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

#
/*.n'rop.c'ROP line discipline'*/

/*
 *      ROP Line Discipline uses:
 *          l_read      = nodev.
 *          l_rcvd      = nodev.
 *          l_write     = ttwrite.
 *          l_xmtd      = hfxint.
 *          l_ioctl     = hfioctl.
 *          l_dst       = rodst.
 *          l_open .    = roopen.
 *          l_close     = ttyclose.
 */

#include "sys/param.h"
#include "sys/conf.h"
#include "sys/confx.h"
#include "sys/user.h"
#include "sys/userx.h"
#include "sys/tty.h"
#include "sys/ttyx.h"
#include "sys/dm11.h"
/*.s'roopen'Open line for ROP use.'*/
roopen(tp)
register struct tty *tp;
{

        (*cdevsw[major(tp->t_dev)].d_mctl)(tp, 's', ROSEND);
        ttyopen(tp);
}
/*.s'rodst'Half duplex modem control interrupt handler'*/
/*
 * rodst - Called by DM11 modem control interrupt service.
 *         Sets appropriate state bits when line state changes.
 */

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

        /*
         * 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;
                tp->t_state  =& ~TIMEOUT;
                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);
                ttrstrt(tp);
        } else if(tp->t_dstat&XMT_ON) {
                tp->t_dstat =& ~SEC_ON;
                tp->t_state =| TIMEOUT;
        }
}
}
```

```
/*      @(#)rp.c        2.5.1.1 */

#
/*
 * RP 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     rpds;
        int     rper;
        int     rpcs;
        int     rpwc;
        int     rpba;
        int     rpca;
        int     rpda;
};

#define RPADDR  0176710
#define NRP     1

struct {
        char    *nblocks;
        int     cyloff;
} rp_sizes[] {
        6000,   0,              /*  0: doc, 30 cyl.             */
        10400,  30,             /*  1: musr, 52 cyl.            */
        30000,  82,             /*  2: nems, 150 cyl.           */
        2000,   232,            /*  3: /dev/rp3, 10 cyl.                */
        9800,   242,            /*  4: rp4, 49 cyl.             */
        22990,  291,            /*  5: mtpt, 114.9 cyl.         */
};

struct  iostat  rpstat[NRP];
struct  iobuf   rptab   tabinit(RP0,&rpstat);

#define GO      01
#define RESET   0
#define HSEEK   014

#define IENABLE 0100
#define READY   0200

#define SUFU    01000
#define SUSU    02000
```

```
#define SUSI    04000
#define HNF     010000

#define SUWP    0400

/*
 * Use av_back to save track+sector,
 * b_resid for cylinder.
 */

#define trksec  av_back
#define cylin   b_resid

/*
 * Monitoring device number
 */
#define DK_N    1

rpopen(dev, flag)
{

#ifdef PWR_FAIL
        extern unsigned pwr_fail;

        if (dev == NODEV) {
                if (flag) {
                        rptab.b_active = 0;
                        if (pwr_fail == NULL)
                                rpstart();
                }
                return;
        }
#endif

        if (dev.d_minor >= (NRP<<3))
                u.u_error = ENXIO;
        rptab.io_addr = RPADDR;
        rptab.io_nreg = NDEVREG;
}

rpstrategy(bp)
register struct buf *bp;
{
        register char *p1, *p2;

        p1 = &rp_sizes[bp->b_dev.d_minor&07];
        if(bp->b_blkno >= p1->nblocks) {
                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->cylin = p1->cyloff;
        p1 = bp->b_blkno;
        p2 = lrem(p1, 10);
        p1 = ldiv(p1, 10);
        bp->trksec = (p1%20)<<8 | p2;
        bp->cylin =+ p1/20;
        bp->b_pri = u.u_procp->p_nice;
        spl5();
        if ((p1 = rptab.b_actf)==0) {
                rptab.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 (rptab.b_active==0)
                rpstart();
        spl0();
}

rpstart()
{
        register struct buf *bp;

