```
SSR2    = 177576
KISA0   = 172340
KISA6   = 172354
KISD0   = 172300
[illegible] = 172522
[illegible] = 172440
UI[illegible] = 177640
UIS[illegible] = 177642
UISD[illegible] = 177600
UISD1[illegible] = 177602
IO      [illegible] 7600

.data
.globl  [illegible], _cputype
_ka6:   KI[illegible]
_cputype:40[illegible]

.globl  _pirr
_pirr:  pirr
pirr:   0

.bss
.globl  nofault, ssr[illegible] badtrap
nofault:.=.+2
ssr:    .=.+6
badtrap:.=.+2
.globl _mapalloc
.globl _msginit
.globl _serrlog
```

```
/       @(#)mch.70.s    2.10.1.1

/ machine language assist
/ set up for 11-45 I and D space

.fpp = 1
.mt = 0                 /1 = tm11       0 = tju16
.pf = 1                 / Power fail restart
.xclist = 0             /1 = external clist,  0 = no external clist
                        /must also define NXCLIST in param.h
CBITS = 37              / mask bits--determine size of clist packet
                        / must also change PACKETSIZ in tty.h

/ non-UNIX instructions
mfpi    = 6500^tst
mtpi    = 6600^tst
mfpd    = 106500^tst
mtpd    = 106600^tst
spl     = 230
.if .fpp
ldfps   = 170100^tst
stfps   = 170200^tst
.endif
wait    = 1
rtt     = 6
reset   = 5

HIPRI   = 340
HIGH    = 7

/ Mag tape dump
/ save registers in low core and
/ write all core onto mag tape.
/ entry is thru 44 abs

.data
.globl  dump
dump:
        bit     $1,SSR0
        bne     dump

/ save regs r0,r1,r2,r3,r4,r5,r6,KIA6
/ starting at abs location 4

        mov     r0,4
        mov     $6,r0
        mov     r1,(r0)+
        mov     r2,(r0)+
        mov     r3,(r0)+
        mov     r4,(r0)+
        mov     r5,(r0)+
        mov     sp,(r0)+
        mov     KDSA0+[6*2],(r0)+

/ dump all of core (ie to first mt error)
/ onto mag tape. (9 track or 7 track 'binary')

.if .mt
        mov     $MTC,r0
        mov     $60004,(r0)+
        clr     2(r0)
1:
        mov     $-512.,(r0)
        inc     -(r0)
2:
        tstb    (r0)
        bge     2b
        tst     (r0)+
        bge     1b
        reset

/ end of file and loop

        mov     $60007,-(r0)
        br      .
.endif

.if .mt-1
        mov     $TUC,r0
        mov     $60,(r0)+
        clr     2(r0)
        clr     6(r0)
        mov     $1300,24.(r0)           / 800 bpi + pdp11 mode + unit zero
1:
        mov     $-256.,(r0)
        mov     $-512.,4(r0)
        inc     -(r0)
2:
        tstb    (r0)
        bge     2b
        tst     (r0)+
        bge     1b
        reset
        mov     $027,-(r0)
        br      .
.endif

.globl  dstart, tstart, _end, _edata, _etext
dstart:
        mov     $stk+4,sp
        reset
        mov     $2,0
        mov     $777,2
        mov     $30000,PS

/ Set KD0-5 to physical

        mov     $KDSA0,r0
        mov     $KDSD0,r1
        clr     r2
        mov     $_end+63.,r3
        ash     $-6,r3
        bic     $!1777,r3               / r3 = size of data+bss
1:
        cmp     r3,$200
        bge     3f
        dec     r3
        bge     4f
        clr     r4
        br      2f
4:
        mov     r3,r4
        ash     $8,r4
        bis     $6,r4
        br      2f
3:
        mov     $77406,r4
2:
        mov     r2,(r0)+
        mov     r4,(r1)+
        add     $200,r2
        sub     $200,r3
        cmp     r0,$KDSA0+[5*2]
        blos    1b

/ Set KD7 to IO page

        mov     $IO,KDSA0+[7*2]
        mov     $77406,KDSD0+[7*2]

/ Set KI0 to Physical 0 and KI7 to IO temporarily

        clr     KISA0
        mov     $77406,KISD0
        mov     $IO,KISA0+[7*2]
        mov     $77406,KISD0+[7*2]

/Set up UI0-7 to eventual values of KI0-7
/ Set up KD6 to ublock
/ Set KI6 to ublock, temporarily

        mov     $UISA0,r0
        mov     $UISD0,r1
        mov     $_end+63.,r2
        ash     $-6,r2
        bic     $!1777,r2       / r2 = sizeof data+bss = final start of text
        mov     $_etext+63.,r3
        ash     $-6,r3
        bic     $!1777,r3       / r3 = size of text

        mov     r2,KDSA0+[6*2]
        add     r3,KDSA0+[6*2]
        mov     $[[usize-1]\<8]|6],KDSD0+[6*2]
        mov     KDSA0+[6*2],KISA0+[6*2]
        mov     KDSD0+[6*2],KISD0+[6*2]

/ Set up SD0-7
```

```
        mov     $77406,SDSD0
        mov     $77406,SDSD0+[1*2]
        mov     $77406,SDSD0+[2*2]
        mov     $77406,SDSD0+[3*2]
        mov     $77406,SDSD0+[4*2]
        mov     $77406,SDSD0+[5*2]
        clr     SDSD0+[6*2]
        mov     KDSD0+[7*2],SDSD0+[7*2]
        mov     KDSA0+[7*2],SDSA0+[7*2]

1:
        cmp     r3,$200
        bge     3f
        dec     r3
        bge     4f
        clr     r4
        br      2f
4:
        mov     r3,r4
        ash     $8,r4
        bis     $6,r4
        br      2f
3:
        mov     $77406,r4
2:
        mov     r2,(r0)+
        mov     r4,(r1)+
        add     $200,r2
        sub     $200,r3
        cmp     r0,$UISA0+[7*2]
        blos    1b

/ Set up UDSA0-7 to present text location

        mov     $UDSA0,r0
        mov     $UDSD0,r1
        mov     $_edata+63.,r2
        ash     $-6,r2
        bic     $!1777,r2       / r2 = current start of text
1:
        mov     r2,(r0)+
        mov     $77402,(r1)+
        add     $200,r2
        cmp     r0,$UDSA0+[7*2]
        blos    1b

/ Turn on memory mgmt, 22 bin, ubmap, U sep, K non sep
/ For the text copy which follows

        mov     $61,SSR3
        bit     $20,SSR3
        beq     1f
        mov     $70.,_cputype
        mov     $3,*SMSCR
1:
```

```
.if .pf
        tstb    _power
        bne     9f
.endif

/ Now copy text
        inc     SSR0
        mov     $_etext,r1
        mov     r1,r0
        asr     r1
        bic     $100000,r1      / r1 = words of text
1:
        mfpd    -(r0)
        mtpi    (r0)
        sob     r1,1b

/ Clear ublock

        mov     $_u,r0
1:
        clr     (r0)+
        cmp     r0,$_u+[usize*64.]
        blo     1b
        dec     SSR0

/ Clear bss

        mov     $_edata,r0
1:
        clr     (r0)+
        cmp     r0,$_end
        blo     1b

/ 22 bit, ubmap, U, S, and K sep
9:
        mov     $67,SSR3

/ Now set up KI0-7 and SI0-7

        mov     $UISA0,r0
        mov     $KISA0,r1
        mov     $UISD0,r2
        mov     $KISD0,r3
        mov     $SISA0,r4
        mov     $SISD0,r5
1:
        mov     (r0),(r1)+
        mov     (r0)+,(r4)+
        mov     (r2)+,(r3)
        bic     $4,(r3)         / turn off write permission
        mov     (r3)+,(r5)+
        cmp     r1,$KISA0+[7*2]
        blos    1b

/ Now give control to text start
        jmp     tstart
```

```
.text

.globl  start, _main
start:
.if     .pf
        tstb    _power
        jne     pwr_ret
.endif

        mov     $_u+[usize*64.],sp

        mov     $30000,PS
        jsr     pc,_main
        mov     $170000,-(sp)
        clr     -(sp)
        rtt

.if .pf
/ Power fail save sequence.  Set up normal interrupt stack and call power
/ fail routine.  This routine must save any necessary info for restart.
/ General registers are saved by normal trap sequence.  When everything
/ is ready for power off, this routine calls pwr_on(); to simulate
/ waiting for power to return.  The following code provides the assist
/ for this.
/ J. A. McGuire 2/78

.globl  pfail, _power, _pwr_flg, _pwrfail

/ Debugging entry for power fail traps.
/ Allows C code to simulate power fail trap in critical
/ regions of code.  The magic_number is placed in the
/ console switch register.
/ Normal use:
/       if (SW->integ == magic_number)
/               pfail();
/ This code can be commented out when not needed.

.globl  _pfail, stk
nop = 0240
halt = 0
_pfail:
        mov     (sp),r0
        mov     PS,(sp)
        mov     r0,-(sp)
        mov     $HIPRI,PS
        nop                     / Can be changed to a halt whenever
                                / necessary (to power off disks, etc.)


pfail:
        tstb    _power          / Power fail state flag.
        beq     0f              / System going down. Get safe.
        jgt     pwr_ret         / Save was completed. Try to come up.
        rtt                     / Ignore traps taken during save routine.
0:
        dech    _power          / Set new state to ignore further traps.
        bit     $1,SSR0         / Ensure memory mgt. enabled
        bne     0f
        halt                    / Halt -- something went wrong.
0:
        cmp     $stk,sp         / Test for temporary stack ptr
        bne     0f              / No, continue

/ Whenever retu is switching KDSA0+[6*2], the stack pointer is set to
/ a temporary location.  In this case, things must be fixed up
/ before taking power fail interrupt.

        mov     r5,KDSA0+[6*2]  / New _u address
        mov     _u,sp           / Saved stack pointer
        mov     _u+2,r5         / Saved r5
        mov     stk+2,-(sp)     / Move interrupted PS
        mov     stk,-(sp)       / Move interrupted PC
0:
        tst     pc              / Clear cond codes (set dev=0 for pwrfail)
        jsr     r0,call; jmp _pwrfail
/ no return

/ Power on sequence.  The C entry _pwr_on is called as a final step
/ in the power fail save sequence.  Before committing to restart,
/ delay a while, just in case power is failing.
/
/ There are two places for the delay.  The first is in pwr_on
/ just before restarting everything.  Here, a loop is entered
/ which increments _pwr_fail until it overflows.  Normally,
/ when power is failing, the instruction loop will never finish.
/ The value of _pwr_fail will then be some indication of the time
/ remaining for the save routine.  This value is then printed
/ out by the restart routine.

/ After power returns, delay a second time.  Timing is done
/ using the clock.  The real reason for the second delay
/ is to allow time for the system console to come back up
/ to speed.  Note that this code does not assume anything
/ about the power fail trap interrupt, other than the first one
/ means power is failing.
/
/ If the variable _pwr_flg is non-zero, then the stack pointer,
/ r5, and return address from _pwr_on have previously been saved.
/ Thus, a recursive call will not be continued past pwr_on.

.globl _pwr_on, _pwr_ka6, _clk_fn, pir, _lks, pwr_ret
.globl restart, _pwr_fail

_pwr_on:
        tstb    _pwr_flg        / Non-zero when pwrfail() was called
        bne     0f              / by PIR trap.

        mov     $pwr_sp,r0
        mov     sp,(r0)+        / Save current stack pointer
        mov     r5,(r0)+        / Save R5
        mov     (sp),(r0)+      / Save return address to pwrfail();
        mov     PS,(r0)+        / Save current and previous mm modes
        bis     $30000,PS
        mfpd    sp              / Save user sp
        mov     (sp)+,(r0)+

        tst     (r0)
        bne     0f
        mov     _clk_fn,(r0)    / Save previous interrupt instruction
0:
        negb    _power          / Make positive for restart

/ When power is failing, count number instructions executed
/ before processor stops.

        clr     _pwr_fail
0:      inc     _pwr_fail
        bne     0b

/ Attempt to come up.  Force a reset to reset and disable all devices
/ on the unibus.

        jmp     restart

/ Power fail recovery sequence.  Re-establish user and stack
/ We get here from restart code.  Return is from call to _pwr_on.

PIR_VEC = 0240                  / Address of PIR vector

pwr_ret:
        mov     $_pwr_ka6,r2
        mov     (r2)+,KDSA0+[6*2]       / Restore _u address
        mov     (r2)+,sp
        mov     (r2)+,r5
        mov     (r2)+,(sp)      / Restore proper return address
        mov     (r2)+,r0
        mov     (r2)+,-(sp)     / Restore user stack pointer
        mov     $30000,PS
        mtpd    sp
        bic     $130000,r0
        bis     r0,PS

        mov     $artspc,_clk_fn / Set up to ignore clock interrupt

        mov     _lks,r0         / Address of clock
        mov     $120.,r1        / 2 seconds at 60 HZ

0:      mov     $0115,(r0)      / Prime clock
        wait
        sob     r1,0b

        clr     (r0)            / Stop clock

        spl     7
        mov     (r2)+,_clk_fn   / Restore normal clock interrupt
artspc:
        rts     pc
```

```
/ Power fail state flags, normally zero.  If _power is negative,
/ subsequent power fail traps are ignored; if greater than zero,
/ saved data is valid, and an attempt will be made to come up.

