisp / return
```

u6   Page 3   add to bottom of writei list

```
     wdsp / T4002A
```

add routines wdsp, chout, and wakdsp

```
/ write routine for the T4002A graphics console
/ a character at a time is taken out of the graphic
/ instruction buffer and sent over to the T4002A

wdsp: / write on the graphic display
        jsr     r0,cpass / set next char from user buffer area
                         / if none, return to syswrite
        tst     r1 / is the character null
        beq     wdsp / yes, get the next character
        jsr     r0,chout / output the character
        br      wdsp / get next character

chout: / do the actual output of the character
        tstb    *$gps / check for output ready
        bge     chout / wait for ready
1:
        tstb    toutt+12 / check time out
        bne     1b / wait for it to be 0
        movb    r1,*$gpb / output the character
        cmpb    r1,$14 / is char ff (erase, home?)
        beq     1f
        cmpb    r1,$30 / is char "can" (erase)?
        beq     1f
        cmpb    r1,$5 / is char enq (digitize joystock)?
```

```
        beq     2f
        rts     r0
1:
        movb    $30,toutt+12 / put 500 ms delay for erase
        jsr     r0,sleep; 23 / put output process to sleep
        rts     r0
2:
        movb    $2,toutt+12 / put in 20 ms delay for joystick
        rts     r0

/ time out subroutine for display

wakdsp: / wakeup the output process
            jsr r0, wakeup; runq+2; 23
            rts    r0
```

```
/ u0 -- unix

cold = 0
orig = 0   .  / orig = 0. relocatable

rkda = 177412 / disk address reg                    rk03/rk11
rkds = 177400 / driv status reg                     rk03/rk11
rkcs = 177404 / control status reg                  rk03/rk11
rcsr = 174000 / receiver status reg                 dc-11
rcbr = 174002 / receiver buffer reg                 dc-11
tcsr = 174004 / xmtr status reg                     dc-11
tcbr = 174006 / xmtr buffer reg                     dc-11
tcst = 177340 / dec tape control status             tc11/tu56
tccm = 177342 / dec tape command reg                tc11/tu56
tcwc = 177344 /          word count                 tc11/tu56
tcba = 177346 /          bus addr                   tc11/tu56
tcdt = 177350 /          data reg                   tc11/tu56
dcs  = 177460 / drum control status                 rf11/rs11
dae  = 177470 / drum address extension              rf11/rs11
lks  = 177546 / clock status reg                    kw11-l
prs  = 177550 / papertape reader status             pc11
prb  = 177552 /                   buffer            pc11
pps  = 177554 /             punch status            pc11
ppb  = 177556 /             punch buffer            pc11
/lps = 177514   line printer status                 (future)
/lpb = 177516   line printer buffer                 (future)
tks  = 177560 / console read status                 asr-33
tkb  = 177562 /            read buffer              asr-33
tps  = 177564 /           punch status              asr-33
tpb  = 177566 /           punch buffer              asr-33
ps   = 177776 / processor status

halt = 0
wait = 1
rti  = 2

nproc = 16. / number of processes
nfiles = 50.
ntty = 8+1
nbuf = 6
 .if cold / ignored if cold = 0
nbuf = 2
 .endif

core = orig+40000  / specifies beginning of user's core
ecore = core+20000 / specifies end of user's core (4096 words)

    loc
/   0;2 4;4       init by copy
/   4;6 unkni;0   " bus error
/   10;12 fpsym;0 " illg in tr
    14;16 unkni;0 / trace and trap (see Sec. B.1 page  )
    20;22 unkni;0 / trap
    24;26 panic;0 / pwr
    30;32 rtssym;0 / emt
    34;36 sysent;0 / sys
```

```
. = orig+60
  60;62 ttyi;240 / interrupt vector tty in      ; processor level 5
  64;66 ttyo;240 / interrupt vector tty out
  70;72 ppti;240 /                  punch papertape in
  74;76 ppto;240 /                  punch papertape out
  100;102 clock;340 / clock interrupt vector    ; processor level 7
. = orig+200
/       lpto; 240  line printer interrupt    ; processor level 5 (future)
. = orig+204
        drum;300 / drum interrupt         ; processor level 6
. = orig+214
        tape;300 / dec tape interrupt
        disk;300 / rk03 interrupt
. = orig+300
        0*4+trcv; 240; 0*4+txmt; 240 / dc11 input;output interrupt vectors
        1*4+trcv; 240; 1*4+txmt; 240
        2*4+trcv; 240; 2*4+txmt; 240
        3*4+trcv; 240; 3*4+txmt; 240
        4*4+trcv; 240; 4*4+txmt; 240
        5*4+trcv; 240; 5*4+txmt; 240
        6*4+trcv; 240; 6*4+txmt; 240
        7*4+trcv; 240; 7*4+txmt; 240

. = orig+400
/ copy in transfer vectors

        mov     $ecore,sp / put pointer to ecore in the stack pointer
        jsr     r0,copyz; 0; 14 / clear locations 0 to 14 in core
        mov     $4,r0
        clr     r1
        mov     r0,(r1)+ / put value of 4 into location 0
        mov     r0,(r1)+ / put value of 4 into location 2
        mov     $unkni,(r1)+ / put value of unkni into location 4;
                             / time out, bus error
        clr     (r1)+ / put value of 0 into location 6
        mov     $fpsym,(r1)+ / put value of fpsym into location 10
        clr     (r1)+ / put value of 0 into location 12
/ clear core
        .if cold / ignored if cold = 0
        halt / halt before initializing rf file system; user has
             / last chance to reconsider
        .endif

        jsr     r0,copyz; systm; ecore / clear locations systm to ecore
        mov     $s.chrgt+2,clockp / intialize clockp
/ allocate tty buffers; see H.0 for description
        mov     $buffer,r0
        mov     $tty+6,r1
1:
        mov     r0,(r1)
        add     $140.,r0 / tty buffers are 140. bytes long
        add     $8,r1
        cmp     r1,$tty+[ntty*8] / has a buffer been assigned for each tty
        blo     1b

/ allocate disk buffers; see H.0 for description
```

```
        mov     $bufp,r1
1:
        mov     r0,(r1)+
        add     $8,r0
        mov     r0,-2(r0)              / bus address
        mov     $-256.,-4(r0)          / word count
        add     $512.,r0               / buffer space
        cmp     r1,$bufp+nbuf+nbuf
        blo     1b
        mov     $sb0,(r1)+             / I/O queue entry drum
        mov     $sb1,(r1)+ / I/O queue entry disk (mounted device)
        mov     $swp,(r1)+ / I/O queue entry core image being swapped
        mov     $[systm-inode]\/2,sb0+4 / sets up initial buffers per
                                        / format given in
        mov     $systm,sb0+6 / memory map
        mov     $-512.,sb1+4
        mov     $mount,sb1+6
        mov     $user,swp+6

/ set devices to interrupt

        mov     $100,*$lks / put 100 into clock status register;
                           / enables clock interrupt

/ set up time out subroutines

        mov     $touts,r0
        mov     $startty,(r0)+ / if toutt = 0 call startty
        mov     $pptito,(r0)+ / if toutt+1 = 0 call pptito
        tst     (r0)+ / add 2 to r0
        mov     $ntty-1,r1
1:
        mov     $xmtto,(r0)+ / if toutt+2 thru toutt+2+ntty=0 call xmtto
        dec     r1
        bne     1b

/ free all character blocks; see H.0 for description

        mov     $510.,r2
        mov     $-1,r1
1:
        jsr     r0,put
        sub     $2,r2
        bgt     1b

/ set up drum swap addresses; see H.0 for description

        mov     $1024.-64.,r1 / highest drum address; high 64 blks allocated
                              / to UNIX
        mov     $p.dska,r2 / p.dska contains disk addresses for processes
1:
        sub     $17.,r1 / 17 blocks per process
        mov     r1,(r2)+
        cmp     r2,$p.dska+nproc+nproc
        bne     1b
```

```
/ free rest of drum

        .if cold
        mov     $128.,systm / initialize word 1 of drum superblock image;
                            / number of bytes in free storage map=128.
        mov     $64.,systm+2+128. / init. wd 66. of superblock image; # of
                                  / bytes in i-node map=64.
1:
        dec     r1 / r1=687.,...,34.
        jsr     r0,free / free block 'r1', i.e., set bit 'r1' in free
                        / storage map in core
        cmp     r1,$34. / first drum address not in i list
        bgt     1b / if block 34 has been freed, zero i list

/ zero i list

1:
        dec     r0 / r0 = 33.,...,1
        jsr     r0,clear / zero block 'r1' on fixed head dis
        tst     r1
        bgt     1b / if blocks 33,...,1 have all been zeroed, done.
        .endif

/ make current program a user

        mov     $41.,r0 / rootdir set to 41 and never changed
        mov     r0,rootdir / rootdir is i-number of root directory
        mov     r0,u.cdir / u.cdir is i-number of process current directory
        mov     $1,r0
        movb    r0,u.uno / set process table index for this process to 1
        mov     r0,mpid / initialize mpid to 1
        mov     r0,p.pid / p.pid identifies process
        movb    r0,p.stat / process status = 1 i.e., active
                          /                = 0 free
        .if cold          /                = 2 waiting for a child to die
                          /                = 3 terminated but not yet waited
                          /                    for

/ initialize inodes for special files (inodes 1 to 40.)

        mov     $40.,r1 / set r1=i-node-number 40.
1:
        jsr     r0,iget / read i-node 'r1' from disk into inode area of
                        / core and write modified inode out (if any)
        mov     $100017,i.flgs / set flags in core image of inode to indi-
                               / cate allocated, read (owner, non-owner),
                               / write (owner, non-owner)
        movb    $1,i.nlks / set no. of links = 1
        movb    $1,i.uid / set user id of owner = 1
        jsr     r0,setimod / set imod=1 to indicate i-node modified, also
                           / stuff time of modification into i-node
        dec     r1 / next i-node no. = present i-node no.-1
        bgt     1b / has i-node 1 been initialized; no, branch

/ initialize i-nodes r1.,...,47. and write the root device, binary, etc.,
/ directories onto fixed head disk.  user temporary, initialization prog.
```

```
        mov     $idata,r0 / r0=base addr. of assembled directories.
        mov     $u.off,u.fofp / pointer to u.off in u.fofp (holds file
                              / offset)
1:
        mov     (r0)+,r1/r1=41.,...,47; "0" in the assembled directory
                        / header signals last
        beq     1f      / assembled directory has been written onto drum
        jsr     r0,imap / locate the inode map bit for i-node 'r1'
        bisb    mq,(r2) / set the bit to indicate the i-node is not
                        / available
        jsr     r0,iget / read inode 'r1' from disk into inode area of
                        / core and write modified i-node on drum (if any)
        mov     (r0)+,i.flgs / set flags in core image of inode from
                             / assembled directories header
        movb    (r0)+.i.nlks / set no. of links from header
        movb    (r0)+,i.uid / set user id of owner from header
        jsr     r0,setimod / set imod=1 to indicate inode modified: also,
                           / stuff time of modification into i-node
        mov     (r0)+,u.count / set byte count for write call equal to
                              / size of directory
        mov     r0,u.base / set buffer address for write to top of directory
        clr     u.off / clear file offset used in 'seek' and 'tell'
        add     u.count,r0 / r0 points to the header of the next directory
        jsr     r0,writei / write the directory and i-node onto drum
        br      1b / do next directory
        .endif

/ next 2 instructions not executed during cold boot.
        bis     $2000,sb0 / sb0 I/O queue entry for superblock on drum;
                          / set bit 10 to 1
        jsr     r0,ppoke / read drum superblock
1:
        tstb    sb0+1 / has I/O request been honored (for drum)?
        bne     1b / no, continue to idle.

1:
        decb    sysflg / mormally sysflag=0, indicates executing in system
        sys     exec; 2f; 1f / generates trap interrupt; trap vector =
                             / sysent; 0
        br      panic / execute file/etc/init

1:
        2f;0          this is file #47 listed on E0,9 (on coldboot) see E0,10
2:
        </etc/init\0> / UNIX looks for strings term, noted by nul\0

panic:
        clr     ps
1:
        dec     $0
        bne     1b
        dec     $5
        bne     1b
        jmp     *$173700 / rom loader address
```

```
rtssym:
        mov     r0,-(sp)
        mov     r1,-(sp)
        mov     4(sp),r0
        mov     -(r0),r0
        bic     $!7,r0
        asl     r0
        jmp     *1f(r0)
1:
        0f;1f;2f;3f;4f;5f;badrts;7f
0:
        mov     2(sp),r0
        br      1f
2:
        mov     r2,r1
        br      1f
3:
        mov     r3,r1
        br      1f
4:
        mov     r4,r1
        br      1f
5:
        mov      r5,r1
        br      1f
7:
        mov     8.(sp),r1
1:
        cmp     r1,$core
        blo     badrts
        cmp     r1,$ecore
        bhis    badrts
        bit     $1,r1
        bne     badrts
        tst     (r1)
        beq     badrts
        add     $1f,r0
        mov     r0,4(sp)
        mov     (sp)+,r1
        mov     (sp)+,r0
        rti
1:
        rts     r0
        rts     r1
        rts     r2
        rts     r3
        rts     r4
        rts     r5
        rts     sp
        rts     pc

badrts:
        mov     (sp)+,r1
        mov     (sp)+,r0

rpsym:
```

```
        jmp     unkni

        .if cold

idata:

/ root

        41.
        140016
        .byte 7,1
        9f-.-2
        41.
        <..\0\0\0\0\0\0>
        41.
        <.\0\0\0\0\0\0\0>
        42.
        <dev\0\0\0\0\0>
        43.
        <bin\0\0\0\0\0>
        44.
        <etc\0\0\0\0\0>
        45.
        <usr\0\0\0\0\0>
        46.
        <tmp\0\0\0\0\0>
9:

/ device directory

        42.
        140016
        .byte 2,1
        9f-.-2
        41.
        <..\0\0\0\0\0\0>
        42.
        <.\0\0\0\0\0\0\0>
        01.
        <tty\0\0\0\0\0>
        02.
        <ppt\0\0\0\0\0>
        03.
        <mem\0\0\0\0\0>
        04.
        <rf0\0\0\0\0\0>
        05.
        <rk0\0\0\0\0\0>
        06.
        <tap0\0\0\0\0>
        07.
        <tap1\0\0\0\0>
        08.
        <tap2\0\0\0\0>
        09.
        <tap3\0\0\0\0>
```

```
        10.
        <tap4\0\0\0\0>
        11.
        <tap5\0\0\0\0>
        12.
        <tap6\0\0\0\0>
        13.
        <tap7\0\0\0\0>
        14.
        <tty0\0\0\0\0>
        15.
        <tty1\0\0\0\0>
        16.
        <tty2\0\0\0\0>
        17.
        <tty3\0\0\0\0>
        18.
        <tty4\0\0\0\0>
        19.
        <tty5\0\0\0\0>
        20.
        <tty6\0\0\0\0>
        21.
        <tty7\0\0\0\0>
        22.
        <lpr\0\0\0\0\0>
        01.
        <tty8\0\0\0\0> / really tty
9:

/ binary directory

        43.
        140016
        .byte 2,3
        9f-.-2
        41.
        <..\0\0\0\0\0\0>
        43.
        <.\0\0\0\0\0\0\0>
9:

/ etcetra directory

        44.
        140016
        .byte 2,3
        9f-.-2
        41.
        <..\0\0\0\0\0\0>
        44.
        <.\0\0\0\0\0\0\0>
        47.
        <init\0\0\0\0>
9:
```

```
/ user directory

        45.
        140016
        .byte 2,1
        9f-.-2
        41.
        <..\0\0\0\0\0\0>
        45.
        <.\0\0\0\0\0\0\0>
9:

/ temporary directory

        46.
        140017
        .byte 2,1
        9f-.-2
        41.
        <..\0\0\0\0\0\0>
        46.
        <.\0\0\0\0\0\0\0>
9:

/ initialization program

        47.
        100036
        .byte 1,3
        9f-.-2
8:
        sys     break; 0
        sys     open; 6f-8b+core; 0
        mov     r0,r1
        sys     seek; 65.; 0
1:
        mov     r1,r0
        sys     read; 9f-8b+core; 512.
        mov     9f,r5               / size
        beq     1f
        sys     creat; 9f-8b+core+4; 0
        mov     r0,r2
        movb    9f+2,0f
        sys     chmod; 9f-8b+core+4; 0:..
        movb    9f+3,0f
        sys     chown; 9f-8b+core+4; 0:..
2:
        tst     r5
        beq     2f
        mov     r1,r0
        sys     read; 9f-8b+core; 512.
        mov     $512.,0f
        cmp     r5,$512.
        bhi     3f
        mov     r5,0f
3:
```

```
        mov     r2,r0
        sys     write; 9f-8b+core; 0:..
        sub     r0,r5
        br      2b
2:
        mov     r2,r0
        sys     close
        br      1b
1:
        mov     r1,r0
        sys     close
        sys     exec; 5f-8b+core; 4f-8b+core
        sys     exit
4:
        5f-8b+core; 0
5:
        </eet/init\0>
6:
        </dev/tap0\0>
        .even
9:

/ end of initialization data

        0

        .endif
```

etc

this file #47 is init program that sets up user console system. ~~Then~~ ~~replaced~~ This init during execution of this init on cold boot!!

```
/ u1 -- unix

unkni: / used for all system calls
sysent:
        incb    sysflg / indicate a system routine is
        beq     1f / in progress
        jmp     panic / called if trap inside system
1:
        mov     $s.syst+2,clockp
        mov     r0,-(sp) / save user registers
        mov     sp,u.r0 / pointer to bottom of users stack in u.r0
        mov     r1,-(sp)
        mov     r2,-(sp)
        mov     r3,-(sp)
        mov     r4,-(sp)
        mov     r5,-(sp)
        mov     ac,-(sp) / "accumulator" register for extended
                         / arithmetic unit
        mov     mq,-(sp) / "multiplier quotient" register for the
                         / extended arithmetic unit
        mov     sc,-(sp) / "step count" register for the extended
                         / arithmetic unit
        mov     sp,u.sp / u.sp points to top of users stack
        mov     18.(sp),r0 / store pc in r0
        mov     -(r0),r0 / sys inst in r0      10400xxx
        sub     $sys,r0 / get xxx code
        asl     r0 / multiply by 2 to jump indirect in bytes
        cmp     r0,$2f-1f / limit of table (35) exceeded
        bhis    badsys / yes, bad system call
        bic     $341,20.(sp) / set users processor priority to 0 and clear
                             / carry bit
        jmp     *1f(r0) / jump indirect thru table of addresses
                        / to proper system routine.
1:
        sysrele / 0
        sysexit / 1
        sysfork / 2
        sysread / 3
        syswrite / 4
        sysopen / 5
        sysclose / 6
        syswait / 7
        syscreat / 8
        syslink / 9
        sysunlink / 10
        sysexec / 11
        syschdir / 12
        systime / 13
        sysmkdir / 14
        syschmod / 15
        syschown / 16
        sysbreak / 17
        sysstat / 18
        sysseek / 19
        systell / 20
```

```
        sysmount / 21
        sysumount / 22
        syssetuid / 23
        sysgetuid / 24
        sysstime / 25
        sysquit / 26
        sysintr / 27
        sysfstat / 28
        sysemt / 29
        sysmdate / 30
        sysstty / 31
        sysgtty / 32
        sysilgins / 33
2:

error:
        mov     u.sp,r1
        bis     $1,20.(r1) / set c bit in processor status word below
                           / users stack

sysret:
        tstb    u.bsys / is a process about to be terminated because
        bne     sysexit / of an error? yes, go to sysexit
        mov     u.sp,sp / no point stack to users stack
        clr     r1 / zero r1 to check last mentioned i-node
        jsr     r0,iget / if last mentioned i-node has been modified
                        / it is written out
        tstb    smod / has the super block been modified
        beq     1f / no, 1f
        clrb    smod / yes, clear smod
        bis     $1000,sb0 / set write bit in I/O queue for super block
                          / output
        jsr     r0,ppoke / write out modified super block to disk
1:
        tstb    mmod / has the super block for the dismountable file
                     / system
        beq     1f / been modified?  no, 1f
        clrb    mmod / yes, clear mmod
        movb    mntd,sb1 / set the I/O queue
        bis     $1000,sb1 / set write bit in I/O queue for detached sb
        jsr     r0,ppoke / write it out to its device
1:
        tstb    uquant / is the time quantum 0?
        bne     1f / no, don't swap it out

sysrele:
        jsr     r0,tswap / yes, swap it out
1:
        mov     (sp)+,sc / restore user registers
        mov     (sp)+,mq
        mov     (sp)+,ac
        mov     (sp)+,r5
        mov     (sp)+,r4
        mov     (sp)+,r3
        mov     (sp)+,r2
        mov     (sp)+,r1
```

```
        mov     (sp)+,r0
        mov     $s.chrgt+2,clockp
        decb    sysflg / turn system flag off
        jsr     r0,isintr / is there an interrupt from the user
                br intract / yes, output gets flushed, take interrupt
                        / action
        rti     / no return from interrupt

badsys:
        incb    u.bsys / turn on the user's bad system flag
        mov     $3f,u.namep / point u.namep to "core\0\0"
        jsr     r0,namei / get the i-number for the core image file
                br 1f / error
        neg     r1 / negate the i-number to open the core image file
                   / for writing
        jsr     r0,iopen / open the core image file
        jsr     r0,itrunc / free all associated blocks
        br      2f
1:
        mov     $17,r1 / put i-node mode (17) in r1
        jsr     r0,maknod / make an i-node
        mov     u.dirbuf,r1 / put i-nodes number in r1
2:
        mov     $core,u.base / move address core to u.base
        mov     $ecore-core,u.count / put the byte count in u.count
        mov     $u.off,u.fofp / more user offset to u.fofp
        clr     u.off / clear user offset
        jsr     r0,writei / write out the core image to the user
        mov     $user,u.base / pt. u.base to user
        mov     $64.,u.count / u.count = 64
        jsr     r0,writei / write out all the user parameters
        neg     r1 / make i-number positive
        jsr     r0,iclose / close the core image file
        br      sysexit /
3:
        <core\0\0>

sysexit: / terminate process
        clr     u.intr / clear interrupt control word
        clr     r1 / clear r1
1: / r1 has file descriptor (index to u.fp list)  Search the whole list
        jsr     r0,fclose / close all files the process opened
                br .+2 / ignore error return
        inc     r1 / increment file descriptor
        cmp     r1,$10. / end of u.fp list?
        blt     1b / no, go back
        movb    u.uno,r1 / yes, move dying process's number to r1
        clrb     p.stat-1(r1) / free the process
        asl     r1 / use r1 for index into the below tables
        mov     p.pid-2(r1),r3 / move dying process's name to r3
        mov     p.ppid-2(r1),r4 / move its parents name to r4
        clr     r2
        clr     r5 / initialize reg
1: / find children of this dying process, if they are zombies, free them
        add     $2,r2 / search parent process table for dying process's name
        cmp     p.ppid-2(r2),r3 / found it?
```

```
        bne     3f / no
        asr     r2 / yes, it is a parent
        cmpb    p.stat-1(r2),$3 / is the child of this dying process a
                                / zombie
        bne     2f / no
        clrb    p.stat-1(r2) / yes, free the child process
2:
        asl     r2
3: / search the process name table for the dying process's parent
        cmp     p.pid-2(r2),r4 / found it?
        bne     3f / no
        mov     r2,r5 / yes, put index to p.pid table (parents
                      / process # x2) in r5
3:
        cmp     r2,$nproc+nproc / has whole table been searched?
        blt     1b / no, go back
        mov     r5,r1 / yes, r1 now has parents process # x2
        beq     2f / no parent has been found.  The process just dies
        asr     r1 / set up index to p.stat
        movb    p.stat-1(r1),r2 / move status of parent to r2
        beq     2f / if its been freed, 2f
        cmp     r2,$3 / is parent a zombie?
        beq     2f / yes, 2f
        movb    u.uno,r3 / move dying process's number to r3
        movb    $3,p.stat-1(r3) / make the process a zombie
        cmp     r2,$2 / is the parent waiting for this child to die
        bne     2f / yes, notify parent not to wait any more
        decb    p.stat-1(r1) / awaken it by putting it (parent)
        mov     $runq+4,r2 / on the runq
        jsr     r0, putlu
2: / the process dies
        clrb    u.uno / put zero as the process number, so "swap" will
        jsr     r0,swap / overwrite process with another process
        0       / and thereby kill it; halt?

intract: / interrupt action
        cmp     *(sp),$rti / are you in a clock interrupt?
        bne     1f / no, 1f
        cmp     (sp)+,(sp)+ / pop clock pointer
1: / now in user area
        mov     r1,-(sp) / save r1
        mov     u.ttyp,r1 / pointer to tty buffer in control to r1
        cmpb    6(r1),$177 / is the interrupt char equal to "del"
        beq     1f / yes, 1f
        clrb    6(r1) / no, clear the byte (must be a quit character)
        mov     (sp)+,r1 / restore r1
        clr     u.quit / clear quit flag
        bis     $20,2(sp) / set trace for quit (sets t bit of ps-trace trap)
        rti               / return from interrupt
1: / interrupt char = del
        clrb    6(r1) / clear the interrupt byte in the buffer
        mov     (sp)+,r1 / restore r1
        cmp     u.intr,$core / should control be transferred to loc core?
        blo     1f
        jmp     *u.intr / user to do rti yes, transfer to loc core
1:
```

```
        sys     1 / exit

syswait: / wait for a process to die
        movb    u.uno,r1 / put parents process number in r1
        asl     r1 / x2 to get index into p.pid table
        mov     p.pid-2(r1),r1 / get the name of this process
        clr     r2
        clr     r3 / initialize reg 3
1:
        add     $2,r2 / use r2 for index into p.ppid table / search table
                       / of parent processes for this process name
        cmp     p.ppid-2(r2),r1 / r2 will contain the childs process number
        bne     3f / branch if no match of parent process name
        inc     r3 / yes, a match, r3 indicates number of children
        asr     r2 / r2/2 to get index to p.stat table
        cmpb    p.stat-1(r2),$3 / is the child process a zombie?
        bne     2f / no, skip it
        clrb    p.stat-1(r2) / yes, free it
        asl     r2 / r2x2 to get index into p.pid table
        mov     p.pid-2(r2),*u.r0 / put childs process name in (u.r0)
        br      sysret1 / return cause child is dead
2:
        asl     r2 / r2x2 to get index into p.ppid table
3:
        cmp     r2,$nproc+nproc / have all processes been checked?
        blt     1b / no, continue search
        tst     r3 / one gets here if there are no children or children
                   / that are still active
        beq     error1 / there are no children, error
        movb    u.uno,r1 / there are children so put parent process number
                         / in r1
        incb    p.stat-1(r1) / it is waiting for other children to die
        jsr     r0,swap / swap it out, because it's waiting
        br      syswait / wait on next process

error1:
        jmp     error / see 'error' routine
sysret1:
        jmp     sysret / see 'sysret' routine

sysfork: / create a new process
        clr     r1
1: / search p.stat table for unused process number
        inc     r1
        tstb    p.stat-1(r1) / is process active, unused, dead
        beq     1f / it's unused so branch
        cmp     r1,$nproc / all processes checked
        blt     1b / no, branch back
        add     $2,18.(sp) / add 2 to pc when trap occured, points
                           / to old process return
        br      error1 / no room for a new process
1:
        movb    u.uno,-(sp) / save parent process number
        movb    r1,u.uno / set child process number to r1
        incb    p.stat-1(r1) / set p.stat entry for child process to
                             / active status
```