        if ((bp = rptab.b_actf) == 0)
                return;
        rptab.b_active++;
        rptab.io_stp = &rpstat[minor(bp->b_dev)>>3];
        rptab.io_stp->io_ops++;
        if ((bp->b_flags&B_MAP) == 0)
                mapalloc(bp);
        RPADDR->rpda = bp->trksec;
        devstart(bp, &RPADDR->rpca, bp->cylin, bp->b_dev.d_minor>>3);
        dk_busy =| 1<<DK_N;
        dk_numb[DK_N] =+ 1;
        dk_wds[DK_N] =+ (bp->b_bcount>>6) & 03777;
}

rpintr()
{
```

```
        register struct buf *bp;
        register int ctr;
        register status;

        if (rptab.b_active == 0)
                return;
        dk_busy =& ~(1<<DK_N);
        bp = rptab.b_actf;
        rptab.b_active = 0;
        if (RPADDR->rpcs < 0) {         /* error bit */
                fmtberr(&rptab,rp_sizes[minor(bp->b_dev)&07].cyloff);
                status = RPADDR->rpds;
                if(RPADDR->rpds & (SUFU|SUSI|HNF)) {
                        RPADDR->rpcs.lobyte = HSEEK|GO;
                        ctr = 0;
                        while ((RPADDR->rpds&SUSU) && --ctr);
                }
                RPADDR->rpcs = RESET|GO;
                ctr = 0;
                while ((RPADDR->rpcs&READY) == 0 && --ctr);
                if (++rptab.b_errcnt <= 10 && (status&SUWP) == 0) {
                        rpstart();
                        return;
                }
                bp->b_flags =| B_ERROR;
        }
        if (rptab.io_erec)
                logberr(&rptab,bp->b_flags&B_ERROR);
        rptab.b_errcnt = 0;
        rptab.b_actf = bp->av_forw;
        bp->b_resid = (-RPADDR->rpwc)<<1;
        iodone(bp);
        rpstart();
}

rpread(dev)
{
        register nblks;

        nblks = rp_sizes[dev.d_minor & 07].nblocks;
        physio(rpstrategy, dev, B_READ, nblks);
}

rpwrite(dev)
{
        register nblks;

        nblks = rp_sizes[dev.d_minor & 07].nblocks;
        physio(rpstrategy, dev, B_WRITE, nblks);
}
```

```
/*      @(#)rx.c        2.5.1.1 */
#
/*
 *      RX01/RX02 Floppy disk driver by E. G. Bradford
 */

#include "sys/param.h"
#include "sys/buf.h"
#include "sys/user.h"
#include "sys/userx.h"
#include "sys/ioctl.h"
#include "sys/elog.h"
#include "sys/iobuf.h"
#include "sys/rx.h"

#define YES             1
#define NO              0

/*
 *      rename some buffer members
 */
#define b_rxcmd         b_error


/*
 *      RX01/RX02 REGISTERS
 */

#define RX2TA           ((char *)rx2cs+2)
#define RX2SA           ((char *)rx2cs+2)
#define RX2WC           ((char *)rx2cs+2)
#define RX2BA           ((char *)rx2cs+2)
#define RX2DB           ((char *)rx2cs+2)
#define RX2ES           ((char *)rx2cs+2)


/*
 *      RX01/RX02 COMMANDS
 */

#define RX2_CMD         016
#define RX2_FILL                000
#define RX2_EMPTY               002
#define RX2_WRITE               004
#define RX2_READ                006
#define RX2_SETDEN              010
#define RX2_RDST                012
#define RX2_DELSEC              014
#define RX2_RDERR               016


/*
 *      RX01/RX02 CONTROL STATUS REGISTER BITS
 */

#define RX2_ER          0100000
```

```
#define RX2_INIT        0040000
#define RX2_ADR         0030000
#define RX2_RX02        0004000
#define RX2_DEN         0000400
#define         RX2_SINGLE      0000000
#define         RX2_DOUBLE      0000400
#define RX2_TR          0000200
#define RX2_IE          0000100
#define RX2_DONE        0000040
#define RX2_U0          0000000
#define RX2_U1          0000020
#define RX2_GO          0000001