/ If _pwr_flg is non-zero, then the previous state of the processor
/ is known -- any interrupt service routine was allowed to complete,
/ and a PIR trap was taken at priority one.  Therefore, the
/ device restart routine can be repeated.

.data

_power:          .byte 0
_pwr_flg:        .byte 0
_pwr_fail:      0

/ The following is the local save area for the power fail assist routines.

_pwr_ka6:       .=.+14.
pwr_sp = _pwr_ka6+2
pwr_r5 = _pwr_ka6+4
pwr_rpc = _pwr_ka6+6
pwr_ps = _pwr_ka6+8.
pwr_usp = _pwr_ka6+10.
pwr_cl = _pwr_ka6+12.
.text
.endif

.if .pf-1
.globl pfail, _pfail

_pfail:
pfail:
        br      .
.endif

.globl  trap, tracet, call
.globl  _trap, _stray

tracet:
        mov     PS,saveps
        cmpb    $377,2(sp)      / test if stray interrupt
        bne     2f
        mov     (sp)+,saveps    / save pc of stray and fix stack
        tst     (sp)+
        jsr     r0,call1; jmp _stray

trap:
        mov     PS,saveps
        tst     nofault
        bne     1f
        mov     SSR0,ssr
        mov     SSR1,ssr+2
        mov     SSR2,ssr+4
        mov     $1,SSR0
2:
	jsr	r0,call1; jmp _trap
	/ no return
1:
	mov	$1,SSR0
	mov	nofault,(sp)
	rtt

.globl	_runrun; _qswtch
call1:
	mov	saveps,-(sp)
	spl	0
	br	1f

call:                                   > inc   _intlvl
	mov	PS,-(sp)
1:
	mov	r1,-(sp)
	mfpd	sp
	mov	4(sp),-(sp)
	bic	$137,(sp)
	bit	$20000,PS
	beq	1f
.if .fpp
	mov	$20,_u+4		/ FP maint mode
.endif
	jsr	pc,(r0)+
2:
	spl	HIGH
	tstb	_runrun
	beq	2f
	spl	0
	jsr	pc,_savfp
	jsr	pc,_qswtch
	br	2b
2:
.if .fpp
	mov	$_u+4,r1
	bit	$20,(r1)
	bne	2f
	mov	(r1)+,r0
	ldfps	r0
	movf	(r1)+,fr0
	movf	(r1)+,fr1
	movf	fr1,fr4
	movf	(r1)+,fr1
	movf	fr1,fr5
	movf	(r1)+,fr1
	movf	(r1)+,fr2
	movf	(r1)+,fr3
	ldfps	r0
2:
.endif
	tst	(sp)+
	mtpd	sp
	br	2f
1:
        bis     $30000,PS
        jsr     pc,(r0)+
        cmp     (sp)+,(sp)+
2:
        mov     (sp)+,r1
        tst     (sp)+
        mov     (sp)+,r0        ->  if (intlvl)
        rtt                             intlvl--;

.globl  _savfp
_savfp:
.if .fpp
        mov     $_u+4,r1
        bit     $20,(r1)
        beq     1f
        stfps   (r1)+
        movf    fr0,(r1)+
        movf    fr4,fr0
        movf    fr0,(r1)+
        movf    fr5,fr0
        movf    fr0,(r1)+
        movf    fr1,(r1)+
        movf    fr2,(r1)+
        movf    fr3,(r1)+
1:
.endif
        rts     pc

.globl  _incupc
_incupc:
        mov     r2,-(sp)
        mov     6(sp),r2        / base of prof with base,leng,off,scale
        mov     4(sp),r0        / pc
        sub     4(r2),r0        / offset
        clc
        ror     r0
        mul     6(r2),r0        / scale
        ashc    $-14.,r0
        inc     r1
        bic     $1,r1
        cmp     r1,2(r2)        / length
        bhis    1f
        add     (r2),r1         / base
        mov     nofault,-(sp)
        mov     $2f,nofault
        mfpd    (r1)
        inc     (sp)
        mtpd    (r1)
        br      3f
2:
        clr     6(r2)
3:
        mov     (sp)+,nofault
1:
        mov     (sp)+,r2
        rts     pc
```

```
.globl  _display
_display:
        dec     dispdly
        bge     2f
        clr     dispdly
        mov     PS,-(sp)
        mov     $HIPRI,PS
        mov     CSW,r1
        bit     $1,r1
        beq     1f
        bis     $30000,PS
        dec     r1
1:
        jsr     pc,fuword
        mov     r0,CSW
        mov     (sp)+,PS

        cmp     r0,$-1
        bne     2f
        mov     $120.,dispdly           / 2 sec delay after CSW fault
2:
        rts     pc

/ Character list get/put

.globl  _getc, _putc
.globl  _cfreelist
.globl  _cfreecnt

.if     .xclist
        .globl  _xcfreelist, _xclistbase
        .globl  _xcfreecnt
.endif

_getc:
        mov     2(sp),r1
        mov     PS,-(sp)
        mov     r2,-(sp)
        spl     6
        mov     2(r1),r2        / first ptr
        beq     9f              / empty
        movb    (r2)+,r0        / character
        bic     $!377,r0
        mov     r2,2(r1)
        dec     (r1)+           / count
        bne     1f
        clr     (r1)+
        clr     (r1)+           / last block
        br      2f
1:
        bit     $CBITS,r2
        bne     3f

        .if     .xclist
                bit     $1,-[CBITS+1](r2)       / system or external?
                beq     3f                      / system

                mov     UISA0,-(sp)     / external--save registers
                mov     UISD0,-(sp)
                mov     r3,-(sp)
                mov     r4,-(sp)
                mov     $30340,PS       / spl 7--previous user
                mov     $77406,UISD0    / 4k-read/write
                mov     -[CBITS+1](r2),r3       / r3->first ext. packet
                mov     r3,r4           / set up mem mgmt and address in r3
                mov     r3,-(sp)        / so mmr0 can always be used.
                dec     (sp)            / make full address even
                ash     $-6,r4          / must save full address for external
                bic     $176177,r4      / linking so put it on the stack
                add     _xclistbase,r4
                mov     r4,UISA0
                bic     $160000,r3
                add     $CBITS,r3       / make r3->end of packet
                mov     $[[CBITS+1]\/2],r4      / loop count
        1:
                mfpi    -(r3)           / copy packet in from end to beginning
                mov     (sp)+,-(r2)
                sob     r4,1b
                mov     _xcfreelist,-(sp)       / put ext. packet on freelist
                mtpi    (r3)
                mov     (sp)+,_xcfreelist       / saved address from above
                inc     _xcfrecnt       / increment xcfreecount
                mov     r2,(r1)         / update first pointer
                add     $2,(r1)
                / if just freed last external packet in
                / this clist, force last system packet
                / to point back to first
                bit     $1,(r2)
                bne     1f
                mov     r2,*(r2)
        1:
                mov     (sp)+,r4        / restore registers
                mov     (sp)+,r3
                mov     (sp)+,UISD0
                mov     (sp)+,UISA0
                br      8f

        3:
        .endif

        mov     -[CBITS+1](r2),(r1)     / next block
        .if     .xclist
                clr     *(r1)
        .endif
        add     $2,(r1)
2:
        dec     r2
        bic     $CBITS,r2
        mov     _cfreelist,(r2)
        mov     r2,_cfreelist
        inc     _cfrecnt                / increment cfreecount
```

```
8:
        mov     (sp)+,r2
        mov     (sp)+,PS
        rts     pc
9:
        clr     4(r1)
        mov     $-1,r0
        br      8b

_putc:
        mov     2(sp),r0
        mov     4(sp),r1
        mov     PS,-(sp)
        mov     r2,-(sp)
        mov     r3,-(sp)
        spl     6
        mov     4(r1),r2        / last ptr
        bne     1f
        mov     _cfreelist,r2
        beq     9f
        mov     (r2),_cfreelist
        dec     _cfrecnt        / decrement cfreecount
        clr     (r2)+
        mov     r2,2(r1)        / first ptr
        br      4f
1:
        bit     $CBITS,r2
        bne     4f

        .if     .xclist
                tst     -[CBITS+1](r2)  / is last-1 packet external?
                beq     3f              / no

                mov     _xcfreelist,r3  / yes--is external free list empty?
                beq     9f              / yes
                mov     UISA0,-(sp)     / no--save mapping registers
                mov     UISD0,-(sp)
                mov     r4,-(sp)        / save r4
                mov     $30340,PS       / spl 7--previous user
                mov     $77406,UISD0    / 4k-read/write
                sub     $[CBITS+1],r2   / r2->last internal packet
                mov     (r2),r4         / r4->last-1 packet
                mov     r3,-(sp)        / (sp)->new external packet
                inc     (sp)            / mark (sp) as external
                bit     $1,r4           / is last-1 packet external?
                beq     1f              / no
                dec     r4              / yes--make it point to new packet
                mov     r4,-(sp)
                ash     $-6,r4
                bic     $176177,r4
                add     _xclistbase,r4
                mov     r4,UISA0
                mov     (sp)+,r4
                bic     $160000,r4
                mtpi    (r4)
                br      2f
        1:
                mov     (sp)+,(r4)      / make first packet point to ext. one
        2:
                mov     r3,r4
                mov     r3,-(sp)
                ash     $-6,r4
                bic     $176177,r4
                add     _xclistbase,r4
                mov     r4,UISA0
                bic     $160000,r3
                mfpi    (r3)            / update xcfreelist
                mov     (sp)+,_xcfreelist
                dec     _xcfrecnt       / decrement xcfreecount
                mov     r2,(r2)         / setup packet to be copied out
                                        / so that it will point to last packet
                mov     $[[CBITS+1]\/2],r4      / set up loop count
                                                / to number of words in packet
        1:
                mov     (r2)+,-(sp)     / copy last packet out
                mtpi    (r3)+
                sob     r4,1b
                sub     $[CBITS+1],r2   / make r2->start of last packet again
                mov     (sp)+,(r2)      / make last int. point to last ext.
                inc     (r2)+           / and make r2->next data slot
                mov     (sp)+,r4        / restore registers
                mov     (sp)+,UISD0
                mov     (sp)+,UISA0
                br      4f

        3:
        .endif

        mov     _cfreelist,r3
        beq     9f
        mov     (r3),_cfreelist
        dec     _cfrecnt        / decrement cfreecount
        sub     $[CBITS+1],r2   / make last packet point to prev. (first) one
        mov     r3,(r2)
        mov     r2,(r3)
        mov     r3,r2
        tst     (r2)+
4:
        movb    r0,(r2)+
        mov     r2,4(r1)
        inc     (r1)            / count
        clr     r0
8:
        mov     (sp)+,r3
        mov     (sp)+,r2
        mov     (sp)+,PS
        rts     pc
9:
        mov     pc,r0
        br      8b

.globl  _backup
```

```
.globl  _regloc
_backup:
        mov     2(sp),r0
        movb    ssr+2,r1
        jsr     pc,1f
        movb    ssr+3,r1
        jsr     pc,1f
        movb    _regloc+7,r1
        asl     r1
        add     r0,r1
        mov     ssr+4,(r1)
        clr     r0
2:
        rts     pc
1:
        mov     r1,-(sp)
        asr     (sp)
        asr     (sp)
        asr     (sp)
        bic     $!7,r1
        movb    _regloc(r1),r1
        asl     r1
        add     r0,r1
        sub     (sp)+,(r1)
        rts     pc


.globl  _fubyte, _subyte
.globl  _fuword, _suword
.globl  _fuibyte, _suibyte
.globl  _fuiword, _suiword
_fuibyte:
        mov     2(sp),r1
        bic     $1,r1
        jsr     pc,giword
        br      2f

_fubyte:
        mov     2(sp),r1
        bic     $1,r1
        jsr     pc,gword

2:
        cmp     r1,2(sp)
        beq     1f
        swab    r0
1:
        bic     $!377,r0
        rts     pc

_suibyte:
        mov     2(sp),r1
        bic     $1,r1
        jsr     pc,giword
        mov     r0,-(sp)
        cmp     r1,4(sp)
        beq     1f
        movb    6(sp),1(sp)
        br      2f
1:
        movb    6(sp),(sp)
2:
        mov     (sp)+,r0
        jsr     pc,piword
        clr     r0
        rts     pc

_subyte:
        mov     2(sp),r1
        bic     $1,r1
        jsr     pc,gword
        mov     r0,-(sp)
        cmp     r1,4(sp)
        beq     1f
        movb    6(sp),1(sp)
        br      2f
1:
        movb    6(sp),(sp)
2:
        mov     (sp)+,r0
        jsr     pc,pword
        clr     r0
        rts     pc

_fuiword:
        mov     2(sp),r1
fuiword:
        jsr     pc,giword
        rts     pc

_fuword:
        mov     2(sp),r1
fuword:
        jsr     pc,gword
        rts     pc

giword:
        mov     PS,-(sp)
        spl     HIGH
        mov     nofault,-(sp)
        mov     $err,nofault
        mfpi    (r1)
        mov     (sp)+,r0
        br      1f

gword:
        mov     PS,-(sp)
        spl     HIGH
        mov     nofault,-(sp)
        mov     $err,nofault
        mfpd    (r1)
        mov     (sp)+,r0
        br      1f

_suiword:
        mov     2(sp),r1
        mov     4(sp),r0
suiword:
        jsr     pc,piword
        rts     pc

_suword:
        mov     2(sp),r1
        mov     4(sp),r0
suword:
        jsr     pc,pword
        rts     pc

piword:
        mov     PS,-(sp)
        spl     HIGH
        mov     nofault,-(sp)
        mov     $err,nofault
        mov     r0,-(sp)
        mtpi    (r1)
        br      1f

pword:
        mov     PS,-(sp)
        spl     HIGH
        mov     nofault,-(sp)
        mov     $err,nofault
        mov     r0,-(sp)
        mtpd    (r1)
1:
        mov     (sp)+,nofault
        mov     (sp)+,PS
        rts     pc

err:
        mov     (sp)+,nofault
        mov     (sp)+,PS
        tst     (sp)+
        mov     $-1,r0
        rts     pc

.globl  _copyin, _copyout
.globl  _copyiin, _copyiout
_copyiin:
        jsr     pc,copsu
1:
        mfpi    (r0)+
        mov     (sp)+,(r1)+
        sob     r2,1b
        br      2f

_copyin:
        jsr     pc,copsu
```

```
1:
        mfpd    (r0)+
        mov     (sp)+,(r1)+
        sob     r2,1b
        br      2f

_copyiout:
        jsr     pc,copsu
1:
        mov     (r0)+,-(sp)
        mtpi    (r1)+
        sob     r2,1b
        br      2f

_copyout:
        jsr     pc,copsu
1:
        mov     (r0)+,-(sp)
        mtpd    (r1)+
        sob     r2,1b
2:
        mov     (sp)+,nofault
        mov     (sp)+,r2
        clr     r0
        rts     pc

copsu:
        mov     (sp)+,r0
        mov     r2,-(sp)
        mov     nofault,-(sp)
        mov     r0,-(sp)
        mov     10(sp),r0
        mov     12(sp),r1
        mov     14(sp),r2
        asr     r2
        mov     $1f,nofault
        rts     pc

1:
        mov     (sp)+,nofault
        mov     (sp)+,r2
        mov     $-1,r0
        rts     pc

.globl  _idle, _waitloc
_idle:
        mov     PS,-(sp)
        spl     0
        wait
_waitloc:
        mov     (sp)+,PS
        rts     pc

.globl  _savu, _retu, _aretu
_savu:
        spl     HIGH
        mov     (sp)+,r1
        mov     (sp),r0
        mov     sp,(r0)+
        mov     r5,(r0)+
        spl     0
        jmp     (r1)

_aretu:
        spl     7
        mov     (sp)+,r1
        mov     (sp),r0
        br      1f

_retu:
        spl     7
        mov     (sp)+,r1
.if .pf
        mov     (sp),r5         / Since power fail traps may catch us
        mov     $stk+4,sp       / here after switching the _u pointer,
        mov     r5,KDSA0+[6*2]  / we must set up a temporary stack ptr.