```
        mov     u.ttyp,r2 / put pointer to parent process' control tty
                          / buffer in r2
        beq     2f / branch, if no such tty assigned
        clrb    6(r2) / clear interrupt character in tty buffer
2:
        mov     $runq+4,r2
        jsr     r0,putlu / put child process on lowest priority run queue
        asl     r1 / multiply r1 by 2 to get index into p.pid table
        inc     mpid / increment m.pid; get a new process name
        mov     mpid,p.pid-2(r1) / put new process name in child process'
                                 / name slot
        movb    (sp),r2 / put parent process number in r2
        asl     r2 / multiply by 2 to get index into below tables
        mov     p.pid-2(r2),r2 / get process name of parent process
        mov     r2,p.ppid-2(r1) / put parent process name in parent
                                / process slot for child
        mov     r2,*u.r0 / put parent process name on stack at location
                         / where r0 was saved
        mov     $sysret1,-(sp) /
        mov     sp,u.usp / contents of sp at the time when user is
                         / swapped out
        mov     $sstack,sp / point sp to swapping stack space
        jsr     r0,wswap / put child process out on drum
        jsr     r0,unpack / unpack user stack
        mov     u.usp,sp / restore user stack pointer
        tst     (sp)+ / bump stack pointer
        movb    (sp)+,u.uno / put parent process number in u.uno
        mov     mpid,*u.r0 / put child process name on stack where r0
                           / was saved
        add     $2,18.(sp) / add 2 to pc on stack; gives parent
                           / process return
        clr     r1
1: / search u.fp list to find the files opened by the parent process
        movb    u.fp(r1),r2 / get an open file for this process
        beq     2f / file has not been opened by parent, so branch
        asl     r2 / multiply by 8
        asl     r2 / to get index into fsp table
        asl     r2
        incb    fsp-2(r2) / increment number of processes using file,
                          / because child will now be using this file
2:
        inc     r1 / get next open file
        cmp     r1,$10. / 10. files is the maximum number which can be
                        / opened
        blt     1b / check next entry
        br      sysret1

sysread:
        jsr     r0,rw1 / get i-number of file to be read into r1
        tst     r1 / negative i-number?
        ble     error1 / yes, error 1 to read it should be positive
        jsr     r0,readi / read data into core
        br      1f

syswrite:
        jsr     r0,rw1 / get i-number in r1 of file to write
```

```
        tst     r1 / positive i-number ?
        bge     error1 / yes, error 1 negative i-number means write
        neg     r1 / make it positive
        jsr     r0,writei / write data
1:
        mov     u.nread,*u.r0 / put no. of bytes transferred into (u.r0)
        br      sysret1

rw1:
        jsr     r0,arg; u.base / get buffer pointer
        jsr     r0,arg; u.count / get no. of characters
        mov     *u.r0,r1 / put file descriptor (index to u.fp table) in r1
        jsr     r0,getf / get i-number of the file in r1
        rts     r0

sysopen:
        jsr     r0,arg2 / get sys args into u.namep and on stack
        jsr     r0,namei / i-number of file in r1
        br      error2 / file not found
        tst     (sp) / is mode = 0 (2nd arg of call; 0 means, open for read)
        beq     1f / yes, leave i-number positive
        neg     r1 / open for writing so make i-number negative
1:
        jsr     r0,iopen / open file whose i-number is in r1
        tst     (sp)+ / pop the stack and test the mode
        beq     op1 / is open for read op1

op0:
        neg     r1 / make i-number positive if open for writing
op1:
        clr     r2 / clear registers
        clr     r3
1: / scan the list of entries in fsp table
        tstb    u.fp(r2) / test the entry in the u.fp list
        beq     1f / if byte in list is 0 branch
        inc     r2 / bump r2 so next byte can be checked
        cmp     r2,$10. / reached end of list?
        blt     1b / no, go back
        br      error2 / yes, error (no files open)
1:
        tst     fsp(r3) / scan fsp entries
        beq     1f / if 0 branch
        add     $8.,r3 / add 8 to r3 to bump it to next entry mfsp table
        cmp     r3,$[nfiles*8.] / done scanning
        blt     1b / no, back
        br      error2 / yes, error
1: / r2 has index to u.fp list; r3, has index to fsp table
        mov     r1,fsp(r3) / put i-number of open file into next available
        mov     cdev,fsp+2(r3) / entry in fsp table, put # of device in
                               / next word
        clr     fsp+4(r3)
        clr     fsp+6(r3) / clear the next two words
        asr     r3
        asr     r3 / divide by 8 to get number of the fsp entry-1
        asr     r3
        inc     r3 / add 1 to get fsp entry number
```

```
        movb    r3,u.fp(r2) / move entry number into next available slot
                            / in u.fp list
        mov     r2,*u.r0 / move index to u.fp list into r0 loc on stack
        br      sysret2

error2:
        jmp     error / see 'error' routine
sysret2:
        jmp     sysret / see 'sysret' routine

syscreat: / name; mode
        jsr     r0,arg2 / put file name in u.namep put mode on stack
        jsr     r0,namei / get the i-number
                br  2f / if file doesn't exist 2f
        neg     r1 / if file already exists make i-number negative
                   / (open for writing)
        jsr     r0,iopen /
        jsr     r0,itrunc / truncate to 0 length
        br      op0
2: / file doesn't exist
        mov     (sp)+,r1 / put the mode in r1
        bic     $!377,r1 / clear upper byte
        jsr     r0,maknod / make an i-node for this file
        mov     u.dirbuf,r1 / put i-number for this new file in r1
        br      op0 / open the file

sysmkdir: / make a directory
        jsr     r0,arg2 / point u.namep to the file name
        jsr     r0,namei / get the i-number
                br .+4 / if file not found branch around error
        br      error2 / directory already exists (error)
        tstb    u.uid / is user the super user
        bne     error2 / no, not allowed
        mov     (sp)+,r1 / put the mode in r1
        bic     $!317,r1 / all but su and ex
        bis     $40000,r1 / directory flag
        jsr     r0,maknod / make the i-node for the directory
        br      sysret2 /

sysclose: / close the file
        mov     *u.r0,r1 / move index to u.fp list into r1
        jsr     r0,fclose / close the file
                br error2 / unknown file descriptor
        br      sysret2

sysemt:
        jsr     r0,arg; 30 / put the argument of the sysemt call in loc 30
        cmp     30,$core / was the argument a lower address than core
        blo     1f / yes, rtssym
        cmp     30,$ecore / no, was it higher than "core" and less than
                          / "ecore"
        blo     2f / yes, sysret2
1:
        mov     $rtssym,30
2:
        br      sysret2
```

```
sysilgins: / calculate proper illegal instruction trap address
        jsr     r0,arg; 10 / take address from sysilgins call     , put
                           / it in loc 8.,
        cmp     10,$core / making it the illegal instruction trap address
        blo     1f / is the address a user core address?  yes, go to 2f
        cmp     10,$ecore
        blo     2f
1:
        mov     $fpsym,10 / no, make 'fpsum' the illegal instruction trap
                          / address for the system
2:
        br      sysret2 / return to the caller via 'sysret'

sysmdate: / change the modification time of a file
        jsr     r0,arg; u.namep / point u.namep to the file name
        jsr     r0,namei / get its i-number
                br error2 / no, such file
        jsr     r0,iget / get i-node into core
        cmpb    u.uid,i.uid / is user same as owner
        beq     1f / yes
        tstb    u.uid / no, is user the super user
        bne     error2 / no, error
1:
        jsr     r0,setimod / fill in modification data, time etc.
        mov     4(sp),i.mtim / move present time to
        mov     2(sp),i.mtim+2 / modification time
        br      sysret2

sysstty: / set mode of typewriter; 3 consequtive word arguments
        jsr     r0,gtty / r1 will have offset to tty block, r2 has source
        mov     r2,-(sp)
        mov     r1,-(sp) / put r1 and r2 on the stack
1: / flush the clist wait till typewriter is quiescent
        mov     (sp),r1 / restore r1 to tty block offset
        movb    tty+3(r1),0f / put cc offset into getc argument
        mov     $240,*$ps / set processor priority to 5
        jsr     r0,getc; 0:../ put character from clist in r1
                br .+4 / list empty, skip branch
        br      1b / get another character until list is empty
        mov     0b,r1 / move cc offset to r1
        inc     r1 / bump it for output clist
        tstb    cc(r1) / is it 0
        beq     1f / yes, no characters to output
        mov     r1,0f / no, put offset in sleep arg
        jsr     r0,sleep; 0:.. / put tty output process to sleep
        br      1b / try to calm it down again
1:
        mov     (sp)+,r1
        mov     (sp)+,r2 / restore registers
        mov     (r2)+,r3 / put reader control status in r3
        beq     1f / if 0, 1f
        mov     r3,rcsr(r1) / move r.c. status to reader control status
                            / register
1:
        mov     (r2)+,r3 / move pointer control status to r3
```

```
        beq     1f / if 0 1f
        mov     r3,tcsr(r1) / move p.c. status to printer control status reg
1:
        mov     (r2)+,tty+4(r1) / move to flag byte of tty block
        jmp      sysret2 / return to user

sysgtty: / get status of typewriter; 3 conseqtive word arguments
        jsr     r0,gtty / r1 will have offset to tty block, r2 has
                        / destination
        mov     rcsr(r1),(r2)+ / put reader control status in 1st word
                               / of dest
        mov     tcsr(r1),(r2)+ / put printer control status in 2nd word
                               / of dest
        mov     tty+4(r1),(r2)+ / put mode in 3rd word
        jmp     sysret2 / return to user

gtty:
        jsr     r0,arg; u.off / put first arg in u.off
        mov     *u.r0,r1 / put file descriptor in r1
        jsr     r0,getf / get the i-number of the file
        tst     r1 / is it open for reading
        bgt     1f / yes
        neg     r1 / no, i-number is negative, so make it positive
1:
        sub     $14.,r1 / get i-number of tty0
        cmp     r1,$ntty-1 / is there such a typewriter
        bhis    error9 / no, error
        asl     r1 / 0%2
        asl     r1 / 0%4 / yes
        asl     r1 / 0%8 / multiply by 8 so r1 points to tty block
        mov     u.off,r2 / put argument in r2
        rts     r0 / return
```

```
/ u2 -- unix

syslink: / name1, name2
        jsr     r0,arg2 / u.namep has 1st arg u.off has 2nd
        jsr     r0,namei / find the i-number associated with the 1st
                        / path name
        br      error9 / cannot be found
        jsr     r0,iget / get the i-node into core
        mov     (sp)+,u.namep / u.namep points to 2nd name
        mov     r1,-(sp) / put i-number of name1 on the stack (a link
                        / to this file is to be created)
        mov     cdev,-(sp) / put i-nodes device on the stack
        jsr     r0,isdir / is it a directory
        jsr     r0,namei / no, get i-number of name2
                br .+4  / not found so r1-i-number of current directory
                        /               ii = i-number of current directory
        br      error9 / file already exists., error
        cmp     (sp)+,cdev / u.dirp now points to end of current dir
        bne     error9
        mov     (sp),u.dirbuf / i-number of name1 into u.dirbuf
        jsr     r0,mkdir / make directory entry for name2 in current
                        / directory
        mov     (sp)+,r1 / r1 has i-number of name1
        jsr     r0,iget / get i-node into core
        incb    i.nlks / add 1 to its number of links
        jsr     r0,setimod / set the i-node modified flag

sysret9:
        jmp     sysret / see 'sysret' routine
error9:
        jmp     error / see 'error' routine

isdir: / if the i-node whose i-number is in r1 is a directory there is an
       / error unless super user made the call
        tstb    u.uid / super user
        beq     1f / yes, don't care
        mov     ii,-(sp) / put current i-number on stack
        jsr     r0,iget / get i-node into core (i-number in r1)
        bit     $40000,i.flgs / is it a directory
        bne     error9 / yes, error
        mov     (sp)+,r1 / no, put current i-number in r1 (ii)
        jsr     r0,iget / get it back in
1:
        rts     r0

sysunlink: / name - remove link name
        jsr     r0,arg; u.namep / u.namep points to name
        jsr     r0,namei / find the i-number associated with the path name
                br error9 / not found
        mov     r1,-(sp) / put its i-number on the stack
        jsr     r0,isdir / is it a directory
        clr     u.dirbuf / no, clear the location that will get written
                        / into the i-number portion of the entry
        sub     $10.,u.off / move u.off back 1 directory entry
        jsr     r0,wdir / free the directory entry
```

```
        mov     (sp)+,r1 / get i-number back
        jsr     r0,iget / get i-node
        jsr     r0,setimod / set modified flag
        decb    i.nlks / decrement the number of links
        bgt     sysret9 / if this was not the last link to file return
        jsr     r0,anyi / if it was, see if anyone has it open.  Then
                        / free contents of file and destroy it.
        br      sysret9

mkdir:
        jsr     r0,copyz; u.dirbuf+2; u.dirbuf+10. / clear this
        mov     u.namep,r2 / r2 points to name of directory entry
        mov     $u.dirbuf+2,r3 / r3 points to u.dirbuf+2
1: / put characters in the directory name in u.dirbuf+2 - u.dirbuf+10
        movb     (r2)+,r1 / move character in name to r1
        beq     1f / if null, done
        cmp     r1,$'/ / is it a "/"?
        beq     error9 / yes, error
        cmp     r3,$u.dirbuf+10. / have we reached the last slot for
                                 / a char?
        beq     1b / yes, go back
        movb    r1,(r3)+ / no, put the char in the u.dirbuf
        br      1b / get next char
1:
        mov     u.dirp,u.off / pointer to empty current directory slot to
                             / u.off

wdir:
        mov     $u.dirbuf,u.base / u.base points to created file name
        mov     $10.,u.count / u.count = 10
        mov     ii,r1 / r1 has i-number of current directory
        jsr     r0,access; 1 / get i-node and set its file up for writing
        jsr     r0,writei / write into directory
        rts     r0

sysexec:
        jsr     r0,arg2 / arg0 in u.namep,arg1 on top of stack
        jsr     r0,namei / namei returns i-number of file named in
                         / sysexec call in r1
                br error9
        jsr     r0,iget / get i-node for file to be executed
        bit     $20,i.flgs / is file executable
        beq     error9
        jsr     r0,iopen / gets i-node for file with i-number given in
                         / r1 (opens file)
        bit     $40,i.flgs / test user id on execution bit
        beq     1f
        tstb    u.uid / test user id
        beq     1f / super user
        movb    i.uid,u.uid / put user id of owner of file as process
                            / user id
1:
        mov     (sp)+,r5 / r5 now contains address of list of pointers to
                         / arguments to be passed
        mov     $1,u.quit / u.quit determines handling of quits;
                          / u.quit = 1 take quit
```

```
        mov     $1,u.intr / u.intr determines handling of interrupts;
                          / u.intr = 1 take interrupt
        mov     $rtssym,*30 / emt trap vector set to take system routine
        mov     $fpsym,*10 / reserved instruction trap vector set to take
                           / system routine
        mov     $sstack,sp / stack space used during swapping
        mov     r5,-(sp) / save arguments pointer on stack
        mov     $ecore,r5 / r5 has end of core
        mov     $core,r4 / r4 has start of users core
        mov     r4,u.base / u.base has start of users core
        mov     (sp),r2 / move arguments list pointer into r2
1:
        tst     (r2)+ / argument char = "nul"
        bne     1b
        tst     -(r2) / decrement r2 by 2; r2 has addr of end of argument
                      / pointer list
1: / move arguments to bottom of users core
        mov     -(r2),r3 / (r3) last non zero argument ptr
        cmp     r2,(sp) / is r2 = beginning of argument ptr list
        blo     1f / branch to 1f when all arguments are moved
2:
        tstb    (r3)+
        bne     2b / scan argument for \0 (nul)
2:
        movb    -(r3),-(r5) / move argument char by char starting at
                            / "ecore"
        cmp     r3,(r2) / moved all characters in this argument
        bhi     2b / branch 2b if not
        mov     r5,(r4)+ / move r5 into top of users core; r5 has
                         / pointer to nth arg
        br      1b / string
1:
        clrb    -(r5)
        bic     $1,r5 / make r5 even, r5 points to last word of argument
                      / strings
        mov     $core,r2
1: / move argument pointers into core following argument strings
        cmp     r2,r4
        bhis    1f / branch to 1f when all pointers are moved
        mov     (r2)+,-(r5)
        br      1b
1:
        sub     $core,r4 / gives number of arguments *2
        asr     r4 / divide r4 by 2 to calculate the number of args stored
        mov     r4,-(r5) / save number of arguments ahead of the argument
                         / pointers
        clr     -(r5) / popped into ps when rti in sysrele is executed
        mov     $core,-(r5) / popped into pc when rti in sysrele
                            / is executed
        mov     r5,0f / load second copyz argument
        tst     -(r5) / decrement r5
        mov     r5,u.r0 /
        sub     $16.,r5 / skip 8 words
        mov     r5,u.sp / assign user stack pointer value, effectively
                        / zeroes all regs when sysrele is executed
        jsr     r0,copyz; core; 0:0 / zero user's core
```

```
        clr     u.break
        mov     r5,sp / point sp to user's stack
        mov     $14,u.count
        mov     $u.off,u.fofp
        clr     u.off / set offset in file to be read to zero
        jsr     r0,readi / read in first six words of user's file, starting
                         / at $core
        mov     sp,r5 / put users stack address in r5
        sub     $core+40.,r5 / subtract $core +40, from r5 (leaves
                             / number of words less 26 available for
                             / program in user core
        mov     r5,u.count /
        cmp     core,$405 / br .+14 is first instruction if file is
                          / standard a.out format
        bne     1f / branch, if not standard format
        mov     core+2,r5 / put 2nd word of users program in r5; number of
                          / bytes in program text
        sub     $14,r5 / subtract 12
        cmp     r5,u.count /
        bgt     1f / branch if r5 greater than u.count
        mov     r5,u.count
        jsr     r0,readi / read in rest of user's program text
        add     core+10,u.nread / add size of user data area to u.nread
        br      2f
1:
        jsr     r0,readi / read in rest of file
2:
        mov     u.nread,u.break / set users program break to end of
                                / user code
        add     $core+14,u.break / plus data area
        jsr     r0,iclose / does nothing
        br      sysret3 / return to core image at $core

sysfstat: / set status of open file
        jsr     r0,arg; u.off / put buffer address in u.off
        mov     u.off,-(sp) / put buffer address on the stack
        mov     *u.r0,r1 / put file descriptor in r1
        jsr     r0,getf / get the files i-number
        tst     r1 / is it 0?
        beq     error3 / yes, error
        bgt     1f / if i-number is negative (open for writing)
        neg     r1 / make it positive, then branch
        br      1f / to 1f

sysstat: / ; name of file; buffer - get files status
        jsr     r0,arg2 / get the 2 arguments
        jsr     r0,namei / get the i-number for the file
                br error3 / no such file, error
1:
        jsr     r0,iget / get the i-node into core
        mov     (sp)+,r3 / move u.off to r3 (points to buffer)
        mov     r1,(r3)+ / put i-number in 1st word of buffer
        mov     $inode,r2 / r2 points to i-node
1:
        mov     (r2)+,(r3)+ / move rest of i-node to buffer
        cmp     r2,$inode+32 / done?
```

```
        bne     1b / no, go back
        br      sysret3 / return through sysret

error3:
        jmp     error / see 'error' routine
sysret3:
        jmp     sysret / see 'sysret' routine

getf: / get the device number and the i-number of an open file
        cmp     r1,$10. / user limited to 10 open files
        bhis    error3 / u.fp is table of users open files, index in
                       / fsp table
        movb    u.fp(r1),r1 / r1 contains number of entry in fsp table
        beq     1f / if its zero, return
        asl     r1
        asl     r1 / multiply by 8 to get index into fsp table entry
        asl     r1
        add     $fsp-4,r1 / r1 is pointing at the 3rd word in the fsp entry
        mov     r1,u.fofp / save address of 3rd word in fsp entry in u.fofp
        mov     -(r1),cdev / remove the device number  cdev
        mov     -(r1),r1 / and the i-number  r1
1:
        rts     r0

namei:
        mov     u.cdir,r1 / put the i-number of current directory in r1
        mov     u.cdev,cdev / device number for users directory into cdev
        cmpb    *u.namep,$'/ / is first char in file name a /
        bne     1f
        inc     u.namep / go to next char
        mov     rootdir,r1 / put i-number of rootdirectory in r1
        clr     cdev / clear device number
1:
        tstb    *u.namep / is the character in file name a nul
        beq     nig / yes, end of file name reached; branch to "nig"
1:
        jsr     r0,access; 2 / get i-node with i-number r1
        bit     $40000,i.flgs / directory i-node?
        beq     error3 / no, got an error
        mov     i.size,u.dirp / put size of directory in u.dirp
        clr     u.off / u.off is file offset used by user
        mov     $u.off,u.fofp / u.fofp is a pointer to the offset portion
                              / of fsp entry
2:
        mov     $u.dirbuf,u.base / u.dirbuf holds a file name copied from
                                 / a directory
        mov     $10.,u.count / u.count is byte count for reads and writes
        jsr     r0,readi / read 10. bytes of file with i-number (r1);
                         / i.e. read a directory entry
        tst     u.nread
        ble     nib / gives error return
        tst     u.dirbuf /
        bne     3f / branch when active directory entry (i-node word in
                   / entry non zero)
        mov     u.off,u.dirp
        sub     $10.,u.dirp
```

```
        br      2b
3:
        mov     u.namep,r2 / u.namep points into a file name string
        mov     $u.dirbuf+2,r3 / points to file name of directory entry
3:
        movb    (r2)+,r4 / move a character from u.namep string into r4
        beq     3f / if char is nul, then the last char in string has been
                   / moved
        cmp     r4,$'/ / is char a </>
        beq     3f
        cmp     r3,$u.dirbuf+10. / have I checked all 8 bytes of file name
        beq     3b
        cmpb    (r3)+,r4 / compare char in u.namep string to file name
                         / char read from
        beq     3b / directory; branch if chars match
        br      2b / file names do not match go to next directory entry
3:
        cmp     r3,$u.dirbuf+10. / if equal all 8 bytes were matched
        beq     3f
        tstb    (r3)+ /
        bne     2b
3:
        mov     r2,u.namep / u.namep points to char following a / or nul
        mov     u.dirbuf,r1 / move i-node number in directory entry to r1
        tst     r4 / if r4 = 0 the end of file name reached, if r4 = </>
                   / then go to next directory
        bne     1b

nig:
        tst     (r0)+ / gives non-error return
nib:
        rts     r0

syschdir: / makes the directory specified in the argument the current
          / directory
        jsr     r0,arg; u.namep / u.namep points to path name
        jsr     r0,namei / find its i-number
                br error3
        jsr     r0,access; 2 / get i-node into core
        bit     $40000,i.flgs / is it a directory?
        beq     error3 / no error
        mov     r1,u.cdir / move i-number to users current directory
        mov     cdev,u.cdev / move its device to users current device
        br      sysret3

isown:
        jsr     r0,arg2 / u.namep points to file name
        jsr     r0,namei / get its i-number
                br error3
        jsr     r0,iget / get i-node into core
        tstb    u.uid / super user?
        beq     1f / yes, branch
        cmpb    i.uid,u.uid / no, is this the owner of the file
        beq     1f / yes
        jmp     error3 / no, error
1:
```

```
        jsr     r0,setimod / indicates i-node has been modified
        mov     (sp)+,r2 / mode is put in r2 (u.off put on stack with
                         / 2nd arg)
        rts     r0

syschmod: / name; mode
        jsr     r0,isown / get the i-node and check user status
        bit     $40000,i.flgs / directory?
        beq     2f / no
        bic     $60,r2 / su & ex / yes, clear set user id and
                               / executable modes
2:
        movb    r2,i.flgs / move remaining mode to i.flgs
        br      1f

syschown: / name; owner
        jsr     r0,isown / get the i-node and check user status
        tstb    u.uid / super user
        beq     2f / yes, 2f
        bit     $40,i.flgs / no, set user id on execution?
        bne     3f / yes error, could create Trojan Horses
2:
        movb    r2,i.uid / no, put the new owners id in the i-node
1:
        jmp     sysret4
3:
        jmp     error

arg:
        mov     u.sp,r1
        mov     *18.(r1),*(r0)+ / put argument of system call into
                                / argument of arg2 or rwl
        add     $2,18.(r1) / point pc on stack to next system argument
        rts     r0

arg2:
        jsr     r0,arg; u.namep / u.namep contains value of first arg in
                                / sys call
        jsr     r0,arg; u.off / u.off contains value of second arg in
                              / sys call
        mov     r0,r1 / r0 points to calling routine
        mov     (sp),r0 / put operation code back in r0
        mov     u.off,(sp) / put pointer to second argument on stack
        jmp     (r1) / return to calling routine

systime: / get time of year
        mov     s.time,4(sp)
        mov     s.time+2,2(sp) / put the present time on the stack
        br      sysret4

sysstime: / set time
        tstb    u.uid / is user the super user
        bne     error4 / no, error
        mov     4(sp),s.time
        mov     2(sp),s.time+2 / set the system time
        br      sysret4
```

```
sysbreak: / set the program break
        mov     u.break,r1 / move users break point to r1
        cmp     r1,$core / is it the same or lower than core?
        blos    1f / yes, 1f
        cmp     r1,sp / is it the same or higher than the stack?
        bhis    1f / yes, 1f
        bit     $1,r1 / is it an odd address
        beq     2f / no, its even
        clrb    (r1)+ / yes, make it even
2: / clear area between the break point and the stack
        cmp     r1,sp / is it higher or same than the stack
        bhis    1f / yes, quit
        clr     (r1)+ / clear word
        br      2b / go back
1:
        jsr     r0,arg; u.break / put the "address" in u.break (set new
                                / break point)
        br      sysret4 / br sysret

maknod: / r1 contains the mode
        bis     $100000,r1 / allocate flag set
        mov     r1,-(sp) / put mode on stack
        mov     ii,r1 / move current i-number to r1
        jsr     r0,access; 1 / get its i-node into core
        mov     r1,-(sp) / put i-number on stack
        mov     $40.,r1 / r1 = 40
1: / scan for a free i-node (next 4 instructions)
        inc     r1 / r1 = r1+1
        jsr     r0,imap / get byte address and bit position in inode map in
                        / r2 & m
        bitb    mq,(r2) / is the i-node active
        bne     1b / yes, try the next one
        bisb    mq,(r2) / no, make it active (put a 1 in the bit map)
        jsr     r0,iget / get i-node into core
        tst     i.flgs / is i-node already allocated
        blt     1b / yes, look for another one
        mov     r1,u.dirbuf / no, put i-number in u.dirbuf
        mov     (sp)+,r1 / get current i-number back
        jsr     r0,iget / get i-node in core
        jsr     r0,mkdir / make a directory entry in current directory
        mov     u.dirbuf,r1 / r1 = new inode number
        jsr     r0,iget / get it into core
        jsr     r0,copyz; inode; inode+32. / 0 it out
        mov     (sp)+,i.flgs / fill flags
        movb    u.uid,i.uid / user id
        movb    $1,i.nlks / 1 link
        mov     s.time,i.ctim / time created
        mov     s.time+2,i.ctim+2 / time modified
        jsr     r0,setimod / set modified flag
        rts     r0 / return

sysseek: / moves read write pointer in an fsp entry
        jsr     r0,seektell / get proper value in u.count
        add     u.base,u.count / add u.base to it
        mov     u.count,*u.fofp / put result into r/w pointer
```

```
        br      sysret4

systell: / get the r/w pointer
        jsr     r0,seektell
        br      error4

error4:
        jmp     error / see 'error' routine
sysret4:
        jmp     sysret / see 'sysret' routine

seektell:
        jsr     r0,arg; u.base / puts offset in u.base
        jsr     r0,arg; u.count / put ptr name in u.count
        mov     *u.r0,r1 / file descriptor in r1 (index in u.fp list)
        jsr     r0,getf / u.fofp points to 3rd word in fsp entry
        mov     r1,-(sp) / r1 has i-number of file, put it on the stack
        beq     error4 / if i-number is 0, not active so error
        bgt     .+4 / if its positive jump
        neg     r1 / if not make it positive
        jsr     r0,iget / get its i-node into core
        cmp     u.count,$1 / is ptr name =1
        blt     2f / no its zero
        beq     1f / yes its 1
        mov     i.size,u.count / put number of bytes in file in u.count
        br      2f
1: / ptr name =1
        mov     *u.fofp,u.count / put offset in u.count
2: / ptrname =0
        mov     (sp)+,r1 / i-number on stack  r1
        rts     r0

sysintr: / set interrupt handling
        jsr     r0,arg; u.intr / put the argument in u.intr
        br      1f / go into quit routine
sysquit: jsr    r0,arg; u.quit / put argument in u.quit
1:
        mov     u.ttyp,r1 / move pointer to control tty buffer to r1
        beq     sysret4 / return to user
        clrb    6(r1) / clear the interrupt character in the tty buffer
        br      sysret4 / return to user

syssetuid: / set process id
        movb    *u.r0,r1 / move process id (number) to r1
        cmpb    r1,u.ruid / is it equal to the real user id number
        beq     1f / yes
        tstb    u.uid / no, is current user the super user?
        bne     error4 / no, error
1:
        movb    r1,u.uid / put process id in u.uid
        movb    r1,u.ruid / put process id in u.ruid
        br      sysret4 / system return

sysgetuid:
        movb    u.ruid,*u.r0 / move the real user id to (u.r0)
        br      sysret4 / system return, sysret
```