/*
 *      RX01/RX02 ERROR BITS
 */
#define RX2_ERDEN       0020            /* RX2ES drive density error bit */
#define RX2_DRDEN       0040            /* RX2ES bit for drive density */
#define RX2_ONLINE      0200            /* RX2ES drive ready bit */

struct
{
        int integ;
};


#define RX2_BC1         128             /* word count of single den flop */
#define RX2_BC2         (2*RX2_BC1)     /* word count of double den flop */

#define NRXBLKS         2002

/*
 *      # of unix blks on single den flop
*/
#define NRX1BLKS        (NRXBLKS/(BSIZE/RX2_BC1)+1)

/*
 *      # of unix blks on double den flop
*/
#define NRX2BLKS        (NRXBLKS/(BSIZE/RX2_BC2))

#define RXLOPRI()       spl0()
#define RXHIPRI()       spl5()

/*
 *      To add more controllers, just add more addresses to this list.
 */
int *rxreg[] =
{
        0177170,                        /* vector address = 0264 */
        0177150,                        /* vector address = 0270 */
};

/*
```

```
 *      total number of drives
*/
#define NRXDEV  (2*sizeof(rxreg)/sizeof(rxreg[0]))

static int rx2state;                    /* current software copy of RX2CS */
static int rx2_density[NRXDEV];         /* set in rxopen                  */
static char rx2_buf[RX2_BC2];

#define switden(a)      (rx2_density[minor((a))] = (rx2_density[minor((a))] \
                        == RX2_DOUBLE) ? RX2_SINGLE : RX2_DOUBLE)
#define setden(a,b)     rx2_density[minor( (a) )] = (b)

/*
 *      Make some easy to remember defines.
 */
struct  iobuf   rxtab;

#define         OFFSET          (bp->b_bcount-bp->b_resid)



#define MAXERRS         8               /* MAX # OF CONSECUTIVE ERRORS */


rxstrategy(bp)
register struct buf *bp;
{
        register int density;
        register int nrxblks;

        if(bp->b_rxcmd != RX2_RDST && bp->b_rxcmd != RX2_SETDEN)
        {
                density = rx2_density[minor(bp->b_dev)];
                nrxblks = (density == RX2_SINGLE) ? NRX1BLKS : NRX2BLKS ;
                if(bp->b_blkno == nrxblks-1 && density == RX2_SINGLE)
                        bp->b_bcount = BSIZE/2;
                bp->b_resid = bp->b_bcount;

                if (bp->b_blkno >= nrxblks)
                {
                        if((bp->b_flags&B_READ) == 0)
                        {
err:
                                bp->b_flags |= B_ERROR;
                                bp->b_error = ENXIO;
                        }
                        else
                                bp->b_error = 0;
done:
                        iodone(bp);
                        return;
                }
                bp->b_rxcmd = ((bp->b_flags&B_READ)==0) ? RX2_FILL : RX2_READ ;
        }
```

```
        else
        {
                if(!(rxreg[minor(bp->b_dev)/2]->integ&RX2_RX02))
                {
                        if(bp->b_rxcmd == RX2_SETDEN)
                                goto err;
                        if(bp->b_rxcmd == RX2_RDST)
                                goto done;
                }
        }

        bp->av_forw = 0;
        RXHIPRI();
        if(rxtab.b_actf == 0)
                rxtab.b_actf = bp;
        else
                rxtab.b_actl->av_forw = bp;
        rxtab.b_actl = bp;
        if(rxtab.b_active == NO)
                rxstart();
        RXLOPRI();
}

rxstart()
{
        register struct buf *bp;
        register char *rxbptr;
        register unsigned bc;
        int sector, track;
        int *rx2cs;
        int rx02;

topstart:
        if((bp=rxtab.b_actf) == 0)
        {
                rxtab.b_active = NO;
                return;
        }