.endif
.if .pf-1
        mov     (sp),KDSA0+[6*2]
.endif
        mov     $_u,r0
1:
        mov     (r0)+,sp
        mov     (r0)+,r5
        spl     0
        jmp     (r1)

.globl  _spl0, _spl1, _spl4, _spl5, _spl6, _spl7, _splx
_spl0:
        mov     PS,r0
        spl     0
        rts     pc

_spl1:
        mov     PS,r0
        spl     1
        rts     pc

_spl4:
        mov     PS,r0
        spl     4
        rts     pc

_spl5:
        mov     PS,r0
        spl     5
        rts     pc

_spl6:
        mov     PS,r0
        spl     6
        rts     pc
```

```
_spl7:
        mov     PS,r0
        spl     HIGH
        rts     pc

_splx:
        mov     2(sp),PS
        rts     pc

/**********************\
/                      *
/       COPYIO.s       *
/                      *
/**********************/
/
/
/       copyio -- is a generalized copy to or from a physical
/               address from or to a virtual address.
/
/       In C, called by:        copyio(paddr,vaddr,nbytes,flag)
/               where:
/                       paddr - a physical address
/                       vaddr - a virtual address as def'd by flag
/                       nbytes - the number of bytes in transfer
/                       flag -
/                           U_WUD (0) - write from user data space
/                           U_RUD (1) - read to user data space
/                           U_WKD (2) - write from kernal data space
/                           U_RKD (3) - read to kernal data space
/                           U_WUI (4) - write from user instr space
/                           U_RUI (5) - read to user instr space
/                           U_WKI (6) - write from kernal instr space
/                           U_RKI (7) - read to kernal instr space
/
/
.globl  _copyio
_copyio:
        mov     r5,-(sp)
        mov     r4,-(sp)
        mov     r3,-(sp)
        mov     r2,-(sp)
        mov     SDSA0+[2*2],-(sp)
        mov     SDSA0+[3*2],-(sp)
        mov     SDSA0+[4*2],-(sp)
        mov     SDSD0+[2*2],-(sp)
        mov     SDSD0+[3*2],-(sp)
    /*****************************************\
    /
    /   CONVERSION OF "PADDR"
    /           Set up SupDatSpRegs #4 to map "paddr".
    /           Make r4 the SupVirAddr of "paddr".
    /
    /*****************************************/
        mov     sp,r5           / make r5 point to first arg
        add     $20.,r5
        mov     (r5)+,r0
        mov     (r5)+,r1        / r0,r1 is "paddr"
        ashc    $10.,r0         / shift page addr of "paddr" to r0
        mov     r0,SDSA0+[4*2]  / consume SupDatSpaAddr [4]
        mov     $1000,r0
        ashc    $6,r0           / r0 = SupVirAddr of "paddr"
        mov     r0,r4           / r4 = SupVirAddr of "paddr"
    /****************************************\
    /
    /   LOAD AND TEST REMAINING ARGUMENTS
    /
    /****************************************/
        clr     r0
        mov     (r5)+,r1        / r1 = "vaddr"
        mov     (r5)+,r2        / r2 = "nbytes"
        beq     8f
        cmp     r2,$[8192.-63.]
        blo     2f
        mov     $-1,r0
        br      8f
2:
        mov     (r5),r3         / r3 = "flag"
    /****************************************\
    /
    /   CONVERSION OF "VADDR"
    /           Find address of PDR of "vaddr".
    /           Put PAR,PDR of "vaddr" into SupDatSpRegs #2.
    /           Put 'next' PAR,PDR of "vaddr" into SupDatSpRegs #3.
    /           Make r0 the SupVirAddr of "vaddr".
    /
    /****************************************/
        bic     $177771,r3      / strips read&write (low) bit
        ashc    $3,r0           / puts PAR no. of "vaddr" into r0
        asl     r0              / doubles r0
        add     cbase(r3),r0    / addr of PageDescrReg of "vaddr" in r0
        mov     40(r0),SDSA0+[2*2]      / eat SupDatSpaAddrReg[2]
        mov     (r0)+,SDSD0+[2*2]       / eat SupDatSpaDesReg[2]
        bit     $17,r0
        bne     1f
        sub     $20,r0  / if next PDR is out of PDR set, get the
                        /  first of this set by decrementing addr
                        /  of PDR by a set (8.*2=20).
1:
        mov     40(r0),SDSA0+[3*2]      / eat SupDatSpaAddrReg[3]
        mov     (r0),SDSD0+[3*2]        / eat SupDatSpaDesReg[3]
        mov     $2,r0
        ashc    $13.,r0         / r0 = Sup virtual addr of "vaddr"
    /****************************************\
    /
    /   MOVE SET UP
    /           If writing (from "vaddr" to "paddr")
    /                   Set r0 = SupVirAddr of "vaddr".
    /                   Set r1 = SupVirAddr of "paddr".
    /           Else if reading (from "paddr" to "vaddr")
    /                   Set r0 = SupVirAddr of "paddr".
    /                   Set r1 = SupVirAddr of "vaddr".
```

```
	/
	/****************************************/
		bit	$1,(r5)
		bne	1f
		mov	r4,r1	/ if write, r1=SupVirAddr("paddr")
		br	2f
1:
		mov	r0,r1
		mov	r4,r0	/ if read, r0=SupVirAddr("paddr")
	/****************************************\
	/
	/	SET "nofault" (TO CATCH MEMORY FAULT)
	/
	/****************************************/
gcio:
2:
		mov	nofault,r5	/ r5 = nofault
		mov	$gcerr,nofault
	/****************************************\
	/
	/	SWITCH TO SUPERVISOR MODE
	/
	/****************************************/
		bis	$40000,PS
	/****************************************\
	/
	/	DO THE MOST EFFICIENT COPY DEPENDING ON 'EVENNESS'
	/	  OF VIRTUAL ADDRESS.
	/
	/****************************************/
		bit	$1,r1
		bne	1f
		bit	$1,r0
		beq	gcete
		br	gcote
1:
		bit	$1,r0
		beq	gceto
gcoto:			/ odd VirAddr to odd VirAddr
		dec	r2
		movb	(r0)+,(r1)+
gcete:			/ even VirAddr to even VirAddr
		asr	r2
		beq	2f
1:
		mov	(r0)+,(r1)+
		sob	r2,1b
2:
		bcc	9f
		movb	(r0)+,(r1)+	/ catch odd-th byte
		br	9f
gceto:			/ even VirAddr to odd VirAddr
gcote:			/ odd VirAddr to even VirAddr
1:
		movb	(r0)+,(r1)+
		sob	r2,1b
```

```
     /****************************************\
     /
     /   CLEAN UP
     /
     /****************************************/
9:
        clr     r0
1:
        bic     $40000,PS
        mov     r5,nofault
8:
        mov     (sp)+,SDSD0+[3*2]
        mov     (sp)+,SDSD0+[2*2]
        mov     (sp)+,SDSA0+[4*2]
        mov     (sp)+,SDSA0+[3*2]
        mov     (sp)+,SDSA0+[2*2]
        mov     (sp)+,r2
        mov     (sp)+,r3
        mov     (sp)+,r4
        mov     (sp)+,r5
        rts     pc
     /****************************************\
     /
     /   IN CASE OF MEMORY FAULT, RETURN -1
     /
     /****************************************/
gcerr:
        mov     $-1,r0
        br      1b

.data
cbase:
        UDSD0
        KDSD0
        UISD0
        KISD0
.text

.globl  _xgetc, _xputc
_xgetc:
        mov     PS,-(sp)
        jsr     pc,7f
        mfpd    (r0)
        mov     (sp)+,r0
        tst     r1
        bpl     1f
        swab    r0
1:
        bic     $!377,r0
        br      9f

_xputc:
        mov     PS,-(sp)
        jsr     pc,7f
        mfpd    (r0)
        tst     r1
        bpl     1f
        movb    12(sp),1(sp)
        br      8f
1:
        movb    12(sp),(sp)
        br      8f

.globl  _xget, _xput
_xget:
        mov     PS,-(sp)
        jsr     pc,7f
        mfpd    (r0)
        mov     (sp)+,r0
        br      9f

_xput:
        mov     PS,-(sp)
        jsr     pc,7f
        mov     10(sp),-(sp)
        br      8f

.globl  _xgetl, _xputl
_xgetl:
        mov     PS,-(sp)
        jsr     pc,7f
        mfpd    (r0)+
        mfpd    (r0)
        mov     (sp)+,r1
        mov     (sp)+,r0
        br      9f

_xputl:
        mov     PS,-(sp)
        jsr     pc,7f
        mov     10(sp),-(sp)
        mtpd    (r0)+
        mov     12(sp),r1
        mov     r1,-(sp)
8:
        mtpd    (r0)
        mov     10(sp),r0
9:
        mov     savesup,SDSA0+[5*2]
        mov     (sp)+,PS
        rts     pc
7:
        mov     6(sp),r0
        mov     10(sp),r1
        ashc    $10.,r0
        mov     $10000+HIPRI,PS
        mov     SDSA0+[5*2],savesup
        mov     r0,SDSA0+[5*2]
        mov     $1200,r0
        ashc    $5,r0
        asl     r0
        rts     pc
```

```
.data
savesup:
        0
.text

.globl  _copyseg
_copyseg:
        mov     SDSA0,-(sp)
        mov     SDSA0+[1*2],-(sp)
        mov     r2,-(sp)
        mov     10(sp),SDSA0
        mov     12(sp),SDSA0+[1*2]
        clr     r0
        mov     $20000,r1
        mov     $32.,r2
        bis     $40000,PS
1:
        mov     (r0)+,(r1)+
        sob     r2,1b
        bic     $40000,PS
        mov     (sp)+,r2
        mov     (sp)+,SDSA0+[1*2]
        mov     (sp)+,SDSA0
        rts     pc

.globl  _clearseg
_clearseg:
        mov     SDSA0,-(sp)
        mov     4(sp),SDSA0
        clr     r0
        mov     $32.,r1
        bis     $40000,PS
1:
        clr     (r0)+
        sob     r1,1b
        bic     $40000,PS
        mov     (sp)+,SDSA0
        rts     pc

.globl  _clear
_clear:
        mov     SDSA0,-(sp)
        mov     r2,-(sp)
        mov     sp,r2
        add     $6,r2
        mov     (r2)+,r0
        mov     (r2)+,r1
        ashc    $10.,r0
        mov     r0,SDSA0
        clr     r0
        ashc    $6,r0
        mov     (r2)+,r1
        bis     $40000,PS
        bit     $1,r0
        beq     1f
```

```
	dec	r1
	clrb	(r0)+
1:
	mov	r1,r2
	asr	r1
	beq	2f
1:
	clr	(r0)+
	sob	r1,1b
2:
	asr	r2
	bcc	1f
	clrb	(r0)+
1:
	bic	$40000,PS
	mov	(sp)+,r2
	mov	(sp)+,SDSA0
	rts	pc

.globl	_dpcmp
_dpcmp:
	mov	2(sp),r0
	mov	4(sp),r1
	sub	6(sp),r0
	sub	8(sp),r1
	sbc	r0
	bge	1f
	cmp	r0,$-1
	bne	2f
	cmp	r1,$-512.
	bhi	3f
2:
	mov	$-512.,r0
	rts	pc
1:
	bne	2f
	cmp	r1,$512.
	blo	3f
2:
	mov	$512.,r1
3:
	mov	r1,r0
	rts	pc

.globl	_ldiv
_ldiv:
	clr	r0
	mov	2(sp),r1
	div	4(sp),r0
	rts	pc

.globl	_lrem
_lrem:
	clr	r0
	mov	2(sp),r1
	div	4(sp),r0
	mov	r1,r0
	rts	pc

/ Long quotient

	.globl	ldiv
ldiv:
	jsr	r5,csv
	mov	10.(r5),r3
	sxt	r4
	bpl	1f
	neg	r3
1:
	cmp	r4,8.(r5)
	bne	hardldiv
	mov	6.(r5),r2
	mov	4.(r5),r1
	bge	1f
	neg	r1
	neg	r2
	sbc	r1
	com	r4
1:
	mov	r4,-(sp)
	clr	r0
	div	r3,r0
	mov	r0,r4		/high quotient
	mov	r1,r0
	mov	r2,r1
	div	r3,r0
	bvc	1f
	sub	r3,r0		/ this is the clever part
	div	r3,r0
	tst	r1
	sxt	r1
	add	r1,r0		/ cannot overflow!
