```
#       @(#)emerg.mk    2.1

VERSION = emerg
CFLAGS = -O -I/usr/include/util

LIB = lib2.$(VERSION).a
COMPOOL=

LIB2OBJS = \
        $(LIB)(bio.o) \
        $(LIB)(tty.o) \
        $(LIB)(malloc.o) \
        $(LIB)(pipe.o) \
        $(LIB)(err.o) \
        $(LIB)(hps.o) \
        $(LIB)(ht.o) \
        $(LIB)(kl.o) \
        $(LIB)(dhfdm.o) \
        $(LIB)(mem.o) \
        $(LIB)(sys.o) \
        $(LIB)(ttdma.o) \
        $(LIB)(partab.o) \
        $(LIB)(rh.o) \
        $(LIB)(devstart.o) \
        $(LIB)(ioctl.o) \
        $(LIB)(fakemx.o)

all:    $(LIB)
        @echo $(LIB) is now up-to-date.

$(LIB):         $(LIB2OBJS)

$(LIB2OBJS):    $(FRC)

FRC:
        rm -f $(LIB)

clobber: cleanup
        -rm -f $(LIB) *.o

clean cleanup:

install:        all

.PRECIOUS:      $(LIB)

.s.a:
        $(AS) $(ASFLAGS) -o $*.o $<
        ar rcv $@ $*.o
        rm $*.o
```

```
/*      @(#)err.c       2.1.1.1 */

#include "sys/param.h"
#include "sys/buf.h"
#include "sys/dir.h"
#include "sys/user.h"
#include "sys/userx.h"
#include "sys/elog.h"

static  short   logging;

erropen(dev,flg)
{
        extern unsigned pwr_fail;

#ifdef POWERFAIL
        if (dev == NODEV) {
                if (pwr_fail == 0)
                        logpower();
                return;
        }
#endif
        if(logging) {
                u.u_error = EBUSY;
                return;
        }
        if (flg || minor(dev) != 0) {
                u.u_error = ENXIO;
                return;
        }
        logstart();
        logging++;
}

errclose(dev,flg)
{
        logging = 0;
}

errread(dev)
{
        register err_t *eup;
        register n;
        err_t   *geterec();

        if(logging == 0)
                return;
        eup = geterec();
        n = min(eup->e_hdr.e_len, u.u_count);
        if (copyout(eup, u.u_base, n))
                u.u_error = EFAULT;
        else
                u.u_count -= n;
        freeslot(eup);
}
```

```
/*      @(#)fakemx.c    2.1     */
sdata() {}

scontrol() {}

mcstart() {}
```

```
/*      @(#)fakevtlp.c  2.1     */
vtpint() {}

vtlpopen(dev, flag) {}
```

```
/*      @(#)hf.c        2.5     */

#
/*,n'hf.c'Half duplex package includes and defines'*/
/*
 *      NOC project DS40 package, June 1976
 */

#include "sys/param.h"
#include "sys/conf.h"
#include "sys/confx.h"
#include "sys/user.h"
#include "sys/userx.h"
#include "sys/ioctl.h"
#include "sys/tty.h"
#include "sys/ttyx.h"
#include "sys/dmll.h"

#define DELTA   1               /* timing for dataset turn around */
#define PAUSE   2               /* timing for line turn around */


#define TTOHOFL 90              /* DS40 input queue shutdown level */

/*.s'hfread'Half duplex line read routine'*/
/*
 * hfread - line read for half duplex line discipline.
 *          Takes care of rawq overflow resets by re-enabling
 *          the line when the rawq falls below the low mark.
 */

hfread(atp)  struct tty *atp;  {

        register struct tty *tp;
        register int    c,cl;

        tp = atp;
        while(tp->t_state&WOPEN && ((tp->t_flags&NOSLEEP) == 0))
                sleep(tp,TTOPRI);

        /*
         * If there are characters in the canon Q, or if the
         * line is on and characters can be processed by canon,
         * give chars to user.
         */

        while(tp->t_canq.c_cc || (tp->t_state&CARR_ON && canon(tp))) {
                c = 0;  cl = -1;
                while(tp->t_canq.c_cc) {
                        if((cl=getc(&tp->t_canq)) == QESC) break;
                        if((c =passc(cl)) < 0)             break;
                }

                /*
                 * Character passing terminated because:
```

```
		 *	1. Enough chars passed to user, or;
		 *	2. Error in passing chars to user, or;
		 *	3. QESC found in canon queue, or;
		 *	4. Canon Q empty.
		 */

		/*
		 * If QESC found, it implies that the line
		 * was probably turned off at one time. Strip
		 * off the QBRK (if one is found), then check
		 * for line turnon.
		 */

		if(c1 == QESC && tp->t_canq.c_cc) {
			if((c1=getc(&tp->t_canq)) == QESC)
				c = passc(c1);
		}

		/*
		 * Now turn on line if raw queue is low enough.
		 */

		spl5();
		if((tp->t_dstat&RCV_WAIT) && c1 == QBRK) {
			tp->t_dstat =& ~RCV_WAIT;
			(*cdevsw[major(tp->t_dev)].d_mctl)(tp,'s',SUPTD);
			wakeup(&tp->t_rawq);
		}
		spl0();

		/*
		 * If we have somehow satisfied the users request for
		 * data, then return.
		 */

		if(c < 0 || (tp->t_canq.c_cc == 0 && c1 != QBRK))
			return;

	}
}				/* hfread */
/*.s'hfrint'Half duplex read interrupt handler'*/
/*
 * hfrint - Called by DH11 read interrupt service routine
 *	whenever receiver interrupts occur.
 *	Primary function is to store received characters
 *	for processing by canon.  One important side
 *	function is to turn off the half duplex line
 *	when the raw queue fill level exceeds the high
 *	water mark TTQHOFL.
 */

hfrint(achar,atp)  struct tty *atp;  {

	register struct tty *tp;

	tp = atp;
```

```
        /*
         * Put the character read into the raw q.
         */

        ttyinput(achar,tp);

        /*
         * If the raw Q has reached the high water mark and if we
         * have not already turned off the line, turn it off.
         */

        if(tp->t_rawq.c_cc >= TTQHOFL && (tp->t_dstat&RCV_WAIT) == 0) {
                (*cdevsw[major(tp->t_dev)].d_mctl)(tp,'c',SUPTD);
                tp->t_dstat =| RCV_WAIT;
                putc(QESC,&tp->t_rawq);
                putc(QBRK,&tp->t_rawq);
                if(putc(0377,&tp->t_rawq) == 0) tp->t_delct++;
                wakeup(&tp->t_rawq);
        }
}                                       /* hfrint */
/*.s'hfwrite'Half duplex line write routine'*/
/*
 * hfwrite - Logical level write routine.  Does appropriate
 *           things to line before and after write.
 */

hfwrite(atp)  struct tty *atp;  {

        ttwrite(atp);

}                                       /* hfwrite */
/*.s'hfxint'Called by DH11 xmtr interrupt handler'*/
/*
 * hfxint - Called by transmitter interrupt handler whenever
 *          a new character is needed for output.
 */

hfxint(atp, flag)  struct tty *atp;  {

        register struct tty *tp;
        register int c;
        extern hftrnrd(),ttrstrt();

        tp = atp;

        /*
         * If currently timing, don't return a character.
         */

        if(tp->t_state & TIMEOUT) return(0);

        /*
         * If there are no characters waiting on the output
         * queue or if transmit stop is on, prepare to turn
         * the line around.
```

```
 */

if(tp->t_outq.c_cc == 0 || tp->t_state&XMTSTOP) {

        /*
         * If line is already off, return; otherwise, check
         * for line turn around currently in progress.
         * If being turned on, this routine will get called
         * again so just return.  If being turned off, and if
         * the turn off is doomed to failure, then set up a
         * timeout entry so this routine is called again.
         * If line is not being turned around, then set up
         * to turn it off.
         */

        if((tp->t_dstat&RQS_ON) == 0)
                return(0);

        if(tp->t_dstat&LNTRNRD) {
                if((tp->t_dstat&(LNTRNON|NCHOUT)) == 0) {
                        timeout(ttrstrt,tp,PAUSE);
                        tp->t_state =| TIMEOUT;
                }
        } else if(flag) {
                timeout(hftrnrd,tp,PAUSE);
                tp->t_dstat =| (LNTRNRD|NCHOUT);
        }
        return(0);
}

/*
 * Turn off NCHOUT in case a data set turn around had
 * been set up.
 */

if(flag) return;
tp->t_dstat =& ~NCHOUT;

/*
 * If line is not turned on, or if line is being
 * turned around, examine more closely.
 */

if((tp->t_dstat&(XMT_ON|SEC_ON)) != (XMT_ON|SEC_ON) ||
   (tp->t_dstat&LNTRNRD)) {

        /*
         * We want the line to be on so if it is off or
         * being turned off, set up to turn it back on.
         * If it is on and being turned on set up for
         * retry.
         */

        if(tp->t_dstat&LNTRNRD) {
                if((tp->t_dstat&LNTRNON) == 0) {
                        timeout(ttrstrt,tp,PAUSE);
```

```
                                tp->t_state =| TIMEOUT;
                        }
                        return(0);
                } else {
                        if(tp->t_dstat&(REC_ON|XMT_ON)) {
                                timeout(ttrstrt,tp,PAUSE);
                                tp->t_state =| TIMEOUT;
                        } else {
                                (*cdevsw[major(tp->t_dev)].d_mctl)(tp,'s',RQSEN '
*  p);
                                tp->t_dstat =| (LNTRNRD|LNTRNON|RQS_ON);
                        }
                        return(0);
                }
        }

        /*
         * Now get a character to be transmitted.
         */

    loop:
        c = getc(&tp->t_outq);
        if(c == QESC) {
                switch(c = getc(&tp->t_outq)) {

                case QESC:
                        break;

                case QBRK:
                        c = CBREAK | 30;
                        goto out;

                case HQEND:
                        tp->t_hqcnt--;
                        goto loop;

                default:
                        if(c > 0177) {
                                c = CTOUT | (c&0177);
                                goto out;
                        }
                }
        }
        c =& 0177;
        c =| CPRES | partab[c]&0200;

    out:
        if(tp->t_outq.c_cc == TTLOWAT || tp->t_outq.c_cc == 0)
                wakeup(&tp->t_outq);
        return(c);
}                                       /* hfxint */
/*.s'hftrnrd'Turn off half duplex line'*/
/*
 * hftrnrd - Turns off request to send on a line.
 */
```

```
hftrnrd(tp)
register struct tty *tp;
{

        /*
         * If a "logical" write is still going on and transmitting
         * has not been stopped then don't turn line around.
         */

        if((tp->t_state&XMTSTOP) == 0 && tp->t_outq.c_cc) {
                tp->t_dstat =& ~LNTRNRD;
                return;
        }

        /*
         * If DH11 is busy or has been busy since this turn around
         * was set up then don't turn line around.