```
fclose:
        mov     r1,-(sp) / put r1 on the stack (it contains the index
                         / to u.fp list)
        jsr     r0,getf / r1 contains i-number, cdev has device =, u.fofp
                        / points to 3rd word of fsp entry
        tst     r1 / is inumber 0?
        beq     1f / yes, i-node not active so return
        tst     (r0)+ / no, jump over error return
        mov     r1,r2 / move i-number to r2
        mov     (sp),r1 / restore value of r1 from the stack which is
                        / index to u.fp
        clrb    u.fp(r1) / clear that entry in the u.fp list
        mov     u.fofp,r1 / r1 points to 3rd word in fsp entry
        decb    2(r1) / decrement the number of processes that have opened
                      / the file
        bge     1f / if all processes haven't closed the file, return
        mov     r2,-(sp) / put r2 on the stack (i-number)
        clr     -4(r1) / clear 1st word of fsp entry
        tstb    3(r1) / has this file been deleted
        beq     2f / no, branch
        mov     r2,r1 / yes, put i-number back into r1
        jsr     r0,anyi / free all blocks related to i-number
                        / check if file appears in fsp again
2:
        mov     (sp)+,r1 / put i-number back into r1
        jsr     r0,iclose / check to see if its a special file
1:
        mov     (sp)+,r1 / put index to u.fp back into r1
        rts     r0

anyi: / r1 contains an i-number
        mov     $fsp,r2 / move start of fsp table to r2
1:
        cmp     r1,(r2) / do i-numbers match?
        beq     1f / yes, 1f
        neg     r1 / no complement r1
        cmp     r1,(r2) / do they match now?
        beq     1f / yes, transfer
                   / i-numbers do not match
        add     $8,r2 / no, bump to next entry in fsp table
        cmp     r2,$fsp+[nfiles*8] / are we at last entry in the table
        blt     1b / no, check next entries i-number
        tst     r1 / yes, no match
        bge     .+4
        neg     r1 / make i-number positive
        jsr     r0,imap / get address of allocation bit in the i-map in r2
        bicb    mq,(r2) / clear bit for i-node in the imap
        jsr     r0,itrunc / free all blocks related to i-node
        clr     i.flgs / clear all flags in the i-node
        rts     r0 / return
1: / i-numbers match
        incb    7(r2) / increment upper byte of the 4th word
        rts     r0 / in that fsp entry (deleted flag of fsp entry)
```

```
/ u3 -- unix

tswap:
        movb    u.uno,r1 / move users process number to r1
        mov     $runq+4,r2 / move lowest priority queue address to r2
        jsr     r0,putlu / create link from last user on Q to u.uno's user

swap:
        mov     $300,*$ps / processor priority = 6
        mov     $runq,r2 / r2 points to runq table
1: / search runq table for highest priority process
        tst     (r2)+ / are there any processes to run in this Q entry
        bne     1f / yes, process 1f
        cmp     r2,$runq+6 / if zero compare address to end of table
        bne     1b / if not at end, go back
        jsr     r0,idle; s.idlet+2 / wait for interrupt; all queues
                                   / are empty
        br      swap
1:
        tst     -(r2) / restore pointer to right Q entry
        mov     r2,u.pri / set present user to this run queue
        movb    (r2)+,r1 / move 1st process in queue to r1
        cmpb    r1,(r2)+ / is there only 1 process in this Q to be run
        beq     1f / yes
        tst     -(r2) / no, pt r2 back to this Q entry
        movb    p.link-1(r1),(r2) / move next process in line into
                                  / run queue
        br      2f
1:
        clr     -(r2) / zero the entry; no processes on the Q
2: / write out core to appropriate disk area and read in new process if
   / required
        clr     *$ps / clear processor status
        cmpb    r1,u.uno / is this process the same as the process in core?
        beq     2f / yes, don't have to swap
        mov     r0,-(sp) / no, write out core; save r0 (address in rout
                         / that called swap)
        mov     sp,u.usp / save stack pointer
        mov     $sstack,sp / move swap stack pointer to the stack pointer
        mov     r1,-(sp) / put r1 (new process #) on the stack
        tstb    u.uno / is the process # = 0
        beq     1f / yes, kill process by overwriting
        jsr     r0,wswap / write out core to disk
1:
        mov     (sp)+,r1 / restore r1 to new process number
        jsr     r0,rswap / read new process into core
        jsr     r0,unpack / unpack the users stack from next to his program
                          / to its normal
        mov     u.usp,sp / location; restore stack pointer to new process
                         / stack
        mov     (sp)+,r0 / put address of where the process that just got
                         / swapped in, left off., i.e., transfer control
                         / to new process
2:
        movb    $30.,uquant / initialize process time quantum
        rts     r0 / return
```

```
wswap:
        mov     *$30,u.emt / determines handling of emts
        mov     *$10,u.ilgins / determines handling of illegal instructions
        mov     u.break,r2 / put process program break address in r2
        inc     r2 / add 1 to it
        bic     $1,r2 / make it even
        mov     r2,u.break / set break to an even location
        mov     u.usp,r3 / put users stack pter at moment of swap in r3
        cmp     r2,$core / is u.break less than $core
        blos    2f / yes
        cmp     r2,r3 / no, is (u.break) greater than stack pointer
        bhis    2f / yes
1:
        mov     (r3)+,(r2)+ / no, pack stack next to users program
        cmp     r3,$ecore / has stack reached end of core
        bne     1b / no, keep packing
        br      1f / yes
2:
        mov     $ecore,r2 / put end of core in r2
1:
        sub     $user,r2 / get number of bytes to write out (user up
                         / to end of stack gets written out)
        neg     r2 / make it negative
        asr     r2 / change bytes to words (divide by 2)
        mov     r2,swp+4 / word count
        movb    u.uno,r1 / move user process number to r1
        asl     r1 / x2 for index
        mov     r2,p.break-2(r1) / put negative of word count into the
                                 / p.break table
        mov     p.dska-2(r1),r1 / move disk address of swap area for
                                / process to r1
        mov     r1,swp+2 / put processes dska address in swp +2 (block
                         / number)
        bis     $1000,swp / set it up to write (set bit 9)
        jsr     r0,ppoke / write process out on swap area of disk
1:
        tstb    swp+1 / is it done writing?
        bne     1b / no, wait
        rts     r0 / yes, return to swap

rswap:
        asl     r1 / process number x2 for index
        mov     p.break-2(r1), swp+4 / word count
        mov     p.dska-2(r1),swp+2 / disk address
        bis     $2000,swp / read
        jsr     r0,ppoke / read it in
1:
        tstb    swp+1 / done
        bne     1b / no, wait for bit 15 to clear (inhibit bit)
        mov     u.emt,*$30 / yes move these
        mov     u.ilgins,*$10 / back
        rts     r0 / return

unpack: / move stack back to its normal place
        mov     u.break,r2 / r2 points to end of user program
```

```
        cmp     r2,$core / at beginning of user program yet?
        blos    2f / yes, return
        cmp     r2,u.usp / is break above the "stack pointer before
                       / swapping"
        bhis    2f / yes, return
        mov     $ecore,r3 / r3 points to end of core
        add     r3,r2
        sub     u.usp,r2 / end of users stack is in r2
1:
        mov     -(r2),-(r3) / move stack back to its normal place
        cmp     r2,u.break / in core
        bne     1b
2:
        rts     r0

putlu: / r1 = user process no.; r2 points to lowest priority queue
        tstb    (r2)+ / is queue empty?
        beq     1f / yes, branch
        movb    (r2),r3 / no, save the "last user" process number in r3
        movb    r1,p.link-1(r3) / put pointer to user on "last users" link
        br      2f /
1:
        movb    r1,-1(r2) / user is only user; put process no. at beginning
                        / and at end
2:
        movb    r1,(r2) / user process in r1 is now the last entry on
                      / the queue
        dec     r2 / restore r2
        rts     r0

copyz:
        mov     r1,-(sp) / put r1 on stack
        mov     r2,-(sp) / put r2 on stack
        mov     (r0)+,r1
        mov     (r0)+,r2
1:
        clr     (r1)+ / clear all locations between r1 and r2
        cmp     r1,r2
        blo     1b
        mov     (sp)+,r2 / restore r2
        mov     (sp)+,r1 / restore r1
        rts     r0

idle:
        mov     *$ps,-(sp) / save ps on stack
        clr     *$ps / clear ps
        mov     clockp,-(sp) / save clockp on stack
        mov     (r0)+,clockp / arg to idle in clockp
        1 / wait for interrupt
        mov     (sp)+,clockp / restore clockp, ps
        mov     (sp)+,*$ps
        rts     r0

clear:
        jsr     r0,wslot / get an I/O buffer set bits 9 and 15 in first
                       / word of I/O queue r5 points to first data word
```

```
                        / in buffer
        mov     $256.,r3
1:
        clr     (r5)+ / zero data word in buffer
        dec     r3
        bgt     1b / branch until all data words in buffer are zero
        jsr     r0,dskwr / write zeroed buffer area out onto physical
                        / block specified
        rts     r0 / in r1
```

```
/ u4 -- unix

setisp:
        mov     r1,-(sp)
        mov     r2,-(sp)
        mov     r3,-(sp)
        mov     clockp,-(sp)
        mov     $s.syst+2,clockp
        jmp     (r0)

clock: / interrupt from 60 cycle clock
        mov     r0,-(sp) / save r0
        tst     *$lks / restart clock?
        mov     $s.time+2,r0 / increment the time of day
        inc     (r0)
        bne     1f
        inc     -(r0)
1:
        mov     clockp,r0 / increment appropriate time category
        inc     (r0)
        bne     1f
        inc     -(r0)
1:
        mov     $uquant,r0 / decrement user time quantum
        decb    (r0)
        bge     1f / if less than 0
        clrb    (r0) / make it 0
1: / decrement time out counts return now if priority was not 0
        cmp     4(sp),$200 / ps greater than or equal to 200
        bge     2f / yes, check time outs
        tstb    (r0) / no, user timed out?
        bne     1f / no
        cmpb    sysflg,$-1 / yes, are we outside the system?
        bne     1f / no, 1f
        mov     (sp)+,r0 / yes, put users r0 in r0
        sys     0 / sysrele
        rti
2: / priority is high so just decrement time out counts
        mov     $toutt,r0 / r0 points to beginning of time out table
2:
        tstb    (r0) / is the time out?
        beq     3f / yes, 3f (get next entry)
        decb    (r0) / no, decrement the time
        bne     3f /isit zero now?
        incb    (r0) / yes, increment the time
3:
        inc     r0 / next entry
        cmp     r0,$touts / end of toutt table?
        blo     2b / no, check this entry
        mov     (sp)+,r0 / yes, restore r0
        rti     / return from interrupt
1: / decrement time out counts; if 0 call subroutine
        mov     (sp)+,r0 / restore r0
        mov     $240,*$ps / set processor priority to 5
        jsr     r0,setisp / save registers
```

```
        mov     $touts-toutt-1,r0 / set up r0 as index to decrement thru
                                  / the table
1:
        tstb    toutt(r0) / is the time out for this entry
        beq     2f / yes
        decb    toutt(r0) / no, decrement the time
        bne     2f / is the time 0, now
        asl     r0 / yes, 2 x r0 to get word index for tout entry
        jsr     r0,*touts(r0) / go to appropriate routine specified in this
        asr     r0 / touts entry; set r0 back to toutt index
2:
        dec     r0 / set up r0 for next entry
        bge     1b / finished? , no, go back
        br      retisp / yes, restore registers and do a rti

ttyi: / console tty input interrupt routine
        jsr     r0,setisp / save reg r1, r2, r3
        mov     *$tkb,r1 / r1 = char in tty reader buffer
        inc     *$tks / set the reader enable bit
        bic     $!177,r1 / clear upper 9 bits of the character (strip off
                         / 8th bit of char)
        cmp     r1,$'a-40 / is character upper case A,..., upper case Z.
                          / note that
        blt     1f / lower case a is represented by 141, upper case by
        cmp     r1,$'z-40 / 101; and lower case z by 172, upper
                          / case Z by 132.
        bgt     1f / if not upper case, branch
        add     $40,r1 / if upper case, calculate the representation of its
                       / lower case counter part
1:
        cmp     r1,$175 / char = "}"?  Note:  may be quit char (fs)
        beq     2f / yes 2f
        cmp     r1,$177 / char = "del"?
        beq     2f / yes, 2f
        jsr     r0,putc; 0 / put char in r1 on clist entry
                br 1f
        movb    r1,ttyoch / put char in ttyoch
        jsr     r0,startty / load char in tty output data buffer
        cmp     r1,$4 / r1 = "eot"
        beq     1f / yes, 1f
        cmp     r1,$12 / r1 = "lf"
        beq     1f / yes 1f
        cmpb    cc+0,$15. / are there less than 15 chars on the input list
        blo     retisp / yes, return
1:
        jsr     r0,wakeup; runq; 0 / no, wakeup the input process
        br      retisp / return
2: / r1 = "}" or "delete" to get here
        mov     tty+[ntty*8]-8+6,r2 / move console tty buffer address to r2
        beq     2f / if 0, wakeall
        movb    r1,6(r2) / move "}" or del into "interrupt char"
                         / byte of buffer
2:
        jsr     r0,wakeall / wakeup all sleeping processes
        br      retisp / return
```

```
wakeall:
        mov     $39.,0f / fill arg2 of wakeup call with 39
1:
        jsr     r0,wakeup; runq+4; 0:.. / wakeup the processes in the
        dec     0b                      / wait list; decrement arg2
        bge     1b / if not done, go back
        rts     r0

ttyo: / console typewriter output interrupt routine
        jsr     r0,setisp / save registers
        jsr     r0,startty / put a char in the console tty output register
        br      retisp / restore registers

retisp:
        mov     (sp)+,clockp / pop values before interrupt off the stack
        mov     (sp)+,r3
        mov     (sp)+,r2
        mov     (sp)+,r1
        mov     (sp)+,r0
        rti     / return from interrupt

ppti: / paper tape input interrupt routine
        jsr     r0,setisp / save registers
        movb    pptiflg,r1 / place "pptiflg" in r1
        jmp     *1f(r1) / jump to location specified by value of "pptiflg"
1:
        retisp / file not open
        1f / file just opened
        2f / file normal
        retisp / file not closed

1: / file just opened
        tstb    *$prs+1 / is error bit set in prs
        bge     1f / no
        jsr     r0,pptito / place 10 in toutt entry for ppt input
        br      retisp
1:
        movb    $4,pptiflg / change "pptiflg" to indicate file "normal"
2:
        jsr     r0,wakeup; runq+2; 2 / wakeup process for ppt input entry
                                     / in wlist
        tstb    *$prs+1 / is error bit set
        blt     1f / yes
        mov     *$prb,r1 / place contents ppt read buffer in r1
        jsr     r0,putc; 2 / place character in clist area for ppt input
                br .+2 / temp / if no space in clist character lost
        cmpb    cc+2,$50. / character count in clist area for ppt input
                          / greater than or equal to 50
        bhis    retisp / yes
        inc     *$prs / no, set reader enable bit in prs
        br      retisp
1:
        movb    $6,pptiflg / set pptiflg to 6 to indicate error bit set
        br      retisp

/lpto:
```

```
/       jsr     r0,setisp
/       jsr     r0,starlpt
/       br      retisp

ppto: / paper tape output interrupt routine
        jsr     r0,setisp / save registers
        jsr     r0,starppt / get next character from clist, and output
                           / if possible
        br      retisp / pop register values from stack

/starlpt:
/       cmpb    cc+5.,$100.
/       bhi     1f
/       jsr     r0,wakeup; runq+2; 5
/1:
/       tstb    *$lps
/       bge     1f
/       jsr     r0,getc; 5
/               br 1f
/       mov     r1,*$lpb
/       br      starlpt
/1:
/       rts     r0

starttty: / start or restart console tty output
        cmpb    cc+1,$5.
        bhi     1f / branch to 1f when character count on tty (? input,
                  / output) list is greater than 5.
        jsr     r0,wakeup; runq+2; 1
1:
        tstb    *$tps / test console output ready bit
        bge     2f / branch if ready bit is clear
        tstb    toutt+0 / is toutt for console a zero
        bne     2f / if not; branch to 2f
        movb    ttyoch,r1 / put character to be output in r1
        bne     1f
        jsr     r0,getc; 1 / if char is nul, get a char from console
                           / output list
                br 2f / if console output list is empty, branch to 2f
1:
        clrb    ttyoch
        mov     r1,*$tpb / put character in console output register
        cmp     r1,$12 / is char a line feed
        bne     1f
        movb    $15,ttyoch / put a cr in ttyoch
1:
        cmp     r1,$11 / char = ht
        bne     1f
        movb    $15.,toutt+0 / set time out to 15 clock tics
1:
        cmp     r1,$15 / char = cr
        bne     2f
        movb    $15.,toutt+0 / set time out to 15 clock ticks
2:
        rts     r0
```

```
pptito: / paper tape input touts subrouting
        cmpb    pptiflg,$2 / does "pptiflg" indicate file just opened
        bne     1f / no, do nothing                          ppt
        movb    $10.,toutt+1 / yes, place 10 in tout entry for tty input
        tstb    *$prs+1 / is error bit set
        blt     1f / yes, return
        inc     *$prs / no, set read enable bit
1:
        rts     r0

starppt: / start ppt output
        cmpb    cc+3,$10. / is character count for ppt output greater
                          / than 10.
        bhi    1f / yes, branch
        jsr     r0,wakeup; runq+2; 3 / no, wakeup process in wlist
                                     / entry for ppt input
1:
        tstb    *$pps / is ready bit set in punch status word
        bge     1f / no, branch
        jsr     r0,getc; 3 / yes, get next char in clist for pptout and
                           / place in r1
                br 1f / if none, branch
        mov     r1,*$ppb / place character in ppt buffer
1:
        rts     r0

wakeup: / wakeup processes waiting for an event by linking them to the
        / queue
        mov     r1,-(sp) / put char on stack
        mov     (r0)+,r2 / r2 points to a queue
        mov     (r0)+,r3 / r3 = wait channel number
        movb    wlist(r3),r1 / r1 contains process number in that wait
                             / channel that was sleeping
        beq     2f / if 0 return, nothing to wakeup
        cmp     r2,u.pri / is runq greater than or equal to users process
                         / priority
        bhis    1f / yes, don't set time quantum to zero
        clrb    uquant / time quantum = 0
1:
        clrb    wlist(r3) / zero wait channel entry
        jsr     r0,putlu / create a link from the last user on the Q
                         / to this process number that got woken
2:
        mov     (sp)+,r1 / restore r1
        rts     r0

sleep: / wait for event
        jsr     r0,isintr / check to see if interrupt or quit from user
                br 2f / something happened / yes, his interrupt so return
                      / to user
        mov     (r0)+,r1 / put number of wait channel in r1
        movb    wlist(r1),-(sp) / put old process number in there, on
                                / the stack
        movb    u.uno,wlist(r1) / put process number of process to put
                                / to sleep in there
        mov     cdev,-(sp) / nothing happened in isintr so
```

```
        jsr     r0,swap / swap out process that needs to sleep
        mov     (sp)+,cdev / restore device
        jsr     r0,isintr / check for interrupt of new process
                br 2f / yes, return to new user
        movb    (sp)+,r1 / no, r1 = old process number that was originally
                         / on the wait channel
        beq     1f / if 0 branch
        mov     $runq+4,r2 / r2 points to lowest priority queue
        mov     $300,*$ps / processor priority = 6
        jsr     r0,putlu / create link to old process number
        clr     *$ps / clear the status; process priority = 0
1:
        rts     r0 / return
2:
        jmp     sysret / return to user

isintr:
        mov     r1,-(sp) / put number of wait channel on the stack
        mov     r2,-(sp) / save r2
        mov     u.ttyp,r1 / r1 = pointer to buffer of process control
                          / typewriter
        beq     1f / if 0, do nothing except skip return
        movb    6(r1),r1 / put interrupt char in the tty buffer in r1
        beq     1f / if its 0 do nothing except skip return
        cmp     r1,$177 / is interrupt char = delete?
        bne     3f / no, so it must be a quit (fs)
        tst     u.intr / yes, value of u.intr determines handling
                       / of interrupts
        bne     2f / if not 0, 2f.  If zero do nothing.
1:
        tst     (r0)+ / bump r0 past system return (skip)
4:
        mov     (sp)+,r2 / restore r1 and r2
        mov     (sp)+,r1
        rts     r0
3: / interrupt char = quit (fs)
        tst     u.quit / value of u.quit determines handling of quits
        beq     1b / u.quit = 0 means do nothing
2: / get here because either u.intr ≠ 0 or u.quit ≠ 0
        mov     $tty+6,r1 / move pointer to tty block into r1
1: / find process control tty entry in tty block
        cmp     (r1),u.ttyp / is this the process control tty buffer?
        beq     1f / block found go to 1f
        add     $8,r1 / look at next tty block
        cmp     r1,$tty+[ntty*8]+6 / are we at end of tty blocks
        blo     1b / no
        br      4b / no process control tty found so go to 4b
1:
        mov     $240,*$ps / set processor priority to 5
        movb    -3(r1),0f / load getc call argument; character list
                          / identifier
        inc     0f / increment
1:
        jsr     r0,getc; 0:.. / erase output char list for control
                br 4b / process tty.  This prevents a line of stuff
                      / being typed out after you hit the interrupt
```

```
                    / key
     br      1b
```

```
/ u5 -- unix

mget:
        mov     *u.fofp,mq / file offset in mq
        clr     ac / later to be high sig
        mov     $-8,lsh / divide ac/mq by 256.
        mov     mq,r2
        bit     $10000,i.flgs / lg/sm is this a large or small file
        bne     4f / branch for large file
        bit     $!17,r2
        bne     3f / branch if r2 greater than or equal to 16
        bic     $!16,r2 / clear all bits but bits 1,2,3
        mov     i.dskp(r2),r1 / r1 has physical block number
        bne     2f / if physical block num is zero then need a new block
                    / for file
        jsr     r0,alloc / allocate a new block
        mov     r1,i.dskp(r2) / physical block number stored in i-node
        jsr     r0,setimod / set inode modified byte (imod)
        jsr     r0,clear / zero out disk/drum block just allocated
2:
        rts     r0
3: / adding on block which changes small file to a large file
        jsr     r0,alloc / allocate a new block for this file; block number
                          / in r1
        jsr     r0,wslot / set up I/O buffer for write, r5 points to first
                          / data word in buffer
        mov     $8.,r3 / next 6 instructions transfer old physical block
                       / pointers
        mov     $i.dskp,r2 / into new indirect block for the new large file
1:
        mov     (r2),(r5)+
        clr     (r2)+
        dec     r3
        bgt     1b
        mov     $256.-8.,r3 / clear rest of data buffer
1:
        clr     (r5)+
        dec     r3
        bgt     1b
        jsr     r0,dskwr / write new indirect block on disk
        mov     r1,i.dskp / put pointer to indirect block in i-node
        bis     $10000,i.flgs / set large file bit in i.flgs word of i-node
        jsr     r0,setimod / set i-node modified flag
        br      mget
4: / large file
        mov     $-8,lsh / divide byte number by 256.
        bic     $!776,r2 / zero all bits but 1,2,3,4,5,6,7,8; gives offset
                          / in indirect block
        mov     r2,-(sp) / save on stack
        mov     mq,r2 / calculate offset in i-node for pointer to proper
                      / indirect block
        bic     $!16,r2
        mov     i.dskp(r2),r1
        bne     2f / if no indirect block exists
        jsr     r0,alloc / allocate a new block
```

```
        mov     r1,i.dskp(r2) / put block number of new block in i-node
        jsr     r0,setimod / set i-node modified byte
        jsr     r0,clear / clear new block
2:
        jsr     r0,dskrd / read in indirect block
        mov     (sp)+,r2 / get offset
        mov     r1,-(sp) / save block number of indirect block on stack
        add     r5,r2 / r5 points to first word in indirect block, r2
                      / points to location of inter
        mov     (r2),r1 / put physical block no of block in file
                        / sought in r1
        bne     2f / if no block exists
        jsr     r0,alloc / allocate a new block
        mov     r1,(r2) / put new block number into proper location in
                        / indirect block
        mov     (sp)+,r1 / get block number of indirect block
        mov     (r2),-(sp) / save block number of new block
        jsr     r0,wslot
        jsr     r0,dskwr / write newly modified indirect block back out
                         / on disk
        mov     (sp),r1 / restore block number of new block
        jsr     r0,clear / clear new block
2:
        tst     (sp)+ / bump stack pointer
        rts     r0

alloc:
        mov     r2,-(sp) / save r2, r3 on stack
        mov     r3,-(sp)
        mov     $systm,r2 / start of inode and free storage map for drum
        tst     cdev
        beq     1f / drum is device
        mov     $mount,r2 / disk or tape is device, start of inode and free
                          / storage map
1:
        mov     (r2)+,r1 / first word contains number of bytes in free
                         / storage map
        asl     r1 / multiply r1 by eight gives, number of blocks in device
        asl     r1
        asl     r1
        mov     r1,-(sp) / save # of blocks in device on stack
        clr     r1 / r1 contains bit count of free storage map
1:
        mov     (r2)+,r3 / word of free storage map in r3
        bne     1f / branch if any free blocks in this word
        add     $16.,r1
        cmp     r1,(sp) / have we examined all free storage bytes
        blo     1b
        jmp     panic / found no free storage
1:
        asr     r3 / find a free block
        bcs     1f / branch when free block found; bit for block k is in
                   / byte k/8 / in bit k (mod 8)
        inc     r1 / increment bit count in bit k (mod8)
        br      1b
1:
```

```
        tst     (sp)+ / bump sp
        jsr     r0,3f / have found a free block
        bic     r3,(r2) / set bit for this block i.e. assign block
        br      2f

free:
        mov     r2,-(sp) / save r2, r3
        mov     r3,-(sp)
        jsr     r0,3f / set up bit mask and word no. in free storage map
                  / for block
        bis     r3,(r2) / set free storage block bit; indicates free block
2:
        mov     (sp)+,r3 / restore r2, r3
        mov     (sp)+,r2
        tst     cdev / cdev = 0, block structured, drum; cdev = 1
                     / mountable device
        bne     1f
        incb    smod / set super block modified for drum
        rts     r0
1:
        incb    mmod / set super block modified for mountable device
        rts     r0

3:
        mov     r1,r2 / block number, k, = 1
        bic     $!7,r2 / clear all bits but 0,1,2; r2 = (k) mod (8)
        clr     r3
        bisb     2f(r2),r3 / use mask to set bit in r3 corresponding to
                          / (k) mod 8
        mov     r1,r2 / divide block number by 16
        asr     r2
        asr     r2
        asr     r2
        asr     r2
        bcc     1f / branch if bit 3 in r1 was 0 i.e., bit for block is in
                   / lower half of word
        swab    r3 / swap bytes in r3; bit in upper half of word in free
                   / storage map
1:
        asl     r2 / multiply block number by 2; r2 = k/8
        add     $systm+2,r2 / address of word of free storage map for drum
                            / with block bit in it
        tst     cdev
        beq     1f / cdev = 0 indicates device is drum
        add     $mount-systm,r2 / address of word of free storage map for
                                / mountable device with bit of block to be
                                / freed
1:
        rts     r0 / return to 'free'
2:
        .byte   1,2,4,10,20,40,100,200 / masks for bits 0,...,7

access:
        jsr     r0,iget / read in i-node for current directory (i-number
                        / passed in r1)
        mov     i.flgs,r2
```

```
        cmpb    i.uid,u.uid / is user same as owner of file
        bne     1f / no, then branch
        asrb    r2 / shift owner read write bits into non owner
                   / read/write bits
        asrb    r2
1:
        bit     r2,(r0)+ / test read-write flags against argument in
                         / access call
        bne     1f
        tstb    u.uid
        beq     1f
        jmp     error
1:
        rts     r0

setimod:
        movb    $1,imod / set current i-node modified bytes
        mov     s.time,i.mtim / put present time into file modified time
        mov     s.time+2,i.mtim+2
        rts     r0

imap: / get the byte that has the allocation bit for the i-number contained
      / in r1
        mov     $1,mq / put 1 in the mq
        mov     r1,r2 / r2 now has i-number whose byte in the map we
                      / must find
        sub     $41.,r2 / r2 has i-41
        mov     r2,r3 / r3 has i-41
        bic     $!7,r3 / r3 has (i-41) mod 8 to get the bit position
        mov     r3,lsh / move the 1 over (i-41) mod 8 positions to the left
                       / to mask the correct bit
        asr     r2
        asr     r2
        asr     r2 / r2 has (i-41) base 8 of the byte no. from the start of
                   / the map
        mov     r2,-(sp) / put (i-41) base 8 on the stack
        mov     $systm,r2 / r2 points to the in-core image of the super
                          / block for drum
        tst     cdev / is the device the disk
        beq     1f / yes
        add     $mount-systm,r2 / for mounted device, r2 points to 1st word
                                / of its super block
1:
        add     (r2)+,(sp) / get byte address of allocation bit
        add     (sp)+,r2 / ?
        add     $2,r2 / ?
        rts     r0

iget:
        cmp     r1,ii / r1 = i-number of current file
        bne     1f
        cmp     idev,cdev / is device number of i-node = current device
        beq     2f
1:
        tstb    imod / has i-node of current file been modified i.e.,
                     / imod set
```