1:
	mov	r0,r1
	mov	r4,r0
	tst	(sp)+
	bpl	9f
	neg	r0
	neg	r1
	sbc	r0
9:
	jmp	cret

hardldiv:
	4

/ Long remainder

	.globl	lrem
lrem:
	jsr	r5,csv
	mov	10.(r5),r3
	sxt	r4
	bpl	1f
	neg	r3
1:
	cmp	r4,8.(r5)
	bne	hardlrem
	mov	6.(r5),r2
	mov	4.(r5),r1
	mov	r1,r4
	bge	1f
	neg	r1
	neg	r2
	sbc	r1
1:
	clr	r0
	div	r3,r0
	mov	r1,r0
	mov	r2,r1
	div	r3,r0
	bvc	1f
	sub	r3,r0
	div	r3,r0
	tst	r1
	beq	9f
	add	r3,r1
1:
	tst	r4
	bpl	9f
	neg	r1
9:
	sxt	r0
	jmp	cret

/ The divisor is known to be >= 2^15.  Only 16 cycles are
/ needed to get a remainder.
hardlrem:
	4

/.globl lmul
/lmul:
/	mov	r2,-(sp)
/	mov	r3,-(sp)
/	mov	8(sp),r2
/	sxt	r1
/	sub	6(sp),r1
/	mov	12.(sp),r0
/	sxt	r3
/	sub	10.(sp),r3
/	mul	r0,r1
/	mul	r2,r3
/	add	r1,r3
/	mul	r2,r0
/	sub	r3,r0
/	mov	(sp)+,r3
/	mov	(sp)+,r2
/	rts	pc
```

```
.globl  csv
csv:
        mov     r5,r0
        mov     sp,r5
        mov     r4,-(sp)
        mov     r3,-(sp)
        mov     r2,-(sp)
        jsr     pc,(r0)

.globl cret
cret:
        mov     r5,r2
        mov     -(r2),r4
        mov     -(r2),r3
        mov     -(r2),r2
        mov     r5,sp
        mov     (sp)+,r5
        rts     pc

.globl  _u
_u      = 140000
usize   = 16.

CSW     = 177570
PS      = 177776
SSR0    = 177572
SSR1    = 177574
SSR2    = 177576
SSR3    = 172516
KISA0   = 172340
KISD0   = 172300
KDSA0   = 172360
KDSD0   = 172320
MTC     = 172522
TUC     = 172440
UISA0   = 177640
UISD0   = 177600
UDSA0   = 177660
UDSD0   = 177620
SISA0   = 172240
SISD0   = 172200
SDSA0   = 172260
SDSD0   = 172220
MSCR    = 017777746     / 11/70 memory control register
PIRR    = 177772        / Programmed Interrupt Request Register
IO      = 177600

.data
.globl  _ka6
.globl  _cputype

_ka6:   KDSA0+[6*2]

.globl  _pirr
_pirr:  PIRR
```

```
_cputype:45.
stk:    .=.+4

.bss
.globl  nofault, ssr
nofault:.=.+2
ssr:    .=.+6
dispdly:.=.+2
saveps: .=.+2
```

```
/*      @(#)messag.c    2.10.1.2        */

#include "sys/param.h"
#include "sys/reg.h"
#include "sys/ipcomm.h"
#include "sys/ipcommx.h"
#include "sys/user.h"
#include "sys/userx.h"
#include "sys/proc.h"
#include "sys/procx.h"
#include "sys/systm.h"

#define MHREAD  U_RKD
#define MHWRITE U_WKD

struct msgqhdr msgqhdr[NMQHDR];
struct  map     msgmap[MMAPSIZ];        /* space for message allocation */
paddr_t msgbase;                        /* pointer to space for messages */
int movrhead, msgslept, mmemwant;

/*
 *      Message System Call
 */

messag()
{       register int n;
        register struct msgqhdr *rqp;
        register unsigned hp;
        struct msghdr mhd;
        int mtest;

        mtest = 0;
        switch(u.u_arg[0]) {
                default:
                        u.u_error = EINVAL;
                        return;
                case MDISAB:
                        msgflush();
                        return;
                case MENAB:
                        if(u.u_msgqhdr != NULL) return;
                        for(rqp = &msgqhdr[0];rqp < &msgqhdr[NMQHDR];rqp++)
                                if(rqp->mq_procp == NULL) {
                                        rqp->mq_forw = NULL;
                                        rqp->mq_last = &rqp->mq_forw;
                                        rqp->mq_cnt = 0;
                                        rqp->mq_flag = 0;
                                        u.u_msgqhdr = rqp;
                                        rqp->mq_procp = u.u_procp;
                                        rqp->mq_meslim = MAXMSGDEF;
                                        return;
                                }
                        u.u_error = ETABLE;
                        return;
                case MSEND:
```

```
                mtest++;
        case MSENDW:
                if((n = u.u_ar0[R0]) < 0 || n > MAXMLEN ||
                  u.u_arg[3] <= 0 || u.u_arg[3] > 128) {
                        u.u_error = EINVAL;
                        return;
                }

        case MSTAT:
        loop:
                if((rqp = mqsrch(u.u_arg[2])) == NULL) {
                        u.u_error = ESRCH;
                        return;
                }
                if (u.u_arg[0] == MSTAT) {
                        struct mstat mstat;

                        mstat.ms_cnt = (unsigned)rqp->mq_cnt;
                        mstat.ms_maxm = rqp->mq_meslim;
                        if (copyout((caddr_t)&mstat, u.u_arg[1],
                            sizeof(mstat)))
                                u.u_error = EFAULT;
                        return;
                }
                if((unsigned)rqp->mq_cnt >= rqp->mq_meslim) {
                        if(mtest) {
                                u.u_error = ETABLE;
                                return;
                        }
                        rqp->mq_flag |= IP_QWANT;
                        sleep(&rqp->mq_flag, PMSG);
                        goto loop;
                }

                if((hp = malloc(msgmap, (n+movrhead)>>6)) == NULL) {
                        if(mtest) {
                                u.u_error = ENOMEM;
                                return;
                        }
                        mmemwant++;
                        msgslept++;
                        sleep(&msgbase, PMSG);
                        goto loop;
                }
                msgmove(hp, n, MSGIN);
                if(u.u_error)
                        msgfree(hp,n);
                else {
                        mhd.mq_size = n;
                        mhd.mq_type = u.u_arg[3];
                        mhd.mq_sender = u.u_procp->p_pid;
                        msgsend(rqp, &mhd, hp);
                }
                return;
        case MRECV:
                mtest++;
                case MRECVW:
                        if((rqp = u.u_msgqhdr) == NULL) {
                                u.u_error = ENOALOC;
                                return;
                        }
                        if((n = u.u_arg[3]) < 0 || n > 128 || u.u_ar0[R0] < 0)
*  {
                                u.u_error = EINVAL;
                                return;
                        }
                        while(!msgrecv(rqp, n, &mhd)) {
                                if(mtest) {
                                        u.u_error = ENOMSG;
                                        return;
                                }
                                rqp->mq_flag =| IP_WANTED);
                                sleep(rqp, PMSG);
                        }
                        return;
                case MSGCTL:
                        if(( rqp =  mqsrch(u.u_arg[1])) == NULL) {
                                u.u_error = ESRCH;
                                return;
                        }
                        switch(u.u_ar0[R0]) {
                                default:
                                        u.u_error = EFUNC;
                                        return;
                                case SETMQLEN:
                                        if(u.u_procp->p_pid != u.u_arg[1]
                                                &&  !suser() )
                                                return;
                                        if( u.u_arg[2] <= MAXMSGL &&
                                                u.u_arg[2]  >= 0) {
                                           rqp->mq_meslim = u.u_arg[2] ;
                                        }
                                        else {
                                           u.u_error = ERANGE;
                                           return;
                                        }
                        }
        }
}

/*
 * Scan a process's message Q for a message of
 * the desired type. If found, try to effect transfer
 * of the message to the process.
 */

msgrecv(qp, type, mhd)
struct msgqhdr *qp;
register struct msghdr *mhd;
{
        register int n;
        register unsigned rhp1, rhp2;
	n = type;
	mhd->mq_forw = qp->mq_forw;
	for(rhp1 = qp; rhp2 = mhd->mq_forw; rhp1 = rhp2) {
		msghmov(mhd,rhp2,MHREAD);
		if(n == 0 || n == mhd->mq_type) {
			n = min(mhd->mq_size, u.u_ar0[R0]);
			msgmove(rhp2, n, MSGOUT);
			if(u.u_error) return(1);
			if(suword((unsigned)u.u_arg[2], mhd->mq_sender) < 0 ||
			  suword((unsigned)u.u_arg[2]+2, mhd->mq_type) < 0)
					u.u_error = EFAULT;
			else {
				msgremov(qp, rhp1, mhd);
				msgfree(rhp2,mhd->mq_size);
				u.u_ar0[R0] = n;
			}
			return(1);
		}
	}
	return(0);
}

/*
 * Place the message pointed to by "hp" on the message Q
 * pointed to by "qp". Awaken the process if it's waiting
 * for arrival of a message.
 */

msgsend(qp, mhd, hp)
register struct msgqhdr *qp;
register struct msghdr *mhd;
unsigned hp;
{
	register int s;

	mhd->mq_forw = NULL;
	msghmov(mhd,hp,MHWRITE);
	s = spl6();
	if(qp->mq_last != qp) {
		msghmov(mhd,qp->mq_last,MHREAD);
		mhd->mq_forw = hp;
		msghmov(mhd,qp->mq_last,MHWRITE);
	}
	else
		qp->mq_forw = hp;
	qp->mq_last = hp;
	qp->mq_cnt++;
	splx(s);
	if(qp->mq_flag & IP_WANTED) {
		qp->mq_flag =& ~IP_WANTED;
		wakeup(qp);
	}
}

/*
```

```
 * Deallocate a message Q header and any messages
 * pending on the Q. Messages requiring an ACK
 * (types 1-63) are returned to the sending process as
 * type 128, if possible.
 */

msgflush()
{
        register struct msgqhdr *rqp1, *rqp2;
        register unsigned rhp;
        struct msghdr mhd;

        if((rqp1 = u.u_msgqhdr) == NULL) {
                u.u_error = ENOALOC;
                return;
        }
        u.u_msgqhdr = NULL;
        rqp1->mq_procp = NULL;
        while((rhp = rqp1->mq_forw) != NULL) {
                msghmov(&mhd,rhp,MHREAD);
                msgremov(rqp1, rqp1, &mhd);
                if((mhd.mq_type) < 64 && (rqp2 = mqsrch(mhd.mq_sender))
                  && (unsigned)rqp2->mq_cnt < rqp1->mq_meslim) {
                        mhd.mq_type = 128;
                        msgsend(rqp2, &mhd, rhp);
                }
                else
                        msgfree(rhp,mhd.mq_size);
        }
}

/*
 * Remove a message from a message Q.
 */

msgremov(qp, rhp1, mhd2)
struct msgqhdr *qp;
unsigned rhp1;
struct msghdr *mhd2;
{
        register int s;
        unsigned rhp2;
        struct msghdr mhd,, *mhd1;

        s = spl6();
        if( rhp1!=qp) {
                msghmov(&mhd,rhp1,MHREAD);
                mhd1 = &mhd;
        }
        else
                mhd1 = qp;
        qp->mq_cnt--;
        if((mhd1->mq_forw = mhd2->mq_forw) == NULL)
                qp->mq_last = rhp1;
        if( rhp1!=qp )
                msghmov(&mhd,rhp1,MHWRITE);
	splx(s);

	if(qp->mq_flag & IP_QWANT) {
		qp->mq_flag &= ~IP_QWANT;
		wakeup(&qp->mq_flag);
	}
}

/*
 * Free the buffer space used by a message and its header.