         */

        if(tp->t_state&(BUSY|TIMEOUT) || (tp->t_dstat&NCHOUT) == 0) {
                tp->t_dstat =& ~LNTRNRD;
                return;
        }

        /*
         * Now turn line around.
         */

        (*cdevsw[major(tp->t_dev)].d_mctl)(tp,'c',ROSEND);
        tp->t_dstat =& ~ROS_ON;
        return;
}                                       /* hftrnrd */
/*.s'hfdst'Half duplex modem control interrupt handler'*/
/*
 * hfdst - Called by DM11 modem control interrupt service.
 *         Sets appropriate state bits when line state changes.
 */

hfdst(atp,acsr,alsr)  struct tty *atp;  {

        register struct tty *tp;
        register int csr,lsr;
        extern ttrstrt();

        tp = atp;  lsr = alsr;  csr = acsr;

        /*
         * Check Carrier Transition
         */

        if(csr&CTRANS) {
                if(lsr&CARRIER) {
                        tp->t_dstat =| REC_ON;
                        tp->t_state =| CARR_ON;
                        tp->t_state =& ~WOPEN;
                        (*cdevsw[major(tp->t_dev)].d_mctl)(tp,'s',SUPTD);
```

```
                wakeup(tp);
        } else {
                tp->t_dstat =& ~(REC_ON|RCV_WAIT);
                (*cdevsw[major(tp->t_dev)].d_mctl)(tp,'c',SUPTD);
        }
}

/*
 * Check Clear to Send Transition
 */

if(csr&CSTRANS) {
        wakeup(tp);
        if(lsr&CLSEND) {
                tp->t_dstat =| XMT_ON;
                if(tp->t_dstat&LNTRNRD) {
                        tp->t_dstat =& ~(LNTRNRD|LNTRNON);
                        if((tp->t_state&TIMEOUT)==0) {
                                tp->t_state =| TIMEOUT;
                                timeout(ttrstrt,tp,0);
                        }
                }
        } else {
                tp->t_dstat  =& ~XMT_ON;
                tp->t_tmflgs =& ~TERM_BIT;
                if(tp->t_dstat&LNTRNRD) {
                        tp->t_dstat =& ~LNTRNRD;
                }
        }
}

/*
 * Check Secondary Carrier Transition
 */

if(csr&SRTRANS) {
        if(lsr&SUPRD) {
                tp->t_dstat =| SEC_ON;
                tp->t_state =| CARR_ON;
                tp->t_state =& ~WOPEN;
                wakeup(tp);
        } else {
                tp->t_dstat =& ~SEC_ON;
                if(tp->t_state&XMTSTOP) {
                        tp->t_state =& ~XMTSTOP;
                        if((tp->t_state&TIMEOUT)==0) {
                                tp->t_state =| TIMEOUT;
                                timeout(ttrstrt,tp,0);
                        }
                } else {
                        if(tp->t_dstat&ROS_ON)
                                tp->t_state =| XMTSTOP;
                }
        }
}
}                               /* hfdst */
```

```
/*.g'hfioctl'Set up line for half duplex'*/
hfioctl(dev, cmd, addr, flag)
register cmd;
caddr_t addr;
{
        register struct tty *tp;
        register int    lsr;

        switch(cmd) {
        case OLDSGTTY:
                if (addr)       /* old gtty case */
                        return;
        case TIOCSETD:
                if(flag == LSET)
                        tp->t_term = 0;
                else if(flag == LUNSET) {
                        flushtty(tp);
                        return;
                }
                spl5();
                lsr = (*cdevsw[major(tp->t_dev)].d_mctl)(tp,'t',0377);
                tp->t_dstat = 0;
                if(lsr&(CARRIER|SUPRD)) {
                        tp->t_state |= CARR_ON;
                        if(tp->t_state&WOPEN) {
                                tp->t_state &= ~WOPEN;
                        }
                        if(lsr&CARRIER) {
                                (*cdevsw[major(tp->t_dev)].d_mctl)
                                        (tp,'s',SUPTD);
                                tp->t_dstat |= REC_ON;
                        }
                        if(lsr&SUPRD)
                                tp->t_dstat |= SEC_ON;
                }
                if(lsr&CLSEND)
                        tp->t_dstat |= XMT_ON;
                if(lsr&RQSEND)
                        tp->t_dstat |= RQS_ON;

                wakeup(tp);
                spl0();
        }
}                                       /* hfioctl */


/*.t'hfclose'Shut down line'*/
hfclose(dev,tp)
register struct tty *tp;
{
        while(tp->t_state&BUSY || tp->t_dstat&(LNTRNRD|XMT_ON))
                sleep(tp,TTOPRI);
        flushtty(tp);
        tp->t_state &= ~ISOPEN;
        tp->t_pgrp  =  0;
}
```

```
/*      @(#)hp45.c      2.9.1.2 */

#

/*
 * Hewlett Packard Model HP 2645A Display Station Interface
 *
 * Modified To Handle Enhanced Video Features. 09/10/79 By J. P. Jenal
 *
 */

#include "sys/param.h"
#include "sys/user.h"
#include "sys/userx.h"
#include "sys/tty.h"
#include "sys/ttyx.h"
#include "sys/crtctl.h"


char maphp45[] {
        0341,   LCA,
        0101,   CUP,
        0102,   CDN,
        0103,   CRI,
        0104,   CLE,
        0150,   HOME,
        0344,   STB,    /* STB and SPB included to accomodate old code. */
        0344,   SPB,
        0344,   DVID,   /* Use of DVID preferred.       */
        0150,   CS,
        0114,   IL,
        0115,   DL,
        0121,   IC,
        0120,   DC,
        0113,   EEOL,
        0112,   EEOP,
        0123,   USCRL,
        0124,   DSCRL,
        0160,   HOME,
        0161,   EEOP,
        0162,   USCRL,
        0163,   DSCRL,
        0164,   DL,             /* kludge for sccs browser. should be CLE */
        0165,   IL,             /* kludge for sccs browser. should be CRI */
        0166,   CUP,
        0167,   CDN,
        0
};

char hp45str[] {' ', ESC, 'R', 010, 0};
char hp_vid_chars[6] {
        'D',    /* Underline    */
        'A',    /* Blink        */
        'B',    /* Reverse vid. */
        'H',    /* Dim video    */
```

```
        'B',    /* Change bold to Reverse vid. */
        'H'     /* Dim for Off  */
};

hp45output(ac, atp)
struct tty *atp;
{
        register struct clist *qp;
        register char *cp, *ap, vid_attr;
        register c;

        c = ac;
        qp = &atp->t_outq;

        if (c == DVSCN) {
                ttdvscn(atp);
                return;
        }
        if (c == UVSCN) {
                ttuvscn(atp);
                return;
        }

        for(cp=maphp45; *cp++;)
        if (c == *cp++) {
                putc(ESC, qp);
                if ((c=cp[-2]) < 0)
                        putc('&', qp);

                qputc(c&0177, qp);

                switch(ac) {
                        case LCA:
                                c = atp->t_row;
                                qputc(c/10 + '0', qp);
                                qputc(c%10 + '0', qp);
                                qputc('r', qp);
                                c = atp->t_col;
                                qputc(c/10 + '0', qp);
                                qputc(c%10 + '0', qp);
                                qputc('C', qp);
                                break;

                        case IC:
                                for(cp=hp45str; *cp; cp++)
                                        putc(*cp, qp);
                                break;

                        case STB:               /* Should use DVID      */
                                putc(001,qp);
                                putc(' ',qp);
                                break;

                        case SPB:               /* Should use DVID      */
                                putc(000,qp);
                                putc(' ',qp);
```

```
                        break;

                case DVID:
                        c = atp->t_dstat & 077;
                        ap = hp_vid_chars;
                        for(vid_attr='@'; c; ap++, c >>= 1)
                            if (c & 01) vid_attr |=  *ap;
                        putc(vid_attr, qp);
                        continue;

                case CS:
                        putc(ESC, qp);
                        putc(0112, qp);
                        break;
            }

            break;
        }
}

hp45input(ac, atp)
struct tty *atp;
{
        register struct tty *tp;
        register char *cp;
        register c;

        c = ac;
        tp = atp;

        if (tp->t_tmflgs&TERM_BIT) {
                tp->t_tmflgs =& ~TERM_BIT;
                for (cp=maphp45; *cp++;)
                        if (c == (cp++)[-1]) {
                                c = cp[-1];
                                if(tp->t_tmflgs&TERM_CTLECHO) {
                                        ttyctl(c, tp);
                                        if(tp->t_tmflgs&TERM_INVIS) {
                                                tp->t_state =& ~XMTSTOP;
                                                ttstart(tp);
                                        }
                                }
                                if(tp->t_tmflgs&TERM_INVIS) {
                                        return(-1);
                                }
                                return(c|CPRES);
                        }
        } else if (c == ESC) {
                tp->t_tmflgs =| TERM_BIT;
                if ((tp->t_flags&RAW) == 0)
                        tp->t_state =| XMTSTOP;

                return(-1);
        }
        return(c);
}
```

```
hp451octl(tp, flag, nvrow)
register struct tty *tp;
register unsigned nvrow;
{

        if (nvrow > 23) {
                u.u_error = EINVAL;
                return;
        }
        if (flag == LSET) {
                tp->t_lrow = 23;
                tp->t_tmflgs = ANL;
                tp->t_flags = NLDELAY|NCDELAY|EVENP|ODDP|CRMOD|
                              ECHO|XTABS|TANDEMI;
        }
}
```

```
/*      @(#)hpmap.c     2.3     */

/*
 *      Disk layout definition.
 *
 *      This diskmap is placed in lib2 of the operating system and
 *      included by:
 *              sacopy/sahp.c
 *              cmd2/errpt.c
 *      When this diskmap is modified be sure to re-build all
 *      relevant systems.  Also keep in mind the fact that
 *      the operating system configuration file (opsys/conf*)
 *      may need to be re-built.
 *
 *      One more thing - the logical file system handler has