        rxtab.b_active  = YES;
        rx2cs           = rxreg[minor(bp->b_dev)/2];
        rx02            = (rx2cs->integ&RX2_RX02) ? 1 : 0;      /* rx02? */
        rx2state        = bp->b_rxcmd|RX2_GO|RX2_IE;            /* cmd   */
        rx2state        |= ( (minor(bp->b_dev)&01 ) << 4);      /* unit */
        rx2state        |= rx2_density[minor(bp->b_dev)];       /* den   */
        if(!rx02 && (bp->b_rxcmd == RX2_FILL || bp->b_rxcmd == RX2_EMPTY))
                        rx2state &= (~RX2_IE);
        bc              = (rx2state&RX2_DOUBLE) ? RX2_BC2 : RX2_BC1 ;
        bc              = min(bc, bp->b_resid);

        rx2cs->integ    = rx2state;
        switch(bp->b_rxcmd)
        {
        case RX2_WRITE:
        case RX2_READ:
```

```
                rx2factr( ((BSIZE/bc)*bp->b_blkno + (OFFSET/bc)),
                        &sector, &track);
                rx02wait();
                RX2DB->integ = sector;
                rx02wait();
                RX2DB->integ = track;
                break;

        case RX2_FILL:
                copyio(bp->b_paddr+OFFSET, rx2_buf, bc, U_RKD);
                if(!rx02)
                {
                        rxbptr = rx2_buf;
                        RXLOPRI();
                        while((rx2cs->integ&RX2_DONE) == 0)
                        {
                                while(rx2cs->integ&RX2_TR)
                                        RX2DB->lobyte = *rxbptr++;
                        }
                        RXHIPRI();
                        bp->b_rxcmd = RX2_WRITE;
                        goto topstart;
                }

        case RX2_EMPTY:
                if(rx02)
                {
                        rx02wait();
                        RX2DB->integ = (bc+1)>>1;
                        rx02wait();
                        RX2DB->integ = rx2_buf;
                }
                else
                {
                        rxbptr = rx2_buf;
                        RXLOPRI();
                        while((rx2cs->integ&RX2_DONE) == 0)
                        {
                                while(rx2cs->integ&RX2_TR)
                                        *rxbptr++ = RX2DB->lobyte;
                        }
                        copyio(bp->b_paddr+OFFSET, rx2_buf, bc, U_WKD);
                        RXHIPRI();
                        bp->b_resid -= bc;
                        if(bp->b_resid == 0)
                                rxdone();
                        else
                                bp->b_rxcmd = RX2_READ;
                        goto topstart;
                }
                break;

        case RX2_RDST:
                break;

        case RX2_SETDEN:
```

```
                rx02wait();
                RX2DB->integ = 0111;            /* 'I' */
                break;

        }
}

rxintr()
{
        register unsigned bc;
        register struct buf *bp;
        int sector, track;
        int *rx2cs;

        if(rxtab.b_active == NO)
                return;

        bp              = rxtab.b_actf;
        rx2cs           = rxreg[minor(bp->b_dev)/2];

        if(rx2cs->integ < 0)
        {
                rxtab.b_errcnt++;
                rxdone();
                goto out;
        }

        rxtab.b_errcnt = 0;
        bc      = (rx2state&RX2_DOUBLE) ? RX2_BC2 : RX2_BC1;
        bc      = min(bc, bp->b_resid);
        switch(bp->b_rxcmd)
        {
        case RX2_READ:
                bp->b_rxcmd = RX2_EMPTY;
                break;

        case RX2_EMPTY:
                copyio(bp->b_paddr+OFFSET, rx2_buf, bc, U_WKD);
        case RX2_WRITE:
                bp->b_resid -= bc;
                if(bp->b_resid == 0)
                {
                        rxdone();
                }
                else if(bp->b_rxcmd == RX2_WRITE)
                        bp->b_rxcmd = RX2_FILL;
                else
                        bp->b_rxcmd = RX2_READ;
                break;