 * Awaken any processes roadblocked because of
 * insufficient buffer space.
 */

msgfree(hp,size)
unsigned hp;
{
	register int s;

	s = spl6();
	mfree(msgmap, (size+movrhead)>>6, hp);
	splx(s);
	if(mmemwant) {
		mmemwant = 0;
		wakeup(&msgbase);
	}
}

/*
 * Message interface to copyio()
 */

msgmove(hp, len, mode)
unsigned hp;
register int len;
{
	register paddr_t paddr;

	if (len==0)
		return;
	paddr = hp;
	paddr <<= 6;
	paddr += msgbase + sizeof(struct msghdr);
	if (copyio(paddr,u.u_arg[1],len,mode))
		u.u_error = EFAULT;
}

/*
 * See if a process is enabled for messages
 */

mqsrch(pid)
{	register struct proc *rpp;
	register struct msgqhdr *rqp;
	register id;
	id = pid;
	for(rqp = &msgqhdr[0]; rqp < &msgqhdr[NMQHDR]; rqp++)
		if((rpp = rqp->mq_procp) != NULL && rpp->p_pid == id)
			return(rqp);
	return(NULL);
}

/*
 * Initialization
 */

msginit()
{
	struct msghdr proto;
	register unsigned core;

	movrhead = sizeof(proto) + 63;
	core = malloc(coremap, MSGMEM);
	if(core) {
		msgbase = core;
		msgbase--;
		msgbase <<= 6;
		mfree(msgmap, MSGMEM, 1);
		return(MSGMEM);
	}
	return(0);
}

/*
 *	copy a message header to/from outer memory
 */
msghmov(mhd,hp,mode)
struct msghdr *mhd;
unsigned hp;
{
	register paddr_t paddr;

	paddr = hp;
	paddr <<= 6;
	paddr += msgbase;
	copyio(paddr, mhd, sizeof(*mhd), mode);
}
```

```
/*      @(#)messagf.c   2.4     */

#include "sys/param.h"
/*
 *      Fake Message System Call
 */

messag()
{
        nosys();
}
msgrecv()
{
}
msgsend()
{
}
msgsetup()
{
        return(NULL);
}
msgflush()
{
}
msgremov()
{
}
msgfree()
{
}
msgmove()
{
}
mgsrch()
{
        return(NULL);
}
msginit()
{
        return(0);
}
```

```
/*      @(#)nami.c      2.10    */

#include "sys/param.h"
#include "sys/inode.h"
#include "sys/inodex.h"
#include "sys/user.h"
#include "sys/userx.h"
#include "sys/systm.h"
#include "sys/buf.h"


/*
 * Convert a pathname into a pointer to
 * an inode. Note that the inode is locked.
 *
 * func = function called to get next char of name
 *       &uchar if name is in user space
 *       &schar if name is in system space
 * flag = 0 if name is sought
 *       1 if name is to be created
 *       2 if name is to be deleted
 */
struct inode *
namei(func, flag)
int (*func)();
{
        register struct inode *dp;
        register c;
        register char *cp;
        struct buf *bp;
        int i;
        dev_t d;
        off_t eo;

        /*
         * If name starts with '/' start from
         * root; otherwise start from current dir.
         */

        if ((c = (*func)()) == '\0') {
                u.u_error = ENOENT;
                return;
        }
        if (c == '/') {
                if ((dp = u.u_rdir) == NULL)
                        dp = rootdir;
                while ((c = (*func)()) == '/')
                        ;
                if(c == '\0' && flag != 0) {
                        u.u_error = ENOENT;
                        return;
                }
        } else
                dp = u.u_cdir;
        iget(dp->i_dev, dp->i_number);
```

```
cloop:
        /*
         * Here dp contains pointer
         * to last component matched.
         */

        if(u.u_error)
                goto out;
        if(c == '\0')
                return(dp);

        /*
         * If there is another component,
         * gather up name into users' dir buffer.
         */


        cp = &u.u_dbuf[0];
        while(c != '/' && c != '\0' && u.u_error == 0) {
                if (mpxip!=NULL && c=='!')
                        break;
                if(cp < &u.u_dbuf[DIRSIZ])
                        *cp++ = c;
                c = (*func)();
        }
        while(cp < &u.u_dbuf[DIRSIZ])
                *cp++ = '\0';
        while(c == '/')
                c = (*func)();
        if (c == '!' && mpxip != NULL) {
                iput(dp);
                plock(mpxip);
                mpxip->i_count++;
                return(mpxip);
        }

seloop:
        /*
         * dp must be a directory and
         * must have X permission.
         */
        if((((dp->i_mode&IFMT) != IFDIR) && ((dp->i_mode&IFMT) != IFLDR))
            || dp->i_nlink==0)
                u.u_error = ENOTDIR;
        access(dp, IEXEC);
        if(u.u_error)
                goto out;

        /*
         * set up to search a directory
         */
        u.u_offset = 0;
        u.u_segflg = 1;
        eo = 0;
        bp = NULL;
        u.u_count = ldiv(dp->i_size1, DIRSIZ+2);
        if (dp == u.u_rdir)
        if (u.u_dbuf[0] == '.')
        if (u.u_dbuf[1] == '.')
        if (u.u_dbuf[2] == '\0')
                goto cloop;

eloop:

        /*
         * If at the end of the directory,
         * the search failed. Report what
         * is appropriate as per flag.
         */

        if(u.u_count == 0) {
                if(bp != NULL)
                        brelse(bp);
                if(flag==1 && c=='\0') {
                        if(access(dp, IWRITE))
                                goto out;
                        u.u_pdir = dp;
                        if(eo)
                                u.u_offset = eo-DIRSIZ-2;
                        else
                                bmap(dp, (daddr_t)(u.u_offset>>BSHIFT), B_WRITE
*  );
                        if (u.u_error)
                                goto out;
                        return(NULL);
                }
                u.u_error = ENOENT;
                goto out;
        }

        /*
         * If offset is on a block boundary,
         * read the next directory block.
         * Release previous if it exists.
         */

        if((u.u_offset&BMASK) == 0) {
                daddr_t bn;

                if(bp != NULL)
                        brelse(bp);
                bn = bmap(dp, (daddr_t)(u.u_offset>>BSHIFT), B_READ);
                if (u.u_error)
                        goto out;
                if (bn < 0) {
                        u.u_error = EIO;
                        goto out;
                }
                bp = bread(dp->i_dev, bn);
                if (u.u_error) {
                        brelse(bp);
                        goto out;
                }
        }

        /*
         * Note first empty directory slot
         * in eo for possible creat.
         * String compare the directory entry
         * and the current component.
         * If they do not match, go back to eloop.
         */

        copyio(paddr(bp)+((unsigned)u.u_offset&BMASK), (caddr_t)&u.u_dent,
                (DIRSIZ+2), U_RKD);
        u.u_offset += DIRSIZ+2;
        u.u_count--;
        if(u.u_dent.u_ino == 0) {
                if(eo == 0)
                        eo = u.u_offset;
                goto eloop;
        }
        for(i=0; i<DIRSIZ; i++)
                if(u.u_dbuf[i] != u.u_dent.u_name[i])
                        goto eloop;

        /*
         * Here a component matched in a directory.
         * If there is more pathname, go back to
         * cloop, otherwise return.
         */

        if(bp != NULL)
                brelse(bp);
        if(flag==2 && c=='\0') {
                if(access(dp, IWRITE))
                        goto out;
                return(dp);
        }
        d = dp->i_dev;
        if(u.u_dent.u_ino == ROOTINO)
        if(dp->i_number == ROOTINO)
        if(u.u_dent.u_name[1] == '.')
                for(i=1; i<NMOUNT; i++)
                        if(mount[i].m_bufp != NULL)
                        if(mount[i].m_dev == d) {
                                iput(dp);
                                dp = mount[i].m_inodp;
                                dp->i_count++;
                                plock(dp);
                                goto seloop;
                        }
        iput(dp);
        dp = iget(d, u.u_dent.u_ino);
        if(dp == NULL)
                return(NULL);
        goto cloop;
```

```
out:
        iput(dp);
        return(NULL);
}

/*
 * Return the next character from the
 * kernel string pointed at by dirp.
 */
schar()
{

        return(*u.u_dirp++ & 0377);
}

/*
 * Return the next character from the
 * user string pointed at by dirp.
 */
uchar()
{
        register c;

        c = fubyte(u.u_dirp++);
        if(c == -1)
                u.u_error = EFAULT;
        return(c);
}
```

```
/*      @(#)prf.c       2.7     */

#
#include "sys/param.h"
#include "sys/seg.h"
#include "sys/buf.h"
#include "sys/bufx.h"
#include "sys/conf.h"
#include "sys/confx.h"
#include "sys/systm.h"
#include "sys/sigdef.h"
#include "sys/proc.h"
#include "sys/procx.h"
#include "sys/dm11.h"
#include "sys/tty.h"
#include "sys/user.h"
#include "sys/userx.h"

/*
 * Address and structure of the
 * KL-11 console device registers.
 */
struct
{
        int     rsr;
        int     rbr;
        int     xsr;
        int     xbr;
};
char    *msgbufp msgbuf;        /* Next saved printf character */

/*
 * In case console is off,
 * panicstr contains argument to last
 * call to panic.
 */

char    *panicstr;

/*
 * Scaled down version of C Library printf.
 * Only %s %l %d (==%l) %o are recognized.
 * Used to print diagnostic information
 * directly on console tty.
 * Since it is not interrupt driven,
 * all system activities are pretty much
 * suspended.
 * Printf should not be used for chit-chat.
 */
printf(fmt,x1)
char fmt[];
{
        register char *s;
        register *adx, c;
        adx = &x1;
loop:
        while((c = *fmt++) != '%') {
                if(c == '\0')
                        return;
                putchar(c);
        }
        c = *fmt++;
        if((c == 'd')&&(*adx < 0)){
                *adx = -*adx;
                putchar('-');
        }
        if(c == 'd' || c == 'l' || c == 'o')
                printn(*adx, c=='o'? 8: 10);
        if(c == 's') {
                s = *adx;
                while(c = *s++)
                        putchar(c);
        }else
        if(c == 'c')
                putchar(*adx);
        adx++;
        goto loop;
}

/*
 * Print an unsigned integer in base b.
 */
printn(n, b)
{
        register a;

        if(a = ldiv(n, b))
                printn(a, b);
        putchar(lrem(n, b) + '0');
}

/*
 * Print a character on console.
 * Attempts to save and restore device
 * status.
 * If the switches are 0, all
 * printing is inhibited.
 *
 * Whether or not printing is inhibited,
 * the last MSGBUFS characters
 * are saved in msgbuf for inspection later.
 */
putchar(c)
{
        register rc, s, timo;

        rc = c;
        if (rc!='\0' && rc!='\r' && rc!=0177) {
                *msgbufp++ = rc;
                if (msgbufp >= &msgbuf[MSGBUFS])
```

```
				msgbufp = msgbuf;
	}
	if(SW->integ == 010)
		return;
	timo = 30000;
	/*
	 * Try waiting for the console tty to come ready,
	 * otherwise give up after a reasonable time.
	 */
	while((KL->xsr&0200)==0 && --timo!=0)
		;
	if(rc == 0)
		return;
	s = KL->xsr;
	KL->xsr = 0;
	KL->xbr = rc;
	if(rc == '\n') {
		putchar('\r');
		putchar(0177);
		putchar(0177);
	}
	putchar(0);
	KL->xsr = s;
}

/*
 * Panic is called on unresolvable
 * fatal errors.
 * It syncs, prints "panic: mesg" and
 * then loops.
 */
panic(s)
char *s;
{
	panicstr = s;
	update();
	printf("panic: %s\n", s);
	for(;;)
		idle();
}

/*
 * prdev prints a warning message of the
 * form "mesg on dev x/y".
 * x and y are the major and minor parts of
 * the device argument.
 */

static char	*emtab[] = {
	"bad block",
	"bad count",
	"no space",
	"Out of inodes"
};

prdev(type, dev)
```

```
register type, dev;
{

        logprdev(type, dev);
        printf("%s on dev %l/%l\n", emtab[type], major(dev), minor(dev));
}

#ifdef PWR_FAIL

/*
 * Power Fail Recovery Routine
 *
 * This routine is called whenever the processor takes a trap
 * through the power fail vector location.  All necessary data
 * is first saved, the power on subroutine is called, and then the
 * registers are reloaded.  The necessary assist to this code is
 * in mch.s.  Code runs at processor priority 7.
 *
 * Written by J. A. McGuire     2/78
 */

struct pwr_save {
        int     (*pf_addr)();           /* PIR function address */
        int     pf_pirr;                /* Previous outstanding request */
        int     pf_uisa[16];            /* User mem. mgt. regs */
        int     pf_uisd[16];
        int     pf_sdsa[6];             /* supv. addr. registers */
        int     pf_sdsd[6];             /* supv. desc. registers */
        int     pf_ubmap[62];
} pwr_save;

pwrfail(dev, sp, r1, nps, r0, pc, ps)
char *sp;
{
        register struct proc *pp;
        int cnt;
        extern *pirr, (*pir_fn)();
        extern char power, pwr_flg;
        extern pwr_ka6;

        power--;

        pwr_ka6 = *ka6;         /* save current u. address */

/*
 * Save user memory management registers
 */
        cnt = 32;
        if (cputype == 40)
                cnt = 16;

        bcopy(UISA, pwr_save.pf_uisa, cnt);
        bcopy(UISD, pwr_save.pf_uisd, cnt);

        if (cputype != 40) {
                bcopy(SDSA, pwr_save.pf_sdsa, sizeof(pwr_save.pf_sdsa));
```

```
		bcopy(SDSD, pwr_save.pf_sdsd, sizeof(pwr_save.pf_sdsd));
	}
	if (cputype == 70)
		bcopy(UBMAP, pwr_save.pf_ubmap, 124);

/*
 * If this is the first time pwrfail has been called,
 * save previous PIR requests and function address.  Then
 * set up to catch new trap and recall pwrfail.