 *      its device coded in.  It might need changing also.
 *
 *         Size  Starting Blk         Description
 */

#define NSECT   22      /* 22 sectors per track */
#define NTRACK  19
#define NCYL    NSECT*NTRACK    /* 418 sectors per cylinder */

struct {
        unsigned        nblocks;
        unsigned        cyloff;
} hp_sizes[32] {
        120*NCYL,       0,      /*  0: overlaps 8,9,10,11.      */
        120*NCYL,       120,    /*  1: overlaps 12,13,14,15.    */
        120*NCYL,       240,    /*  2: overlaps 16,17,18,19.    */
                                /*  2: musr (bmusr drive 1)     */
        120*NCYL,       360,    /*  3: overlaps 7,20,21,22,23.  */
        120*NCYL,       480,    /*  4: overlaps 24,25,26,27.    */
        120*NCYL,       600,    /*  5: overlaps 28,29,30.       */
        95*NCYL,        720,    /*  6: overlaps 31.             */
                                /*     tssrc (bsrc drive 1)     */
                        /* END RP06 */
        25*NCYL,        455,    /*  7: swapdev (drive 1)        */

        /* SECTION 0                                            */
        12*NCYL,        0,      /*  8: util (tutil drive 1)     */
        11*NCYL,        12,     /*  9: unused (emerg swap)      */
        97*NCYL,        23,     /* 10: r1 (rcmas) (br1 drive 1) */
        0,              0,      /* 11: (unused)                 */

        /* SECTION 1                                            */
        13*NCYL,        120,    /* 12: savutil (bsavutil drive 1)*/
        13*NCYL,        132,    /* 13: mrcs (rcmas)             */
                                /*     bmrcs (drive 1)          */
        20*NCYL,        145,    /* 14: man (bman drive 1)       */
        75*NCYL,        165,    /* 15: rootdev (brootdev drive 1)*/

        /* SECTION 2                                            */
        0*NCYL,         0,      /* 16: (unused)                 */
```

```
        0*NCYL,         0,      /* 17: (unused)                 */
        0*NCYL,         0,      /* 18: (unused)                 */
        0,              0,      /* 19: (unused)                 */

        /* SECTION 3                                            */
        95*NCYL,        360,    /* 20: src (tsrc drive 1)       */
        51*NCYL,        360,    /* 21: (end of RP04/5)          */
                        /* END RP04/5 */
        0*NCYL,         0,      /* 22: (unused)                 */
        0*NCYL,         0,      /* 23: (unused)                 */

        /* SECTION 4                                            */
        20*NCYL,        480,    /* 24: rhh (tmp drive 1)        */
        75*NCYL,        500,    /* 25: ses2 (troot drive 1)     */
        25*NCYL,        575,    /* 26: (unused)                 */
        0*NCYL,         0,      /* 27: (unused)                 */

        /* SECTION 5                                            */
        75*NCYL,        600,    /* 28: amacc (drive 1)          */
        45*NCYL,        675,    /* 29: (unused)                 */
        0*NCYL,         0,      /* 30: (unused)                 */
        0*NCYL,         0,      /* 31: (unused)                 */
};
```

```
/*      @(#)hpmap.tu.c  2.2     */

/*      UTILITY VERSION runs in first 12 cylinders of disk.     */

/*
 *      Disk layout definition.
 *
 *      This diskmap is placed in lib2 of the operating system and
 *      included by:
 *              sacopy/sahp.c
 *              cmd2/errpt.c
 *      When this diskmap is modified be sure to re-build all
 *      relevant systems.  Also keep in mind the fact that
 *      the operating system configuration file (opsys/conf*)
 *      may need to be re-built.
 *
 *      One more thing - the logical file system handler has
 *      its device coded in.  It might need changing also.
 *
 *         Size  Starting Blk          Description
 */

#define NSECT   22      /* 22 sectors per track */
#define NTRACK  19
#define NCYL    NSECT*NTRACK    /* 418 sectors per cylinder */

struct {
        unsigned        nblocks;
        unsigned        cyloff;
} hp_sizes[32] {

        11*NCYL,        0,      /*  0: util                     */
        1*NCYL,         11,     /*  1: swap for util            */
        55*NCYL,        12,     /*  2: rootdev                  */
        10*NCYL,        67,     /*  3: swapdev                  */
        65*NCYL,        77,     /*  4: src                      */
        15*NCYL,        142,    /*  5: man                      */
        65*NCYL,        157,    /*  6: hp6                      */
        65*NCYL,        222,    /*  7: hp7                      */
};
```

```
/*      @(#)hpmap.util.c        2.2     */

/*      UTILITY VERSION runs in first 12 cylinders of disk.     */

/*
 *      Disk layout definition.
 *
 *      This diskmap is placed in lib2 of the operating system and
 *      included by:
 *              sacopy/sahp.c
 *              cmd2/errpt.c
 *      When this diskmap is modified be sure to re-build all
 *      relevant systems.  Also keep in mind the fact that
 *      the operating system configuration file (opsys/conf*)
 *      may need to be re-built.
 *
 *      One more thing - the logical file system handler has
 *      its device coded in.  It might need changing also.
 *
 *         Size  Starting Blk         Description
 */

#define NSECT   22      /* 22 sectors per track */
#define NTRACK  19
#define NCYL    NSECT*NTRACK    /* 418 sectors per cylinder */

struct {
        unsigned        nblocks;
        unsigned        cyloff;
} hp_sizes[32] {

        11*NCYL,        0,      /*  0: util                     */
        1*NCYL,         11,     /*  1: swap for util            */
        55*NCYL,        12,     /*  2: rootdev                  */
        10*NCYL,        67,     /*  3: swapdev                  */
        65*NCYL,        77,     /*  4: src                      */
        15*NCYL,        142,    /*  5: man                      */
        65*NCYL,        157,    /*  6: hp6                      */
        65*NCYL,        222,    /*  7: hp7                      */
};
```

```
/*      @(#)hps.c       2.14.1.1        */

#
#include "sys/param.h"
#include "sys/buf.h"
#include "sys/bufx.h"
#include "sys/conf.h"
#include "sys/user.h"
#include "sys/userx.h"
#include "sys/proc.h"
#include "sys/seg.h"
#include "sys/systm.h"
#include "sys/elog.h"
#include "sys/iobuf.h"

struct device {
        int     hpcs1;  /* Control and Status register 1 */
        int     hpwc;   /* Word count register */
        int     hpba;   /* UNIBUS address register */
        int     hpda;   /* Desired address register */
        int     hpcs2;  /* Control and Status register 2*/
        int     hpds;   /* Drive Status */
        int     hper1;  /* Error register 1 */
        int     hpas;   /* Attention Summary */
        int     hpla;   /* Look ahead */
        int     hpdb;   /* Data buffer */
        int     hpmr;   /* Maintenance register */
        int     hpdt;   /* Drive type */
        int     hpsn;   /* Serial number */
        int     hpof;   /* Offset register */
        int     hpdc;   /* Desired Cylinder address register*/
        int     hpcc;   /* Current Cylinder */
        int     hper2;  /* Error register 2 */
        int     hper3;  /* Error register 3 */
        int     hppos;  /* Burst error bit position */
        int     hppat;  /* Burst error bit pattern */
        int     hpbae;  /* Bus address extension */
        int     hpcs3;  /* Control and status register 3 */
};

#ifndef HPADDR
#define HPADDR  0176700
#endif
#ifndef NHP
#define NHP     1
#endif

extern struct {
        daddr_t         nblocks;
        unsigned        cyloff;
} hp_sizes[];

struct  iobuf   hptab   tabinit(HP0,0);
qhsh_t  hpqtab[QHSHSZ]  = {
                0, &hpqtab[0], &hpqtab[0],
```

```
                0, &hpqtab[1], &hpqtab[1],
                0, &hpqtab[2], &hpqtab[2],
                0, &hpqtab[3], &hpqtab[3],
                0, &hpqtab[4], &hpqtab[4],
                0, &hpqtab[5], &hpqtab[5],
                0, &hpqtab[6], &hpqtab[6],
                0, &hpqtab[7], &hpqtab[7]
};

struct  iostat  hpstat[NHP];
struct  iobuf   hputab[8] {
                tabinit(HP0,&hpstat[0]), tabinit(HP0,&hpstat[1]),
                tabinit(HP0,&hpstat[2]), tabinit(HP0,&hpstat[3]),
                tabinit(HP0,&hpstat[4]), tabinit(HP0,&hpstat[5]),
                tabinit(HP0,&hpstat[6]), tabinit(HP0,&hpstat[7])
};
#ifdef PWR_FAIL
char    hpopenf;                /* Power fail drive open flag */
char    hp_pwron;               /* Power fail drive not online flag */
#endif

int     hpsioc[NHP];
int     hpsios[NHP];

#define NSECT   22
#define NTRACK  19

                        /* Drive Commands */
#define GO      01
#define UNLOAD  02
#define SEEK    04
#define RECAL   06
#define DCLR    010
#define RELEASE 012
#define OFFSET  014
#define RTC     016
#define PRESET  020
#define SEARCH  030
#define READ    070

#define ERR     040000  /* hpds - Error */
#define PIP     020000  /* hpds - Positioning in progress */
#define MOL     010000  /* hpds - Medium online */
#define PGM     01000   /* Programmable (Dual Port) */
#define DPR     0400    /* hpds - drive present */
#define DRY     0200    /* hpds - drive ready */
#define VV      0100    /* hpds - volume valid */
#define SC      0100000 /* hpcs1 - Special condition */
#define TRE     040000  /* hpcs1 - transfer error */
#define MCPE    020000  /* hpcs1 - massbus control parity err */
#define DVA     04000   /* hpcs1 - drive available */
#define IE      0100    /* hpcs1 - interrupt enable */
#define NED     010000  /* hpcs2 - Nonexistent drive */
#define DCK     0100000 /* hper1 - Data Check */
#define WLE     04000   /* hper1 - Write lock error */
#define ACL     0100    /* hper3 - AC power low */
```

```
#define UWR     010     /* hper3 - Unsafe - power fail */
#define FMT22   010000  /* hpof - 16 bit /word format */

#define status  b_flags
#define acts    io_s1
#define qcnt    io_s2

#define trksec  av_back
#define cylin   b_resid

#define rh70    (cputype == 70)

#define sunit(x)        ((minor(x) >> 5) & 07)
#define spart(x)        (minor(x) & 037)


hpopen(dev, flag)