        case RX2_FILL:
                bp->b_rxcmd = RX2_WRITE;
                break;

        case RX2_SETDEN:
        case RX2_RDST:
```

```
                rxdone();
                break;
        }
out:
        rxstart();
}

rxdone()
{
        register struct buf *bp;
        register int *rx2cs;
        register int db;

        bp = rxtab.b_actf;
        rxtab.b_active = NO;
        rx2cs = rxreg[minor(bp->b_dev)/2];

        if(bp->b_rxcmd == RX2_RDST)
        {
                db = RX2DB->integ;
                if((db&RX2_ONLINE) == 0)
                        rxtab.b_errcnt = MAXERRS+1;
                else if((db&RX2_ERDEN) == 0 && rxtab.b_errcnt == 0)
                {
                        bp->b_flags &= (~B_ERROR);
                        setden(bp->b_dev,
                                ((db&RX2_DRDEN)?RX2_DOUBLE:RX2_SINGLE));
                }
                else if(bp->b_flags&B_ERROR)
                        rxtab.b_errcnt = MAXERRS+1;
                else
                {
                        bp->b_flags |= B_ERROR;
                        switden(bp->b_dev);
                        return;
                }
        }
        if(rxtab.b_errcnt)
                if(rxtab.b_errcnt < MAXERRS)
                        return;

        if(rxtab.b_errcnt)
        {
                bp->b_flags |= B_ERROR;
                bp->b_error = ENXIO;
        }
        else
                bp->b_error = 0;
        rxtab.b_errcnt = 0;
        rxtab.b_actf = bp->av_forw;
        iodone(bp);
}

/*
 *      rx2factr -- calculates the physical sector and physical
 *      track on the disk for a given logical sector.
```

```
 *      call:
 *              rx2factr(logical_sector,&p_sector,&p_track);
 *      the logical sector number (0 - 2001) is converted
 *      to a physical sector number (1 - 26) and a physicial
 *      track number (0 - 76).
 *      the logical sectors specify physical sectors that
 *      are interleaved with a factor of 3; thus the sectors
 *      are read in the following order for increasing logical
 *      sector numbers (1,4, ... 22,25,2,5, ... 23,26,3,6, ... 18,21,24).
 *      Logical sectors start at track 1, sector 1; go to
 *      track 76, sector 26; and then to track 0, sector 1 thru
 *      track 0, sector 26.  Thus, for example, unix block number
 *      498 starts at track 0, sector 8 and runs thru track 0, sector 26.
 */
rx2factr(sectr, psectr, ptrck)
register int sectr;
int *psectr, *ptrck;
{
        register int i;

        *psectr = (sectr * 3) % 26 + 1;
        i = sectr / 26 + 1;
        if (i == 77)
                i = 0;
        *ptrck = i;
}

rx02wait()
{
        int i;
        int *rx2cs;

        i = 0;
        rx2cs = rxreg[minor(rxtab.b_actf->b_dev)/2];
        while( (rx2cs->integ&RX2_TR) == 0)
                if(i++ > 10)
                        return;
}

rxopen(dev)
{
        register struct buf *bp;

        if(minor(dev) >= NRXDEV)
        {
                u.u_error = ENXIO;
                return;
        }
        bp = getbfh();
        bp->b_flags |= B_BUSY;
        bp->b_rxcmd = RX2_RDST;
        bp->b_dev = dev;
        rxstrategy(bp);
        iowait(bp);
        hrelse(bp);
}
```