```
        beq     1f
        clrb    imod / if it has, we must write the new i-node out on disk
        mov     r1,-(sp)
        mov     cdev,-(sp)
        mov     ii,r1
        mov     idev,cdev
        jsr     r0,icalc; 1
        mov     (sp)+,cdev
        mov     (sp)+,r1
1:
        tst     r1 / is new i-number non zero
        beq     2f / branch if r1=0
        tst     cdev / is the current device number non zero (i.e., device
                     / ≠ drum)
        bne     1f / branch if cdev ≠ 0
        cmp     r1,mnti / mnti is the i-number of the cross device
                        / file (root directory of mounted device)
        bne     1f
        mov     mntd,cdev / make mounted device the current device
        mov     rootdir,r1
1:
        mov     r1,ii
        mov     cdev,idev
        jsr     r0,icalc; 0 / read in i-node ii
2:
        mov     ii,r1
        rts     r0

icalc: / i-node i is located in block (i+31.)/16. and begins 32.*
       / (i+31)mod16 bytes from its start
        add     $31.,r1 / add 31. to i-number
        mov     r1,-(sp) / save i+31. on stack
        asr     r1 / divide by 16.
        asr     r1
        asr     r1
        asr     r1 / r1 contains block number of block in which
                   / i-node exists
        jsr     r0,dskrd / read in block containing i-node i.
        tst     (r0)
        beq     1f / branch to wslot when argument in icalc call = 1
        jsr     r0,wslot / set up data buffer for write (will be same buffer
                         / as dskrd got)
1:
        bic     $!17,(sp) / zero all but last 4 bits; gives (i+31.) mod 16
        mov     (sp)+,mq / calculate offset in data buffer; 32.*(i+31.)mod16
        mov     $5,lsh / for i-node i.
        add     mq,r5 / r5 points to first word in i-node i.
        mov     $inode,r1 / inode is address of first word of current i-node
        mov     $16.,r3
        tst     (r0)+ / branch to 2f when argument in icalc call = 0
        beq     2f / r0 now contains proper return address for rts r0
1:
        mov     (r1)+,(r5)+ / over write old i-node
        dec     r3
        bgt     1b
        jsr     r0,dskwr / write inode out on device
```

```
        rts     r0
2:
        mov     (r5)+,(r1)+ / read new i-node into "inode" area of core
        dec     r3
        bgt     2b
        rts     r0

itrunc:
        jsr     r0,iget
        mov     $i.dskp,r2 / address of block pointers in r2
1:
        mov     (r2)+,r1 / move physical block number into r1
        beq     5f
        mov     r2,-(sp)
        bit     $10000,i.flgs / test large file bit?
        beq     4f / if clear, branch
        mov     r1,-(sp) / save block number of indirect block
        jsr     r0,dskrd / read in block, 1st data word pointed to by r5
        mov     $256.,r3 / move word count into r3
2:
        mov     (r5)+,r1 / put 1st data word in r1; physical block number
        beq     3f / branch if zero
        mov     r3,-(sp) / save r3, r5 on stack
        mov     r5,-(sp)
        jsr     r0,free / free block in free storage map
        mov     (sp)+,r5
        mov     (sp)+,r3
3:
        dec     r3 / decrement word count
        bgt     2b / branch if positive
        mov     (sp)+,r1 / put physical block number of indirect block
4:
        jsr     r0,free / free indirect block
        mov     (sp)+,r2
5:
        cmp     r2,$i.dskp+16.
        bne     1b / branch until all i.dskp entries chec
        bic     $10000,i.flgs / clear large file bit
        clr     i.size / zero file size
        jsr     r0,copyz; i.dskp; i.dskp+16. / zero block pointers
        jsr     r0,setimod / set i-node modified flag
        mov     ii,r1
        rts     r0
```

```
/ u6 -- unix

readi:
        clr     u.nread / accumulates number of bytes transmitted
        tst     u.count / is number of bytes to be read greater than 0
        bgt     1f / yes, branch
        rts     r0 / no, nothing to read; return to caller
1:
        mov     r1,-(sp) / save i-number on stack
        cmp     r1,$40. / want to read a special file (i-nodes 1,...,40 are
                        / for special files)
        ble     1f / yes, branch
        jmp     dskr / no, jmp to dskr; read file with i-node number (r1)
                     / starting at byte ((u.fofp)), read in u.count bytes
1:
        asl     r1 / multiply inode number by 2
        jmp     *1f-2(r1)
1:
        rtty    / tty; r1=2
        rppt    / ppt; r1=4
        rmem    / mem; r1=6
        rrf0    / rf0
        rrk0    / rk0
        rtap    / tap0
        rtap    / tap1
        rtap    / tap2
        rtap    / tap3
        rtap    / tap4
        rtap    / tap5
        rtap    / tap6
        rtap    / tap7
        rcvt    / tty0
        rcvt    / tty1
        rcvt    / tty2
        rcvt    / tty3
        rcvt    / tty4
        rcvt    / tty5
        rcvt    / tty6
        rcvt    / tty7
        rcrd/ crd

rtty: / read from console tty
        mov     tty+[8*ntty]-8+6,r5 / r5 is the address of the 4th word of
                                    / of the control and status block
        tst     2(r5) / for the console tty; this word points to the console
                      / tty buffer
        bne     1f / 2nd word of console tty buffer contains number
                   / of chars.  Is this number non-zero?
        jsr     r0,canon; ttych / if 0, call 'canon' to get a line
                                / (120 chars.)
1:
        tst     2(r5) / is the number of characters zero
        beq     ret1 / yes, return to caller via 'ret1'
        movb    *4(r5),r1 / no, put character in r1
        inc     4(r5) / 3rd word of console tty buffer points to byte which
                      / contains the next char.
```

```
        dec     2(r5) / decrement the character count
        jsr     r0,passc / move the character to core (user)
        br      1b / get next character

ret1:
        jmp     ret / return to caller via 'ret'

rppt: / read paper tape
        jsr     r0,pptic / gets next character in clist for ppt input and
                         / places
                br ret / it in r1; if there is no problem with reader, it
                       / also enables read bit in prs
        jsr     r0,passc / place character in users buffer area
        br      rppt

rmem: / transfer characters from memory to a user area of core
        mov     *u.fofp,r1 / save file offset which points to the char to
                           / be transferred to user
        inc     *u.fofp / increment file offset to point to 'next' char in
                        / memory file
        movb    (r1),r1 / get character from memory file, put it in r1
        jsr     r0,passc / move this character to the next byte of the
                         / users core area
        br      rmem / continue
1:
rcrd:
        jmp     error / see 'error' routine

dskr:
        mov     (sp),r1 / i-number in r1
        jsr     r0,iget / get i-node (r1) into i-node section of core
        mov     i.size,r2 / file size in bytes in r2
        sub     *u.fofp,r2 / subtract file offset
        blos    ret
        cmp     r2,u.count / are enough bytes left in file to carry out read
        bhis    1f
        mov     r2,u.count / no, just read to end of file
1:
        jsr     r0,mget / returns physical block number of block in file
                        / where offset points
        jsr     r0,dskrd / read in block, r5 points to 1st word of data in
                         / buffer
        jsr     r0,sioreg
2:
        movb    (r2)+,(r1)+ / move data from buffer into working core
                            / starting at u.base
        dec     r3
        bne     2b / branch until proper number of bytes are transferred
        tst     u.count / all bytes read off disk
        bne     dskr
        br      ret

passc:
        movb    r1,*u.base / move a character to the next byte of the
                           / users buffer
        inc     u.base / increment the pointer to point to the next byte
```

```
                        / in users buffer
        inc     u.nread / increment the number of bytes read
        dec     u.count / decrement the number of bytes to be read
        bne     1f / any more bytes to read?; yes, branch
        mov     (sp)+,r0 / no, do a non-local return to the caller of
                        / 'readi' by;
ret: / (1) pop the return address off the stack into r0
        mov     (sp)+,r1 / (2) pop the i-number off the stack into r1
1:
        clr     *$ps / clear processor status
        rts     r0 / return to address currently on top of stack

writei:
        clr     u.nread / clear the number of bytes transmitted during
                        / read or write calls
        tst     u.count / test the byte count specified by the user
        bgt     1f / any bytes to output; yes, branch
        rts     r0 / no, return - no writing to do
1:
        mov     r1,-(sp) / save the i-node number on the stack
        cmp     r1,$40. / does the i-node number indicate a special file?
        bgt     dskw / no, branch to standard file output
        asl     r1 / yes, calculate the index into the special file
        jmp     *1f-2(r1) / jump table and jump to the appropriate routine
1:
        wtty    / tty
        wppt    / ppt
        wmem    / mem
        wrf0    / rf0
        wrk0    / rk0
        wtap    / tap0
        wtap    / tap1
        wtap    / tap2
        wtap    / tap3
        wtap    / tap4
        wtap    / tap5
        wtap    / tap6
        wtap    / tap7
        xmtt    / tty0
        xmtt    / tty1
        xmtt    / tty2
        smtt    / tty3
        xmtt    / tty4
        xmtt    / tty5
        xmtt    / tty6
        xmtt    / tty7
/       wlpr / lpr

wtty:
        jsr     r0,cpass / get next character from user buffer area; if
                        / none go to return address in syswrite
        tst     r1 / is character = null
        beq     wtty / yes, get next character
1:
        mov     $240,*$ps / no, set processor priority to five
```

```
        cmpb    cc+1,$20. / is character count for console tty greater
                          / than 20
        bhis    2f / yes; branch to put process to sleep
        jsr     r0,putc; 1 / find place in freelist to assign to console
                           / tty and
                br 2f / place character in list; if none available
                      / branch to put process to sleep
        jsr     r0,startty / attempt to output character on tty
        br      wtty
2:
        mov     r1,-(sp) / place character on stack
        jsr     r0,sleep; 1 / put process to sleep
        mov     (sp)+,r1 / remove character from stack
        br      1b / try again to place character in clist and output

wppt:
        jsr     r0,cpass / get next character from user buffer area,
                         / if none return to writei's calling routine
        jsr     r0,pptoc / output character on ppt
        br      wppt

/wlpr:
/       jsr     r0,cpass
/       cmp     r0,$'a
/       blo     1f
/       cmp     r1,$'z
/       bhi     1f
/       sub     $40,r1
/1:
/       jsr     r0,lptoc
/       br      wlpr

wmem: / transfer characters from a user area of core to memory file
        jsr     r0,cpass / get next character from users area of core and
                         / put it in r1
        mov     r1,-(sp) / put character on the stack
        mov     *u.fofp,r1 / save file offset in r1
        inc     *u.fofp / increment file offset to point to next available
                        / location in file
        movb    (sp)+,(r1) / pop char off stack, put in memory loc assigned
                           / to it
        br      wmem / continue
1:
        jmp     error / ?

dskw: / write routine for non-special files
        mov     (sp),r1 / get an i-node number from the stack into r1
        jsr     r0,iget / write i-node out (if modified), read i-node 'r1'
                        / into i-node area of core
        mov     *u.fofp,r2 / put the file offset [(u.off) or the offset in
                           / the fsp entry for this file] in r2
        add     u.count,r2 / no. of bytes to be written + file offset is
                           / put in r2
        cmp     r2,i.size / is this greater than the present size of
                          / the file?
        blos    1f / no, branch
```

```
        mov     r2,i.size / yes, increase the file size to file offset +
                          / no. of data bytes
        jsr     r0,setimod / set imod=1 (i.e., core inode has been
                           / modified), stuff time of modification into
                           / core image of i-node
1:
        jsr     r0,mget / get the block no. in which to write the next data
                        / byte
        bit     *u.fofp,$777 / test the lower 9 bits of the file offset
        bne     2f / if its non-zero, branch; if zero, file offset = 0,
                   / 512, 1024,...(i.e., start of new block)
        cmp     u.count,$512. / if zero, is there enough data to fill an
                              / entire block? (i.e., no. of
        bhis    3f / bytes to be written greater than 512.?  Yes, branch.
                   / Don't have to read block
2: / in as no past info. is to be saved (the entire block will be
   / overwritten).
        jsr     r0,dskrd / no, must retain old info.. Hence, read block 'r1'
                         / into an I/O buffer
3:
        jsr     r0,wslot / set write and inhibit bits in I/O queue, proc.
                         / status=0, r5 points to 1st word of data
        jsr     r0,sioreg / r3 = no. of bytes of data, r1 = address of data,
                          / r2 points to location in buffer in which to
                          / start writing data
2:
        movb    (r1)+,(r2)+ / transfer a byte of data to the I/O buffer
        dec     r3 / decrement no. of bytes to be written
        bne     2b / have all bytes been transferred?  No, branch
        jsr     r0,dskwr / yes, write the block and the i-node
        tst     u.count / any more data to write?
        bne     1b / yes, branch
        jmp     ret / no, return to the caller via 'ret'

cpass: / get next character from user area of core and put it in r1
        tst     u.count / have all the characters been transferred (i.e.,
                        / u.count, # of chars. left
        beq     1f / to be transferred = 0?) yes, branch
        dec     u.count / no, decrement u.count
        movb    *u.base,r1 / take the character pointed to by u.base and
                           / put it in r1
        inc     u.nread / increment no. of bytes transferred
        inc     u.base / increment the buffer address to point to the
        rts     r0 / next byte
1:
        mov     (sp)+,r0 / put return address of calling routine into r0
        mov     (sp)+,r1 / i-number in r1
        rts     r0 / non-local return

sioreg:
        mov     *u.fofp,r2 / file offset (in bytes) is moved to r2
        mov     r2,r3 / and also to r3
        bis     $177000,r3 / set bits 9,...,15. of file offset in r3
        bic     $!777,r2 / calculate file offset mod 512.
        add     r5,r2 / r2 now points to 1st byte in system buffer where
                      / data is to be placed
```

```
        mov     u.base,r1 / address of data is in r1
        neg     r3 / 512 - file offset (mod512.) in r3 (i.e., the number
                   / of free bytes in the file block
        cmp     r3,u.count / compare this with the number of data bytes to
                           / be written to the file
        blos    2f / if less than branch.  Use the number of free bytes
                   / in the file block as the number to be written
        mov     u.count,r3 / if greater than, use the number of data bytes
                           / as the number to be written
2:
        add     r3,u.nread / r3 + number of bytes xmitted during write is
                           / put into u.nread
        sub     r3,u.count / u.count = no. of bytes that still must be
                           / written or read
        add     r3,u.base / u.base points to the 1st of the remaining data
                          / bytes
        add     r3,*u.fofp / new file offset = number of bytes done + old
                           / file offset
        rts     r0
```

```
/ u7 -- unix

canon:
        mov     r5,r1 / move tty buffer address to r1
        add     $10.,r1 / add 10 to get start of data
        mov     r1,4(r5) / canp = 10(r5) / move buffer addr + 10 to 3rd
                         / word in buffer (char. pointer)
        clr     2(r5) / ncan / clear 2nd word in buffer, 0 char. count
1:
        jsr     r0,*(r0) / jump to arg get char off Q of characters, sleep
                         / if none
        jsr     r0,cesc; 100 / test for @ (kill line)
                br canon / character was @ so start over
        jsr     r0,cesc; 43 / test for # (erase last char. typed)
                br 1b / character was #, go back
        cmp     r1,$4 / is char eot?
        beq     1f / yes, reset and return
        movb    r1,*4(r5) / no, move char to address in 3rd word of buffer
                          / (char. pointer)
        inc     2(r5) / increment 2nd word (char. count)
        inc     4(r5) / increment 3rd word (char. pointer)
        cmp     r1,$'\n / is char = newline
        beq     1f / yes, 1f
        cmp     2(r5),$120. / is byte count greater than or equal to 120
        bhis    1f / yes, 1f
        br      1b / no, get another char off the Q
1: / get here if line is full, a new line has been received or an eot
   / has been received
        mov     r5,r1 / move buffer address to r1
        add     $10.,r1 / add 10
        mov     r1,4(r5) / canp = 10(r5) / reset char pointer
        tst     (r0)+ / skip over argument
        rts     r0 / return

cesc: / test for erase or kill char
        cmp     r1,(r0)+ / char in r1 = erase or kill character?
        bne     1f / no, skip return
        tst     2(r5) / yes, is char. count = 0
        beq     2f / yes, don't skip return
        dec     2(r5) / no, decrement char count
        dec     4(r5) / decrement character pointer
        cmpb    *4(r5),$'\\/ was previous character a "\"
        bne     2f / no, don't skip
1:
        tst     (r0)+ / yes, skip
2:
        rts     r0 / return

ttych: / get characters from Q of characters inputted ~~to~~ from tty
        mov     $240,*$ps / set processor priority to 5
        jsr     r0,getc; 0 / takes char. off clist and puts it in r1
                br 1f / list is empty, go to sleep
        clr     *$ps / clear process priority
        rts     r0 / return
1: / list is empty
```

```
        mov     r5,-(sp) / save r5
        jsr     r0,sleep; 0 / put process to sleep in input wait channel
        mov     (sp)+,r5 / restore r5
        br      ttych / try again

pptic: / paper tape input control
        mov     $240,*$ps / set processor priority to five
        cmpb    cc+2,$30. / is character count for paper tape input in
                        / clist greater than or equal to 30
        bhis    1f / yes, branch
        bit     *$prs,$104200 / is there either an error, an unread char
                          / in buffer, or reader busy
        bne     1f / yes, don't enable reader
        inc     *$prs / set reader enable bit
1:
        jsr     r0,getc; 2 / get next character in clist for ppt input and
                br 1f / place in r1; if no char in clist for ppt input
                      / branch
        tst     (r0)+ / pop stack so that return will be four locations past
                      / subroutine call
2:
        clr     *$ps / set process priority equal to zero
        rts     r0 / return
1:
        cmpb    pptiflg,$6 / does pptiflg indicate file "not closed"
        beq     2b / yes, return to calling routine at instruction
                   / immediately following jsr
        jsr     r0,sleep; 2 / no, all characters to be read in not yet in
                         / clist, put process to sleep
        br      pptic

pptoc: / paper tape output control
        mov     $240,*$ps / set processor priority to five
        cmpb    cc+3,$50. / is character count for paper tape output in
                        / clist greater than or equal to 50
        bhis    1f / yes
        jsr     r0,putc; 3 / find place in freelist to assign ppt output
                         / and place
                br 1f / character in list; if none available branch to put
                      / process to sleep
        jsr     r0,starppt / try to output character
        clr     *$ps / clear processor priority
        rts     r0 / return
1:
        mov     r1,-(sp) / place character on stack
        jsr     r0,sleep; 3 / put process to sleep
        mov     (sp)+,r1 / place character in r1
        br      pptoc / try again to place character in clist and output

/lptoc: / line printer output control
/       mov     $240,*$ps / set processor priority to five
/       cmpb    cc+5,$200. / is character count for printer greater than or
                         / equal to 200
/       bhis    1f / yes
/       jsr     r0,putc; 5 / find place in freelist to assign to printer
                         / and place
```

```
                br 1f / char in list, if none available branch to put
                      / process to sleep
/       jsr     r0,starlpt / try to output character
/       clr     *$ps / set processor priority = 0
/       rts     r0 / return
/1:
/       mov     r1,-(sp) / place character on stack
/       jsr     r0,sleep; 5 / put process to sleep
/       mov     (sp)+,r1 / place character on stack
/       br      lptoc

getc: / get a character off character list
        mov     (r0)+,r1 / put argument in getc call in r1 (char list id)
        jsr     r0,get
                br 1f / empty char list return
        decb    cc(r1) / decrement number of char in char list
        mov     $-1,r1 / load minus 1 in r1
        jsr     r0,put / put char back on free list
        movb    clist-2(r2),r1 / put char in r1
        tst     (r0)+ / bump r0 for non blank char list return
1:
        rts     r0

putc:
        mov     r1,-(sp) / save char on stack
        mov     $-1,r1 / put free list list id in r1
        jsr     r0,get / take char off free list / clist slot taken
                       / identified by r2
                br 1f / branch when no chars in free list
        mov     (r0)+,r1 / put putc call arg in r1 (i.e., list identifier)
        incb    cc(r1) / increment character count for list (r1)
        jsr     r0,put / put clist entry on list
        movb    (sp),clist-2(r2) / put character in new entry
1:
        tst     (r0)+
        mov     (sp)+,r1
        rts     r0

get:
        movb    cf+1(r1),r2 / move current first char offset to r2
        beq     2f / no characters in char list
        tst     (r0)+ / bump r0, second return
        cmpb    r2,cl+1(r1) / r2 equal to last char offset
        beq     1f / yes, (i.e., entire char list scanned), branch to 1f
        bic     $!377,r2 / clear bits 8-15 in r2
        asl     r2 / multiply r2 by 2 to get offset in clist
        movb    clist-1(r2),cf+1(r1) / move next char in list pointer to
                                     / first char offset ptr
        br      2f
1:
        clrb    cf+1(r1) / clear first char clist offset
        clrb    cl+1(r1) / clear last char clist offset
        bic     $!377,r2 / zero top half of r2
        asl     r2 / multiply r2 by 2
2:
        rts     r0
```

```
put:
        asr     r2 / divide r2 by 2; r2 is offset in clist
        mov     r2,-(sp) / save r2 on stack
        movb    cl+1(r1),r2 / move offset of last char in list (r1) into r2
        beq     1f / offset = 0 then go to 1f (i.e., start a new list)
        bic     $!377,r2 / zero top half of r2
        asl     r2 / multiply offset by 2, r2 now has offset in clist
        movb    (sp),clist-1(r2) / link new list entry to current last
                                 / entry in list (r1)
        br      2f
1:
        movb    (sp),cf+1(r1) / put new list entry offset into first char
                              / offset of list (r1)
2:
        mov     (sp)+,r2 / pop stack into r2; offset of new list
                         / entry in r2
        movb    r2,cl+1(r1) / make new list entry the last entry in list
                            / (r1)
        asl     r2 / multiply r2 by 2; r2 has clist offset for new
                   / list entry
        rts     r0

iopen: / open file whose i-number is in r1
        tst     r1 / write or read access?
        blt     2f / write, go to 2f
        jsr     r0,access; 2 / get inode into core with read access
        cmp     r1,$40. / is it a special file
        bgt     3f / no, 3f
        mov     r1,-(sp) / yes, figure out
        asl     r1
        jmp     *1f-2(r1) / which one and transfer to it
1:
        otty    / tty
        oppt    / ppt
        sret    / mem
        sret    / rf0
        sret    / rk0
        sret    / tap0
        sret    / tap1
        sret    / tap2
        sret    / tap3
        sret    / tap4
        sret    / tap5
        sret    / tap6
        sret    / tap7
        ocvt    / tty0
        ocvt    / tty1
        ocvt    / tty2
        ocvt    / tty3
        ocvt    / tty4
        ocvt    / tty5
        ocvt    / tty6
        ocvt    / tty7
        error / crd
```

```
2: / check open write access
        neg     r1 / make inode number positive
        jsr     r0,access; 1 / get inode in 0 core
        bit     $40000,i.flgs / is it a directory?
        bne     2f / yes, transfer (error)
        cmp     r1,$40. / no, is it a special file?
        bgt     3f / no, return
        mov     r1,-(sp) / yes
        asl     r1
        jmp     *1f-2(r1) / figure out which special file it is
                          / and transfer
1:
        otty    / tty
        leadr   / ppt
        sret    / mem
        sret    / rf0
        sret / rk0
        sret    / tap0
        sret    / tap1
        sret    / tap2
        sret    / tap3
        sret    / tap4
        sret    / tap5
        sret    / tap6
        sret    / tap7
        ocvt    / tty0
        ocvt    / tty1
        ocvt    / tty2
        ocvt    / tty3
        ocvt    / tty4
        ocvt    / tty5
        ocvt    / tty6
        ocvt    / tty7
/       ejec / lpr

otty: / open console tty for reading or writing
        mov     $100,*$tks / set interrupt enable bit (zero others) in
                           / reader status reg
        mov     $100,*$tps / set interrupt enable bit (zero others) in
                           / punch status reg
        mov     tty+[ntty*8]-8+6,r5 / r5 points to the header of the
                                    / console tty buffer
        incb    (r5) / increment the count of processes that opened the
                     / console tty
        tst     u.ttyp / is there a process control tty (i.e., has a tty
                       / buffer header
        bne     sret / address been loaded into u.ttyp yet)?  Yes, branch
        mov     r5,u.ttyp / no, make the console tty the process control
                          / tty
        br      sret / ?

sret:
        clr     *$ps / set processor priority to zero
        mov     (sp)+,r1 / pop stack to r1
3:
        rts     r0
```

```
oppt: / open paper tape for reading or writing
        mov     $100,*$prs / set reader interrupt enable bit
        tstb    pptiflg / is file already open
        bne     2f / yes, branch
1:
        mov     $240,*$ps / no, set processor priority to 5
        jsr     r0,getc; 2 / remove all entries in clist
                br .+4 / for paper tape input and place in free list
        br      1b
        movb    $2,pptiflg / set pptiflg to indicate file just open
        movb    $10.,toutt+1 / place 10 in paper tape input tout entry
        br      sret
2:
        jmp     error / file already open

iclose: / close file whose i-number is in r1
        tst     r1 / test i-number
        blt     2f / if neg., branch
        cmp     r1,$40. / is it a special file
        bgt     3b / no, return
        mov     r1,-(sp) / yes, save r1 on stack
        asl     r1
        jmp     *1f-2(r1) / compute jump address and transfer
1:
        ctty    / tty
        cppt    / ppt
        sret    / mem
        sret    / rf0
        sret    / rk0
        sret    / tap0
        sret    / tap1
        sret    / tap2
        sret    / tap3
        sret    / tap4
        sret    / tap5
        sret    / tap6
        sret    / tap7
        ccvt    / tty0
        ccvt    / tty1
        ccvt    / tty2
        ccvt    / tty3
        ccvt    / tty4
        ccvt    / tty5
        ccvt    / tty6
        ccvt    / tty7
        error / crd

2: / negative i-number
        neg     r1 / make it positive
        cmp     r1,$40. / is it a special file
        bgt     3b / no, return
        mov     r1,-(sp)
        asl     r1 / yes, compute jump address and transfer
        jmp     *1f-2(r1)
1:
```

```
        ctty   / tty
        leadr  / ppt
        sret   / mem
        sret   / rf0
        sret   / rk0
        sret   / tap0
        sret   / tap1
        sret   / tap2
        sret   / tap3
        sret   / tap4
        sret   / tap5
        sret   / tap6
        sret   / tap7
        ccvt   / tty0
        ccvt   / tty1
        ccvt   / tty2
        ccvt   / tty3
        ccvt   / tty4
        ccvt   / tty5
        ccvt   / tty6
        ccvt   / tty7
/       ejec / lpr

ctty: / close console tty
        mov    tty+[ntty*8]-8+6,r5 / point r5 to the console tty buffer
        decb   (r5) / dec number of processes using console tty
        br     sret / return via sret

cppt: / close paper tape
        clrb   pptiflg / set pptiflg to indicate file not open
1:
        mov    $240.*$ps / set process or priority to 5
        jsr    r0,getc; 2 / remove all ppt input entries from clist
                          / and assign to free list
               br sret
        br     1b

/ejec:
/       mov    $100,*$lps / set line printer interrupt enable bit
/       mov    $14,r1 / 'form feed' character in r1 (new page).
/       jsr    r0,lptoc / space the printer to a new page
/       br     sret / return to caller via 'sret'

leadr: / produce paper tape leader
        mov    $100,*$pps / set paper tape punch interrupt enable
        mov    $100.,-(sp) / 101. characters of 'nul' will be output as
                           / leader
1:
        clr    r1 / r1 contains a 'nul' character
        jsr    r0,pptoc / output the 'nul' character
        dec    (sp)
        bge    1b / last leader character output?  no, branch
        tst    (sp)+ / bump stack pointer
        br     sret / return to caller via 'sret'

sysmount: / mount file system; args special; name
```

```
        jsr     r0,arg2 / get arguments special and name
        tst     mnti / is the i-number of the cross device file zero?
        bne     errora / no, error
        jsr     r0,getspl / get special files device number in r1
        mov     (sp)+,u.namep / put the name of file to be placed on the
                           / device
        mov     r1,-(sp) / save the device number
        jsr     r0,namei / get the i-number of the file
                br errora
        mov     r1,mnti / put it in mnti
1:
        tstb    sb1+1 / is 15th bit of I/O queue entry for dismountable
                      / device set?
        bne     1b / (inhibit bit) yes, skip writing
        mov     (sp),mntd / no, put the device number in mntd
        movb    (sp),sb1 / put the device number in the lower byte of the
                         / I/O queue entry
        mov     (sp)+,cdev / put device number in cdev
        bis     $2000,sb1 / set the read bit
        jsr     r0,ppoke / read in entire file system super block
1:
        tstb    sb1+1 / done reading?
        bne     1b / no, wait
        br      sysreta / yes

sysumount: / special dismount file system
        jsr     r0,arg; u.namep / point u.namep to special
        jsr     r0,getspl / get the device number in r1
        cmp     r1,mntd / is it equal to the last device mounted?
        bne     errora / no error
1:
        tstb    sb1+1 / yes, is the device still doing I/O (inhibit
                      / bit set)?
        bne     1b / yes, wait
        clr     mntd / no, clear these
        clr     mnti
        br      sysreta / return

getspl: / get device number from a special file name
        jsr     r0,namei / get the i-number of the special file
                br errora / no such file
        sub     $4,r1 / i-number-4 rk=1,tap=2+n
        ble     errora / less than 0?  yes, error
        cmp     r1,$9. / greater than 9  tap 7
        bgt     errora / yes, error
        rts     r0 / return with device number in r1

errora:
        jmp     error / see 'error' routine

sysreta:
        jmp     sysret / see 'sysret' routine
```

```
/ u8 -- unix

rtap: / read from the dec tape
        asr     r1 / divide the i-number by 2
        sub     $4.,r1 / (i-number/2)-4  r1
        mov     r1,cdev / cdev now has device number
        jsr     r0,bread; 578. / read in block thats in *u.fofp

wtap:
        asr     r1 / divide i-number by 2
        sub     $4.,r1 / r1 = i-number minus 4
        mov     r1,cdev / this is used as the device number
        jsr     r0,bwrite; 578. / write block (u.fofp) on dec tape
                                / Maximum

rrk0:
        mov     $1,cdev / set current device to i., disk
        jsr     r0,bread; 4872. / read block from disk (maximum block
                                / number allowed on  evice is 4872.)
                                / - (u.fofp) contains block number

wrk0:
        mov     $1,cdev / set current device to 1; disk
        jsr     r0,bwrite; 4872. / write block (u.fofp) on disk

rrf0:
        clr     cdev / set current device to 0., fixed head disk
        jsr     r0,bread; 1024. / read block (u.fofp) from fixed head
                                / disk (max. block number allowed on
                                / device is 1024.)