{
        register unsigned unit, unitbit;
        register struct iobuf *dp;

#ifdef PWR_FAIL
        extern hp_pwrup();
        extern unsigned pwr_fail;

        if (dev == NODEV) {
                if (flag) {
                        hp_pwron = hpopenf;
                        hptab.b_active = 0;
                        hptab.b_actf = 0;
                        if (pwr_fail == NULL) {
                                HPADDR->hpcs1 = IE;
                                unitbit = 1;
                                for (unit=0; unit<NHP; unit++, unitbit =<< 1)
                                        if (hpopenf & unitbit)
                                                hpustart(unit);
                                kiltout(hp_pwrup, 0);
                                timeout(hp_pwrup, 0, 7200);  /* Two minutes */
                        }
                }
                return;
        }
#endif

        if (sunit(dev) >= NHP)
                u.u_error = ENXIO;
        dp = &hputab[sunit(dev)];
        dp->io_addr = HPADDR;
        dp->io_nreg = rh70?NDEVREG:NDEVREG-2;
        (qhsh_t *)hptab.b_forw = hpqtab;
        hptab.b_flags = B_QHASH;
}

hpstrategy(abp)
struct buf *abp;
{
```

```
        register struct buf *bp;
        register char *p1, *p2;
        struct devtab *dp;
        unsigned        unit;
        int     co;

        bp = abp;
        unit = sunit(bp->b_dev);
        dp = &hputab[unit];
        p1 = &hp_sizes[spart(bp->b_dev)];
        if (unit >= NHP) {
                printf("\nhpstrategy: unit out of range\n");
                goto bad;
        }
        if (dp->status || bp->b_blkno >= p1->nblocks) {
                if (dp->status == 0 && (bp->b_flags&B_READ))
                        bp->b_resid = bp->b_bcount;
                else {
        bad:
                        bp->b_flags |= B_ERROR;
                        bp->b_error = ENXIO;
                }
                iodone(bp);
                return;
        }
        dqlen++;
        bp->cylin = bp->b_blkno/(NSECT*NTRACK) + p1->cyloff;
        co = bp->b_blkno%(NSECT*NTRACK);
        bp->trksec =  ((co/NSECT)<<8)+(co%NSECT);
        bp->b_pri = (u.u_procp->p_nice - 1) & ~03;
        spl5();
        hpsioc[unit]++;
        if ((p1 = dp->b_actf)==0) {
                dp->b_actf = bp;
                bp->av_forw = 0;
        } else {
                for (; p2 = p1->av_forw; p1 = p2) {
                        if (p2->b_pri < bp->b_pri)
                                continue;
                        if (p2->b_pri > bp->b_pri)
                                break;
                        if (p1->cylin <= bp->cylin
                         && bp->cylin <  p2->cylin
                         || p1->cylin >= bp->cylin
                         && bp->cylin >  p2->cylin)
                                break;
                }
                bp->av_forw = p2;
                p1->av_forw = bp;
                while (p2) {
                        if (p2->b_pri != bp->b_pri)
                                p2->b_pri--;
                        p2 = p2->av_forw;
                }
        }
        if (dp->b_active==0)
```

```
                hpustart(unit);
        spl0();
}

hpustart(unit)
register unsigned unit;
{
        register struct buf *bp;
        register struct devtab *dp;
#ifdef PWR_FAIL
        extern hp_pwrup();
#endif
        long *misc;
        int     search, unitbit;
        struct device hpregs[0];

        unitbit = 1 << unit;
        dp = &hputab[unit];
        misc = &(hpstat[unit].io_misc);
        bp = 0;
        HPADDR->hpcs2.lobyte = unit;
        HPADDR->hpas = unitbit;
        dp->status = 0;
        if ((HPADDR->hpcs1&DVA)==0)
                if (HPADDR->hpcs2&NED)
                        goto abort;
                else
                        return;
        if ((HPADDR->hpds&MOL)==0)
                goto abort;

        bp = dp->b_actf;
        if (HPADDR->hpds&ERR || (HPADDR->hpcs1&(TRE|MCPE))) {
                dp->b_dev = makedev(HP0,unit);
                fmtberr(dp,(bp==NULL)?0:hp_sizes[spart(bp->b_dev)].cyloff);
#ifdef PWR_FAIL
                if (hp_pwrfail(unitbit))
                        dp->b_active = 0;
#endif
                HPADDR->hpcs1.hibyte =| TRE>>8;
                HPADDR->hpcs1.lobyte = IE|DCLR|GO;
                (*misc)++;
        }

        if (bp == 0) {
                HPADDR->hpcs1.lobyte = IE|RELEASE|GO;
                (*misc)++;
                dp->b_active = 0;
                return;
        }
        if ((HPADDR->hpds&VV) == 0) {
                HPADDR->hpcs1.lobyte = IE|PRESET|GO;
                HPADDR->hpof = FMT22;
                (*misc)++;
        }
        dp->b_active++;
```

```
#ifdef CYLHIST

/* cylinder access profiling */

        if (dp->b_active==1 && unit==dk_unit) {
                search = HPADDR->hpcc - bp->cylin;
                if(search < 0)
                        search = -search;
                dk_scyl[(search+7)>>3]++;       /* seek distance */
                search = bp->cylin;
                dk_acyl[search>>3]++;   /* absolute cylinder */
        }

#endif

        HPADDR->hpdc = bp->cylin;
        search = bp->trksec.lobyte-(HPADDR->hpla)>>6)-1;
        if (search<0) search =+ NSECT;
        if ((bp->cylin!=HPADDR->hpcc || search>6) &&
            dp->b_active<3) {
                search = bp->trksec;
                search.lobyte =- 4;
                if (search.lobyte<0) search.lobyte =+ NSECT;
                hpsios[unit]++;
                HPADDR->hpda = search;
                HPADDR->hpcs1.lobyte = IE|SEARCH|GO;
                (*misc)++;
                dk_busy =| unitbit;
        } else {

#ifdef PWR_FAIL
                hpopenf =| unitbit;             /* This drive is open */
#endif

                dp->b_forw = 0;
                if (hptab.b_actf == 0)
                        hptab.b_actf = dp; else
                        hptab.b_actl->b_forw = dp;
                hptab.b_actl = dp;
                hpstart();
        }
        return;

abort:
#ifdef PWR_FAIL
/*
 *  When performing a Power Fail Restart, disk drives will
 *  appear offline while they are recycling.  Do not
 *  generate errors.
 */
        if (hp_pwron&unitbit || hp_pwrfail(unitbit))
                return;
#endif

        if (dp->io_erec)
```

```
                logberr(dp,B_ERROR);
        dp->status++;
        while(bp = dp->b_actf) {
                bp->b_flags =| B_ERROR;
                dp->b_actf = bp->av_forw;
                dqlen--;
                iodone(bp);
        }
        dp->b_active = 0;
        printf("\nRP06/5/4 drive %d offline\n", unit);
#ifdef PWR_FAIL
        hpopenf =& ~unitbit;
#endif
}

hpstart()
{
        register struct buf *bp;
        register struct devtab *dp;
        register unsigned       unit;

        if (hptab.b_active || (dp = hptab.b_actf) == 0)
                return;
        bp = dp->b_actf;
        unit = sunit(bp->b_dev);
        HPADDR->hpcs2.lobyte = unit;
        hpstat[unit].io_ops++;
        hptab.b_active++;
        HPADDR->hpdc = bp->cylin;
        rhstart(bp, &HPADDR->hpda, bp->trksec, &HPADDR->hpbae);
        blkacty =| (1<<HP0);
        dk_busy =| (1<<unit);
        dk_numb[unit] =+ 1;
        dk_wds[unit] =+ (bp->b_bcount>>6) & 03777;
}

hpintr()
{
        register struct buf *bp;
        register struct devtab *dp;
        register cnt;
        unsigned unit;
        int i, b;
        long mask, adr, *misc, l;
        static int recal;
        struct device hpregs[0];
        int *hpb;
        int     ghperl;
        long    xgetl();

        unit = -1;
        if (hptab.b_active) {   /* data transfer underway */
                dp = hptab.b_actf;
                bp = dp->b_actf;
                unit = sunit(bp->b_dev);
                misc = &hpstat[unit].io_misc;
```

```
blkacty =& ~(1<<HP0);
dk_busy =& ~(1<<unit);
HPADDR->hpcs2.lobyte = unit;
if((HPADDR->hpcs1&(TRE|MCPE)) || (HPADDR->hpds&ERR) || recal) {
        /*
         * ERROR
         */
        ghperl = HPADDR->hperl;
        fmtberr(dp,hp_sizes[spart(bp->b_dev)].cyloff);
        if(HPADDR->hperl == DCK) {      /* correctable error */
                cnt = (HPADDR->hppos-1)/16;
                if ((mask=HPADDR->hppat) == 0) {
                        HPADDR->hpof = FMT22;
                        goto bad;
                }
                mask =<< (HPADDR->hppos-1)%16;
                if(cnt == 256)
                        mask.hiword = 0;
                cnt =<< 1;
                if(rh70)
                        adr.hiword = HPADDR->hpbae;
                else
                        adr.hiword =
                          (HPADDR->hpcs1&01400)>>8;
                adr.loword = HPADDR->hpba;
                adr -= 512;
                adr += cnt;
                l = xgetl(adr);
                l.loword ^= mask.hiword;
                l.hiword ^= mask.loword;
                xputl(adr, l);

                /*
                 * If transfer was not yet complete,
                 *      restart it.
                 */
                if(HPADDR->hpwc) {
                        i = (int)bp->trksec;
        b = ((HPADDR->hpwc + (bp->b_bcount>>1) - 1)>>8)&0377;
                        i = NSECT*(i>>8) + (i&0377) + b + 1;
                        if(i >= NSECT*NTRACK) {
                                i -= NSECT*NTRACK;
                                HPADDR->hpdc = bp->cylin + 1;
                        } else
                                HPADDR->hpdc = bp->cylin;
                        HPADDR->hpda=((i/NSECT)<<8)+(i%NSECT);
                        cnt = HPADDR->hpda;
                        HPADDR->hpcs1.lobyte = IE|DCLR|GO;
                        (*misc)++;
                        HPADDR->hpda = cnt;
                        HPADDR->hpcs1.lobyte = IE|READ|GO;
                        hpstat[unit].io_ops++;
                        return;
                } else {
                        HPADDR->hpcs1 = TRE|IE|DCLR|GO;
                        (*misc)++;
```

```
                        }
                } else {        /* Error was not correctable */
        bad:
                        /*
                         * Clear error indication
                         */