 */
	if (pwr_save.pf_addr == 0) {
		pwr_save.pf_addr = pir_fn;
		pwr_save.pf_pirr = *pirr;
		pir_fn = pwrfail;
	}
/*
 * Save floating point registers, if necessary.
 */
	savfp();

/*
 * Everything has been saved.  Wait for power to be restored.
 *	CAUTION: General registers, other than r5 and sp,
 *		are not restored by the pwr_on() routine.

 * The machine assist guarantees the following: Once pwr_on
 * has been called, and the value of power is greater than
 * zero, any power fail trap appears as a return to the
 * call to pwr_on.  If the value of power is zero, then
 * pwrfail is called again to save all data, but pwr_on
 * will return to the first caller if pwr_flg is
 * non-zero.  In other words, the PIR entry to
 * pwrfail calls pwr_on to set up a return address, and
 * then proceeds to reinitialize each device.  A new power
 * fail trap will push the stack pointer down even further,
 * but when it calls pwr_on, the original stack pointer is
 * restored.
 *
 * All this is done so the device routines can be guaranteed
 * to be called only once when power is restored, regardless
 * of how often, or when, power fails.
 */

	pwr_on();

/*
 * Restore user memory mgt. regs
 */
	bcopy(pwr_save.pf_uisa, UISA, cnt);
	bcopy(pwr_save.pf_uisd, UISD, cnt);

	if (cputype != 40) {
		bcopy(pwr_save.pf_sdsa, SDSA, sizeof(pwr_save.pf_sdsa));
		bcopy(pwr_save.pf_sdsd, SDSD, sizeof(pwr_save.pf_sdsd));
	}
```

```
        if (cputype == 70)
                bcopy(pwr_save.pf_ubmap, UBMAP, 124);

/*
 * Reset power fail indicator.  If power fails again, we must
 * save data all over again.  While pwr_flg is set, calls will
 * not be recursive.
 */

        pwr_flg = 1;
        power = 0;

/*
 * Call device restart function.
 */

        restart(dev);

/*
 * Determine current state.  If this routine was called by a
 * power fail trap, and we haven't previously finished any
 * possible interrupt servicing routine, set up a PIR request
 * at priority 1 and return.
 *
 * Otherwise, send signal SIGPWR to all processes.
 */

        if (dev == NULL)
                *pirr = 1 << (1+8);     /* Priority 1 */
        else
                for (pp = &proc[0]; pp<procend; pp++)
                        if (pp->p_stat)
                                psignal(pp, SIGPWR);

        pwr_flg = 0;
}

/*
 * Power fail restart for devices
 *
 * This routine is called after power returns.  If the state
 * argument is NULL, then any previous interrupt service
 * routine processing has not been completed; device
 * open routines must set the appropriate flags to prevent
 * erroneous results.
 *
 * If the state flag is true, then restart everything --
 * routines may manipulate clists, queues, timeout, etc.,
 * as needed.
 *
 * When the previous interrupt processing is allowed to
 * complete, it must not initiate any DMA transfers,
 * since the appropriate registers are probably no longer
 * valid.  During this period, the external variable
 * pwr_fail is set to a one; otherwise, it has the value
 * zero.
 *
 * Pwr_fail is also used as a measurement of time remaining
 * when power fails.
 */

restart(state)
register state;
{
	register i;
	register int uerror;
	extern unsigned pwr_fail;

	uerror = u.u_error;

	if (state == 0) {
		if (pwr_fail)
			printf("\177\n%d", pwr_fail);
		printf("\177\n\n*** POWER FAIL \177");
		pwr_fail = 1;
	} else {
		pwr_fail = 0;
		printf("RESTART ***\n\n");
	}

/*
 * Reinitialize character devices.
 */

	for (i=0; i<nchrdev; i++)
		(*cdevsw[i].d_open)(NODEV, 0);


/*
 * Reset Programmed Interrupt Request, if any, and start clock
 */
	if (state) {
		pir_fn = pwr_save.pf_addr;
		*pirr = pwr_save.pf_pirr;
		pwr_save.pf_addr = 0;
		*lks = 0115;
	}

/*
 * Condition block devices
 */
	for (i=0; i<nblkdev; i++)
		(*bdevsw[i].d_open)(NODEV, 1);

	u.u_error = uerror;
}

/*
 * Character device restart routine.
 *
 * Called by device open routines whenever power-fail flag
 * is set.  This routine performs most routine initialization
 * for DH11's, KL11's, etc.
 *
 * Save state of previous carrier flag for each line, and then call
 * device open routine.  Start transmitters for kl like devices,
 * and, finally, generate hangup signals when carrier is no longer
 * there.
 */

pwr_init(base, cnt, xmtint)
struct tty *base;
int (*xmtint)();
{
	register struct tty *tp;
	register dev, carr;

	for (tp=base; tp < &base[cnt]; tp++)
	if (tp->t_state&ISOPEN) {
		dev = tp->t_dev;
		carr = tp->t_state & (CARRIER|SUPRD);
		tp->t_state =& ~ISOPEN;

		(*cdevsw[tp->t_dev.d_major].d_open)(dev, 0);

		if (xmtint)
			(*xmtint)(tp->t_dev.d_minor);

		if (carr && (tp->t_state&(CARRIER|SUPRD))==0)
			(*linesw[tp->t_ltype].l_dst)(tp,
				CTRANS|CSTRANS|SRTRANS|(dev%16), 0);
	}
}
#endif
```

```
/*      @(#)rdwri.c     2.7     */

#include "sys/param.h"
#include "sys/inode.h"
#include "sys/inodex.h"
#include "sys/user.h"
#include "sys/userx.h"
#include "sys/buf.h"
#include "sys/bufx.h"
#include "sys/conf.h"
#include "sys/confx.h"
#include "sys/systm.h"
#include "sys/ipcomm.h"

/*
 * Read the file corresponding to
 * the inode pointed at by the argument.
 * The actual read arguments are found
 * in the variables:
 *      u_base          core address for destination
 *      u_offset        byte offset in file
 *      u_count         number of bytes to read
 *      u_segflg        read to kernel/user/user I
 */
readi(ip)
register struct inode *ip;
{
        struct buf *bp;
        dev_t dev;
        daddr_t lbn, bn;
        long diff;
        register unsigned on, n;
        register type;

        if(u.u_count == 0)
                return;
        dev = (dev_t)ip->i_un.i_rdev;
        type = ip->i_mode&IFMT;
        if (type==IFCHR || type==IFMPC) {
                ip->i_flag |= IACC;
                (*cdevsw[major(dev)].d_read)(dev);
                return;
        }
        do {
                lbn = bn = u.u_offset >> BSHIFT;
                on = u.u_offset & BMASK;
                n = min((unsigned)BSIZE-on, u.u_count);
                if (type!=IFBLK && type!=IFMPB) {
                        diff.loword = ip->i_size1;
                        diff.hiword = ip->i_size0&0377;
                        diff -= u.u_offset;
                        if(diff <= 0)
                                return;
                        if (diff < n)
                                n = diff;
```

```
				bn = bmap(ip, lbn, B_READ);
				if (u.u_error)
					return;
				dev = ip->i_dev;
			} else {
				rablock = bn+1;
			}
			if (bn == (daddr_t)-1) {
				bp = getablk(0);
				clear(paddr(bp), BSIZE);
				bp->b_resid = 0;
			} else if (ip->i_un.i_lastr+1 == lbn && (on+n) == BSIZE) {
				bp = breada(dev, bn, rablock);
			} else
				bp = bread(dev, bn);
			if ((on+n) == BSIZE)
				ip->i_un.i_lastr = lbn;
			if (bp->b_resid)
				n = 0;
			pimove(paddr(bp)+on, n, B_READ);
			brelse(bp);
			ip->i_flag |= IACC;
	} while(u.u_error==0 && u.u_count!=0 && n!=0);
}

/*
 * Write the file corresponding to
 * the inode pointed at by the argument.
 * The actual write arguments are found
 * in the variables:
 *	u_base		core address for source
 *	u_offset	byte offset in file
 *	u_count		number of bytes to write
 *	u_segflg	write to kernel/user/user I
 */
writei(ip)
register struct inode *ip;
{
	struct buf *bp;
	dev_t dev;
	daddr_t bn;
	register unsigned n, on;
	register type;
	long lng;

	dev = ip->i_un.i_rdev;
	type = ip->i_mode&IFMT;
	if (type==IFCHR || type==IFMPC) {
		ip->i_flag =| IACC|IUPD;
		(*cdevsw[major(dev)].d_write)(dev);
		return;
	}
	while (u.u_error==0 && u.u_count!=0) {
		bn = u.u_offset >> BSHIFT;
		on = u.u_offset & BMASK;
		n = min((unsigned)BSIZE-on, u.u_count);
```

```
                if(type!=IFBLK && type!=IFMPB) {
                        if ((bn = bmap(ip, bn, B_WRITE)) == (daddr_t)-1
                            || bn == 0)
                                return;
                        dev = ip->i_dev;
                }
                if(n == BSIZE)
                        bp = getblk(dev, bn);
                else
                        bp = bread(dev, bn);
                pimove(paddr(bp)+on, n, B_WRITE);
                if(u.u_error != 0)
                        brelse(bp);
                else if ((u.u_offset&BMASK)==0)
                        bawrite(bp);
                else
                        bdwrite(bp);
                lng.loword = ip->i_size1;
                lng.hiword = ip->i_size0&0377;
                if (u.u_offset > lng &&
                  (type==IFREG || type==IFDIR || type==IFLRG || type==IFLDR)) {
                        ip->i_size0 = u.u_offset.hiword;
                        ip->i_size1 = u.u_offset.loword;
                }
                ip->i_flag |= IACC|IUPD;
        }
}

/*
 * Return the logical maximum
 * of the 2 arguments.
 */
max(a, b)
unsigned a, b;
{

        if(a > b)
                return(a);
        return(b);
}

/*
 * Return the logical minimum
 * of the 2 arguments.
 */
min(a, b)
unsigned a, b;
{

        if(a < b)
                return(a);
        return(b);
}

pimove(cp, n, flag)
paddr_t cp;
```

```
register unsigned n;
{

        if (u.u_error || n == 0)
                return;
        if (copyio(cp, u.u_base, n, (u.u_segflg<<1)|flag))
                u.u_error = EFAULT;
        else {
                u.u_base += n;
                u.u_offset += n;
                u.u_count -= n;
        }
}
```

```
/*      @(#)sig.c       2.6.1.1 */

#include "sys/param.h"
#include "sys/systm.h"
#include "sys/user.h"
#include "sys/userx.h"
#include "sys/proc.h"
#include "sys/procx.h"
#include "sys/inode.h"
#include "sys/inodex.h"
#include "sys/reg.h"
#include "sys/text.h"
#include "sys/textx.h"
#include "sys/seg.h"
#include "sys/vtmn.h"

/*
 * Priority for tracing
 */
#define IPCPRI  -1

/*
 * Structure to access an array of integers.
 */
struct
{
        int     inta[];
};

/*
 * Tracing variables.
 * Used to pass trace command from
 * parent to child being traced.
 * This data base cannot be
 * shared and is locked
 * per user.
 */
struct
{
        int     ip_lock;
        int     ip_req;
        int     ip_addr;
        int     ip_data;
} ipc;

/*
 * Send the specified signal to
 * all processes with 'pgrp' as
 * process group.
 * Called by tty.c for quits and
 * interrupts.
 */
signal(pgrp, sig)
register pgrp;
{
        register struct proc *p;

        if(pgrp == 0)
                return;
        for(p = &proc[0]; p < &proc[NPROC]; p++)
                if(p->p_pgrp == pgrp)
                        psignal(p, sig);
}

/*
 * Send the specified signal to
 * the specified process.
 */
psignal(p, sig)
register struct proc *p;
register sig;
{

        if((unsigned)sig >= NSIG)
                return;
        if(sig)
                p->p_sig =| 1L<<(sig-1);
        if(p->p_pri > PUSER) {
                p->p_pri = PUSER;
                VTPROCENT(p,PR_PRI);
        }
        if(p->p_stat == SSLEEP && p->p_pri >= PZERO)
                setrun(p);
        VTPROCENT(p,PR_SIG);
}

/*
 * Returns true if the current
 * process has a signal to process.
 * This is asked at least once
 * each time a process enters the
 * system.
 * A signal does not do anything
 * directly to a process; it sets
 * a flag that asks the process to
 * do something to itself.