wrf0:
        clr     cdev / set current device to 0., fixed head disk
        jsr     r0,bwrite; 1024. / write block '(u.fofp)' on fixed head
                                 / disk

bread: / read a block from a block structured device
        jsr     r0,tstdeve / error on special file I/O (only works on
                           / tape)
        mov     *u.fofp,r1 / move block number to r1
        mov     $2.-cold,-(sp) / "2-cold" to stack
1:
        cmp     r1,(r0) / is this block # greater than or equal to
                        / maximum block # allowed on device
        bhis    1f / yes, 1f (error)
        mov     r1,-(sp) / no, put block # on stack
        jsr     r0,preread / read in the block into an I/O buffer
        mov     (sp)+,r1 / return block # to r1
        inc     r1 / bump block # to next consecutive block
        dec     (sp) / "2-1-cold" on stack
        bgt     1b / 2-1-cold = 0?  No, go back and read in next block
1:
        tst     (sp)+ / yes, pop stack to clear off cold calculation
        mov     *u.fofp,r1 / restore r1 to initial value of the
                           / block #
```

```
        cmp     r1,(r0)+ / block # greater than or equal to maximum
                         / block number allowed
        bhis    error10 / yes, error
        inc     *u.fofp / no, *u.fofp has next block number
        jsr     r0,preread / read in the block whose number is in r1
        bis     $40000,(r5) / set bit 14 of the 1st word of the I/O
                            / buffer
1:
        bit     $22000,(r5) / are 10th and 13th bits set (read bits)
        beq     1f / no
        cmp     cdev,$1 / disk or drum?
        ble     2f / yes
        tstb    uquant / is the time quantum = 0?
        bne     2f / no, 2f
        mov     r5,-(sp) / yes, save r5 (buffer address)
        jsr     r0,sleep; 31. / put process to sleep in channel 31 (tape)
        mov     (sp)+,r5 / restore r5
        br      1b / go back
2: / drum or disk
        jsr     r0,idle; s.wait+2 / wait
        br      1b
1: / 10th and 13th bits not set
        bic     $40000,(r5) / clear bit 14
        jsr     r0,tstdeve / test device for error (tape)
        add     $8,r5 / r5 points to data in I/O buffer
        jsr     r0,dioreg / do bookkeeping on u.count etc.
1: / r5 points to beginning of data in I/O buffer, r2 points to beginning
   / of users data
        movb    (r5)+,(r2)+ / move data from the I/O buffer
        dec     r3 / to the user's area in core starting at u.base
        tst     u.count / done
        beq     1f / yes, return
        tst     -(r0) / no, point r0 to the argument again
        br      bread / read some more
1:
        mov     (sp)+,r0 / jump to routine that called readi
        jmp     ret

bwrite: / write on block structured device
        jsr     r0,tstdeve / test the device for an error
        mov     *u.fofp,r1 / put the block number in r1
        cmp     r1,(r0)+ / does block number exceed maximum allowable #
        bhis    error10 / yes, error
        inc     *u.fofp / no, increment block number
        jsr     r0,wslot / get an I/O buffer to write into
        jsr     r0,dioreg / do the necessary bookkeeping
1: / r2 points to the users data; r5 points to the I/O buffers data area
        movb    (r2)+,(r5)+ /    ; r3, has the byte count
        dec     r3 / area to the I/O buffer
        bne     1b
        jsr     r0,dskwr / write it out on the device
        tst     u.count / done
        beq     1f / yes, 1f
        tst     -(r0) / no, point r0 to the argument of the call
        br      bwrite / go back and write next block
1:
```

```
        mov     (sp)+,r0 / return to routine that called writei
        jmp     ret

tstdeve: / check whether permanent error has occured on special file
         / I/O
        mov     cdev,r1 / only works on tape; r1 has device #
        tstb    deverr(r1) / test error bit of device
        bne     1f / error
        rts     r0 / device okay
1:
        clrb    deverr(r1) / clear error

error10:
        jmp     error / see 'error' routine

dioreg:
        mov     u.count,r3 / move char count to r3
        cmp     r3,$512. / more than 512. char?
        blos    1f / no, branch
        mov     $512.,r3 / yes, just take 512.
1:
        mov     u.base,r2 / put users base in r2
        add     r3,u.nread / add the number to be read to u.nread
        sub     r3,u.count / update count
        add     r3,u.base / update base
        rts     r0 / return

preread:
        jsr     r0,bufaloc / get a free I/O buffer (r1 has block number)
                br 1f / branch if block already in a I/O buffer
        bis     $2000,(r5) / set read bit (bit 100 in I/O buffer)
        jsr     r0,poke / perform the read
1:
        clr     *$ps / ps = 0
        rts     r0

dskrd:
        jsr     r0,bufaloc / shuffle off to bufaloc; get a free I/O buffer
                br 1f
        bis     $2000,(r5) / set bit 10 of word 1 of I/O queue entry
                           / for buffer
        jsr     r0,poke / just assigned in bufaloc; bit 10=1 says read
1:
        clr     *$ps
        bit     $22000,(r5) / if either bits 10, or 13 are 1; jump to idle
        beq     1f
        jsr     r0,idle; s.wait+2
        br      1b
1:
        add     $8,r5 / r5 points to first word of data in block just read
                      / in
        rts     r0

wslot:
        jsr     r0,bufaloc / get a free I/O buffer; pointer to first
                br 1f / word in buffer in r5
```

```
1:
        bit     $22000,(r5) / check bits 10, 13 (read, waiting to read)
                            / of I/O queue entry
        beq     1f / branch if 10, 13 zero (i.e., not reading, or waiting
                   / to read)
        jsr     r0,idle; s.wait+2 / if buffer is reading or writing to read,
                                  / idle
        br      1b / till finished
1:
        bis     $101000,(r5) / set bits 9, 15 in 1st word of I/O queue
                             / (write, inhibit bits)
        clr     *$ps / clear processor status
        add     $8,r5 / r5 points to first word in data area for this
                      / block
        rts     r0

dskwr:
        bic     $100000,*bufp / clear bit 15 of I/O queue entry at
                              / bottom of queue

ppoke:
        mov     $340,*$ps
        jsr     r0,poke
        clr     *$ps
        rts     r0

poke:
        mov     r1,-(sp)
        mov     r2,-(sp)
        mov     r3,-(sp)
        mov     $bufp+nbuf+nbuf+6,r2 / r2 points to highest priority I/O
                                     / queue pointer
1:
        mov     -(r2),r1 / r1 points to an I/O queue entry
        bit     $3000,(r1) / test bits 9 and 10 of word 1 of I/O queue
                           / entry
        beq     2f / branch to 2f if both are clear
        bit     $130000,(r1) / test bits 12, 13, and 15
        bne     2f / branch if any are set
        movb    (r1),r3 / get device id
        tstb    deverr(r3) / test for errors on this device
        beq     3f / branch if no errors
        mov     $-1,2(r1) / destroy associativity
        clrb    1(r1) / do not do I/O
        br      2f
3:
        cmpb    r3,$1 / device id = 1; device is disk
        blt     prf / device id = 0; device is drum
        bgt     ptc / device id greater than or equal to 1; device is
                    / dec tape
        bit     $2,active / test disk busy bit
        bne     2f / branch if bit is set
        bis     $2,active / set disk busy bit
        mov     r1,rkap / rkap points to current I/O queue entry for disk
        mov     2(r1),mq / put physical block number in mq
        mov     $12.,div / divide physical block number by 12.
```

```
        mov     $rkda+2,r3 /
        mov     ac,-(sp) / put remainder from divide on stack; gives
                         / sector number
        mov     $4,lsh / shift quotient 4 bits, to align with cyl and surf
                       / bits in rkda
        bis     mq,(sp) / or mq with sector; gives total disk address
        br      3f
prf: / drum
        bit     $1,active / test drum busy bit
        bne     2f / branch if bit is set
        bis     $1,active / set drum busy bit
        mov     r1,rfap / rfap points to current I/O queue entry for drum
        mov     $dae+2,r3
        clr     -(sp)
        movb    2(r1),1(sp) / move low byte of physical block number into
                            / high byte of stack
        clr     -(sp) / word
        movb    3(r1),(sp) / move high byte of physical block number into
                           / low byte of stack
        mov     (sp)+,-(r3) / load dae with high byt  of physical block
                            / number
3:
        mov     (sp)+,-(r3) / load rkda register; load dar register
        mov     6(r1),-(r3) / load bus address register
        mov     4(r1),-(r3) / load word count register
        mov     $103,-(sp) / 103 indicates write operation when loaded
                           / in csr
        bit     $2000,(r1) / if bit 10 of word 1 of I/O queue entry is
                           / a one
        beq     3f / then read operation is indicated
        mov     $105,(sp) / 105 indicates read operation
3:
        mov     (sp)+,-(r3) / load csr with interrupt enabled, command, go
        br      seta
ptc: / tape I/O
        bit     $4,active
        bne     2f
        mov     tccm,r3
        swab    r3
        bic     $!7,r3
        add     $2,r3
        cmpb    r3,(r1)
        beq     3f
        movb    $1,tccm / stop transport if not same unit
3:
        bis     $4,active
        mov     r1,tcap
        mov     $20.,tcerrc
        mov     $tape1,tcstate
        movb    (r1),r3 / device
        sub     $2,r3 / now unit
        swab    r3
        bis     $103,r3 / now rbn,for,unit,ie
        mov     r3,tccm
seta: / I/O queue bookkeeping; set read/write waiting bits.
        mov     (r1),r3 / move word 1 of I/O queue entry into r3
```

```
        bic     $!3000,r3 / clear all bits except 9 and 10
        bic     $3000,(r1) / clear only bits 9 and 10
        rol     r3
        rol     r3
        rol     r3
        bis     r3,(r1) / or old value of bits 9 and 10 with bits 12
                        / and 13
2:
        cmp     r2,$bufp / test to see if entire I/O queue has been
                         / scanned
        bhi     1b
        mov     (sp)+,r3
        mov     (sp)+,r2
        mov     (sp)+,r1
        rts     r0

bufaloc:
        mov     r2,-(sp) / save r2 on stack
        mov     $340,*$ps / set processor priority to 7
1:
        clr     -(sp) / vacant buffer
        mov     $bufp,r2 / bufp contains pointers to I/O queue entrys
                         / in buffer area
2:
        mov     (r2)+,r5 / move pointer to word 1 of an I/O queue entry
                         / into r5
        bit     $173000,(r5) / lock+keep+active+outstanding
        bne     3f / branch when any of bits 9,10,12,13,14,15 are set
                   / (i.e., buffer busy)
        mov     r2,(sp) / save pointer to last non-busy buffer found
                        / points to word 2 of I/O queue entry)
3:
        cmpb    (r5),cdev / is device in I/O queue entry same as current
                          / device
        bne     3f
        cmp     2(r5),r1 / is block number in I/O queue entry, same as
                         / current block number
        bne     3f
        tst     (sp)+ / bump stack pointer
        br      1f / use this buffer
3:
        cmp     r2,$bufp+nbuf+nbuf
        blo     2b / go to 2b if r2 less than bufp+nbuf+nbuf (all
                   / buffers not checked)
        mov     (sp)+,r2 / once all bufs are examined move pointer to
                         / last free block
        bne     2f / if (sp) is non zero, i.e., if a free buffer is
                   / found branch to 2f
        jsr     r0,idle; s.wait+2 / idle if no free buffers
        br      1b
2:
        tst     (r0)+ / skip if warmed over buffer
1:
        mov     -(r2),r5 / put pointer to word 1 of I/O queue entry in r5
        movb    cdev,(r5) / put current device number in I/O queue entry
        mov     r1,2(r5) / move block number into word 2 of I/O queue
```

```
                        / entry
1:
       cmp     r2,$bufp / bump all entrys in bufp and put latest assigned
       blos    1f / buffer on the top (this makes if the lowest priority)
       mov     -(r2),2(r2) / job for a particular device
       br      1b
1:
       mov     r5,(r2)
       mov     (sp)+,r2 / restore r2
       rts     r0

tape: / dec tape interrupt
       jsr     r0,setisp / save registers and clockp on stack
       mov     tcstate,r3 / put state of dec tape in r3
       jsr     r0,trapt; tccm; tcap; 4 / busy bit
               mov r3,pc / device control status register
                         / if no errors, go to device state (an address)

taper: / dec tape error
       dec     tcerrc / decrement the number of errors
       bne     1f / if more than 1 branch
       movb    1(r2),r3 / r2+1 points to command register upper byte
       bic     $!7,r3 / clear all but bits 8-10 (Unit Selection)
       incb    deverr+2(r3) / set error bit for this tape unit
       br      tape3
1: / more than 1 error
       bit     $4000,(r2) / direction of tape
       beq     1f / if forward go to 1f
       bic     $4000,(r2) / reverse, set to forward
       mov     $tape1,tcstate / put tape 1 in the state
       br      0f
1: / put tape in reverse
       bis     $4000,(r2) / set tape to reverse direction
       mov     $tape2,tcstate / put tape 2 as the state
0:
       bis     $4,active / check active bit of tape
       movb    $103,(r2) / set read function and interrupt enable
       br      4f / go to retisp
tape1: / read bn forward
       mov     $tcdt,r0 / move address of data register to r0
       cmp     (r0),2(r1) / compare block addresses
       blt     0b / if 1t, keep moving
       bgt     taper / if gt, reverse
       mov     6(r1),-(r0) / put bus address in tcba
       mov     4(r1),-(r0) / put word count in tcwc
       mov     $115,-(sp) / put end interrupt enable
       bit     $20000,(r1) / is "waiting to read bit" of I/O queue set?
       beq      1f / no, 1f
       mov     $105,(sp) / yes, put and interrupt enable
1:
       movb    (sp)+,(r2) / move function into command register (tccm)
       bis     $4,active / set active bit
       mov     $tape3,tcstate / get ready for I/O transfer
       br      4f / go to retisp (rti)

tape2: / read bn bakasswards
```

```
        mov     tcdt,r0 / r0 has contents of data register
        add     $3,r0 / overshoot
        cmp     r0,2(r1)
        bgt     0b / if gt keep reading
        br      taper / else reverse

tape3: / I/O transfer
        bic     $30000,(r1) / clear bits 12 and 13 of I/O queue entry
        jsr     r0,poke / do the I/O
        bit     $4,active / still busy see if pick up r-ahead, w-behind
        bne     1f / yes
        movb    $1,(r2) / no, indicate too bad
1:
        jsr     r0,wakeup; runq; 31. / wait up
        br      4f / retisp

drum: / interrupt handler
        jsr     r0,setisp / save r1,r2,r3, and clockp on the stack
        jsr     r0,trapt; dcs; rfap; 1 / check for stray interrupt or
                                       / error
                br 3f / no, error
        br      2f / error

disk:
        jsr     r0,setisp / save r1,r2,r3, and clockp on the stack
        jmp     *$0f
0:
        jsr     r0,trapt; rkcs; rkap; 2
                br 3f / no, errors
        mov     $115,(r2) / drive reset, errbit was set
        mov     $1f,0b-2 / next time jmp *$0f is executed jmp will be
                         / to 1f
        br      4f
1:
        bit     $20000,rkcs
        beq     4f / wait for seek complete
        mov     $0b,0b-2
        mov     rkap,r1
2:
        bit     $3000,(r1) / are bits 9 or 10 set in the 1st word of
                           / the disk buffer
        bne     3f / no, branch ignore error if outstanding
        inc     r1
        asr     (r1)
        asr     (r1)
        asr     (r1) / reissue request
        dec     r1
3:
        bic     $30000,(r1) / clear bits 12 and 13 in 1st word of buffer
        mov     ac,-(sp)
        mov     mq,-(sp) / put these on the stack
        mov     sc,-(sp)
        jsr     r0,poke
        mov     (sp)+,sc
        mov     (sp)+,mq / pop them off stack
        mov     (sp)+,ac
```

```
4:
        jmp     retisp / u4-3

trapt:                                          / r2 points to the
        mov     (r0)+,r2 / device control register
        mov     *(r0)+,r1 / transaction pointer points to buffer
        tst     (sp)+
        tstb    (r2) / is ready bit of dcs set?
        bge     4b / device still active so branch
        bit     (r0),active / was device busy?
        beq     4b / no, stray interrupt
        bic     (r0)+,active / yes, set active to zero
        tst     (r2) / test the err(bit is) of dcs
        bge     2f / if no error jump to 2f
        tst     (r0)+ / skip on error
2:
        jmp     (r0)
```

```
/ u9 -- unix

trcv:                        / ttyx receiver interrupt handler
        jsr     r0,1f
        jsr     r0,1f
        jsr     r0,1f
        jsr     r0,1f
        jsr     r0,1f
        jsr     r0,1f
        jsr     r0,1f
        jsr     r0,1f
1:
        mov     r1,-(sp)
        mov     r2,-(sp)
        mov     r3,-(sp)
        mov     clockp,-(sp)
        mov     $s.syst+2,clockp
        sub     $trcv+4,r0 / 0%4 / calculate offset for tty causing
        asl     r0 / 0%8 / this interrupt
        mov     rcsr(r0),r2
        mov     rcbr(r0),r1
        tst     r2
        blt     1f / error
        tst     tty+6(r0)
        beq     1f
        bit     $40,r2 / parity
        bne     3f / branch if set
        tstb    tty+4(r0)
        blt     4f / 37 parity not allowed
        br      2f
3:
        bitb    $100,tty+4(r0)
        beq     2f / non-37 parity not allowed
4:
        bic     $177,r1          ? !177
        bit     $40,tty+4(r0)
        bne     3f / raw
        cmp     r1,$177
        beq     5f
        cmp     r1,$34
        bne     3f
5:
        mov     tty+6(r0),r0
        beq     2f
        movb    r1,6(r0) / interrupt or quit
        jsr     r0,wakeall
        br      2f
3:
        cmp     r1,$15 / or
        bne     3f
        bit     $20,tty+4(r0)
        beq     3f
        mov     $12,r1
3:
        bitb    $4,tty+4(r0)
```

```
        beq     3f
        cmp     r1,$'A
        blo     3f
        cmp     r1,$'Z
        bhi     3f
        add     $40,r1
3:
        movb    tty+3(r0),0f
        jsr     r0,putc; 0:.. / put char on input clist
                br 2f
        bitb    $10,tty+4(r0) / echo
        bne     4f / branch echo bit set
        cmp     r1,$12
        bne     3f
        bitb    $20,tty+4(r0) / cr
        beq     3f
4:
        cmp     r1,$4 / is char input an eot
        beq     1f
        mov     r1,-(sp) / put char on stack
        movb    tty+3(r0),0f
        inc     0f
        jsr     r0,putc; 0:.. / put char just input on output clist
                br .+2
        jsr     r0,starxmt
        mov     (sp)+,r1
3:
        bitb    $40,tty+4(r0) / raw
        bne     1f / branch if raw bit set
        cmp     r1,$12
        beq     1f
        movb    tty+3(r0),r1
        cmpb    cc(r1),$15.
        blo     2f
1:
        movb    tty+3(r0),0f
        jsr     r0,wakeup; runq; 0:.. / call wakeup for process
2:
        jmp     retisp

txmt:                   / ttyx transmitter interrupt handler.
        jsr     r0,1f
        jsr     r0,1f
        jsr     r0,1f
        jsr     r0,1f
        jsr     r0,1f
        jsr     r0,1f
        jsr     r0,1f
        jsr     r0,1f
1:
        mov     r1,-(sp)
        mov     r2,-(sp)
        mov     r3,-(sp)
        mov     clockp,-(sp)
        mov     $s.syst+2,clockp
        sub     $txmt+4,r0 / 0%4 / offset in cc
```

```
        asl     r0 / 0%8
        jsr     r0,starxmt
        jmp     retisp

xmtto:
        mov     r0,-(sp)
        mov     2(sp),r0 / 0%2+6
        sub     $6,r0
        asl     r0
        asl     r0 / 0%8
        jsr     r0,starxmt
        mov     (sp)+,r0
        rts     r0

starxmt:
        mov     (sp),r1 / 0%8 r1 contains 8xtty number
        movb    tty+3(r1),r1 / place contents of 4th byte of "tty"
                             / buf in r1 (cc,cf,cl offset)
        cmpb    cc+1(r1),$10. / is char count for tty output greater
                              / than or equal to 10
        bhi     1f / yes
        mov     r1,0f / no, make offset an arg of "wakeup"
        inc     0f / increment arg of wakeup
        jsr     r0,wakeup; runq+2; 0:.. / wakeup process identified
                                        / by wlist
1: / entry specified by argument in 0:
        mov     (sp),r1 / 0%8 / r1 contains 8xtty number
        asr     r1
        asr     r1
        asr     r1 / 0%1 r1 contains tty number
        tstb    toutt+3(r1) / is tout entry for tty output = 0
        bne     1f / no, return to calling routine
        mov     (sp),r2 / yes, place (8xtty number) into r2
        tstb    tcsr(r2) / does tty's tcsr register = 0 (is ready
                         / bit = 0)
        bge     1f / yes, return to calling routine
        movb    tty+2(r2),r1 / no, place third byte of "tty" buf
                             / into r1 (char left over after lf)
        clrb    tty+2(r2) / clear third byte
        tst     r1 / is third byte = 0
        bne     3f / no, r1 contains a non nul character
        movb    tty+3(r2),0f / yes, make byte 4 arg of "getc"
        inc     0f / increment arg to make it tty output list of
                   / clist
        jsr     r0,getc; 0:.. / obtain next character in clist for tty
                              / out and place in r1
                br 1f / if no entry in clist to be output, return to
                      / calling routine
3:
        bic     $!177,r1 / zero out bits 7-15 of r1
        movb    partab(r1),r3 / move "partab" entry (identified by
                              / r1) into r3
        bge     3f / if entry is greater than or equal to 0 (digit
                   / 2, far left digit = 0) branch
        bisb    200,r1 / if entry is less than 0 add 128 to ASC11
                       / code for char to be output
```

```
        bic     $!177,r3 / to make it teletype code and then clear
                         / bits 7-15 of r3
3:
        mov     (sp),r2 / r2 contains 8xtty number
        bit     $4,rcsr(r2) / is carrier present for tty
        beq     starxmt / no carrier flush
        mov     r1,-(sp) / yes, place character to be output on stack
        cmp     r1,$11 / is character "ht"
        bne     3f / no
        bitb    $2,tty+4(r2) / is tab to space flag "for" tty set
                             / (bit 1 of byte 5 in "tty" buffer area)
        beq     3f / no
        mov     $240,(sp) / yes, change character to space
3:
        mov     (sp)+,tcbr(r2) / place char to be output in tty output
                               / buffer
        add     $tty+1,r2 / place addr of 2nd byte of "tty" buf
        jmp     1f-2(r3) / area in r2 (which is the column count) and
                         / then
        incb    (r2) / normal / jmp to location determined by digits
                     / 0 and 1 of character's entry in "partab" which
                     / is now in r3
1:      rts     r0 / non-printing                          A error, label
        br      1f / bs                                     lost!!
        br      2f / nl (line feed)
        br      3f / tab (horizontal tab)
        br      4f / vert (vertical tab)
        br      5f / cr

1:
        decb    (r2) / col decrement column count in byte 2 of "tty"
                     / area
        bge     1f / if count >0 return to calling routine
        clrb    (r2) / col set column count = 0
        br      1f
2:
        bit     $1,r1 / is bit 0 of ASC11 char = 1 (char = lf)
        bne     2f / yes
        bitb    $20,3(r2) / cr flag is bit 4 of 5th byte of "tty"
                          / area = 1
        beq     2f / no (only lf to be handled)
        movb    $15,1(r2) / place "cr" in 3rd byte of "tty" area
                          / (character leftover after "lf")
2:
        movb    (r2),r3 / place present column count in r3
        beq     1f / return to calling routine if count = 0
        clrb    (r2) / col clear column count
        asr     r3
        asr     r3
        asr     r3
        asr     r3 / delay = col/16
        add     $3,r3 / start to determine tout entry for tty output
        br      2f
3:
        bitb    $2,3(r2) / is bit 1 of 5th byte of "tty" area = 1
                         / (tab to space bit set)
```

```
        beq     3f / no
        incb    (r2) / increment column count
        bitb    $7,(r2) / are bits 0, 1 and 2 set at col 0%8
        beq     1f / no
        movb    $11,1(r2) / yes, place ht in another tab next time
        br      1f / 3rd byte of tty area (character left over after
                   / "lf")
3:
        movb    (r2),r3 / place column count in r3
        bisb    $7,(r2) / make bits 0, 1 and 2 of column count = 1
        incb    (r2) / increment column count
        bis     $!7,r3 / clear bits 3-15 of r3
        neg     r3 / delay = dcol start to determine tout entry for
                   / tty out
        br      2f / by neg r3
4:
        mov     $176.,r3 / delay = lots start to determine tout entry
        br      2f
5:
        mov     $10.,r3 / cr delay 160ms for tn300 start to determine
                        / tout
        clrb    (r2) / set column count = 0 entry
2:
        add     $5,r3 / time for this char, increment value for tout
                      / entry by 5
        mov     (sp),r2 / 0%8 r2 contains 8xtty number
        asr     r2
        asr     r2
        asr     r2 / 0%1 r2 contains tty number
        movb    r3,toutt+3(r2) / place value for tout entry into tout
                               / table
1:
        rts     r0 / return

partab: / contains 3 digits for each character; digit 2 is used
        / to determine if 200 is to added to ASC11 code digits 0
        / and 1 are used to determine value for jump table.
        .byte 002,202,202,002,002,002,002,202
        .byte 204,010,006,212,012,214,202,002      parity error EOT
        .byte 202,002,002,202,002,002,202,002                   NAK
        .byte 002,202,202,002,202,002,002,202
        .byte 200,000,000,200,000,200,200,000
        .byte 000,200,200,000,200,000,000,200
        .byte 000,200,200,000,200,000,000,200
        .byte 200,000,000,200,000,200,200,000
        .byte 200,000,000,200,000,200,200,000
        .byte 000,200,200,000,200,000,000,200
        .byte 000,200,200,000,200,000,000,200
        .byte 200,000,000,200,000,200,200,000
        .byte 000,200,200,000,200,000,000,200
        .byte 200,000,000,200,000,200,200,000
        .byte 200,000,000,200,000,200,200,000
        .byte 000,200,200,000,200,000,000,202

xmtt:
        jsr     r0,cpass / get next character from user buffer area
```

```
        tst     r1 / is character nul
        beq     xmtt / yes, get next character
1:
        mov     $240,*$ps / set processor priority equal to 5
        mov     (sp),r2 / r2 contains i node number of file
        asl     r2 / 0%2+28 / multiply inode number by 2
        sub     $21.,r2 / 0%2+7 / subtract 21 from 2x inumber to
                        / get cc, cf, cl offset
        mov     r2,0f / make offset arg of putc
        cmpb    cc(r2),$50. / is char count for device greater than
                           / or equal to 50
        bhis    2f / yes
        jsr     r0,putc; 0:.. / find location in freelist to assign to
                             / device and
                br 2f / place char in list, if none available branch
                     / to put process to sleep
        mov     r0,-(sp) / place calling routines return address on
                        / stack
        mov     0b,r0 / place offset into cc, cl and cf tables in r0
        sub     $7,r0 / subtract seven from offset
        asl     r0 / multiply by 2
        asl     r0 / 0%8 / multiply by 2 (r0 contains 8xtty number)
        jsr     r0,starxmt / attempt to output character
        mov     (sp)+,r0 / pop stack
        br      xmtt / get next character
2:
        mov     r1,-(sp) / place character on stack
        mov     0b,0f / make offset into cc, cf, cl table arg of
                     / sleep (identifies location in wlist)
        jsr     r0,sleep; 0:.. / put process to sleep
        mov     (sp)+,r1 / remove character from stack
        br      1b / try again

rcvt: / read tty
        sub     $28.,r1 / 0%2 r1 contains 2xtty number
        asl     r1
        asl     r1 / r1 contains 8xtty number
        mov     r1,-(sp)
        mov     tty+6(r1),r5 / r5 contains address of 4th word in
                            / tty area
        tst     2(r5) / is char count = 0
        bne     1f / no
        bitb    $40,tty+4(r1) / raw flag set?
        beq     2f / no
        tst     -(sp) / yes, decrement sp
        jsr     r0,rcvch / get character from clist
        tst     (sp)+ / increment sp
        mov     (sp)+;r2 / r2 contains 8xtty number
        bitb    $4,rcsr(r2) / is carrier detect bit on
        beq     3f / no
        jsr     r0,passc / yes, place character in users buffer area
3:
        jmp     ret
2:
        jsr     r0,canon; rcvch / process a line of characters in
                               / clist and place results in tty buffer
```