#ifdef PWR_FAIL
                        if (hp_pwrfail(1<<unit)) {
                                hptab.b_active = 0;
                                hptab.b_actf = 0;
                                hpustart(unit);
                                goto out;
                        }
#endif
                        HPADDR->hpcs1 = TRE|IE|DCLR|GO;
                        (*misc)++;
                        if (++hptab.b_errcnt > 12 ||
                          ghper1&WLE ) {        /* Give up */
                                bp->b_flags =| B_ERROR;
                        } else {        /* Retry */
                                if (recal==0 && hptab.b_errcnt==4) {
                                        HPADDR->hpcs1=RECAL|IE|GO;
                                        hptab.b_errcnt--;
                                        recal++;
                                        (*misc)++;
                                        return;
                                }
                                hptab.b_active = 0;
                                recal = 0;
                        }
                }
        }
        if (hptab.b_active) {
                if(dp->io_erec)
                        logberr(dp,bp->b_flags&B_ERROR);
                hptab.b_active = 0;
                hptab.b_errcnt = 0;
                hptab.b_actf = dp->b_forw;
                dp->b_active = 0;
                dp->b_actf = bp->av_forw;
                bp->b_resid = (-HPADDR->hpwc)*2;
                dqlen--;
                iodone(bp);
                if ((hpstat[unit].io_ops&017) == 0) {
                        HPADDR->hpcs1 = IE|RELEASE|GO;
                        (*misc)++;
                }

#ifdef PWR_FAIL
                hp_pwron =& ~(1<<unit);
#endif
                hpustart(unit);
        }
        hpstart();
}
out:
```

```
        for (cnt=0; cnt<NHP; cnt++) {
                if ((HPADDR->hpas&(1<<cnt)) && (cnt != unit))
                        hpustart(cnt);
        }
        if ((HPADDR->hpcs3&IE)==0) {
                HPADDR->hpcs1.hibyte =| TRE;
                HPADDR->hpcs3 = IE;
        }
}

hpread(dev)
{
        register nblks;

        nblks = hp_sizes[spart(dev)].nblocks;
        physio(hpstrategy, dev, B_READ, nblks);
}

hpwrite(dev)
{
        register nblks;

        nblks = hp_sizes[spart(dev)].nblocks;
        physio(hpstrategy, dev, B_WRITE, nblks);
}

#ifdef PWR_FAIL

/*
 * Since disk drives lost power, the restart sequence must
 * wait until they come back online.  If a previously
 * open drive fails to come online after a couple of
 * minutes, send a message to the system tty asking
 * for manual assistance.
 *
 * The original timeout to call the routine is setup
 *  hpopen().
 */

hp_pwrup()
{
        register unsigned unit, unitbit;

        if (hp_pwron == NULL)           /* All open drives are back */
                return;

        printf("\nRP06/5/4 Disk Drive(s)");

        unitbit = 1;
        for (unit=0; unit<NHP; unit++, unitbit =<< 1) {
                HPADDR->hpcs2.lobyte = unit;
                if (HPADDR->hpcs1&DVA && HPADDR->hpds&MOL)
                        hp_pwron =& ~unitbit;
                HPADDR->hpcs1.lobyte = IE|RELEASE|GO;
                if (hp_pwron & unitbit)
                        printf(" %d", unit);
```

```
        }

        printf(" offline.  Manual attention required.\n\n");
}

/*
 * Drive power fail test routine
 */

hp_pwrfail(unitbit)
register unitbit;
{
        if (hpopenf&unitbit && HPADDR->hper3&(ACL|UWR)) {
                if (hp_pwron == 0)
                        timeout(hp_pwrup, 0, 600);
                hp_pwron =| unitbit;
                return(1);
        }
        return(0);
}
#endif
```

```
/*      @(#)hs.c        2.6     */

#
/*
 * RS03/04 disk driver
 */

#include "sys/param.h"
#include "sys/systm.h"
#include "sys/buf.h"
#include "sys/bufx.h"
#include "sys/conf.h"
#include "sys/user.h"
#include "sys/userx.h"
#include "sys/proc.h"
#include "sys/elog.h"
#include "sys/iobuf.h"

struct device {
        int     hscs1;  /* Control and Status register 1 */
        int     hswc;   /* Word count register */
        int     hsba;   /* UNIBUS address register */
        int     hsda;   /* Desired address register */
        int     hscs2;  /* Control and Status register 2 */
        int     hsds;   /* Drive Status */
        int     hser;   /* Error register */
        int     hsas;   /* not used */
        int     hsla;   /* not used */
        int     hsdb;   /* not used */
        int     hsmr;   /* not used */
        int     hsdt;   /* not used */
        int     hsbae;  /* 11/70 bus extension */
        int     hscs3;  /* 11/70 Control and status register 3 */
};

#define NHS     1
#define HSADDR  0172040

struct  iostat  hsstat[NHS];
struct  iobuf   hstab   tabinit(HS0,&hsstat);

#ifdef PWR_FAIL
char    hs_pwron;
char    hsopenf;
#endif

#define GO      01
#define RCLR    010
#define DRY     0200    /* hsds - Drive Ready */
#define MOL     010000  /* hsds - medium online */
#define DVA     04000   /* hscs1 - drive available */
#define ERR     040000  /* hscs1 - composite error */

#define DK_N    0
```

```
hsopen(dev, flag)
{

#ifdef PWR_FAIL
        extern hs_pwrup();
        extern unsigned pwr_fail;

        if (dev == NODEV) {
                if (flag) {
                        hs_pwron = hsopenf;
                        hstab.b_active = 0;
                        if (pwr_fail == 0) {
                                hsstart();
                                kiltout(hs_pwrup, 0);
                                timeout(hs_pwrup, 0, 7200);
                        }
                }
                return;
        }
#endif

        if((minor(dev)&07) >= NHS)
                u.u_error = ENXIO;
        hstab.io_addr = HSADDR;
        hstab.io_nreg = (cputype == 70)?NDEVREG:NDEVREG-2;
}

hsstrategy(abp)
struct buf *abp;
{
        register struct buf *bp;
        register struct buf *p1, *p2;
        int mblks;

        bp = abp;
        mblks = 1024; /* RJS03 */
        if(minor(bp->b_dev) >= 8)
                mblks = 2048; /* RJS04 */
        if(bp->b_blkno >= mblks) {
                if (bp->b_flags&B_READ)
                        bp->b_resid = bp->b_bcount;
                else {
                        bp->b_flags |= B_ERROR;
                        bp->b_error = ENXIO;
                }
                iodone(bp);
                return;
        }
        bp->b_pri = u.u_procp->p_nice;
        spl5();
        if ((p1 = hstab.b_actf) == 0) {
                hstab.b_actf = bp;
                bp->av_forw = 0;
        } else {
                for (; p2 = p1->av_forw; p1 = p2)
                        if (p2->b_pri > bp->b_pri)
```

```
                                break;
                        bp->av_forw = p2;
                        p1->av_forw = bp;
                        while (p2) {
                                if (p2->b_pri > bp->b_pri)
                                        p2->b_pri--;
                                p2 = p2->av_forw;
                        }
                }
                if (hstab.b_active==0)
                        hsstart();
                spl0();
}

hsstart()
{
        register struct buf *bp;
        register addr, minr;

        if ((bp = hstab.b_actf) == 0)
                return;
        hstab.b_active++;
        addr = bp->b_blkno;
        if((minr = minor(bp->b_dev)) < 8)
                addr =<< 1; /* RJS03 */
        minr =& 07;
        hsstat[minr].io_ops++;
        blkacty =| (1<<HS0);
        HSADDR->hscs2 = minr;
#ifdef PWR_FAIL
        minr = 1<<minr;
        if(HSADDR->hscs1&DVA){
                if(HSADDR->hsds&MOL){
                        hs_pwron =& ~minr;
                        hsopenf =| minr;
                }
                else if(hs_pwron&minr){
                        hstab.b_active = 0;
                        return;
                }
        }
#endif
        rhstart(bp, &HSADDR->hsda, addr<<1, &HSADDR->hsbae);
        dk_busy =| 1<<DK_N;
        dk_numb[DK_N] =+ 1;
        dk_wds[DK_N] =+ (bp->b_bcount>>6) & 03777;
}

hsintr()
{
        register struct buf *bp;
        register unit;
        struct device hsregs[0];

        if (hstab.b_active == 0){
#ifdef PWR_FAIL
```

```
                if(hs_pwron)
                        hsstart();
                else
#endif
                        logstray(HSADDR);
                return;
        }
        bp = hstab.b_actf;
        blkacty =& ~(1<<HS0);
        dk_busy =& ~(1<<DK_N);
        hstab.b_active = 0;
        if(HSADDR->hscs1 & ERR){        /* error bit */
                unit = bp->b_dev&07;
                hstab.io_stp = &hsstat[unit];
                fmtberr(&hstab,0);
                HSADDR->hscs1 = RCLR|GO;
                hsstat[unit].io_misc++;
                if (++hstab.b_errcnt < 10) {
                        hsstart();
                        return;
                }
                bp->b_flags =| B_ERROR;
        }
        if (hstab.io_erec)
                logberr(&hstab, bp->b_flags&B_ERROR);
        hstab.b_errcnt = 0;
        hstab.b_actf = bp->av_forw;
        bp->b_resid = (-HSADDR->hswc)<<1;
        iodone(bp);
        hsstart();
}

hsread(dev)
{

        physio(hsstrategy, dev, B_READ,
                minor(dev) >= 8 ? 2048 : 1024);
}

hswrite(dev)
{

        physio(hsstrategy, dev, B_WRITE,
                minor(dev) >= 8 ? 2048 : 1024);
}



#ifdef PWR_FAIL
hs_pwrup()
{
        if(hs_pwron == 0)
                return;
        printf("\n**RS04 Disk Drive(s) offline.  Manual attention required.\n")
*   ;
}
```

```
#endif
```

```
/*      @(#)ht.c        2.7.1.1 */

#
/*
 * TU16 Tape Driver
 *
 * Handles one TM02 controller, up to 4 TU16 slave transports
 * minor device classes:
 * bits 0,1: slave select
 * bit 2 off: rewind on close; on: position after first TM
 * bit 3 off: 800 bpi; on: 1600 bpi
 */

#include "sys/param.h"
#include "sys/systm.h"
#include "sys/buf.h"
#include "sys/bufx.h"
#include "sys/conf.h"
#include "sys/file.h"
#include "sys/user.h"
#include "sys/userx.h"
#include "sys/elog.h"
#include "sys/iobuf.h"

#define NHT 1
#define HTADDR  0172440

struct device {
        int     htcs1, htwc, htba, htfc;