```
rxread(dev)
{
        register int nblks;

        nblks = (rx2_density[minor(dev)]==RX2_SINGLE) ? NRX1BLKS : NRX2BLKS;
        physio(rxstrategy, dev, B_READ, nblks);
}

rxwrite(dev)
{
        register int nblks;

        nblks = (rx2_density[minor(dev)]==RX2_SINGLE) ? NRX1BLKS : NRX2BLKS;
        physio(rxstrategy, dev, B_WRITE, nblks);
}
rxioctl(dev, cmd, addr, flag)
caddr_t addr;
{
        register int den;
        register struct buf *bp;

        if(cmd == RIOCGETD)
        {
                suword( addr, (rx2_density[minor(dev)]==RX2_SINGLE)?1:2 );
        }
        else if(cmd == RIOCSETD)
        {
                if((den=fuword(addr)) != 1 && den != 2)
                {
                        u.u_error = EINVAL;
                        return;
                }
                bp = getbfh();
                bp->b_flags |= B_BUSY;
                bp->b_rxcmd             = RX2_SETDEN;
                bp->b_dev               = dev;
                rx2_density[minor(dev)] = (den==1)?RX2_SINGLE:RX2_DOUBLE;
                rxstrategy(bp);
                iowait(bp);
                hrelse(bp);
        }
        else
                u.u_error = EINVAL;
}
```

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

#
/*
 */

/*
 *      indirect driver for controlling tty.
 */
#include "sys/param.h"
#include "sys/conf.h"
#include "sys/confx.h"
#include "sys/user.h"
#include "sys/userx.h"
#include "sys/tty.h"
#include "sys/proc.h"
#include "sys/procx.h"

syopen(dev, flag)
{
#ifdef PWR_FAIL
        if (dev == NODEV)
                return;
#endif

        if(u.u_ttyp == NULL) {
                u.u_error = ENXIO;
                return;
        }
        (*cdevsw[u.u_ttyd.d_major].d_open)(u.u_ttyd, flag);
}

syread(dev)
{

        (*cdevsw[u.u_ttyd.d_major].d_read)(u.u_ttyd);
}

sywrite(dev)
{

        (*cdevsw[u.u_ttyd.d_major].d_write)(u.u_ttyd);
}

syioctl(dev, com, addr, flag)
caddr_t addr;
{

        (*cdevsw[u.u_ttyd.d_major].d_ioctl)(u.u_ttyd, com, addr, flag);
}
```

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

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

char maptec[] {
        0154,   LCA,
        0170,   CUP,
        0150,   CDN,
        0147,   CRI,
        0167,   CLE,
        0151,   HOME,
        0173,   STB,
        0174,   SPB,
        0146,   CS,
        0166,   CM,
        0145,   IL,
        0165,   DL,
        0144,   IC,
        0164,   DC,
        0143,   EEOL,
        0163,   EEOP,
        0175,   KBL,
        0160,   KBU,
        0200 | DL,      UVSCN,
        0200 | IL,      DVSCN,
        0
};

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

        c = ac;
        qp = &atp->t_outq;
        for(cp = maptec; *cp++;)
        if(c == *cp++) {
                if( c == UVSCN ) {
                        ttuvscn(atp);
                        break;
                }
                if( c == DVSCN ) {
                        ttdvscn(atp);
                        break;
                }
                qputc(cp[-2], qp);
                if(c == LCA) {
                        qputc(~atp->t_col, qp);
```

```
                        qputc(~atp->t_row, qp);
                }
                break;
        }
}

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


        c = ac;
        for(cp=maptec; *cp++;)
        if(c == (cp++)[-1])
                return(cp[-1] | 0100000);
        return(c);