```
                        / area
1:
        tst     (sp)+ / increment sp
1:
        tst     2(r5) / is char count for tty buffer = 0
        beq     1f / yes
        movb    *4(r5),r1 / no, move character pointer to r1
        inc     4(r5) / increment character pointer
        dec     2(r5) / decrement character count
        jsr     r0,passc / place character, whose address is in
                        / r1, in
        br      1b / user buffer area.  Then get next character.
1:
        jmp     ret

rcvch:
        mov     4(sp),r2 / 0%8 r2 contains 8xtty number
        mov     $4,r1
        bit     r1,rcsr(r2) / is carrier detection bit on
        bne     1f / yes
        bic     $1,rcsr(r2) / no, clear data terminal ready bit
        rts     r0
1:
        movb    tty+3(r2),0f / make cc offset arg for "getc"
        mov     $240,*$ps / set processor priority = 5
        jsr     r0,getc; 0:.. / get next character off clist
                br 2f / clist empty
        clr     *$ps / set processor priority = 0
        rts     r0
2:
        mov     0b,0f / make "getc" arg an arg for "sleep"
        mov     r5,-(sp) / save tty buffer address on stack
        jsr     r0,sleep; 0:..
        mov     (sp)+,r5
        br      rcvch

ocvt:
        sub     $28.,r1 / 0%2 calculate tty table offset
        mov     r1,r2
        asl     r1 / 0%4
        asl     r1 / 0%8
        mov     r1,-(sp)
        add     $6,r2 / calculate clist id  clist offset
        movb    r2,tty+3(r1) / put clist id in tty table
1:
        mov     (sp),r1
        bit     $4,rcsr(r1) / carrier detect bit set
        bne     1f / if so, branch
        mov     $511,rcsr(r1) / set ready, speed, interrupt enable,
                        / supervisor transmit
        movb    tty+3(r1),0f / put clist id in sleep argument
        jsr     r0,sleep; 0:..
        br      1b
1:
        mov     tty+6(r1),r5 / put tty buffer address in r5
        tstb    (r5) / first byte of tty buffer = 0
```

```
        bne     1f / if not, branch
        mov     $511,rcsr(r1) / set control bits for receiver
        mov     $511,tcsr(r1) / set control bits for transmitter
        movb    $210,tty+4(r1) / put 210 in tty table word 3 / set flags
1:
        incb    (r5) / inc first byte of tty buffer
        tst     (sp)+
        tst     u.ttyp / is there a process control tty
        bne     1f / yes, then branch
        mov     r5,u.ttyp / no, make this tty the process control tty
        br      1f / return

ccvt:
        sub     $28.,r1
        asl     r1 / 0%4
        asl     r1
        mov     tty+6(r1),r5
        decb    (r5)
1:
        jmp     sret
```

```
/ ux -- unix

systm:

        .=.+2
        .=.+128.
        .=.+2
        .=.+64.
        s.time: .=.+4
        s.syst: .=.+4
        s.wait: .=.+4
        s.idlet:.=.+4
        s.chrgt:.=.+4
        s.drerr:.=.+2
inode:
        i.flgs: .=.+2
        i.nlks: .=.+1
        i.uid:  .=.+1
        i.size: .=.+2
        i.dskp: .=.+16.
        i.ctim: .=.+4
        i.mtim: .=.+4
        . = inode+32.
mount:  .=.+1024.
proc:
        p.pid:  .=.+[2*nproc]
        p.dska: .=.+[2*nproc]
        p.ppid: .=.+[2*nproc]
        p.break:.=.+[2*nproc]
        p.link: .=.+nproc
        p.stat: .=.+nproc
tty:
        . = .+[ntty*8.]
fsp:    .=.+[nfiles*8.]
bufp:   .=.+[nbuf*2]+6
sb0:    .=.+8
sb1:    .=.+8
swp:    .=.+8
ii:     .=.+2
idev:   .=.+2
cdev:   .=.+2
deverr: .=.+12.
active: .=.+2
rfap:   .=.+2
rkap:   .=.+2
tcap:   .=.+2
tcstate:.=.+2
tcerrc: .=.+2
mnti:   .=.+2
mntd:   .=.+2
mpid:   .=.+2
clockp: .=.+2
rootdir:.=.+2
toutt:  .=.+16.; touts: .=.+32.
runq:   .=.+6
```

```
wlist:  .=.+40.
cc:     .=.+30.
cf:     .=.+31.
cl:     .=.+31.
clist:  .=.+510.
imod:   .=.+1
smod:   .=.+1
mmod:   .=.+1
uquant: .=.+1
sysflg: .=.+1
pptiflg:.=.+1
ttyoch: .=.+1
 .even
 .=.+100.; sstack:
buffer: .=.+[ntty*140.]
        .=.+[nbuf*520.]

 . = core-64.
user:
        u.sp:     .=.+2
        u.usp:    .=.+2
        u.r0:     .=.+2
        u.cdir:   .=.+2
        u.fp:     .=.+10.
        u.fofp:   .=.+2
        u.dirp:   .=.+2
        u.namep:  .=.+2
        u.off:    .=.+2
        u.base:   .=.+2
        u.count:  .=.+2
        u.nread:  .=.+2
        u.break:  .=.+2
        u.ttyp:   .=.+2
        u.dirbuf:.=.+10.
        u.pri:    .=.+2
        u.intr:   .=.+2
        u.quit:   .=.+2
        u.emt:    .=.+2
        u.ilgins:.=.+2
        u.cdev:   .=.+2
        u.uid:    .=.+1
        u.ruid:   .=.+1
        u.bsys:   .=.+1
        u.uno:    .=.+1
 . = core
```

```
/ sh -- command interpreter

        mov     sp,r5
        mov     r5,shel larg / save orig sp in shel larg
        cdpb    B(r5),$'- / was this sh calleZd by init or loginx~
        bne     2f / no
        sys     intr; 0 / yes, turn off interrupts
        sys     quit; 0
2:
        sys     getuid / who is user
        tst     r0 / is it superuser
        bne     2f / no
        movb    $'#,at / yes, set new prompt symbol
2:
        cmp     (r5),$1 / tty input?
        ble     newline / yes, call with '-(or with no command
                        / file name)
        clr     r0 / no, set tty
        sys     close / close it
        mov     4(r5),0f / get new file name
        sys     open; 0:..; 0 / open it
        bec     1f / branch if no error
        jsr     r5,error / error in file name
                /<Input not found\n\0>; .even
        sys     exit
1:
        clr     at / clear prompt character, if reading non-tty
                   / input file
newline:
        tst     at / is there a prompt symbol
        beq     newcom / no
        mov     $1,r0 / yes
        sys     write; at; 2. / print prompt
newcom:
        mov     shellarg,sp /
        mov     $parbuf,r3 / initialize command list area
        mov     $parp,r4 / initialize command list pointers
        clr     infile / initialize alternate input
        clr     outfile / initialize alternate output
        clr     glflag / initialize global flag
newarg:
        jsr     pc,blank / squeeze out leading blanks
        jsr     r5,delim / is new character a ; \n or &
                br 2f / yes
        mov     r3,-(sp) / no, push arg pointer onto stack
        cmp     r0,$'< / new input file?
        bne     1f / no
        mov     (sp),infile / yes, save arg pointer
        clr     (sp) / clear pointer
        br      3f
1:
        cmp     r0,$'> / new output file?
        bne     newchar / no
        mov     (sp),outfile / yes, save arg pointer
        clr     (sp) / clear pointer
        br      3f
```

```
newchar:
        cmp     $' ,r0 / is character a blank
        beq     1f / branch if it is (blank as arg separator)
        cmp     $'\n+200,r0 / treat \n preceded by \
        beq     1f / as blank
        jsr     pc,putc / put this character in parbuf list
3:
        jsr     pc,getc / get next character
        jsr     r5,delim / is char a ; \n or &
                br 1f / yes
        br      newchar / no, start new character tests
1:
        clrb    (r3)+ / end name with \0 when read blank, or
                      / delim
        mov     (sp)+,(r4)+ / move arg ptr to parp location
        bne     1f / if (sp)=0, in file or out file points to arg
        tst     -(r4) / so ignore dummy (0), in pointer list
1:
        jsr     r5,delim / is char a ; \n or &
                br 2f / yes
        br      newarg / no, start newarg processing
2:
        clr     (r4) / \n, &, or ; takes to here(end of arg list)
                     / after 'delim' call
        mov     r0,-(sp) / save delimter in stack
        jsr     pc,docom / go to exec command in parbuf
        cmpb    (sp),$'& / get a new command without wait?
        beq     newcom / yes
        tst     r1 / was chdir just executed or line ended with
                   / ampersand?
        beq     2f / yes
1:
        sys     wait / no, wait for new process to terminate
                     / command executed)
        bcs     2f / no, children not previously waited for
        cmp     r0,r1 / is this my child
        bne     1b
2:
        cmp     (sp),$'\n / was delimiter a new line
        beq     newline / yes
        br      newcom / no, pick up next command

docom:
        sub     $parp,r4 / put arg count in r4
        bne     1f / any arguments?
        clr     r1 / no, line ended with ampersand
        rts     pc / return from call
1:
        jsr     r5,chcom; qchdir / is command chdir?
                br 2f / command not chdir
        cmp     r4,$4 / prepare to exec chdir, 4=arg count x 2
        beq     3f
        jsr     r5,error / go to print error
                <Arg count\n\0>; .even
        br      4f
3:
```

```
        mov     parp+2,0f / more directory name to sys call
        sys     chdir; 0:0 / exec chdir
        bec     4f / no error exit
        jsr     r5,error / go to print error
                <Bad directory\n\0>; .even / this diagnostic
4:
        clr     r1 / set r1 to zero to dkip wait
        rts     pc / and return
2:
        jsr     r5,chcom; qlogin / is command login?
                br 2f / not login, go to fork
        sys     exec; parbuf; parp / exec login
        sys     exec; binpb; parp / or /bin/login
2: / no error return??
        sys     fork / generate sh child process for command
                br newproc / exec command with new process
        bec     1f / no error exit, old process
        jsr     r5,error / go to print error
                <Try again\n\0>; .even / this diagnostic
        jmp     newline / and return for next try
1:
        mov     r0,r1 / save id of child sh
        rts     pc / return to "jsr pc, docom" call in parent sh

error:
        movb    (r5)+,och / pick up diagnostic character
        beq     1f / 0 is end of line
        mov     $1,r0 / set for tty output
        sys     write; och; 1 / print it
        br      error / continue to get characters
1:
        inc     r5 / inc r5 to point to return
        bic     $1,r5 / make it even
        clr     r0 / set for input
        sys     seek; 0; 2 / exit from runcom, skip to end of
                        / input file
chcom: / has no effect if tty input
        mov     (r5)+,r1 / glogin gchdir  r1, bump r5
        mov     $parbuf,r2 / command address  r2 'login'
1:
        movb     (r1)+,r0 / is this command 'chdir'
        cmpb    (r2)+,r0 / compare command name byte with 'login'
                       / or 'chdir'
        bne     1f / doesn't compare
        tst     r0 / is this
        bne     1b / end of names
        tst     (r5)+ / yes, bump r5 again to execute login
                    / chdir
1:
        rts     r5 / no, return to exec command

putc:
        cmp     r0,$'' / single quote?
        beq     1f / yes
        cmp     r0,$'" / double quote
        beq     1f / yes
```

```
        bic     $!177,r0 / no, remove 200, if present
        movb    r0,(r3)+ / store character in parbuf
        rts     pc
1:
        mov     r0,-(sp) / push quote mark onto stack
1:
        jsr     pc,getc / get a quoted charactef
        cmp   ; r0,$'\n / is it end of line
        bne     2f / no
        jsr     r5,error / yes, indicate missing quote mark
                <"' imbalance\n\0>; .even
        jmp     newline / ask for new line
2:
        cmp     r0,(sp) / is this closing quote mark
        beq     1f / yes
        bic     $!177,r0 / no, strip off 200 if present
        movb    r0,(r3)+ / store quoted character in parbuf
        br      1b / continue
1:
        tst     (sp)+ / pop quote mark off stack
      ' rts     pc`, return

/ thp`e new process

newproc:
        mov     infile,0f / move pointer to new file name
        beq     1f / branch if no alternate read file given
        tstb    *0f
        beq     3f / branch if no file name given
        clr     r0 / set tty input file name
        sys     close / close it
        sys     open; 0:..; 0 / open new input file for reading
        bcc     1f / branch if input file ok
3:
        jsr     r5,error / file not ok, print error
                <Input file\n\0>; .even / this diagnostic
        sys     exit / terminate this process and make parent sh
1:
        mov     outfile,r2 / more pointer to new file name
        beq     1f / branch if no alternate write file
        cmpb    (r2),$'> / is > at beginning of file name?
        bne     4f / branch if it isn't
        inc     r2 / yes, increment pointer
        mov     r2,0f
        svs     open; 0:..; 1 / open file for writing
        bec     3f / if no error
4:
        mov     r2,0f
        sys     creat; 0:..; 17 / create new file with this name
        bec     3f / branch if no error
2:
        jsr     r5,error
                <Output file\n\0>; .even
        sys     exit
3:
        sys     close / close the new write file
```

```
        mov     r2,0f / move new name to open
        mov     $1,r0 / set tty file name
        sys     close / close it
        sys     open; 0:..; 1 / open new output file, it now has
                               / file descriptor 1
        sys     seek; 0; 2 / set pointer to current end of file
1:
        tst     glflag / was *, ? or [ encountered?
        bne     1f / yes
        sys     exec; parbuf; parp / no, execute this command
        sys     exec; binpb; parp / or /bin/this command
2:
        sys     stat; binpb; inbuf / if can't execute does it
                                   / exist?
        bes     2f / branch if it doesn't
        mov     $shell,parp-2 / does exist, not executable
        mov     $binpb,parp / so it must be
        sys     exec; shell; parp-2 / a command file, get it with
                                    / sh /bin/x (if x name of file)
2:
        jsr     r5,error / a return for exec is the diagnostic
                <No command\n\0>; .even
        sys     exit
1:
        mov     $glob,parp-2 / prepare to process *,?
        sys     exec; glob; parp-2 / execute modified command
        br      2b

delim:
        cmp     r0,$'\n / is character a newline
        beq     1f
        cmp     r0,$'& / is it &
        beq     1f / yes
        cmp     r0,$'; / is it ;
        beq     1f / yes
        cmp     r0,$'? / is it ?
        beq     3f
        cmp     r0,$'[ / is it beginning of character string
                / (for glob)
        bne     2f
3:
        inc     glflag / ? or * or [ set flag
2:
        tst     (r5)+ / bump to process all except \n,;,&
1:
        rts     r5

blank:
        jsr     pc,getc / get next character
        cmp     $' ,r0 / leading blanks
        beq     blank / yes, 'squeeze out'
        cmp     r0,$200+'\n / new-line preceded by \ is translated
        beq     blank / into blank
        rts     pc

getc:
```

```
        tst     param / are we substituting for $n
        bne     2f / yes
        mov     inbufp,r1 / no, move normal input pointer to r1
        cmp     r1,einbuf / end of input line?
        bne     1f / no
        jsr     pc,getbuf / yes, put next console line in buffer
        br      getc
1:
        movb    (r1)+,r0 / move byte from input buffer to r0
        mov     r1,inbufp / increment routine
        bis     escap,r0 / if last character was \ this adds
                     / 200 to current character
        clr     escap / clear, so escap normally zero
        cmp     r0,$'\\ / note that \\ is equal \ in as
        beq     1f
        cmp     r0,$'$ / is it $
        beq     3f / yes
        rts     pc / no
1:
        mov     $200,escap / mark presence of \ in command line
        br      getc / get next character
2:
        movb    *param,r0 / pick up substitution character put in
                      / r0
        beq     1f / if end of substitution arg, branch
        inc     param / if not end, set for next character
        rts     pc / return as though character in ro is normal
                   / input
1:
        clr     param / unset substitution pointer
        br      getc / get next char in normal input
3:
        jsr     pc,getc / get digit after $
        sub     $'0,r0 / strip off zone bits
        cmp     r0,$9. / compare with digit 9
        clos    1f / less than or equal 9
        mov     $9.,r0 / if larger than 9, force 9
1:
        mov     shellarg,r1 / get pointer to stack for
                          / this call of shell
        inc     r0 / digit +1
        cmp     r0,(r1) / is it less than # of args in this call
        bge     getc / no, ignore it, so this $n is not replaced
        asl     r0 / yes, multiply by 2 (to skip words)
        add     r01,r0 / form pointer to arg pointer (-2)
        mov     2(r0),param / more arg pointer to param
        br      getc / go to get substitution arg for $n

getbuf:
        mov     $inbuf,r0 / move input buffer address
        mov     r0,inbufp / to input buffer pointer
        mov     r0,einbuf / and initialize pointer to end of
                      / character string
        dec     r0 / decrement pointer so can utilize normal
                   / 100p starting at 1f
        mov     r0,0f / initialize address for reading 1st char
```

```
1:
        inc     0f / this routine filles inbuf with line from
                   / console - if there is one
        clr     r0 / set for tty input
        sys     read; 0:0; 1 / read next char into inbuf
        bcs     xit1 / error exit
        tst     r0 / a zero input is end of file
        beq     xit1 / exit
        inc     einbuf / eventually einbuf points to \n
                       / (+1) of this line
        cmp     0b,$inbuf+256. / have we exceeded input buffer size
        bhis    xit1 / if so, exit assume some sort of binary
        cmpb    *0b,$'\n / end of line?
        bne     1b / no, go to get next char
        rts     pc / yes, return

xit1:
        sys     exit

quest:
        <?\n>

at:
        <@ >

qchdir:
        <chdir\0>
glogin:
        <login\0>
shell:
        </bin/sh\0>
glob:
        </etc/glob\0>
binpb:
        </bin/>
parbuf: .=.+1000.
        .even
param:  .=.+2
glflag: .=.+2
infile: .=.+2
outfile:.=.+2
        .=.+2 / room for glob
parp:   .=.+200.
inbuf:  .=.+256.
escap:  .=.+2
inbufp: .=.+2
einbuf: .=.+2
och:    .=.+2
shellarg:.=.+2
```

```
/ init -- process control initialization

mount = 21.

        sys     intr; 0 / turn off interrupts
        sys     quit; 0
        cmp     csw,$73700 / single user?
        bne     1f / no
help:
        clr     r0 / yes
        sys     close / close current read
        mov     $1,r0 / and write
        sys     close / files
        sys     open; ctty; 0 / open control tty
        sys     open; ctty; 1 / for read and write
        sys     exec; shell; shellp / execute shell
        br      help / keep trying
1:
        mov     $'0,r1 / prepare to change
1:
        movb    r1,tapx+8 / mode of dec tape drive x, where
        sys     chmod; tapx; 17 / x=0 to 7, to read/write by owner or
        inc     r1 / non-owner mode
        cmp     r1,$'8 / finished?
        blo     1b / no
        sys     mount; rk0; usr / yes, root file on mounted rko5
                                / disk is /usr
        sys     creat; utmp; 16 / truncate /tmp/utmp
        sys     close / close it
        movb    $'x,zero+8. / put identifier in output buffer
        jsr     pc.wtmprec / go to write accting info
        mov     $itab,r1 / address of table to r1

/ create shell processes

1:
        mov     (r1)+,r0 / 'x, x=0, 1... to r0
        beq     1f / branch if table end
        movb    r0,ttyx+8 / put symbol in ttyx
        jsr     pc,dfork / go to make new init for this ttyx
        mov     r0,(r1)+ / save child id in word offer '0, '1,...etc.
        br      1b / set up next child

/ wait for process to die

1:
        sys     wait / wait for user to terminate process
        mov     $itab,r1 / initialize for search

/ search for process id

2:
        tst     (r1)+ / bump r1 to child id location
        beq     1b / ? something silly
        cmp     r0,(r1)+ / which process has terminated
```

```
        bne     2b / not this one

/ take name out of utmp

        sub     $4,r1 / process is found, point x' to 'x
                      / for it
        mov     r1,-(sp) / save address on stack
        mov     (r1),r1 / move 'x to r1
        sub     $'0,r1 / remove zone bits from character
        asl     r1 / generate proper
        asl     r1 / offset
        asl     r1 / for
        asl     r1 / seek
        mov     r1,0f / move it to offset loc for seek
        mov     $zero,r1
2:
        clr     (r1)+ / ccear-
        cmp     r1,$zero+16. / output buffer
        blo     2b / area
        sys     open; utmp; 1 / open file for writing
        bes     2f / if can't open, create user anyway
        mov     r0,r1 / save file desc
        sys     seek; 0:..; 0 / move to proper pointer position
        mov     r1,r0 / not required
        sys     write; zero; 16. / zero this position in
        mov     r1,r0 / restore file descriptor
        sys     close / close file

/ re-create user process

2:
        mov     (sp)+,r1 / restore 'x to r1
        mov     (r1)+,r0 / move it to r0
        movb    r0,ttyx+8 / get correct ttyx
        movb    r0,zero+8 / move identifier to output buffer
        jsr     pc,wtmprec / go to write accting into
        jsr     pc,dfork / fork
        mov     r0,(r1)+ / save id of child
        br      1b / go to wait for next process end

dfork:
        mov     r1,r2
        sub     $itab+2,r2 / left over
        asl     r2 / from previous
        asl     r2 / version of code
        mov     r2,offset
        sys     fork
                br 1f / to new copy of init
        bes     dfork / try again
        rts     pc / return
1:
        sys     quit; 0 / new init turns off
        sys     intr; 0 / interrupts
        sys     chown; ttyx; 0 / change owner to super user
        sys     chmod; ttyx; 15 / changemode to read/write owner,
                                / write non-owner
```

```
        sys     open; ttyx; 0 / open this ttyx for reading
                              / and wait until someone calls
        bes     help1 / branch if trouble
        sys     open; ttyx; 1 / open this ttyx for writing after
                              / user call
        bes     help1 / branch if trouble
        sys     exec; getty; gettyp / getty types <login> and
                                    / executes login which logs user
                                    / in and executes sh-
        sys     exit / HELP!

help1:
        jmp     help / trouble

wtmprec:
        sys     time / get time
        mov     ac,zero+10. / more to output
        mov     mq,zero+12. / buffer
        sys     open; wtmp; 1 / open accounting file
        bes     2f
        mov     r0,r2 / save file descriptor
        sys     seek; 0; 2 / move pointer to end of file
        mov     r2,r0 / not required
        sys     write; zero; 16. / write accting info
        mov     r2,r0 / restore file descriptor
        sys     close / close file
2:
        rts     pc

ctty:   </dev/tty\0>
shell:  </bin/sh\0>
shellm: <-\0>
tapx:   </dev/tapx\0>
rk0:    </dev/rk0\0>
utmp:   </tmp/utmp\0>
wtmp:   </tmp/wtmp\0>
ttyx:   </dev/ttyx\0>
getty:  </etc/getty\0>
usr:    </usr\0>
        .even

shellp: shellm
        0
gettyp: getty
        0
itab:
        '0; ..
        '1; ..
        '2; ..
        '3; ..
        '4; ..
        '5; ..
        '6; ..
        '7; ..
         0
```

```
offset: .=.+2
zero:   .=.+8.; .=.+6; .=.+2
```

1. Overview

The code of UNIX is divided into 11 files, named u0 through u9 and ux. ux contains the definitions of the system tables and data areas; the actual code is in the other sections. These files are assembled together in the order u0 ... u9 ux. The boot procedures section of the UPM explains how to test and install a newly assembled system.

There are three major portions of UNIX: the file system, the process control system, and the rest. "The rest" refers mostly to the code implementing several miscellaneous system calls which do not fit neatly into any category. Unfortunately the various parts of UNIX are fairly well strewn about its constituent source files. The following is a rough key:

- u0 initialization
- u1 system entry; some system calls
- u2 most remaining system calls
- u3 process switching, swapping
- u4 character-oriented device interrupt time routines, except DC-11
- u5 basic file system routines
- u6 more file system routines
- u7 more file system, character-oriented device non-interrupt time routines
- u8 interrupt and non-interrupt time routines for block structured devices (disks, tape)
- u9 almost all code for DC-11 asynchronous communications interfaces

It has been mentioned parenthetically that UNIX is not very modular. Its lact of modularity is reflected in this document. Therefore (to paraphrase Fenichel and McIlroy referring to their description of TMGL) no single order of reading can be recommended; instead a chimneying technique is suggested, climbing not one wall at a time, but all simultaneously.

2. Overview of the data base.

A description of each item in the data base is given in Section F. In core data is defined in ux

3. System entry and exit

The system can legitimately be entered only by some sort of trap. The trap caused by the trap instruction (that is, sys) and all otherwise unknown traps are directed to one of the synonymous labels unkni or sysent. There the registers are saved in the following order:

```
r0
 ...
r5
ac
```

mq
sc

A pointer to the stack (after the save) is retained. Then the instruction being executed at the time of the trap is examined to see whether it represents a legitimate system call. If so, a jump is made to the proper routine; if not, to the label badsys. Whenever the system is entered by this route, a flag is set to indicate that system code is being executed. No traps, including system calls, are allowed within the system.

To exit from a system call, a call handler jumps either to sysret to error. The only difference is that in the latter case the error bit (c-bit) is set in the word from which the processor status will be restored.

At sysret, a check is made to determine the last-mentioned i-node the super-block, or the dismountable super block have been modified; if so, the I/O to write out the appropriate area is started via ppoke. Then a check is made to determine if the user's time quantum ran out during his execution in the system. If so, tswap is called to give another user a chance to run. The registers are restored and an rti is executed to return to the user's program.

Label badsys is reached either because the user executed an illegal trap-type instruction or because a t-bit trap occurred. (The t-bit is used to implement the quit function.) badsys calls the appropriate internal routines to write out a core image file in the user's current directory, then jumps to the sysexit routine to terminate the process.

4. Fork, Exit, Wait

Fork and exit implement the creation and destruction respectively of processes.

There is a fixed maximum number of processes. Each possible process has a slot in the process tables and a swap area on the RF disk associated with it.

Label sysfork implements the fork primitive. It searches the p.stat portion of the process table to find an idle process slot, and gives an error if none is found. An entry for the new process is placed on the run queue and wswap is called to swap out a copy of the current process' core image onto the new process' disk area. The fsp entry for each file open in the process is incremented to indicate that each such file is open in another process.

sysexit implements process destruction. It is more complicated than one might think. First each open file is closed by fclose. The process' status is set to unused. Then the process table is searched to find any children of the process. Any of these that have died but not waited for are marked free.

When the parent of the dying process is found, it is awakened (by putlu) if it is waiting. Then 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.

syswait implements the process wail facility. It searches the process table for a child process. If none is found, and error is returned. If a child is found in the zombie state (terminated but not buried by wait) its process ID is returned and its process slot is freed.

If all children are still active, syswait calls swap to give up the processor.

The possible states of a process (p.stat values) are:

0 free, i.e., no process associated with this slot number
1 active
2 waiting for a child to die
3 terminated, but not yet waited for (zombie).

5. Process swapping

The important routine is swap. When swap is called, the run queues are searched for the highest priority process. It is not the same as the process in core, core is written out to the appropriate disk area, the image of the new process is read in, and swap returns to the point in which it was called in the new process.

If there is no process in the queues, idle is called. idle consists essentially of a wait instruction; the effect of wait is such that idle returns after every interrupt. swap searches the queues again in the hopes of finding a process entered on a queue by the interrupt routine.

The I/O to write out a core image is done by wswap. It must operate on a stack internal to the system. wswap uses the program break u.break to determine how much to write out. Usually, the process' stack area is copied down to the top of the program area to speed up I/O. The I/O queue entry reserved for swapping is set up and ppoke is called to initiate the I/O.

The core image reading routine is rswap; it also uses the system stack. The core image is unpacked by unpack.

It is important to realize that running processes are not on the run queues. Therefore, processes which call swap must already have arranged to be put back on the run queues in some way.

The tswap entry to swap is used for timer runouts; it puts the process on the lowest priority queue before flowing into swap.

6. File System

A detailed description of the file system is given in the UPM under Format of File System and Format of Directories. The diagrams on the following pages support that write up.

FORMAT OF FILE SYSTEM