        int     htcs2, htds, hter, htas;
        int     htck, htdb, htmr, htdt;
        int     htsn, httc, htbae, htcs3;
};

struct  iostat  htstat[NHT];
struct  iobuf   httab   tabinit(HT0,&htstat);
struct  buf     chtbuf;

#ifdef PWR_FAIL
char    ht_pwroff;
#endif

char    h_openf[NHT];
int     h_den[NHT];
daddr_t h_blkno[NHT], h_nxrec[NHT];

#define GO      01
#define NOP     0
#define WEOF    026
#define SFORW   030
#define SREV    032
#define ERASE   024
#define REW     06
#define DCLR    010
#define P800    01300           /* 800 + pdp11 mode */
```

```
#define P1600   02300           /* 1600 + pdp11 mode */
#define IENABLE 0100
#define RDY     0200
#define TMARK   04
#define DRY     0200
#define EOT     02000
#define CS      02000
#define COR     0100000
#define PES     040
#define WRL     04000
#define MOL     010000
#define PIP     020000
#define ERR     040000
#define FCE     01000
#define TRE     040000
#define HARD    064023  /* UNS|OPI|NEF|FMT|RMR|ILR|ILF */

#define SSEEK   1
#define SIO     2
#define SABORT  3
#define SRETRY  4
#define SCOM    5
#define SOK     6
#define SERR    7
#define SBACK   8

#define rh70    (cputype = 70)

htopen(dev, flag)
{
        register unit, ds, i;

#ifdef PWR_FAIL
        extern unsigned pwr_fail;

        if (dev == NODEV) {
                if (pwr_fail)
                        return;
                for (unit=0; unit<NHT; unit++)
                        if (h_openf[unit]) {
                                h_openf[unit] = -1;
                                ht_pwroff =| (1<<unit);
                        }
                if (ht_pwroff) {
                        spl5();
                        htstart();
                        httab.b_active = 0;
                }
                return;
        }
#endif

        unit = dev&03;
        if (unit >= NHT) {
                u.u_error = ENXIO;
                return;
```

```
        }
        if (h_openf[unit]) {
                u.u_error = EBUSY;
                return;
        }
        h_openf[unit]++;
        httab.b_flags |= B_TAPE;
        httab.io_addr = HTADDR;
        httab.io_nreg = rh70?NDEVREG:NDEVREG-2;
        h_den[unit] = (dev&010 ? P1600 : P800)|unit;
        flag =& FWRITE;
        for(i = 0; i < 75; i++) {       /* max 5 min wait on rewind */
                h_blkno[unit] = 0;
                h_nxrec[unit] = -1;
                ds = hcommand(unit, NOP);
                if ((ds&MOL)==0)
                        goto error;
                if (flag && (ds&WRL)) {
                        u.u_error = EIO;
                        h_openf[unit] = 0;
                        return;
                }
                if((ds&PIP)==0)
                        return;
                sleep(&lbolt,-1);
        }
error:
        u.u_error = ENXIO;
        h_openf[unit] = 0;
}

htclose(dev, flag)
register dev;
{
        register int unit;
        register struct buf *bp;

        unit = dev&03;
        flag =& FWRITE;

#ifdef PWR_FAIL
        if (ht_pwroff & (1<<unit)) {
                ht_pwroff =& ~(1<<unit);
                goto out;
        }
#endif

        if (flag) {
                hcommand(unit, WEOF);
                hcommand(unit, WEOF);
        }
        if ((dev&04) == 0)
                hcommand(unit, REW);
        else if(flag)
                hcommand(unit, SREV);
        else
```

```
                hcommand(unit, NOP);
out:
        for(bp = httab.b_forw; bp != &httab; bp = bp->b_forw)
                if(bp->b_dev == dev)
                        bp->b_flags |= B_STALE;
        h_openf[unit] = 0;
}

hcommand(unit, com)
{
        register struct buf *bp;

        bp = &chtbuf;
        spl5();
        while(bp->b_flags&B_BUSY) {
                bp->b_flags =| B_WANTED;
                sleep(bp, PRIBIO);
        }
        spl0();
        bp->b_dev = unit;
        bp->b_resid = com;
        bp->b_blkno = 0;
        bp->b_flags = B_BUSY|B_READ;
        htstrategy(bp);
        iowait(bp);
        if(bp->b_flags&B_WANTED)
                wakeup(bp);
        bp->b_flags = 0;
        return(bp->b_resid);
}

htstrategy(abp)
struct buf *abp;
{
        register struct buf *bp;
        register char **p;

        bp = abp;
        p = &h_nxrec[bp->b_dev&03];
        if (*p < bp->b_blkno || (*p == bp->b_blkno && bp->b_flags&B_READ)) {
                if (bp->b_flags&B_READ)
                        bp->b_resid = bp->b_bcount;
                else {
                        bp->b_flags =| B_ERROR;
                        bp->b_error = ENXIO;
                }
                iodone(bp);
                return;
        }
        if ((bp->b_flags&B_READ)==0)
                *p = bp->b_blkno + 1;
        bp->av_forw = 0;
        spl5();
        if (httab.b_actf==0)
                httab.b_actf = bp;
        else
```

```
                httab.b_actl->av_forw = bp;
        httab.b_actl = bp;
        if (httab.b_active==0)
                htstart();
        spl0();
}

htstart()
{
        register struct buf *bp;
        register int unit;
        register char *blkno;

    loop:
        if ((bp = httab.b_actf) == 0)
                return;
        unit = bp->b_dev&03;
        HTADDR->htcs2 = 0;
        if((HTADDR->httc&03777)!=h_den[unit])
                HTADDR->httc = h_den[unit];
        if (bp == &chtbuf && bp->b_resid == NOP) {
                bp->b_resid = HTADDR->htds;
                goto next;
        }
        blkno = h_blkno[unit];
        if ((HTADDR->htds&MOL)==0)
                goto abort;
        if (bp == &chtbuf) {

#ifdef PWR_FAIL
                if (ht_pwroff & (1<<unit))
                        goto abort;
#endif

                httab.b_active = SCOM;
                htstat[unit].io_misc++;
                HTADDR->htfc = 0;
                HTADDR->htcs1 = bp->b_resid|IENABLE|GO;
                return;
        }
        if (h_openf[unit] < 0) {
                if(h_openf[unit] == -2) {
                        if(bp->b_flags&B_READ) {
                                bp->b_resid = bp->b_bcount;
                                goto next;
                        }
                } else
                        goto abort;
        }
        if (blkno == bp->b_blkno) {
                httab.b_active = SIO;
                htstat[unit].io_ops++;
                blkacty =| (1<<HT0);
                rhstart(bp, &HTADDR->htfc, -bp->b_bcount, &HTADDR->htbae);
        } else {
                httab.b_active = SSEEK;
```

```
                htstat[unit].io_misc++;
                if (blkno < bp->b_blkno) {
                        HTADDR->htfc = blkno - bp->b_blkno;
                        HTADDR->htcs1 = SFORW|IENABLE|GO;
                } else {
                        if(h_openf[unit] == -2)
                                httab.b_active = SBACK;
                        HTADDR->htfc = bp->b_blkno - blkno;
                        HTADDR->htcs1 = SREV|IENABLE|GO;
                }
        }
        return;
    abort:
        bp->b_flags =| B_ERROR;
    next:
        httab.b_actf = bp->av_forw;
        iodone(bp);
        goto loop;
}

htintr()
{
        register struct buf *bp;
        register int unit, state;
        struct device htregs[0];
        int     err;

        if ((bp = httab.b_actf)==0)
                return;
        blkacty =& ~(1<<HT0);
        unit = bp->b_dev&03;
        state = httab.b_active;
        httab.b_active = 0;
        if ((HTADDR->htcs1&TRE) || (HTADDR->htds&EOT)) {
                err = HTADDR->hter;
                if (HTADDR->htcs2.hibyte>0 || err&HARD)
                        state = SERR;
                if (bp->b_flags&B_HEAD) {
                        err =& ~FCE;
                }
                if ((bp->b_flags&B_READ) && (HTADDR->htds&PES))
                        err =& ~(CS|COR);
                if(HTADDR->htds&EOT) {
                        bp->b_error = ENOSPC;
                        h_openf[unit] = -1;
                }
                else if(HTADDR->htds&TMARK) {
                        HTADDR->htwc = -(bp->b_bcount>>1);
                        if((bp->b_flags&B_HEAD) == 0)
                                h_openf[unit] = (state == SBACK) ? 1 : -2;
                        if (state != SBACK)
                                state = 0;
                    }
                else if(state != SERR && err == 0)
                        state = SOK;
                else {
```

```
                httab.io_stp = &htstat[unit];
                fmtberr(&httab,0);
        }
        HTADDR->htcs1 = TRE|DCLR|GO;
        htstat[unit].io_misc++;
        h_blkno[unit]++;
        if (state==SIO && ++httab.b_errcnt < 10) {
                htstat[unit].io_misc++;
                httab.b_active = SRETRY;
                HTADDR->htfc = -1;
                HTADDR->htcs1 = SREV|IENABLE|GO;
                return;
        }
        if (state != SOK && state != SBACK) {
                if(state)
                        bp->b_flags =| B_ERROR;
                state = SABORT;
        }
} else if (HTADDR->htcs1 < 0) { /* SC */
        if (HTADDR->htds&ERR) {
                httab.io_stp = &htstat[unit];
                fmtberr(&httab,0);
                HTADDR->htcs1 = DCLR|GO;
                htstat[unit].io_misc++;
        }
}
switch(state) {
case SIO:
case SOK:
        h_blkno[unit]++;
case SABORT:
case SCOM:
        if (httab.io_erec)
                logberr(&httab, bp->b_flags&B_ERROR);
        httab.b_errcnt = 0;
        httab.b_actf = bp->av_forw;
        bp->b_resid = (-HTADDR->htwc)*2;
        iodone(bp);
        break;
case SRETRY:
        if((bp->b_flags&B_READ)==0) {
                htstat[unit].io_misc++;
                httab.b_active = SSEEK;
                HTADDR->htcs1 = ERASE|IENABLE|GO;
                return;
        }
case SSEEK:
case SBACK:
        h_blkno[unit] = bp->b_blkno;
        break;
default:
        return;
}
htstart();
}
```

```
htread(dev)
{
        htphys(dev);
        physio(htstrategy, dev, B_READ, 0);
}

htwrite(dev)
{
        htphys(dev);
        physio(htstrategy, dev, B_WRITE, 0);
}

htphys(dev)
{
        register unit, a;

        unit = dev&03;
        a = u.u_offset>>BSHIFT;
        h_blkno[unit] = a;
        h_nxrec[unit] = ++a;
}
```