}

tecioctl(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 = LCF|ANL;
                tp->t_flags = NLDELAY|EVENP|ODDP|
                    CRMOD|ECHO|LCASE|XTABS|NCDELAY;
        }
}
```

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

#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 maptex[] {
        0234,   LCA,
        0127,   CUP,
        0136,   CDN,
        0126,   CRI,
        0210,   CLE,
        0137,   CLE,
        0234,   HOME,
        0120,   STB,
        0100,   SPB,
        0014,   CS,
        0014,   CM,
        0131,   IL,
        0134,   DL,
        0130,   IC,
        0133,   DC,
        0132,   EEOL,
        0132,   EEOP,
/*      0xxx,   KBL,    */
/*      0xxx,   KBU,    */
        IL,     DVSCN,
        DL,     UVSCN,
        0
};

#define US 037

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

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

        for(cp=maptex; *cp++;)
        if (c == *cp++) {
                if (c == DVSCN ) {
                        ttdvscn(atp);
                        break;
                }
                if (c == UVSCN ) {
                        ttuvscn(atp);
                        break;
```

```
                }
                if ((c=cp[-21]) > 0) {
                        putc(ESC, qp);
                        if (ac == STB || ac == SPB) {
                                putc(0126,qp);
                                putc(US, qp);
                        }
                }
                qputc(c&0177, qp);
                switch (ac) {
                        case EEOP:
                                for(c=atp->t_row+1; c<24; c++) {
                                        putc(QESC, qp);
                                        putc(0203, qp);
                                        qputc(034, qp);
                                        qputc(32, qp);
                                        qputc(c+32, qp);
                                        putc(QESC, qp);
                                        putc(0201, qp);
                                        putc(ESC, qp);
                                        putc(0132, qp);
                                }

                        case EEOL:
                                putc(QESC, qp);
                                putc(0203, qp);
                                putc(034, qp);

                        case LCA:
                        case HOME:
                                qputc(atp->t_col+32, qp);
                                qputc(atp->t_row+32, qp);

                        case IL:
                        case CS:
                        case DL:
                                putc(QESC, qp);
                                putc(0201, qp);
                                break;
                }
                break;
        }
}

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

        c = ac;
        tp = atp;
        if (tp->t_tmflgs&TERM_BIT) {
                tp->t_tmflgs =& ~TERM_BIT;
                for (cp=maptex; *cp++;)
```

```
                        if (c == (cp++)[-1])
                                return(cp[-1] | CPRES);

        } else if (c == ESC) {
                tp->t_tmflgs =| TERM_BIT;
                if ((tp->t_flags&RAW) == 0)
                        tp->t_state =| XMTSTOP;


                return(-1);
        }

        return(c);
}

texioctl(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|EVENP|ODDP|CRMOD|
                        ECHO|XTABS|NCDELAY;
        }
}
```

```
/*      @(#)tm.c        2.7.1.2 */

#
/*
 * TM tape driver
 * Conventions:
 *      minor devices 0-3 rewind
 *      minor devices 4-7 no rewind
 */

#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"

struct device {
        int tmer;
        int tmcs;
        int tmbc;
        int tmba;
        int tmdb;
        int tmrd;
};

#define NTM     2
#define TMADDR  0172520

struct  buf     ctmbuf;

struct  iostat  tmstat[NTM];
struct  iobuf   tmtab   tabinit(TM0,&tmstat);

#ifdef PWR_FAIL
char    tm_pwroff;
#endif

char    t_openf[NTM];
char    *t_blkno[NTM];
char    *t_nxrec[NTM];

#define GO      01
#define RCOM    02
#define WCOM    04
#define WEOF    06
#define SFORW   010
#define SREV    012
#define WIRG    014
#define REW     016
#define DENS    060000          /* 9-channel */
```

```
#define IENABLE 0100
#define CRDY    0200
#define TUR     1
#define WRL     4
#define SELR    0100
#define GSD     010000
#define HARD    0102200 /* ILC, EOT, NXM */
#define EOF     0040000

#define NOP     0
#define SSEEK   01
#define SIO     02
#define SBACK   04
#define TCMD    010
#define SBAD    020

tmopen(dev, flag)
{
        register int unit, ds, i;

#ifdef PWR_FAIL