```
Block
Number
          |-------------------------------------|
  0       | number of bytes in free storage map |     0
          |-------------------------------------|
          |                  .                  |
          |                  .                  |  See page 2
          |            free storage             |
          |                map                  |
          |                  .                  |
          |-------------------------------------|
          |    number of bytes in i-node map    |
          |-------------------------------------|
          |                  .                  |
          |                  .                  |  See page 3
          |                  .                  |
          |              inode map              |
          |                  .                  |
          |                  .                  |
          |-------------------------------------|
  .       |                                     |
          |-------------------------------------|
  2       |               inode 1               |
          |                  .                  |
          |                  .                  |  See page 4
          |                  .                  |
          |               inode 16              |
          |-------------------------------------|
  3       |               inode 17              |
          |                  .                  |
          |                  .                  |
          |                  .                  |
          |               inode 32              |
          |-------------------------------------|
  4       |               inode 33              |
  .       |                  .                  |
  .       |                  .                  |
  .       |                  .                  |
  .       ~                                     ~
  .
          |                                     |
          |-------------------------------------|
          |                files                |  See page 6
          |                                     |
```

Notes: There are 256 words/block

FREE STORAGE MAP

Notes:

1. There is 1 bit for each block on the device.
2. If the bit is a 1, the block is free.
3. The bit for block k of the device is in byte k/8 of the map; it is offset k (mod8) bits from the right ex. Find the bit for block 100

   $\frac{100}{8}$ = byte 12 offset = 4

```
block numbers              f.s. map                     byte

   |--------------------|--------------------|
   |15                 8|             2  1  0|          0
   |--------------------|--------------------|
   |31                  |                  16|          2
   |--------------------|--------------------|
   |                    |                  32|          4
   |--------------------|--------------------|
   |                    |                  48|          6
   |--------------------|--------------------|
   |                    |                  64|          8
   |--------------------|--------------------|
   |                    |                  80|         10
   |--------------------|--------------------|
   |                    |    | 4 |3 |2 |1 |0 |
   |                    |    |---|--|--|--|--|         12
   |                    |    |100|99|98|97|96|
   |--------------------|----|---|--|--|--|--|
   |                                         |
   |                           ^             |
   |                                         |
   |                     bit 4 of the 12th   |byte
   |                                         |
   |                                         |
   |                                         |
```

INODE MAP

Notes:

1. The map begins with inode 41.
2. There is 1 bit for each i-node.
3. If the bit is a 0, the inode is free.
4. The byte number for i-node i is byte number = (i-41)/8
   The offset or bit position = (i-41) mod8
   Ex. i = 100
   byte number = $\frac{100-41}{8}$ = byte 7
   offset = (100-41) mod8 = bit 3

```
i-node number                                                  byte

        |----------------------|------------------------|
        |56                    |                      41|   0
        |----------------------|------------------------|
   3    |                      |                      57|   2
        |----------------------|------------------------|
   3    |                      |                      73|   4
        |----------------------|------------------------|
   7    |         | 3 |2 |1 |0 |                        |
        |         |---|--|--|--|                        |
        |         |100|99|98|97|96                    89|   6
        |---------|---|--|--|--|------------------------|
        |                      |                        |   8
        |-----------^----------|------------------------|
        |    bit 3 of the 7th byte                      |
        |                                               |
        |                                               |
        |                                               |
```

I-NODES

Notes:

1. Each i-node represents 1 file.
2. I-numbers start at 1.
3. Storage begins in block 2.
4. i-nodes are 32 bytes long.
   16 inodes fit in 1 block.
5. The block number for i-node i is found by:
   block number = (i+31)/16
   The byte number from the start at the block is found by:
   byte number = 32 ((i+31)(mod16))

   Ex. Find where i-node 50 is.
   block number = (50+31)/16 = 5
   it begins at byte number 32. ((81)mod16)
   = 32 (1) = 32

```
block number
              |--------------|
     2        |   i-node 1   |    32 bytes/i-node
              |--------------|
              |      .       |
              |      .       |
              |      .       |
              |  i-node 16   |
              |--------------|
     3        |     17       |
              |      .       |
              |      .       |
              |      .       |
              |     32       |
              |--------------|
     4        |     33       |
              |      .       |
              |      .       |
              |      .       |
              |     48       |
              |--------------|
     5        |     49       |    32 bytes
              |--------------|
              |     50       |    _ block 5, byte 32
              |--------------|
              |      .       |
              |      .       |
              |      .       |
              |--------------|
              |              |
              |              |
```

6. i-nodes below 41 are for special files.

AN I-NODE IN DETAIL

byte			byte
	flags (see below)		0
3	user id of owner	number of links	2
	size in bytes		4
	1st indirect block or contents block		6
	2nd indirect or contents block		8
	. . . .		
	8th indirect or contents block		20
	creation		22
25	time		24
	modification		26
29	time		28
	unused		30

The flags are as follows:

100000	i-node is allocated
040000	directory
020000	file has been modified (always on)
010000	large file
000040	set user ID on execution
000020	executable
000010	read, owner
000004	write, owner
000002	read, non-owner
000001	write, non-owner

FILES

1) A small file is a file less than 8 blocks long. 2) A large file is greater than 8 blocks long. 3) Byte number "n" of a file is addressed as follows:

block number = n/512 = b

a) If the file is small (see flags)

physical block = bth entry in address portion of i-node

ex. ii = 1500

$$b = \frac{1500}{512} = 2$$

physical block = 2nd contents block in bytes 8 and 9 of the inode

b) If the file is large (greater than 8 blocks) then

indirect block # = b/256

byte offset in indirect block = 2 (b (mod256))

word found in this byte is the address of the block corresponding to b

ex. b = 1000
indirect block number = 1000/256 = 3
byte offset = 2 (1000 mod256) = 2.232=464

```
                inode              byte          indirect   blocks
block                    entry                |--------------------|
|-------------------------|
  1    |                    |  6              |                    |
       |--------------------|      start      ~                    ~
  2    |                    |  8    of
       |--------------------|      block     |--------------------|
  3    | contains block no. | 10              |         .          |
464
       | of indirect block  |                 |         .          |
bytes
       |--------------------|                 |         .          |
                                               |--------------------|
                                               | address of block b |
                                               |--------------------|
                                               |                    |
                                               |                    |
```

DIRECTORIES

Notes:

1) Like a file except no user (except superuser) may write into a directory.
2) A file is identified as a directory by a bit in the flag word of its i-node. (See i-node flag page 5)
3) Directory entries are 10 bytes long.

Entry		
1	i-number of directory itself (.)	10 bytes
	8 character file name	
2	i-number of parent directory (..)	
	8 character file name	
3	i-number of file represented by entry	
	8 character file name	
4	.	
.	.	
.	.	
.	.	

FSP TABLE

Notes:

1) The fsp table is an incore table containing information about open files.
2) It is 4 words/entry.
3) The same file can be opened more than once, and have more than one entry in the fsp table.

```
entry   15
       |---|-----------------------------------------|
  1    |r/w|          i-number of open file          |
       |---|-----------------------------------------|
       |               device number                 |
       |---------------------------------------------|
       | offset pointer, i.e., r/w pointer to file   |
       |----------------------|----------------------|
       |flag that says file   | number of processes  |
       | has been deleted     | that have file open  |
       |----------------------|----------------------|
  2    |                                             |
       |---------------------------------------------|
       |                                             |
       |---------------------------------------------|
       |                                             |
       |---------------------------------------------|
       |                                             |
       |---------------------------------------------|
  3    |                                             |
       |                                             |
```

7. Process Scheduling

Processes are scheduled to run according to a priority structure which is implemented via the runq table and the p.link table. These two tables are described below. (diagram on page 9)

THE RUNQ TABLE

runq:

is a table of length 3, with one entry for each of the three ready-to-run queues of processes. The low byte of each entry contains the process number of the first process in the queue; the high byte contains the process number of the last process. The entry is 0 if there are no processes on the queue. Each queue is linked by the p.link entry in the process table.

	process number of last process on queue	process number of first process on queue	
highest priority queue	7	2	runq
	6	3	runq+2
lowest priority queue	10	4	runq+4

To demonstrate the interaction of p.link and runq:
If the priority of process numbers was arranged as follows: 2, 8, 7, 3, 1, 6, 4, 5, 10, p.link would look like. So, the process 2 is found in the 2nd slot of the p.link table. In this case process 8.

slot numbers - ()

8 (2)	6 (1)	p.link
5 (4)	1 (3)	p.link+2
4 (6)	10 (5)	p.link+4
7 (8)	3 (7)	p.link+6
		p.link+nproc (16)

8. Terminal Control

The handling of character oriented devices (tty, lineprinter, console tty) is done via several tables and buffers, namely: The character count table "cc", the first character pointer table "cf", the last character pointer table "cl", the character list "clist", the tty control blocks "tty", the tty buffers "buffer" and the time out tables toutt and touts.

The tables cc, cf, cl are structured such that each entry is associated with the input or output of a specific tty or other device. The exact structure is shown in the diagram for these tables. The clist contains linked lists of characters associated with each device. See discussion in Section F.

When an input interrupt occurs from a specific device the interrupt routine puts the character received at the end of the clist string for inputs from that device. When an output interrupt occurs the next character on the clist string for outputs to the device is popped off the list and is transmitted. If the character being output generates a delay (lf, cr, ht, vt) the appropriate entry in the toutt table is set no output will be generated while the toutt entry is non-zero. Each clock generated input causes every non-zero toutt entry to be decremented. When a toutt entry becomes zero, the associated routine named in the touts table is called.

The tty buffers are used for editing the input clist strings for the tty's. When a sysread on a tty is done the clist input string for the device is scanned and put in buffer 28 #, @ or deletes are found they are stripped from the input and appropriate action is taken.

TTY BLOCK AND BUFFER

I. TTY BLOCK

column tty is in		tty
sleep queue, wakeup queue, cc offset	char left over after "lf"	tty+2
	flags cr, tab, sp, raw, echo	tty+4
pointer to tty buffer		tty+6

```
tty+4: bit 7 - parity 37
           6 - parity non 37
           5 - raw
           4 - cr
           3 - echo
           2 - caps to lower case
           1 - tab to space
           0 - no delay
```

II. TTY BUFFER

	number of processes using this tty	buffer
char count		buffer+2
character pointer		buffer+4
	interrupt character	buffer+6
		buffer+8
char 2	char 1	buffer+10
char 4	char 3	
		data area
. . .		
		buffer+130
		buffer+138.

TOUTT, TOUTS TABLES

time count decremented by clock interrupt

when = 0 call corresponding touts routine

ppt entry	console tty entry	toutt
tty0 "	(lp)	toutt+2
tty2 "	tty1 entry	toutt+4
tty4 "	tty3 "	toutt+6
tty6 "	tty5 "	toutt+8
	tty7 "	toutt+10
		toutt+12
		toutt+14
console tty subroutine entry point ~~entry~~ startty		touts (tout+16)
ppt " " " pptito		
(lp) ?		
xmtto xmtt1 ⋮ } 8 of these		
		touts+30

CC, CF, CL & CLIST TABLES

console output char count	console input char count	cc
ppt " "	ppt " "	cc+2
lp " "		cc+4
tty0 " "	tty0 " "	cc+6
tty1 " "	tty1 " "	cc+8
tty2 " "	tty2 " "	cc+10
tty3 " "	tty3 " "	cc+12
⋮	⋮	
tty7 " "	tty7 " "	cc+20
		cc+22
⋮	⋮	
		cc+28
console input 1st char offset	free list 1st char offset	cc+30 cf:
ppt input 1st char offset	console output 1st char offset	cf+2
	ppt output 1st char offset	cf+4
tty0 input first char offset	lp " " " "	cf+6
tty1 " " " "	tty0 " " " "	cf+8
⋮	⋮	
	tty7 " " " "	cf+12
⋮	⋮	
cl(cc+31) freelist last char offset		cf+30

console tty output last char off	console tty input last char off	cl+1
ppt " " " "	ppt " " " "	cl+3
lp " " " "		cl+5
tty0 " " " "	tty0 " " " "	cl+7
⋮	⋮	
tty7 " " " "	tty7 " " " "	cl+21
		cl+23
⋮	⋮	
		cl+29
pointer to next char (0)	character (0)	clist (cl+3
⋮	⋮	
pointer to next char (204)	character (204)	clist+508

active -
is a word whose bits encode the activity states of the various block-structured device controllers. If the RK disk bit is on, that device is running and should not be molested. The devices for the bits are:

bit	device
0	drum
1	disk
2	dectape

buffer -
start of the buffers used for block-structured device I/O (there are "nbuf" of them) and typewriter input (there are ntty of them).

From buffer to buffer + 1119., are the 8 tty buffers. From buffer + 1120. to buffer + 1259. is the console tty buffer. Each of these buffers is 70. words long. From buffer + 1260. to buffer + 4381. are the disk buffers. They are 256. words each plus 4 words which represent an I/O queue entry. Thus each block is 260. words. Pointers to these 260. word buffers are contained in bufp. bufp contains pointers to the I/O queue entrys of each buffer. For more information, see H.O, p. 2.

bufp -
contains pointers to the block-structured device buffers. It is 9 words long. The first 6 entries point to the I/O queue entries of the 6 buffers. The last 3 words contain:

sbo - address of I/O queue entry for the super block of the RF disk.

sbi - address of I/O queue entry for the super block of the dismountable device.

swp - address of I/O queue entry for the core image being swapped in or out.

cc -
is a 30. byte table. Each entry contains a count of the number of characters in the associated queue for that entry. The characters have either been received from a character oriented device, or are waiting to be output.

cdev -
The current device number. It is set up during the scan of a file name, and is an implicit argument to the routines which do I/O by device block number. cdev= 0-drum, 1-disk, 2... dec tape. This parameter is 1 word.

cf -
is a 31. byte table. Each entry points to the first character in an associated character queue. The first entry refers to the free list of character blocks. The pointers are offsets, divided by 2, in the "clist" table.

cl -
is a 31. byte table. Each entry points to the last character in its associated character queue. The pointers are offsets, divided by 2, in the "clist" table.

clist -
is a 510. byte table containing linked lists of input or output characters. Each entry is a word; the low byte contains the character; the high byte contains a pointer to the next byte in the list. The pointer is a word offset in "clist".

clockp -
points to one of the clock cells in the super block (1 word).

core -
address of the beginning of user core.

dae -
disk address extension error reg. for RF-11 disk. (See Section G, pg 35)

dcs -
disk control and status register. (See Section G, pg 34)

deverr -
a seven word table containing the error status of devices. The index into this table is the device no. 'cdev'.

word	device	codes
1	drum	0= no error, 1= error
2	disk	"
3	dectape units	"
4	"	"
5	"	"
6	"	"
7	"	"

ecore -
address of the end of users core.

fsp -
this table contains 8 bytes for each currently open file. It must be kept on a per-system basis since the same instance of an open file can be referred to by more than 1 process. This table has 1 entry for each "open" or "creat" call. Each entry contains information about an open file. The fsp table is indexed by the u.fp list. (See Section F, pg 8 for details.) The table is 400 bytes long.

idata -
This 448. byte area contains assembled root, device, binary, etcetra, user and temporary directories and the cold boot initialization program directory. (See Section F, page 7 for a description of directory structure.) Preceeding each of these assembled directories establishing i-nodes for the directories. Namely:

```
          A                    A = i-node number
          B                    B = i-node flags (See Section F, p. 5)
    D             C            C = number pf links
          E                    D = user id of owner
                               E = directory size in bytes
```

Following the 4 word area is the directory associated with it. These directories are used in initializing the system during cold boot.

idev -
the device number of the current i-node (1 word). See ii.

ii -
the i-number of the i-node currently in the 'inode' area of core (1 word).

imod -
a flag set when the current i-node (ii) is modified. Whenever the current inode is changed, or whenever an exit to a user program takes place, this flag causes the i-node to be written out. This flag is 1 byte.

inode -
lays out the structure of an i-node. Each i-node (32 bytes) specifies a file. While a particular file is under consideration, a copy of its i-node resides here. The current i-node number is kept in "ii" and its device in "idev". Labels beginning "i." refer to locations in this area. (See Section F, pg. 5.)

i.ctim -
creation time of the file. (2 words)

i.dskp -
start location of an 8 word 'address' portion of the i-node. Each word contains a physical block number, from which a physical block address can be calculated. The index into this 8 word section of the inode can be considered a logical block number. if the file associated with the i-node is small (< 8 blocks). If the file is large (> 8 blocks), the physical block number indicates an indirect block which contains 256. words, each of which contains a physical block no. for a block associated with this file. A zero physical block no. in either the address words of the i-node or in an indirect block indicates that the corresponding block has never been allocated.

i.flgs -
flags (1 word) for the file are coded as follows:

Bit	0	set indicates	- write, non-owner
"	1	"	- read, non-owner
"	2	"	- write, owner
"	3	"	- read, owner
"	4	"	- executable
"	5	"	- set user ID on execution
"	6	"	- These bits are not assigned
"	7	"	- "
"	8	"	- "
"	9	"	- "
"	10	"	- "
"	11	"	-
"	12	"	- large file
"	13	"	- file has been modified (always on)
"	14	"	- directory
"	15	"	- i-node is allocated

i.mtim -
modification time of the file (2 words).

i.nlks -
number of links (directories) this file appears in. (1byte)

i.size -
size of file in bytes. (1 word)

i.uid -
id of the file owner (1 byte)

lks -
clock status register. (See Section G, pg 36)

mmod -
corresponding byte flag of imod above for the currently mounted desmountable file system.

mntd -
is the internal device number corresponding to the device on which a removable file system is mounted. It is used with "mnti". (1 word)

mnti -
records the i-number of the (inique) cross device file. That is, whenever this i-number is referred to on the RF disk, it will be translated into the root directory on the mounted device. (1 word)

mount -
is the in core image of the super block for the dismountable file system currently mounted. It contains the i-node map and free map for the device.

mpid -
is the source of unique identifiers (names) for processes. It is incremented as each process is created. (1 word)

nbuf -
number of block-structured I/O buffers. Presently its 6 (for cold boot 2).

nfiles -
allowable number of open files in system. Presently 50.

nproc -
number of processes. Presently 16.

ntty -
number of tty's. Presently 9

orig -

partab -
128. byte table.

ppb -
papertape punch buffer register. (See Section G, p. 38)

pps -
paper tape punch status register (See Section G, p. 37).

ptiflg -
indicates the status of the paper tape file. (1 byte)
0 - file not open
2 - file just opened
4 - file is normal
6 - file not closed, error situation

prb -
paper tape reader buffer register. (See Section G, p. 37).

proc -
is a table with an entry for each possible process. The number of processes is given, by 'nproc'. Its length limits the number of processes which can be created, since it is always in core. Subtables in the process table have names beginning with "p.".

prs -
paper tape reader status register. (See Section G, p. 37).

ps -
processor status register. (See Section G, p.)

p.break -
a 16 word table. Each word is associated with a unique process and contains the first core address not used by the process.

p.dska -
is a table of disk addresses for the swap area of the 16 processes. p.dska is 16 words long. Each word contains a block number for each process.

p.link -
is a 16 byte table indexed by process number. Given that a process is on the run queue, its p.link byte is 0 (in which case the process has no successors) or it contains the process number of the next process to be run after the process that owns that slot. If process number 2 was running next on the queue and process number 8 was next, the 2nd byte of the p.link table would contain an 8. This is how the next process in line is linked to the one ahead of it.

p.pid -
is a 16 word table that contains the unique identifier (or name) of a process. It is indexed by 2 X (the process number). The name of the process is actually a unique number.

p.ppid -
is the unique identifier (name) of the parent of the particular process. The table is 16 words long and is indexed by 2 X (the childs process number). This is where a child searches for its parent. Process number 2 would look in the 2nd word of the p.ppid table for its parent.

p.stat -
is 16 bytes long. Each byte represents the status of a process. Each byte is indexed by the process number. The status's are as follows:
0 - indicates the process is unused or free.
1 - indicates the process is active ~~as free~~.
2 - indicates the process is waiting for a process to die.
3 - indicates a zombie (the process has died but it has not been waited for.)

rcbr -
receiver buffer register for the DC-11.

rcsr -
receiver status register for the DC-11. (See Section G, p. 26)

rfap -
address of the drum buffer I/O queue entry. It is passed as an argument to "trapt".

rkap -
address of the disk buffer I/O queue entry. It is used as an argument to "trapt".

rkcs -
control status register of the disk. (See Section G, p. 30)

rkda -
disk address register. (See Section G, p. 29)

rkds -
disk drive status register. (See Section G, p. 28)

rootdir -
is the i-number of the root directory. It is set to 41. by the initialization code and is never changed.

runq -
is a table of length 3, with one entry for each of the three ready-to-run queues of processes. The low byte of each entry contains the process number of the first process in the queue; the high byte contains the process number of the last process. The entry is 0 if there are no processes on the queue. Each queue is linked by the p.link entry in the process table (see above).

sb0 -
is the I/O queue entry for the super block for the permanent device (RF disk). It is 4 words long.

sb1 -
is the I/O queue entry for the super block for the dismountable device. It is 4 words long.

smod -
is a byte flag that is set whenever the super block is modified. During an exit to a user program, the super-block is written out if this flag is set.

swp -
is the I/O queue entry for the core image being swapped. It is 4 words long.

sysflag -
tells whether execution is going on inside the system or not. It is 0 if a system routine is executing and -1 if a user program is running. This is a byte flag.

sstack -
is a temporary stack used to store the stack during swaps.

systm -
is the in-core image of the super block for the RF fixed head disk. It is updated onto the RF wherever it is changed. This area consists of 130. bytes of free-storage map (described in Section F, p.), 64. bytes of I-node map (described in Section F, p.), and 22. bytes of time accounting and error count information. Labels in this area start with 's.'

s.chargt -
is the time charged to users.

s.drerr -
is the drum error count.

s.idlet -
the time the system is idling.

s.syst -
is the overhead time during which the processor is executing in the operating system code.

s.time -
is the total time since the system was last cold booted.

s.wait -
is the disk I/O wait time.

tcap -
is the pointer to the dec tape I/O queue entry (1 word).

tcba -
is the bus address register of the DEC TAPE. (See Section G, p. 32.)

tcbr -
is the transmitter buffer register of the DC-11.

tccm -
is the command register for the DEC TAPE. (See Section G, p. 32)

tcdt -
is the data register for the DEC TAPE. (See Section G, p. 33)

tcerrc -
(1 word)

tcsr -
is the transmitter status register of the DC-11. (See Section G, p. 27)

tcst -
is the control and status register of the DEC TAPE. (See Section G, p. 31)

tcstate -
is the state of the DEC TAPE, e.g., idling, searching doing I/O. (1 word)

tcwc -
is the word count register of the DEC TAPE. (See Section G, p. 32)

touts -
is a 16. word table. Each word, if non-zero, is the entry point of a subroutine. The table is used to implement

'toutt' byte is decremented, if it reaches zero

interval timing in conjunction with the 'toutt' table described below.

toutt -
is a 16. byte table. Each byte is a count. At each clock interrupt each non-zero, the corresponding "touts" subroutine is called. All entries in these tables are fixed.

tkb -
is the tty reader buffer register. See Section G, p. 39.

tks -
is the tty reader status register. See Section G, p. 39.

tpb -
is the tty punch buffer register. See Section G, p. 39.

tps -
is the tty punch status register. See Section G, p. 39.

tty -
contains 8 bytes for each DC-11 communications interface configured. Control and status information is kept therein. These are referred to as tty blocks. There are ntty (9) of them . The last one is for the consold tty. For their contents see F, page 11.

ttyoch -
is used during output to the console typewriter. (1 byte)

user -
is the start of each users data base. It resides just below the users core area and is swapped with the user. All locations in this section begin with "u".

u.base -
holds the "users buffer" address in core during read and write calls. Also points to u.dirbuf in "mkdir".

u.break -
holds the process program break point as set by sysexec or by a sysbreak. It is the location at the end of the users program used in the swap routines. (1 word)

u.bsys -
is set while a process is about to be terminated for some error. A core image is produced. (1 byte)

u.cdev -
holds the device number of the users current directory. (1 word)

cdev	device
0	drum
1	disk
other	dectape

u.cdir -
is the i-number of the processes current directory. (1 word)

u.count -
is the number of bytes to be transferred during read or write operations. This variable is 1 word.

u.dirbuf -
usually holds the i-number of an i-node in "maknod" and "mkdir". (The i-number of a new i-node) u.dirbuf + 2... u.dirbuf + 9 hold the name of the file in the directory entry.

u.dirp -
is either an offset within a directory for a file mentioned by the user or a pointer to an empty directory slot during a "creat". It also points to a directory entry in "namei". (1 word)

u.fofp -
is a word that contains a pointer to the 3rd word of an fsp table entry. This (3rd) word contains an offset (in bytes) into the file associated with the fsp table entry, and is used during read/write operations. In initializing special files, u.fofp points to u.off. For bread and bwrite, u.fofp contains a block number.

u.fp -
is a list of users open files. An entry is either 0, for a non open file, or is an index into the systems fsp table (table of open files). Each byte in the list contains an entry. The list is 10 bytes long, because 10 is the maximum number of files a user can open at once. The index into this u.fp list is called a "file descriptor". It has a value from 0 to 9.

u.ilgins -
determines handling of illegal instructions. If u.ilgins is 0 - the normal instruction trap handling is done the process is terminated and a core image is produced.

If u.ilgins is

a location - control is passed to that location when the trap occurs. This feature is used to implement the floating point instructions. (1 byte)

u.intr -
determines the handling of interrupts. If u.intr is zero - interrupts (ASCII delete) are ignored.
is one - interrupts cause there normal result, ie, force an exit.
is a location - control is passed to that location when an interrupt occurs. (1 word)

u.namep -

is a pointer to a file name mentioned by a user to the system during system calls. (1 word)

u.nread -
accumulates the number of bytes transmitted during read or write calls. (1 word) It is passed back in r0 on return.

u.off -
is either a pointer to a file offset mentioned by a user during "seek" and "tell" calls or a pointer to an empty directory slot in "mkdir" or a pointer to a directory entry as in "sysunclink". (1 word)

u.pri -
holds the process priority expressed as a pointer to one of the three run queues (in one word). If another process with higher priority becomes ready to run while this process is running, the remainiig time quantum is set to zero.

u.quit -
determines the handling of quits. If u.quit is.
0 - quit signals are ignored (ASCII FS).
1 - quits are re-enabled and cause execution to cease and a core image to be produced.
a location - control is transferred to that location when a quit signal is received (1 byte).

u.r0 -
points to the location where the users r0 was stored on entry into the system (and where it will be restored on return). It is used to pick up and pass arguments. Most often it passes file descriptors. (1 word)

u.ruid -
holds the real user id number. It is not changed by the set-user id bit being on in an inode during a "sysexec" (1 byte).

u.sp -
is used to save the value of the users sp register after all the other registers have been saved. It is used to restore the sp when returning to a user so the system need not take care to pop everything off the stack before returning (1 word).

u.ttyp -
is a pointer to the buffer of the tty that is in control of the process. The control tty (typewriter) is the only one which may quit or interrupt a process.

u.uid -
holds the user id number used to determine protection (1 byte).

u.uno -
is the process number. In sysfork it is the parent process

number. In "sysexit" it is the process number of the dying process. In "swap" it is the number of the process being swapped out.

u.usp -
is the contents of the sp at the moment the user is swapped out. It must be saved so that the appropriate return can take place after the user is swapped back in. (1 word)

u.quant -
is the users time quantum. It is set to 30. when a new user is swapped in. At every clock tick it is decremented. When it reaches zero the user is swapped out (1 byte).

wlist -
is a 40. byte table of "wait channels". Each byte is considered a channel. Each entry in this table is associated with a particular event. When a process wishes to wait for one of these events, it calls a routine (sleep) which enters the process number in the appropriate channel in this table. When the event occurs, another routine (wakeup) wakes up the process.

ID - uo; 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.O, page 3.

CALLING SEQUENCE -

ARGUMENTS -

INPUTS -

ntty (number of DC-11 interfaces)

OUTPUTS -

(see diagrams H.O page 3), r0, r1

ID - uo; 3/allocate disk buffers

FUNCTION -

Block I/O devices (drum, disc, 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 "sb0", "sb1", 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.O, page 3.

CALLING SEQUENCE -

ARGUMENTS -

INPUTS -

r0 (points to first block I/O buffer)

OUTPUTS -

(see diagrams H.O page 3) r1 (internal counter, r2 (internal pointer)

ID - uo; 3/free all character blocks

FUNCTION -
this segment of code initializes the cf, cl and clist blocks in core to the following state:

```
                 255.   255.   cf
                          .
                          .
                          .
   (cf+31.) cl     1
                          .
                          .
                          .
                               clist (cf + 31.)'
                   1
                          .
                          .
                          .
                 253.          clist + 506.
                 254.
```

CALLING SEQUENCE -

ARGUMENTS -

INPUTS -

OUTPUTS -

CALLED BY -

CALLS - PUT

ID - uc; 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:

```
                943.     p.dska
                926.
             .
             .
             .
        960.-nproc*17.   p.dska + 2*nproc -2
```

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. System (which represents the number of bytes in the free storage map) is set to 128. System + 130. (which represents the number of bytes in the i-node map) is set to 64.. (See Section 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)

ID - uo; 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 RF03 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 Section 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.)

ID U1;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 area then set up and the core image is written out in the users directory. Then the users area of core is 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 mode by "maknod".

OUTPUTS -

u.bsys - turn on. Its the users bad system flag.
u.base - holds address of "core", and user during write i-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 U1;7 error 2

FUNCTION - See 'error' routine

CALLING SEQUENCE - "

ARGUMENTS - "

INPUTS - "

OUTPUTS - "

ID U1;5 error 1

FUNCTION - See 'error'

ARGUMENTS - "

CALLING SEQUENCE - "

INPUTS - "

OUTPUTS - "

ID U1;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 U1;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 exists an 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 U1-4 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), u.intr is checked to see if it is equal to the process "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 sysintr in the UNIX Programmers Manual).

OUTPUTS -

clock pointer is popped.

- If the interrupt char is a quit character,
(r1)+6, the interrupt character in the control tty's buffer, is cleard
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 U1;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. If 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 U1;8 sysclose

FUNCTION -
"sysclose", 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" H.2.)

CALLING SEQUENCE -
sysclose

ARGUMENTS -

INPUTS -
(u.r0) - file descriptor

OUTPUTS -
See fclose outputs

ID U1;7 syscreat

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 searched 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 -

syscreat; 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
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 U1;7 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) instructor 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 U1;3 sysexit

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 -

sysexit or conditional branch

ARGUMENTS -

INPUTS -

u.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

ID U1;5 sysfork

FUNCTION -

sysfork creates a new process. This process is referred to as the child process. This new process core image is a copy of that of the caller of "sysfork". The only distinction is the return location and the fact that (u.r0) 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.r0).

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.r0).

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 - pointers to parents process control tty buffer.
mpid - process name generator
u.fp - list index into the tsp table.
fsp - table of open files.

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 runq + 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.r0) containing the parents process name.
u.uno - is restored to the parents process number.
(u.r0) - 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 U1;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 - contains the printer control status
(r2)+4 - contains the mode control status

ID U1;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 modification time is overwritten by whatever get 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 - new modification time of the file

ID U1;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

ID U1;6 sysopen

FUNCTION -

"sysopen" opens a file in the following manner:

1) The second argument in a sysopen 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 file descriptor index to the u.fp list is pointed to by u.r0.

CALLING SEQUENCE -

sys open; name; mode

ARGUMENTS -

name - file name or path name

mode - 0 - open for reading
       1 - open for writing

INPUTS -

r1 - contains an I-number (positive or negitive depending on whether and open for read or open for write is desired.

OUTPUT -

entry in fsp table and u.fp list

*u.r0 - 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 beginning of an fsp entry.

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 U.1, page 1.) The i-number of the file is obtained via "rw1" and the data is read into core via "readi".

CALLING SEQUENCE -

sysread; buffer; nchars. 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.

ID U1;2 sysrele

FUNCTION -

"sysrele" first calls tswap it the time quantum for a user is zero (see sysret). It then restores the users registers and torns 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.
sysflag - turned off
clockp - points to s.chrgt+2

ID U1;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 bia "ppoke".

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

5) A check is made to see if the users time quantum (u-quant) 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 brsysret

ARGUMENTS -

INPUTS -

u.bsys - user's bad system flag
u.sp - user's 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
u.quant - user's time quantum

OUTPUTS -

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

ID U1;5 sysret 1

FUNCTION - see 'sysret'

CALLING SEQUENCE - "

ARGUMENTS - "

INPUTS - "

OUTPUTS - "

ID U1;7 sysret 2

FUNCTION - see 'sysret' routine

CALLING SEQUENCE - "

ARGUMENTS - "

INPUTS - "

OUTPUTS - "

ID U1;9 sysstty

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 or 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.r0). 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 an error return is made. If there are, the parents status is set 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 keep 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.r0)
If no zombies found - status of parent is set to 2 (waiting for child to die)
- parent is swapped out

ID U1-6 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 successful open or creat call (see sysopen or syscreat). The i-number of the file is obtained via "rw1" and the buffer is written on the output file via "writei".

CALLING SEQUENCE -

syswrite; buffer; nchar

ARGUMENTS -

buffer - location of contiguous bytes to be written
nchara - 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

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 onto 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.

CALLING SEQUENCE -

jsr r0, anyi

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 points 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

ID U2-6 arg

FUNCTION -

arg extracts an argument for a routine whose call is of form:

```
sys 'routine'; arg1
     or
sys 'routine'; arg1; arg2
     or
sys 'routine'; arg1;...; arg10 (sysexec)
```

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 pointer 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, sysent, sysilgins, sysmdate, gtty, sysunlink, sysfstat, syschdir, arg2, sysbreak, seektell, sysintr, sysquit, sysumount

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 pushed on stack
r1

ID U2-4 error 3

FUNCTION - See 'error' routine

CALLING SEQUENCE - "

ARGUMENTS - "

INPUTS - "

OUTPUTS - "

ID U2-1 error 4

FUNCTION - See 'error' routine

CALLING SEQUENCE - "

ARGUMENTS - "

INPUTS "

OUTPUTS - "

ID U2-1 error 9

FUNCTION - See 'error' routine

CALLING SEQUENCE - "

ARGUMENTS - "

INPUTS - "

OUTPUTS - "

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 $\neq 0$) 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.

CALLING SEQUENCE -

jsr r0, fclose

ARGUMENTS -

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, page 0)
the "number of processes" byte in the fsp entry is decremented
(see F, page 8)
r2 - contains i-number.

ID U2-4 getf

FUNCTION -
"getf" first checks to see that the user has 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.

CALLING SEQUENCE -
jsr r0, getf

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.

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" does not 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

ID 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 is not, 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.

CALLING SEQUENCE -

jsr r0, isown

ARGUMENTS -

INPUTS -

arguments of syschmod or syschown calls

OUTPUTS -

u.uid - id of user
imod - set to a 1
r2 - contains second argument of the system call

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 makdir E.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 0 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. If not the i-number is put in u.dir bit and a directory entry is made via mkdir. Then the new i-node is fetched into core and its parameters are set (see outputs).

CALLING SEQUENCE -

jsr r0, mknod

ARGUMENTS -

INPUTS -

r1 - contains mode ii - current i-number - should be at the current directory mq, r2 - bit position & byte address in i-node map

OUTPUTS -

u.dirbut - contains i-number of free i-node
i.flgs - clag 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

ID U2-2 mkdir

FUNCTION -

"mkdir" makes a directory entry from the name pointed to by u.unamep 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 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 i-number, the last 8 bytes a file name associated with the i-number.

namei scans the file path name until it reaches a "/" or a nul 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 nul 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 nul, etc. If no file is found return to nofile; otherwise normal.

CALLING SEQUENCE -

jsr r0, namei; nofile; 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
r2, r3, r4 (internal)
u.dirb - points to the directory entry where a match occurs in the search for the file path name.
If no match u.dirb points to the end of the directory and
r1 = i-number of the current directory

ID U2-8 seektell

FUNCTION -
seektell puts the arguments from a sysseek 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 brought 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

CALLING SEQUENCE -
jsr r0, seektell

ARGUMENTS -

INPUTS -
u.base - puts offset from sysseek or systell call
u.count - put pfrname from sysseek or systell call
*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

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, 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

INPUTS -
u.break - the current break point

OUTPUTS -
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 U2-6 syschdir

FUNCTION -
syschdir makes the directory specified in its argument the current working directory.

CALLING SEQUENCE -
syschdir; name

ARGUMENTS -
name - address of the path name of a directory terminated by a nul 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

ID U2-2 sysexec

FUNCTION -

sysexec initiates execution of a file whose path name if 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.ro and u.sp.

9. zeros user's core down to u.ro.

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.

continued on page 17

The layout of core when sysexec calls sysret is:

```
                 |                |
                 |----------------|  core
                 |----------------|
   user prog     |       .        |
                 |       .        |
                 |       .        |
                 |----------------|  (u.break)
                 |----------------|
       zeros     |       .        |
                 |       .        |
                 |       .        |
                 |----------------|  (u.sp) = (sp)
                 |       0        |
                 |----------------|
       zeros     |       .        |
                 |       .        |
                 |       .        |
                 |----------------|  (u.r0) : (u.sp) + 16
                 |       0        |
                 |----------------|
                 |     core       |
                 |----------------|
                 |       0        |
                 |----------------|
                 |       n        |
                 |----------------|
                 |     argp1      |
                 |----------------|
                 |       .        |
                 |       .        |
                 |       .        |
                 |----------------|
                 |     argpn      |
                 |----------------|
                 |----------------|  argp1
     <...\0>     |       .        |
                 |       .        |
                 |       .        |
                 |----------------|
                 |----------------|  argp2
                 |       .        |
                 |       .        |
                 |       .        |
                 |----------------|
                 |----------------|  argpn
     <...\0>     |       .        |
                 |       .        |
                 |       .        |
                 |----------------|
                 |----------------|  ecore
```

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)

INPUTS -
    namep
    argp

OUTPUTS -

ID U2-4 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 negitive) the i-number is set positive and a branch into sysstat is made.

CALLING SEQUENCE -
sysfstat; 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 -
"sysgetuid" 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 U2-1 syslink

FUNCTION -

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

```
          |---------------|      |-----------|     |-----------|
          |               |      |  i-node   |     |  name 1   |
          |---------------|      |   for     |     |   file    |
current   |               |      |  name 1   |     |           |
directory |---------------|      |   file    |     |           |
          |               |      |           |     |           |
          |---------------|      |           |     |           |
          | name 2 entry  |      |           |     |           |
          |---------------|      |-----------|     |-----------|

                   |---------------|
                   |  some other   |
                   |   directory   |
                   |---------------|
                   |    name 1     |
                   |---------------|
```

CALLING SEQUENCE -

syslink; name1; name2

ARGUMENTS -

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

INPUTS -

u.namep - points to the arguments above.

OUTPUTS -

entry in the current directory with name, name 2.
r1 - contains i-number of name 1 on exit and i-number of current directory intermittently during subr.
i.nlks - incremented by 1 to indicate another link added.
imod - set by call to setimod.

ID U2-8 sysquit

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 -
sysquit; arg

ARGUMENT -
arg - if 0 this call disables quit signals from the typewriter (ASC11 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 U2-4 sysret 3

FUNCTION - See "sysret" routine

CALLING SEQUENCE - "

ARGUMENTS - "

INPUTS - "

OUTPUTS - "

ID U2-1 sysret 4

FUNCTION - See "sysret" routine

CALLING SEQUENCE - "

ARGUMENTS - "

INPUTS - "

OUTPUTS - "

ID U2-1 sysret 9

FUNCTION - See "sysret" routine

CALLING SEQUENCE - "

ARGUMENTS - "

INPUTS - "

OUTPUTS - "

ID U2-C 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 ptrname is 0, the pointer is set to offset.

if ptrname is 1, the pointer is set to its current location plus offset.

if ptrname is 2, the pointer is set to the size of the file plus offset.

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

CALLING SEQUENCE -

sysseek; offset; ptrname

ARGUMENTS -

offset - number of bytes desired to move the r/w pointer by
ptrname - a switch indicated above

INPUTS -

u.base
u.count (See seektell)

OUTPUTS -

u.fofp - points to the r/w pointer in the fsp entry.
The r/w pointer is changed according to offset and ptrname.

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 sysstat (II).

CALLING SEQUENCE -

sysstat; 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.

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 and u.uid and the real user u.ruid are set to this. Only the super use and make this call.

CALLING SEQUENCE -
syssetuid

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 - set equal to the process id (u.r0)

ID U2-7 systime

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 systime

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

CALLING SEQUENCE -
systime

ARGUMENTS -

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

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

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 (e-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).

CALLING SEQUENCE -

syslink; name

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

ID U2-2 wdir

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

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

ARGUMENTS -

INPUTS -
u.dirbuf - address of where name of directory is kept
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

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

Bits 9 and 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'.

CALLING SEQUENCE -
jsr r0, clear

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.

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 restored to original value

ID U3-3 putlu

FUNCTION -

"putlu" is called with a process number in r1 and a pointer to the lowest priority Q (runq+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 process's link. (The last process's 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:

```
|------------|------------|           |------------|
|      n     |            |  runq+4   |      n     | p.link + L-1
|------------|------------|           |------------|
 previously held "L"                   1 byte in length
```

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
r2 - points to lowest priority queue

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.

CALLING SEQUENCE -
jsr r0, rswap

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 emt's
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

ID U3-1 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.

ARGUMENTS -

INPUTS -

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

OUTPUTS -

present process to its disk block
new process into core
u.quant = 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 -
r0 - 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 cf users program)

OUTPUTS -

stack gets moved if proper conditions stated above are met.

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" is shown 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 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
p.dska - holds block number of process

OUTPUTS -

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 zero (≠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
u.quant - 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
u.quant - decremented
toutt - entries decremented
r0 - contains users r0 if conditions of (2) above are met

ID U4-3 ppti - paper tape 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 I/O 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 less than 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 paper tape input interrupt routine

INPUTS -

pptiflg - flag which indicates function tube performed
prs - paper tape read status bits
cc+2 - character count for ppt input in clist
prb - input character

OUTPUTS -

pptiflg - (see above)

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. Nothing happens except the return is skipped when:

Case I
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 - determines handling of interrupts if 0 - nothing happens
tty+6 - pointer to buffer of first tty block

OUTPUTS -
Case I - nothing except return is skipped
Case II - processor priority = 5
gets - erases the output character list

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 - values of this parameter indicates to pptito the function it is to perform
prs - status of ppt reader

OUTPUTS -

toutt+1 - contains tic count (= 10) 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 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 "sys-ret" 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
w.list - wait channel list
rung+1 - lowest priority run Q

OUTPUTS -

sleeping process number onto wlist
sleeping process onto disk

ID U4-2 ttyi

FUNCTION -

"ttyi" puts a character from the tty reader buffer 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 clist via "putc". It is then put on the tty output buffer via "startty". 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 last tty blocks buffer pointer is zero wakeall is called and all processes are put on the low priority queue.

If the last tty blocks buffer pointer to 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 - tty 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.

ID U4-3 ttyo

FUNCTION -
"ttyo" is the console typewriter output interrupt routine. It calls setisp to save registers during the interrupt then calls startty 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 startty

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 to gether 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 U4-5 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 awakened.

INPUTS -

wlist - wait channel
u.pri - users process priority

OUTPUTS -

if u.pri > arg1 uquant = 0
wlist (r3) = 0 - 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 U4-5 starppt

FUNCTION -

"starppt" checks the character count for ppt output in the clist. If it is greater than 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 U4-3 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 an rt1 and returns.

CALLING SEQUENCE -
jmp retisp

ARGUMENTS -

INPUTS -

OUTPUTS -
r0, r1, r2, r3, clockp

CALLED BY -
trapt

CALLS -

ID U4-1 setisp

FUNCTION -
"setisp" stores r1, r2, r3 and clockp on the stack. Puts $s.systt2 in clockp and returns via a jump without popping the stack.

CALLING SEQUENCE -
jsr r0, setisp

ARGUMENTS -

INPUTS -

OUTPUTS -

CALLED BY -
drum

CALLS

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 read bit, if bit is clear; return.

3 - if bit is set check time out 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.

CALLING SEQUENCE -
jsr r0, startty

ARGUMENTS -

INPUTS -
ttyoch (character to be output), toutt

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

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 whether user can open file for reading or writing based on file protection bits in "i.flgs" (see Section G).

CALLING SEQUENCE -
jsr r0, access; arg.

ARGUMENTS -
arg0 (user, owner flagmask)

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

OUTPUTS -
inode, r2 (internal)

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 super block), mount (dismountable super block), r2 (internal), r3 (internal).

ID U5-2 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, nonzero=mountable device

OUTPUTS -

mount - systm+(r2) word in free storage map portion of the in core image of the super block 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 super block for the fixed head disk.

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 - saved on stack and restored on return

ID U5-4 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 Section F for general discussion of inodes.)

CALLING SEQUENCE -

jsr r0, icalc; arg

ARGUMENTS -

arg - arg - 0 read inode
arg = 1 write inode

INPUTS -

inode - r1 (i-number)

OUTPUTS -

inode - r1 (internal), r5 (internal), r3 (internal)

ID U5-4 iget

FUNCTION -
"iget" gets 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 i-node 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".

CALLING SEQUENCE -
jsr r0, iget

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

ID U5-3 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

CALLING SEQUENCE -

jsr r0, imap

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

ID U5-5 itrunc

FUNCTION -

"itrunc" truncates a file whose i-number is given in r1 to zero length. "itrunc" gets an 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.

CALLING SEQUENCE -

jsr r0, itrunc

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 Section F, page 5
i.size - size of file

OUTPUTS -

i.flags - "large file" flag is cleared
i.size - set to 0
i.dskp - idskp+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

ID U5-1 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 (i.e. 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 8 * 512. 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 assigned to the file. The block pointers in i-node are moved into 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 new entry 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. A new 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)

ID U5-3 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 U6-4 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.

CALLING SEQUENCE -
jsr r0, cpass

ARGUMENTS -

INPUTS -
u.count - users character count
u.base - points to a users character buffer

OUTPUTS -
if u.count $\neq$ 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

ID U6-1 readi

FUNCTION -
"readi" reads from an 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.

CALLING SEQUENCE -
jsr r0, readi

ARGUMENTS -

INPUTS -
u.count - byte count user desires
u.base - points to user buffer
u.foft - points to word with current file offset

OUTPUTS -
u.nread - accumulates total bytes passed back
see "dskr"

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.

CALLING SEQUENCE -
jmp dskr

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

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 number 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 (imod) is set. 'dskw' then uses (u.fofp) to calculate 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 overwritten. 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 - number 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 number is in sp).
see inputs for "iget", "setimod" "mget", "dskrd", "wslot", "sioreg"
r1 - pointer to callers data area
(r1), (r1), +1,..., (r1) + (u.count-1) - the callers data

OUTPUTS -

i.size - file size (may have been modified by (dskw)
see outputs for "iget", "setimod", "mget" "dskrd", "wslot", "sioreg"
r1 - points to the location succeeding the last caller data byte transferred
r2 - points to the location (in the system buffer) succeeding the last system buffer byte overwritten.
r3 - 0
u.count - 0
modofied drum file

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 if 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 number of bytes transferred
u.count - the number 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 and 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 U6-2 rcrd

FUNCTION - See "error" routine

CALLING SEQUENCE - "

ARGUMENTS - "

INPUTS - "

OUTPUTS - "

ID U6-2 ret

FUNCTION -

"ret" is a special subroutinevreturn, used by the following subroutines:

1. reti
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.

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.

ARGUMENTS -

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 paper tape

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 r0, 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 "sysopen" for reading 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 "sysread" 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 U6-3 wppt - write paper tape

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 "sysopen" for writing. This places entries in the fsp table and the user's fp area.

Once the file is open a "syswrite" 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 syswrite. 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. "writei" 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 "syswrite". If not "pptoc" is called. "pptoc" first checks to see if character count for ppt output in the "clist" is 250. 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 less than 10. If it is it will wake up the process in the wlist entry for ppt output.

As seen from above a process puts itself to sleep when it has 250. characters in clist and is "awakened" by the paper tape output interrupt routine (ppto) when the count becomes less than 10.

CALLING SEQUENCE -

jmp ppt

INPUTS -

(see inputs for "cpass" and "pptoc")

OUTPUTS -

(see outputs for "cpass" and "pptoc")

ID U6-5 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 preceding 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 user's 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)	wrf0
RK03/RK11 : movable head disk	wrk0
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 -
n srvr0, writei

INPUTS -
u.count - contains a count of the number of bytes to be written
vr1 - 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 U6-3 wtty

FUNCTION -

"wtty" uses "cpass" to obtain the next character in the user buffer area. If the character count for console tty is greater than or equal to 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 awakened by "wakeup", it again tries to find a location available in "freelist" and the character count for the console tty less than 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 U7-1 canon

FUNCTION -

canon handles the erase kill processing on the teletypewriters. (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) an eot (004) has been received, or (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 '@' a return to the beginning is made. Actually one starts over.

4) If the character is an erase '#', the next character 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 when 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 new line (\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 $\neq$ 0 the character pointer, 4 (r5) points to the first character not yet picked up.

CALLING SEQUENCE -

jsr r0, canon, arg

ARGUMENTS -

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

```
                    |-----------------------------|
                    |                             | tty buffer
                    |-----------------------------|
                    |  number of char in buffer   | +2
                    |-----------------------------|
                    | char pointer (buffer +10 to | +4
                    |             start)          |
                    |-----------------------------| +6
                    |                             |
                    |-----------------------------| +8
                    |                             |
                    |--------------|--------------| +10
                    |              |              |
                    |--------------|--------------|
  character         |                             |
   storage          |                             |
    area            |                             |
```

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 return is not skipped.

CALLING SEQUENCE -
```
jsr r0, cesc; arg
```

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

INPUTS -
r1 - character to be tested
2(r5) - character count
*4(r5) - previous character

OUTPUTS -
skip return if test char is not erase or kill
if character was erase or kill
2(r5) - character count gets decremented
4(r5) - character pointer gets decremented

ID U7-7 cppt - close paper tape file

FUNCTION -
"cppt" assigns all ppt input locations in clist to freelist and sets "ppt, flg" to indicate file closed (=0).

CALLING SEQUENCE -
jmp cppt

INPUTS -

OUTPUTS -
See outputs for "getc".
ps - processor priority set to 5
pptiflg - set to "0" to indicate file closed

ID U7-6 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 error a

FUNCTION - See "error" routine

CALLING SEQUENCE - "

ARGUMENTS - "

INPUTS - "

OUTPUTS - "

ID U7-3 get

FUNCTION -

Removes the first clist entry from the 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.

CALLING SEQUENCE -

jsr r0, get; empty: ; normal:

ARGUMENTS -

INPUTS -

r1 (list identifier), cf+1(r1), cf+1(r1) (see Section 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)

ID U7-2 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.

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

ARGUMENTS -
arg - list identifier

INPUTS -
r2 (clist offset from put)

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

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 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.

CALLING SEQUENCE -

jsr r0, iclose

ARGUMENTS -

INPUTS -

r1 - contains i-number of file being closed

OUTPUTS -

If special file, r1 is put on the stack, i.e., the i-number is put on the stack.

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 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 U7-5 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 ti - contains count for ppt input
See outputs for "getc".

ID U7-5 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 number 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 (number 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"

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 less than 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 vreturns 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 clist character count for ppt input
prs - contains status bits for ppt reader
pptiflg - indicates condition of ppt file

OUTPUTS -
prs - contains reader enable bit
see outputs for "getc"
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 greater than 50. If it is, the process is put to sleep. If it isn't "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 fluf
see outputs for "starppt" and "sleep" and "putc"

ID U7-3 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.

CALLING SEQUENCE -

jsr r0, put

ARGUMENTS -

INPUTS -

r1 (list identifier)
r2 (clist offset)

OUTPUTS -

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

ID U7-3 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.

CALLING SEQUENCE -

jsr r0, putc; arg

ARGUMENTS -

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

INPUTS -

r1 - character from device buffer.

OUTPUTS -

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

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, i.e. the first block on the mountable file system is read in. This block is the super block for the file system. This call is super user restricted.

CALLING SEQUENCE -

sysmount; special; nami

ARGUMENTS -

special - pointer to name of special file (device)
name - pointer to the name of the "toot directory of the newly mounted file system. "name" should alwasy 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 U7-8 sysumount

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 U7-8 sysreta

FUNCTION - See "sysret" 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 number 7, page 2 for others.

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 us 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.8, page 9) 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
u.quant - 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 U8-3 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 - updated by adding r3
u.count - updated by subtracting r3
r2 - has value of u.base before it gets updated

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 bookkeeping 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 block number
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 -
(u.fofp) is the next block to be written into
r3=0 (used internally)

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 clockp are saved on the stack (see setisp) calls "trapt" to check for stray interrupt or error. If neither, it clears bits 12 and 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 -

same as setisp, trapt and retisp

CALLED BY -

interrupt vector

CALLS -

setisp, trapt

ID U8-4 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 structured 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:

```
       write bit -----------------------|
          read bit -----------------|   |
   waiting to write bit ----------|   |   |
                 read
waiting to write bit ---------|   |   |   |
     inhibit bit ---------|   |   |   |   |
                          |   |   |   |   |
                          |   |   |   |   |
                         15  13  12  10   9    7             0
                        |---------------------|---------------|
                        |                     |  device id    |
                        |---------------------|---------------|
                        |   physical block number             |
                        |-------------------------------------|
                        |     word count (-256)               |
                        |-------------------------------------|
                        |        bus address                  |
                        |-------------------------------------|
```

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 Section F, 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~~12 or ~~12~~.13 The routine called upon completion of the I/O operation will clear bit ~~11~~ 12 or ~~12~~ 13 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

$$\text{sector number} = \text{remainder } \frac{N}{12.}$$

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

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

"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.

```
          1     0  15           11  10                  0
          |-------|----------------|---------------------|
    dae   |       |                |                     |   dar
          |-------|----------------|---------------------|

                      track                  word
```

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.

CALLING SEQUENCE -
jsr r0, poke

ARGUMENTS -

INPUTS -
buffer pointers,
I/O queue entries

OUTPUTS -
sets bits 12 and 13 on I/O queue entries where I/O operation is initiated.

ID U8-5 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 buffers 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.

CALLING SEQUENCE -

jsr r0, dskrd

ARGUMENTS -

INPUTS -

OUTPUTS -

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

ID U8-3 dskwr

? clear!

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.

CALLING SEQUENCE -

jsr r0, dskwr

ARGUMENTS -

INPUTS -

OUTPUTS -

(bufp)

ID U8-3 error 10

FUNCTION - See "error" routine

CALLING SEQUENCE - "

ARGUMENTS - "

INPUTS - "

OUTPUTS - "

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 stat 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 U8-8 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 U8-2 tst devc

FUNCTION -
"tstdevc" 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, tstdevc

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 U8-3 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-sec 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 p. 1. 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. If it is empty, the process is put to sleep. When the process is awakened, rcvch again tries to obtain a character from clist.

CALLING SEQUENCE -

jsr r0, rcvch

INPUTS -

r2 - contains 8xtty no.
mcsr + 8xttyn - carrier detect and 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 users buffer area.

CALLING SEQUENCE -

jmp rcvt

INPUTS -

r1 - contains 2xttyno.
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 and rcvch

OUTPUTS -

ps - "set processor priority to 5"
See "canon", "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 less than 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 to ASC11 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 present. 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 (ASC11 40).

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 ASC11 codes 40-176, increments column pointer which is in byte 2 of tty area.

b. For ASC11 codes 0-7, 16-37 and 177, does nothing.

c. For ASC11 0 10 (bs), decrements column pointer.

d. For ASC11 012 (lf), checks for setting of cr flag (bit 4 of 4th byte in "tty" area). If it is set ASC11 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 both "lf" and "cr".

e. For ASC11 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 ASC11 013 (vt), determines value for the tty's output entry in tout table.

g. For ASC11 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 8xtty number
tty+3+8xttynumber - 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+8xttynumber - contains column pointer for tty
tty+2+8xttynumber - contains character left over after lf for tty
tty+4+8xttynumber - contains flags for tty

See outputs for "getc".

rcsr+8xttynumber - contains carrier present flag for tty
tcsr+8xttynumber - contains ready flag for tty

OUTPUTS -
    See inputs to "getc"
    cc+(tty+3+8xttynumber)
    tty+1+8xttynumber   see inputs above
    tty+2+8xttynumber
    tcbr+8xttynumber - contains character to be output on tty
    toutt+3+ttynumber - 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 greater than 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 awakened by "awake", it trys again to find a location available in freelist and a character count for the tty output less than 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 uponvreturn from "cpass"

OUTPUTS -
See inputs for "starxmt" and "putc"
processor priority set to 5