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

#include "sys/param.h"
#include "sys/user.h"
#include "sys/userx.h"
#include "sys/ioctl.h"
#include "sys/tty.h"
#include "sys/inode.h"
#include "sys/file.h"
#include "sys/conf.h"
#include "sys/confx.h"

#define OTHERBITS       (HDPLX|NOHUP|XCLUDE|NOSLEEP|TANDEMI|STDTTY)

/*
 * ioctl system call
 * Check legality, execute common code, and switch out to individual
 * device routine.
 */
ioctl()
{
        register struct file *fp;
        register struct inode *ip;
        register struct a {
                int     fdes;
                int     cmd;
                caddr_t cmarg;
        } *uap;
        register dev_t dev;
        register fmt;

        uap = (struct a *)u.u_arg;
        if ((fp = getf(uap->fdes)) == NULL)
                return;
        if (uap->cmd==FIOCLEX) {
                u.u_pofile[uap->fdes] |= EXCLOSE;
                return;
        }
        if (uap->cmd==FIONCLEX) {
                u.u_pofile[uap->fdes] &= ~EXCLOSE;
                return;
        }
        ip = fp->f_inode;
        fmt = ip->i_mode & IFMT;
        if (fmt != IFCHR && fmt != IFMPC) {
                u.u_error = ENOTTY;
                return;
        }
        dev = (dev_t)ip->i_un.i_rdev;
        (*cdevsw[major(dev)].d_ioctl)(dev, uap->cmd, uap->cmarg, fp->f_flag&(FR
*   EAD|FWRITE));
}

/*
 * Common code for several tty ioctl commands
```

```
 */
ttioccomm(com, tp, addr, dev)
register struct tty *tp;
caddr_t addr;
{
        unsigned t;
        register i, j;
        struct ttiocb iocb;
        struct ttiothcb iooth;
        extern nodev();

        switch(com) {

        /*
         * get discipline number
         */
        case TIOCGETD:
                t = tp->t_line;
                if (copyout((caddr_t)&t, addr, sizeof(t)))
                        u.u_error = EFAULT;
                break;

        /*
         * set line discipline
         */
        case TIOCSETD:
                if (copyin(addr, (caddr_t)&t, sizeof(t))) {
                        u.u_error = EFAULT;
                        break;
                }
                if (t >= nldisc || *linesw[t].l_ioctl == &nodev) {
                        u.u_error = ENXIO;
                        break;
                }
                if (t == tp->t_line)
                        break;
                (*linesw[tp->t_line].l_ioctl)(com, tp, addr, LUNSET);
                (*linesw[tp->t_line=t].l_ioctl)(com, tp, addr, LSET);
                break;

        /*
         * prevent more opens on channel
         */
        case TIOCEXCL:
                tp->t_flags |= XCLUDE;
                break;
        case TIOCNXCL:
                tp->t_flags &= ~XCLUDE;
                break;

        /*
         * Set new parameters
         */
        case TIOCSETP:
                wflushtty(tp);
        case TIOCSETN:
```

```
                if (copyin(addr, (caddr_t)&iocb, sizeof(iocb))) {
                        u.u_error = EFAULT;
                        break;
                }
                if ((unsigned)iocb.ioc_ispeed > 15) {
                        u.u_error = EINVAL;
                        break;
                }
                tp->t_speeds = ((iocb.ioc_ospeed<<8)|iocb.ioc_ispeed);
                tp->t_erase = iocb.ioc_erase;
                tp->t_kill = iocb.ioc_kill;
                /*
                 * Map the mh sgflags to the cb format.
                 */
                i = tp->t_flags;
                j = iocb.ioc_flags;
                i &= (HDPLX|XCLUDE|NOSLEEP|TANDEMI|STDTTY);
                i |= j&(XTABS|LCASE|ECHO|CRMOD|RAW|ANYP);
                if ((j&HUPCL) == 0)
                        i |= NOHUP;
                if ((j&MHNLDELAY) == 0)
                        i |= NLDELAY;
                if ((j&TBDELAY) == 0)
                        i |= NTDELAY;
                if ((j&CRDELAY) == 0)
                        i |= NCDELAY;
                tp->t_flags = i;

                if (tp->t_flags&RAW && tp->t_state&XMTSTOP) {
                        tp->t_state &= ~XMTSTOP;
                        ttstart(tp);
                }
                break;

        /*
         * send current parameters to user
         */
        case TIOCGETP:
                iocb.ioc_ispeed = tp->t_speeds&0377;
                iocb.ioc_ospeed = ((tp->t_speeds>>8)&0377);
                iocb.ioc_erase = tp->t_erase;
                iocb.ioc_kill = tp->t_kill;
                /*
                 * Map the cb sgflags to the mh format
                 */
                i = tp->t_flags;
                j = i&(XTABS|LCASE|ECHO|CRMOD|RAW|ANYP);
                if ((i&NLDELAY) == 0)
                        j |= MHNLDELAY;
                if ((i&NOHUP) == 0)
                        j |= HUPCL;
                if ((i&NTDELAY) == 0)
                        j |= TBDELAY;
                if ((i&NCDELAY) == 0)
                        j |= CRDELAY;
                iocb.ioc_flags = j;
```

```
                if (copyout((caddr_t)&iocb, addr, sizeof(iocb)))
                        u.u_error = EFAULT;
                break;