 */
issig()
{
        register n;
        register struct proc *p, *q;

        p = u.u_procp;
        while(p->p_sig) {
                n = fsig(p);
                if(n == SIGCLD) {
                        if(u.u_signal[SIGCLD]&01) {
                                for(q = &proc[0];q < &proc[NPROC];q++)
                                        if(p->p_pid==q->p_ppid && q->p_stat==SZ
* OMB)
                                                freeproc(q, 0);
```

s=spl6()

splx(s)

```
                        }
                        else if(u.u_signal[SIGCLD])
                                return(n);
                }
#ifdef PWR_FAIL
                else if (n == SIGPWR) {
                        if (u.u_signal[SIGPWR] && (u.u_signal[SIGPWR]&1)==0)
                                return(n);
                }
#endif
                else if((u.u_signal[n]&1) == 0 || (p->p_flag&STRC))
                        return(n);
                p->p_sig =& ~(1L<<(n-1));
                VTPROCENT(p,PR_SIG);
        }
        return(0);
}

/*
 * Enter the tracing STOP state.
 * In this state, the parent is
 * informed and the process is able to
 * receive commands from the parent.
 */
stop()
{
        register struct proc *pp, *cp;

loop:
        cp = u.u_procp;
        if(cp->p_ppid != 1)
        for (pp = &proc[0]; pp < &proc[NPROC]; pp++)
                if (pp->p_pid == cp->p_ppid) {
                        wakeup(pp);
                        cp->p_stat = SSTOP;
                        swtch();
                        if ((cp->p_flag&STRC)==0 || procxmt())
                                return;
                        goto loop;
                }
        exit();
}

/*
 * Perform the action specified by
 * the current signal.
 * The usual sequence is:
 *      if(issig())
 *              psig();
 */
psig()
{
        register n, p;
        register *rp;

        rp = u.u_procp;
	if (rp->p_flag&STRC)
		stop();
	n = fsig(rp);
	if (n==0)
		return;
	rp->p_sig =& ~(1L<<(n-1));
	VTPROCENT(rp,PR_SIG);
	if((p=u.u_signal[n]) != 0) {
		u.u_error = 0;
		if(n != SIGINS && n != SIGTRC)
			u.u_signal[n] = 0;
		n = u.u_ar0[R6] - 4;
		grow(n);
		suword(n+2, u.u_ar0[RPS]);
		suword(n, u.u_ar0[R7]);
		u.u_ar0[R6] = n;
		u.u_ar0[RPS] =& ~TBIT;
		u.u_ar0[R7] = p;
		return;
	}
	switch(n) {

	case SIGQIT:
	case SIGINS:
	case SIGTRC:
	case SIGIOT:
	case SIGEMT:
	case SIGFPT:
	case SIGBUS:
	case SIGSEG:
	case SIGSYS:
		u.u_arg[0] = n;
		if(core())
			n =+ 0200;
	}
	u.u_arg[0] = (u.u_ar0[R0]<<8) | n;
	exit();
}

/*
 * find the signal in bit-position
 * representation in p_sig.
 */
fsig(p)
struct proc *p;
{
	register i;
	long n;

	n = p->p_sig;
	if(n & (1L << (SIGINS-1)))
		return(SIGINS);
	for(i=1; i<NSIG; i++) {
		if(n & 1L)
			return(i);
		n =>> 1;
	}
	return(0);
}

/*
 * Create a core image on the file "core"
 * If you are looking for protection glitches,
 * there are probably a wealth of them here
 * when this occurs to a suid command.
 *
 * It writes USIZE block of the
 * user.h area followed by the entire
 * data+stack segments.
 */
core()
{
	register s, *ip;
	extern schar();

	u.u_error = 0;
	u.u_dirp = "core";
	ip = namei(schar, 1);
	if(ip == NULL) {
		if(u.u_error)
			return(0);
		ip = maknode(0666);
		if (ip==NULL)
			return(0);
	}
	if(!access(ip, IWRITE) &&
	   ((ip->i_mode&IFMT) == IFREG || (ip->i_mode&IFMT) == IFLRG)) {
		itrunc(ip);
		u.u_offset = 0;
		u.u_base = (caddr_t)&u;
		u.u_count = USIZE*64;
		u.u_segflg = 1;
		writei(ip);

/*	Clear the maus bit map so that those registers are available	*/
/*	to be used when establishing the mapping registers for the	*/
/*	core dump.							*/

		u.u_mbitm = 0 ;
		s = u.u_procp->p_size - USIZE;
		estabur((unsigned)0, s, (unsigned)0, 0, RO);
		u.u_base = 0;
		u.u_count = s*64;
		u.u_segflg = 0;
		writei(ip);
	} else
		u.u_error = EPERM;
	iput(ip);
	return(u.u_error==0);
}

/*
```

```
 * grow the stack to include the SP
 * true return if successful.
 */

grow(sp)
unsigned sp;
{
        register a, si, i;

        if(sp >= -u.u_ssize*64)
                return(0);
        si = ldiv(-sp, 64) - u.u_ssize + SINCR;
        if(si <= 0)
                return(0);
        if(estabur(u.u_tsize, u.u_dsize, u.u_ssize+si, u.u_sep, RO))
                return(0);
        expand(u.u_procp->p_size+si);
        a = u.u_procp->p_addr + u.u_procp->p_size;
        for(i=u.u_ssize; i; i--) {
                a--;
                copyseg(a-si, a);
        }
        for(i=si; i; i--)
                clearseg(--a);
        u.u_ssize =+ si;
        return(1);
}

/*
 * sys-trace system call.
 */
ptrace()
{
        register struct proc *p;
        register struct text *xp;

        if (u.u_arg[2] <= 0) {
                u.u_procp->p_flag =| STRC;
                VTPROCENT(u.u_procp,PR_FLAG);
                return;
        }
        for (p=proc; p < &proc[NPROC]; p++)
                if (p->p_stat==SSTOP
                 && p->p_pid==u.u_arg[0]
                 && p->p_ppid==u.u_procp->p_pid)
                        goto found;
        u.u_error = ESRCH;
        return;

    found:
        while (ipc.ip_lock)
                sleep(&ipc, IPCPRI);
        ipc.ip_lock = p->p_pid;
        ipc.ip_data = u.u_ar0[R0];
        ipc.ip_addr = u.u_arg[1] & ~01;
        ipc.ip_req = u.u_arg[2];
        p->p_flag =& ~SWTED;
        setrun(p);
        while (ipc.ip_req > 0)
                sleep(&ipc, IPCPRI);
        u.u_ar0[R0] = ipc.ip_data;
        if (ipc.ip_req < 0)
                u.u_error = EIO;
        ipc.ip_lock = 0;
        wakeup(&ipc);
}

/*
 * Code that the child process
 * executes to implement the command
 * of the parent process in tracing.
 */
procxmt()
{
        register int i;
        register int *p;
        register struct text *xp;

        if (ipc.ip_lock != u.u_procp->p_pid)
                return(0);
        i = ipc.ip_req;
        ipc.ip_req = 0;
        wakeup(&ipc);
        switch (i) {

        /* read user I */
        case 1:
                if (fuibyte(ipc.ip_addr) == -1)
                        goto error;
                ipc.ip_data = fuiword(ipc.ip_addr);
                break;

        /* read user D */
        case 2:
                if (fubyte(ipc.ip_addr) == -1)
                        goto error;
                ipc.ip_data = fuword(ipc.ip_addr);
                break;

        /* read u */
        case 3:
                i = ipc.ip_addr;
                if (i<0 || i >= (USIZE<<6))
                        goto error;
                ipc.ip_data = u.inta[i>>1];
                break;

        /* write user I */
        /* Must set up to allow writing */
        case 4:
                /*
                 * If text, must assure exclusive use
```

```
		 */
		if (xp = u.u_procp->p_textp) {
			if (xp->x_count!=1 || xp->x_iptr->i_mode&ISVTX)
				goto error;
			xp->x_iptr->i_flag =& ~ITEXT;
		}
		estabur(u.u_tsize, u.u_dsize, u.u_ssize, u.u_sep, RW);
		i = suiword(ipc.ip_addr, 0);
		suiword(ipc.ip_addr, ipc.ip_data);
		estabur(u.u_tsize, u.u_dsize, u.u_ssize, u.u_sep, RO);
		if (i<0)
			goto error;
		if (xp)
			xp->x_flag =| XWRIT;
		break;

	/* write user D */
	case 5:
		if (suword(ipc.ip_addr, 0) < 0)
			goto error;
		suword(ipc.ip_addr, ipc.ip_data);
		break;

	/* write u */
	case 6:
		p = &u.inta[ipc.ip_addr>>1];
		if (p >= u.u_fsav && p < &u.u_fsav[25])
			goto ok;
		for (i=0; i<9; i++)
			if (p == &u.u_ar0[regloc[i]])
				goto ok;
		goto error;
	ok:
		if (p == &u.u_ar0[RPS]) {
			ipc.ip_data =| 0170000; /* assure user space */
			ipc.ip_data =& ~0340;   /* priority 0 */
		}
		*p = ipc.ip_data;
		break;

	/* set signal and continue */
	case 7:
		u.u_procp->p_sig = 0;
		psignal(u.u_procp, ipc.ip_data);
		return(1);

	/* force exit */
	case 8:
		exit();

	default:
	error:
		ipc.ip_req = -1;
	}
	return(0);
}
```

/* one version causes a trace trap */

case 9:
    u.u_ar0[RPS] |= TBIT;

```
/*      @(#)slp.c       2.9.1.1 */

#
#include "sys/param.h"
#include "sys/user.h"
#include "sys/userx.h"
#include "sys/lock.h"
#include "sys/proc.h"
#include "sys/procx.h"
#include "sys/text.h"
#include "sys/textx.h"
#include "sys/systm.h"
#include "sys/file.h"
#include "sys/filex.h"
#include "sys/inode.h"
#include "sys/inodex.h"
#include "sys/buf.h"
#include "sys/bufx.h"
#include "sys/ipcomm.h"
#include "sys/vtmn.h"
#include "sys/sysmes.h"
#include "sys/sysmesx.h"

struct proc *procend {&proc[1]};

#define SQSIZE 0100     /* Must be power of 2 */
#define HASH(x) (( (int) x >> 5) & (SQSIZE-1))
struct proc *slpque[SQSIZE];

/*
 * Give up the processor till a wakeup occurs
 * on chan, at which time the process
 * enters the scheduling queue at priority pri.
 * The most important effect of pri is that when
 * pri<PZERO a signal cannot disturb the sleep;
 * if pri>PZERO signals will be processed with non-local goto;
 * if pri==PZERO a signal causes the sleep to return to caller.
 * Callers of this routine must be prepared for
 * premature return, and check that the reason for
 * sleeping has gone away.
 */
sleep(chan, pri)
caddr_t chan;
{
        register struct proc *rp;
        register s, h;

        rp = u.u_procp;
        s = spl6();
        rp->p_stat = SSLEEP;
        rp->p_wchan = chan;
        rp->p_pri = pri;
        rp->p_ctime = 0;
        h = HASH(chan);
        rp->p_link = slpque[h];
```

spl7()

```
        slpque[h] = rp;
        if(pri >= PZERO) {
                if(issig()) {
                        rp->p_wchan = 0;
                        rp->p_stat = SRUN;
                        slpque[h] = rp->p_link;
                        spl0();
                        if (pri > PZERO)
                                goto psig;
                        else
                                goto ret;
                }
                spl0();
                if(runin != 0) {
                        runin = 0;
                        wakeup((caddr_t)&runin);
                }
                swtch();
                if(pri > PZERO && issig())
                        goto psig;
        } else {
                spl0();
                swtch();
        }
ret:
        splx(s);
        return;

        /*
         * If priority was low (>PZERO) and
         * there has been a signal,
         * execute non-local goto to
         * the qsav location.
         * (see trap1/trap.c)
         */
psig:
        aretu(u.u_qsav);
}

/*
 * Wake up all processes sleeping on chan.
 */
wakeup(chan)
register caddr_t chan;
{
        register struct proc *p, *q;
        register i;
        int s;

        s = spl6();              spl7() ?
        i = HASH(chan);
        p = slpque[i];
        q = NULL;
        while(p != NULL) {
                if(p->p_wchan==chan && p->p_stat!=SZOMB) {
                        if (q == NULL)
```

```
					slpque[i] = p->p_link;
				else
					q->p_link = p->p_link;
				p->p_wchan = 0;
				setrun(p);
				p = slpque[i];
				q = NULL;
				continue;
			}
			q = p;
			p = p->p_link;
		}
		splx(s);
}

/*
 * when you are sure that it
 * is impossible to get the
 * 'proc on q' diagnostic, the
 * diagnostic loop can be removed.
 */
setrq(p)
struct proc *p;
{
		register struct proc *q;
		register s;

		s = spl6();          spl7();
/*
		for(q=runq; q!=NULL; q=q->p_link)
			if(q == p) {
				printf("proc on q\n");
				goto out;
			}
*/
		p->p_link = runq;
		runq = p;
out:
		splx(s);
}

/*
 * Set the process running;
 * arrange for it to be swapped in if necessary.
 */
setrun(p)
register struct proc *p;
{
		register caddr_t w;

		if (p->p_stat==0 || p->p_stat==SZOMB)
			panic("Running a dead proc");
		if (w = p->p_wchan) {
			wakeup(w);
			return;
		}
```

```
        p->p_ctime = 0;
        p->p_stat = SRUN;
        VTPROCENT(p,PR_WKP);
        setrq(p);
        if(p->p_pri < curpri)
                runrun++;
        else if(p->p_pri < nxtpri)
                nxtpri = p->p_pri;
        if(runout != 0 && (p->p_flag&SLOAD) == 0) {
                runout = 0;
                wakeup((caddr_t)&runout);
        }
}

/*
 * Set user priority.
 */
setpri(up)
{
        register *pp, p;

        pp = up;
        p = (pp->p_cpu & 0377) / 16;
        p =+ PUSER + pp->p_nice;
        if(p > 127)
                p = 127;
        pp->p_pri = p;
        VTPROCENT(pp,PR_PRI);
        return(p);
}

/*
 * The main loop of the scheduling (swapping)
 * process.
 * The basic idea is:
 *  see if anyone wants to be swapped in;
 *  swap out processes until there is room;
 *  swap him in;
 *  repeat.
 * The runout flag is set whenever someone is swapped out.
 * Sched sleeps on it awaiting work.
 *
 * Sched sleeps on runin whenever it cannot find enough
 * core (by swapping out or otherwise) to fit the
 * selected swapped process.  It is awakend when the
 * core situation changes and in any event once per second.
 */
int maxsize;
int ns;
struct proc *p1, *p2;

sched()
{
        register struct proc *rp;
        register a, n;
static x;
```

```
	/*
	 * find user to swap in;
	 * of users ready, select one out longest
	 * of lowest nice.
	 */

loop:
	spl6();
#ifdef	PTLOCK
	if(runlock)
	{
		runlock = 0;
		shuffle();
	}
#endif
	n = -1;
	a = 100;
	for(rp = runq; rp != NULL; rp = rp->p_link)
	if(rp->p_stat==SRUN && (rp->p_flag&SLOAD)==0 &&
	    (rp->p_nice < a || (rp->p_nice==a && rp->p_time > n))) {
		p1 = rp;
		n = rp->p_time;
		a = rp->p_nice;
	}
	/*
	 * if there is none, wait.
	 */
	if(n == -1) {
		runout++;
		sleep(&runout, PSWP);
		goto loop;
	}
	spl0();
	rp = p1;
	if (rp->p_nice < 0)
		proc[0].p_nice = rp->p_nice;
	else
		proc[0].p_nice = 0;

	/*
	 * see if there is core for that process;
	 * if so, swap it in.
	 */

	if (swapin(rp))
	{
		goto loop;
	}

	/*
	 * none found,
	 * if process has positive nice, don't
	 * try to bring in unless it has been
	 * out for at least "nice" seconds.
	 */
	if((a = rp->p_nice) > 0 && n < a)
		goto sloop;

	/*
	 * look around for core.
	 * select the largest of those
	 * sleeping at positive priority;
	 * if none, select process which has
	 * been sleeping at negative priority for
	 * longer than usual;
	 * if none, select process which has "nice"
	 * of equal or greater value than this one
	 * and which has been in core longest.
	 */

	spl6();
	p2 = 0;
	ns = 1;
	maxsize = -1;
	n = -1;
	for(rp = &proc[0]; rp < procend; rp++) {
		if((rp->p_flag&(SSYS|SLOCK|SLOAD))!=SLOAD)
			continue;
		if (rp->p_textp && rp->p_textp->x_flag&XLOCK)
			continue;
		if(rp->p_stat==SSLEEP && rp->p_pri>=PZERO || rp->p_stat==SSTOP)
*   {
			if(maxsize < rp->p_size) {
				p2 = rp;
				maxsize = rp->p_size;
			}
		} else if(maxsize < 0) {
			if(rp->p_stat==SSLEEP && rp->p_ctime > ns) {
				p2 = rp;
				ns = rp->p_ctime;
			} else if(ns == 1 &&
				(rp->p_stat==SRUN || rp->p_stat==SSLEEP) &&
				(rp->p_nice > a ||
				    (rp->p_nice==a && rp->p_time > n))) {
					p2 = rp;
					n = rp->p_time;
					a = rp->p_nice;
			}
		}
	}

	spl0();
	/*
	 * Swap found user out if sleeping at bad pri,
	 * or if he has spent at least 2 seconds in core and
	 * the swapped-out process has spent at least 3 seconds out.
	 * Otherwise wait a bit and try again.
	 */
	if(p2!=0 && (maxsize>=0 || ns!=1 ||
	    a!=p1->p_nice || (p1->p_time>=3 && n>=2))) {
		rp = p2;
```

state time

resident time

```
                rp->p_flag =& ~SLOAD;
                xswap(rp, 1, 0);
                goto loop;
        }

sloop:
        spl6();
        runin++;
        sleep(&runin, PSWP);
        goto loop;
}

/*
 * Swap a process in.
 * Allocate data and possible text separately.
 * It would be better to do largest first.
 */
swapin(pp)
struct proc *pp;
{
        register struct proc *p;
        register int a;
        int x;

        p = pp;
        if ((a = malloc(coremap, p->p_size)) == NULL)
                return(0);
        if(p->p_textp)
        {
                if(xswapin(p->p_textp) == 0)
                {
                        mfree(coremap,p->p_size,a);
                        return(0);
                }
        }
        swap(p->p_addr, a, p->p_size, B_READ);
        mfree(swapmap, ctod(p->p_size), p->p_addr);
        p->p_addr = a;
        p->p_flag =| SLOAD;
        p->p_time = 0;
        VTPROCENT(p,PR_SWP);
        return(1);
}

qswtch()
{
        setrq(u.u_procp);
        swtch();
}

/*
 * This routine is called to reschedule the CPU.
 * if the calling process is not in RUN state,
 * arrangements for it to restart must have
 * been made elsewhere, usually by calling via sleep.
 */
```

```
swtch()
{
        register n;
        register struct proc *p, *q;
        static struct proc *pp, *pq;

        VTPROCENT(u.u_procp,PR_SWH);
        meas.m_swtch++;
        /*
         * Remember stack of caller
         * and switch to schedulers stack.
         */
        savu(u.u_rsav);
        retu(proc[0].p_addr);

loop:
        spl6();          spl7()
        runrun = 0;
        pp = NULL;
        q = NULL;
        n = 128;
        /*
         * Search for highest-priority runnable process
         */
        for(p=runq; p!=NULL; p=p->p_link) {
                if((p->p_stat==SRUN) && (p->p_flag&SLOAD)) {
                        if(p->p_pri <= n) {
                                pp = p;
                                pq = q;
                                nxtpri = n;
                                n = p->p_pri;
                        }
                }
                q = p;
        }
        /*
         * If no process is runnable, idle.
         */
        p = pp;
        if(p == NULL) {
                VTPROCENT(0,PR_BNKOF);
                idle();
                spl0();
                goto loop;
        }
        q = pq;
        if(q == NULL)
                runq = p->p_link; else
                q->p_link = p->p_link;
        curpri = n;
        spl0();
        /*
         * Switch to stack of the new process and set up
         * his segmentation registers.
         */
        retu(p->p_addr);
        sureg();
        /*
         * If the new process paused because it was
         * swapped out, set the stack level to the last call
         * to savu(u_ssav).  This means that the return
         * which is executed immediately after the call to aretu
         * actually returns from the last routine which did
         * the savu.
         */
        if(p->p_flag&SSWAP) {
                p->p_flag =& ~SSWAP;
                aretu(u.u_ssav);
        }
        /*
         * The value returned here has many subtle implications.
         * See the newproc comments.
         */
        VTPROCENT(p,PR_BLINK);
        return(1);
}

/*
 * Create a new process-- the internal version of
 * sys fork.
 * It returns 1 in the new process.
 * How this happens is rather hard to understand.
 * The essential fact is that the new process is created
 * in such a way that appears to have started executing
 * in the same call to newproc as the parent;
 * but in fact the code that runs is that of swtch.
 * The subtle implication of the returned value of swtch
 * (see above) is that this is the value that newproc's
 * caller in the new process sees.
 */
newproc()
{
        int a1, a2;
        struct proc *p, *up;
        register struct proc *rpp;
        register *rip, n;
        struct proc *pend;

        p = NULL;
        /*
         * First, just locate a slot for a process
         * and copy the useful info from this process into it.
         * The panic "cannot happen" because fork has already
         * checked for the existence of a slot.
         */
retry:
        mpid++;
        if(mpid < 0) {
                mpid = 0;
                goto retry;
        }
        for(rpp = &proc[0]; rpp < &proc[NPROC]; rpp++) {
```

```
		if(rpp->p_stat == NULL){
			if(p == NULL)
				p = rpp;
		}else
			pend = rpp;
		if ((rpp->p_pid==mpid)||(rpp->p_pgrp == mpid))
			goto retry;
	}
	if ((rpp = p)==NULL)
		panic("no procs");

	if(rpp>pend)
		pend = rpp;
	pend++;
	procend = pend;
	/*
	 * make proc entry for new proc
	 */

	rip = u.u_procp;
	up = rip;
	rpp->p_stat = SRUN;
	meas.m_fork++;
	rpp->p_clktim = 0;
	rpp->p_flag = SLOAD;
	rpp->p_uid = rip->p_uid;
	rpp->p_pgrp = rip->p_pgrp;
	rpp->p_nice = rip->p_nice;
	rpp->p_textp = rip->p_textp;
	rpp->p_pid = mpid;
	rpp->p_ppid = rip->p_pid;
	rpp->p_time = 0;
	rpp->p_cpu = 0;

	/*
	 * make duplicate entries
	 * where needed
	 */

	for(rip = &u.u_ofile[0]; rip < &u.u_ofile[NOFILE];)
		if((rpp = *rip++) != NULL)
			rpp->f_count++;
	if((rpp=up->p_textp) != NULL) {
		rpp->x_count++;
		rpp->x_ccount++;
	}
	u.u_cdir->i_count++;
	u.u_rdir->i_count++;
	/*
	 * Partially simulate the environment
	 * of the new process so that when it is actually
	 * created (by copying) it will look right.
	 */
	savu(u.u_rsav);
	rpp = p;
	u.u_procp = rpp;
```

```
        rip = up;
        n = rip->p_size;
        a1 = rip->p_addr;
        rpp->p_size = n;
        VTNEWPROC(rpp,PR_NEW,rip,0);
        a2 = malloc(coremap, n);              savfp();
        /*
         * If there is not enough core for the
         * new process, swap out the current process to generate the
         * copy.
         */
        if(a2 == NULL) {
                rip->p_stat = SIDL;
                rpp->p_addr = a1;
                VTPROCENT(rip,PR_STATE);
                savu(u.u_ssav);
                xswap(rpp, 0, 0);
                rpp->p_flag =| SSWAP;
                rip->p_stat = SRUN;
                VTPROCENT(rpp,PR_FLAG);
                VTPROCENT(rip,PR_STATE);
        } else {
        /*
         * There is core, so just copy.
         */
                rpp->p_addr = a2;
                while(n--)
                        copyseg(a1++, a2++);
        }
        u.u_procp = rip;
        setrq(rpp);
        return(0);
}

/*
 * Change the size of the data+stack regions of the process.
 * If the size is shrinking, it's easy-- just release the extra core.
 * If it's growing, and there is core, just allocate it
 * and copy the image, taking care to reset registers to account
 * for the fact that the system's stack has moved.
 * If there is no core, arrange for the process to be swapped
 * out after adjusting the size requirement-- when it comes
 * in, enough core will be allocated.
 * Because of the ssave and SSWAP flags, control will
 * resume after the swap in swtch, which executes the return
 * from this stack level.
 *
 * After the expansion, the caller will take care of copying
 * the user's stack towards or away from the data area.
 */
expand(newsize)
{
        int i, n;
        register *p, a1, a2;

        p = u.u_procp;
        n = p->p_size;
        p->p_size = newsize;
        VTPROCENT(p,PR_SIZE);
        a1 = p->p_addr;
        if(n >= newsize) {
                mfree(coremap, n-newsize, a1+newsize);
                return;
        }
        savu(u.u_rsav);
        a2 = malloc(coremap, newsize);
        if(a2 == NULL) {                          savfp();
                savu(u.u_ssav);
                xswap(p, 1, n);
                p->p_flag =| SSWAP;
                if(u.u_lock&PROCLOCK)
                        runlock++;
                qswtch();
                /* no return */
        }
        p->p_addr = a2;
        for(i=0; i<n; i++)
                copyseg(a1+i, a2++);
        mfree(coremap, n, a1);
        if(u.u_lock&PROCLOCK)
                runlock++;
        retu(p->p_addr);
        sureg();
}

#ifdef  PTLOCK
shuffle()
{
        struct proc *pp;
        struct text *xp;

        for(pp= &proc[1]; pp< procend; pp++)
        {
                if((pp->p_flag&SLOCK) || (pp->p_stat == NULL)
                  || (pp->p_textp && pp->p_textp->x_flag&XLOCK))
                        continue;
                if(pp->p_flag&SLOAD)
                {
                        /* swap out complete process */
                        pp->p_flag =& ~SLOAD;
                        xswap(pp,1,0);
                }
                if((xp=pp->p_textp) != NULL && xp->x_lcount
                        && xp->x_ccount==1)
                        /*swap text out */
                        xccdec(xp);
        }
        for(pp= &proc[1]; pp< procend;pp++)
        {
                if((xp=pp->p_textp) != NULL && xp->x_lcount != 0
                        && xp->x_ccount == 0)
```

```
				/*swap text in only */
				xswapin(xp);
			if((pp->p_flag&SSYS) && (pp->p_flag&SLOAD) == 0)
				/*swap data only in */
				swapin(pp);
		}
		wakeup(&runlock);
}
#endif
```