        /*
         * Set new "other" bits
         */
        case TIOCSETO:
                if (copyin(addr, (caddr_t)&iooth, sizeof(iooth))) {
                        u.u_error = EFAULT;
                        break;
                }
                tp->t_flags &= ~OTHERBITS;
                tp->t_state &= ~STANDEMO;
                tp->t_flags |= iooth.ioth_flags & OTHERBITS;
                if (iooth.ioth_flags & TANDEMO)
                        tp->t_state |= STANDEMO;
                break;

        /*
         * send "other" bits to user
         */
        case TIOCGETO:
                iooth.ioth_flags = tp->t_flags & OTHERBITS;
                if (tp->t_state & STANDEMO)
                        iooth.ioth_flags |= TANDEMO;
                if (copyout((caddr_t)&iooth, addr, sizeof(iooth)))
                        u.u_error = EFAULT;
                break;

        /*
         * Hang up line on last close
         */

        case TIOCHPCL:
                tp->t_flags &= ~NOHUP;
                break;

        /*
         * toggle TTSTOP from user program
         */
        case TIOCTSTP:
                tp->t_state ^= XMTSTOP;
                ttstart(tp);
                break;

        default:
                if (((com)>>8)&0377) == 'd') {
                        /*
                         * ioctl entries to line discipline
                         */
                        (*linesw[tp->t_line].l_ioctl)(com, tp, addr, LRESET);
                        break;
                }
                return(0);
```

```
        }
        return(1);
}
```

```
/*      @(#)jy.c        2.3     */

#include "sys/param.h"
#include "sys/user.h"
#include "sys/userx.h"

#define JYS     0167760         /* DR11C status word */
#define JYR     0167764         /* DR11C input buffer */
#define JYI     040             /* DR11C initial status word */

#define JYSPRI  5               /* Joystick sleep priority */

#define LENGTH  20              /* Length of joystick queue */

#define VTNFRM  10              /* Maximum number of vt11 frames */

#define JYLASTA 9               /* Last index in joystick cursor */
#define JYX     1               /* Index of X cursor coordinate */
#define JYY     2               /* Index of Y cursor coordinate */

struct {
        char x;
        char y;
};

struct vtframp {
        int vtjump;                     /* Filled with VTDJMP's */
        int vtfradr;                    /* Address of frame contents */
};

extern struct {
        int vtstop;                     /* Point at which DPU will stop */
        int vtsync;                     /* Sync instruction */
        struct vtframp vtjyfr;          /* Special joystick frame */
        struct vtframp vtlpfr;          /* Special lp frame */
        struct vtframp vtfrm[VTNFRM];   /* Regular vt11 frames */
        struct vtframp vtloop;          /* Loop back to stop */
} vt11;

char jygo;                              /* Polling switch */

int jyc;                                /* Latest sampled coordinates */

int jypi;                               /* Input index */
int jypo;                               /* Output index */
int jyq[LENGTH];                        /* Joystick event queue */

int jyframe[] {
        0117624,0,0,0104000,024000,050000,024120,040040,0160000,0
};

/* Open joystick */
jyopen(dev)
{
        register int *p;
```

```
        int jypoll();

#ifdef PWR_FAIL
        if (dev == NODEV)
                return;
#endif

        /* If device already in use */
        if (jygo)
                u.u_error = ENXIO;

        else
        {

                /* Indicate joystick busy */
                jygo++;

                /* Insert joystick frame */
                jyframe[JYLASTA] = &vtll.vtlpfr.vtjump;
                vtll.vtjyfr.vtfradr = jyframe;

                /* Start polling coordinates */
                timeout(jypoll,0,2);

                /* Empty joystick queue */
                jypi = 0;
                jypo = 0;

                /* Enable button interrupt */
                p = JYS;
                *p = JYI;
        }
}

/* Close joystick */
jyclose()
{
        register int *p;

        /* Stop polling */
        jygo++;

        /* Remove joystick frame */
        vtll.vtjyfr.vtfradr = &vtll.vtlpfr.vtjump;

        /* Disable button interrupt */
        p = JYS;
        *p = 0;
}

/* Read event from joystick queue */
jyread()
{
        register int *ubase;
        int state[3];
```

```
        /* If odd address for result */
        if (u.u_base & 1)
                u.u_error = ENXIO;

        else
        {
                ubase = u.u_base;
                spl5();

                /* If sleep call */
                if (u.u_count != 6)
                        sleep(jyq,JYSPRI);

                else
                {

                        /* If nonempty queue */
                        if (jypo != jypi)
                        {
                                jypo = (jypo+1) % LENGTH;
                                state[0] = 1;
                                state[1] = (jyq[jypo].x & 0377)<<2;
                                state[2] = (jyq[jypo].y & 0377)<<2;
                        }

                        else
                        {
                                state[0] = 0;
                                state[1] = (jyc.x & 0377)<<2;
                                state[2] = (jyc.y & 0377)<<2;
                        }
                        spl0();

                        if (copyout(state,ubase,6) == -1)
                                u.u_error = EFAULT;
                        else
                        {
                                u.u_base =+ 6;
                                u.u_count = 0;
                        }
                }
        }
}

/* Sample joystick coordinates */
jypoll()
{
        register int *p;

        if (jygo == 1)
        {
                p = JYR;
                jyc = *p;
                jyframe[JYX] = (jyc.x&0377)<<2;
                jyframe[JYY] = (jyc.y&0377)<<2;
                timeout(jypoll,0,2);
```

```
        }
        else
                jygo = 0;
}

jyint()
{
        /* Queue event */
        jypi = (jypi+1) % LENGTH;
        if (jypi == jypo)
                jypi = (jypi+LENGTH-1) % LENGTH;
        else
                jyq[jypi] = jyc;

        wakeup(jyq);
}
```

```
/*      @(#)kl.c        2.6     */

#
/*
 */

/*
 *   KL/DL-11 driver
 */
#include "sys/param.h"
#include "sys/conf.h"
#include "sys/confx.h"
#include "sys/user.h"
#include "sys/userx.h"
#include "sys/ioctl.h"
#include "sys/tty.h"

/* base address */
#define KLADDR  0177560 /* console */
#ifndef KLBASE
#define KLBASE  0176500 /* kl and dl11-a */
#endif
#ifndef DLBASE
#define DLBASE  0175610 /* dl-e */
#endif
#ifndef NKL11
#define NKL11   1
#endif
#ifndef NDL11
#define NDL11   0
#endif
#define DSRDY   02
#define RDRENB  01

struct  tty kl11[NKL11+NDL11];
int     nkl11   NKL11+NDL11;

struct klregs {
        int klrcsr;
        int klrbuf;
        int kltcsr;
        int kltbuf;
}

klcntrl(dev, action)
{
        register struct tty *tp;
        register *addr;

        tp = &kl11[dev.d_minor];
        tp->t_dev = dev;
        /*
         * set up minor 0 to address KLADDR
         * set up minor 1 thru NKL11-1 to address from KLBASE
         * set up minor NKL11 on to address from DLBASE
```

```
         */
        addr = KLADDR;
        if(dev.d_minor)
                addr = KLBASE + 8*(minor(dev)-1);
        if(dev.d_minor >= NKL11)
                addr = DLBASE + 8*(minor(dev)-NKL11);
        tp->t_addr = addr;
        addr->klrcsr =| IENABLE|DSRDY|RDRENB;
        addr->kltcsr =| IENABLE;
        tp->t_state =| CARR_ON;
}

klopen(dev, flag)
{
        register char *addr;
        register struct tty *tp;
        extern klxint();

#ifdef PWR_FAIL
        extern unsigned pwr_fail;

        if (dev == NODEV) {
                if (pwr_fail == NULL)
                        pwr_init(kl11, NKL11+NDL11, klxint);
                return;
        }
#endif

        if (dev.d_minor >= NKL11+NDL11) {
                u.u_error = ENXIO;
                return;
        }
        klcntrl(dev, 1);
        tp = &kl11[dev.d_minor];
        if ((tp->t_state&ISOPEN) == 0) {
                if ((tp->t_state&EVEROPEN) == 0) {
                        tp->t_flags = ODDP|EVENP|CRMOD|
                                        ECHO|XTABS|NCDELAY|NLDELAY;
                        tp->t_speeds = ((B300<<8)|B300);
                }
        }
        (*linesw[tp->t_ltype].l_open)(tp);
}

klclose(dev)
{
        register struct tty *tp;

        tp = &kl11[dev.d_minor];
        (*linesw[tp->t_ltype].l_close)(dev, tp);
}

klread(dev)
{
        register struct tty *tp;
```

```
        tp = &kl11[dev.d_minor];
        (*linesw[tp->t_ltype].l_read)(tp);
}

klwrite(dev)
{
        register struct tty *tp;

        tp = &kl11[dev.d_minor];
        (*linesw[tp->t_ltype].l_write)(tp);
}

klxint(dev)
{
        register struct tty *tp;

        tp = &kl11[dev.d_minor];
        ttstart(tp);
        if (tp->t_outq.c_cc == 0 || tp->t_outq.c_cc == TTLOWAT)
                wakeup(&tp->t_outq);
}

klrint(dev)
{
        register int c, *addr;
        register struct tty *tp;

        tp = &kl11[dev.d_minor];
        addr = tp->t_addr;
        c = addr->klrbuf;
        addr->klrcsr =| RDRENB;
        if ((c&0177)==0)
                addr->kltbuf = c;       /* hardware botch */
        (*linesw[tp->t_ltype].l_rcvd)(c, tp);
}

klioctl(dev, cmd, addr, flag)
caddr_t addr;
{
        register struct tty *tp;

        tp = &kl11[dev.d_minor];
        if (cmd == OLDSGTTY) {
                lsgtty(addr, tp);
        } else if (ttioccomm(cmd, tp, addr, dev) == 0)
                u.u_error = ENOTTY;
}
```