        extern unsigned pwr_fail;

        if (dev == NODEV) {
                if (pwr_fail)
                        return;
                for (unit=0; unit<NTM; unit++)
                        if (t_openf[unit]) {
                                tm_pwroff =| (1<<unit);
                                t_openf[unit] = -1;
                        }

                if (tm_pwroff) {
                        spl5();
                        tmstart();
                        tmtab.b_active = 0;
                }
                return;
        }
#endif

        unit = dev.d_minor&03;
        if (unit >= NTM) {
                u.u_error = ENXIO;
                return;
        }
        if (t_openf[unit]) {
                u.u_error = EBUSY;
                return;
        }
        tmtab.b_flags |= B_TAPE;
        tmtab.io_addr = TMADDR;
        tmtab.io_nreg = NDEVREG;
        t_openf[unit]++;
        flag =& FWRITE;
        for(i = 0; i < 75; i++) {       /* only wait 5 min for TUR */
```

```
                t_blkno[unit] = 0;
                t_nxrec[unit] = -1;
                ds = tcommand(unit, NOP);
                if ((ds&SELR) == 0)
                        goto error;
                if (flag && (ds&WRL) && (ds&TUR)) {
                        u.u_error = EIO;
                        t_openf[unit] = 0;
                        return;
                }
                if(ds&TUR)
                        return;
                sleep(&lbolt, -1);
        }
error:
        u.u_error = ENXIO;
        t_openf[unit] = 0;
}

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

        unit = dev&03;
        flag =& FWRITE;

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

        if (flag){
                tcommand(unit, WEOF);
                tcommand(unit, WEOF);
        }
        if ((dev&04) == 0)
                tcommand(unit, REW);
        else if(flag)
                tcommand(unit, SREV);
        else
                tcommand(unit, NOP);
out:
        for(bp = tmtab.b_forw; bp != &tmtab; bp = bp->b_forw)
                if(bp->b_dev == dev)
                        bp->b_dev.d_minor = -1;
        t_openf[unit] = 0;
}

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

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

tmstrategy(bp)
register struct buf *bp;
{
        register char **p;

        if((bp->b_flags&B_MAP) == 0)
                mapalloc(bp);
        p = &t_nxrec[bp->b_dev.d_minor&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 (tmtab.b_actf==0)
                tmtab.b_actf = bp;
        else
                tmtab.b_actl->av_forw = bp;
        tmtab.b_actl = bp;
        if (tmtab.b_active==0)
                tmstart();
        spl0();
}

tmstart()
{
        register struct buf *bp;
        register int com, unit;
        char *blkno;
```

```
    loop:
        if ((bp = tmtab.b_actf) == 0)
                return;
        unit = bp->b_dev.d_minor&03;
        com = (unit<<8);
        TMADDR->tmcs = com;      /* gives unit time to be selected */
        blkno = t_blkno[unit];
        tmtab.io_stp = &tmstat[unit];
        if(bp == &ctmbuf) {
#ifdef PWR_FAIL
                if (tm_pwroff&(1<<unit) || bp->b_resid == NOP) {
#endif
#ifndef PWR_FAIL
                if(bp->b_resid == NOP) {
#endif
                        bp->b_resid = TMADDR->tmer;
                        tmtab.b_actf = bp->av_forw;
                        iodone(bp);
                        goto loop;
                }
                tmtab.b_active = TCMD;
                tmtab.io_stp->io_misc++;
                TMADDR->tmbc = 0;
                TMADDR->tmcs = com|bp->b_resid|DENS|IENABLE|GO;
                return;
        }
        if (t_openf[unit] < 0 || (TMADDR->tmcs & CRDY)==0) {
                if((t_openf[unit] != -2) || (bp->b_flags&B_READ)) {
                        if (t_openf[unit] == -2)
                                bp->b_resid = bp->b_bcount;
                        else
                                bp->b_flags |= B_ERROR;
                        tmtab.b_actf = bp->av_forw;
                        iodone(bp);
                        goto loop;
                }
        }
        com |= DENS |  ((bp->b_paddr.hiword & 03)<<4) | IENABLE ;
        if (blkno != bp->b_blkno) {
                tmtab.b_active = SSEEK;
                if (blkno < bp->b_blkno) {
                        com |= SFORW|GO;
                        TMADDR->tmbc = blkno - bp->b_blkno;
                } else {
                        com |= SREV|GO;
                        TMADDR->tmbc = bp->b_blkno - blkno;
                        if(t_openf[unit] == -2)
                                tmtab.b_active = SBACK;
                }
                tmtab.io_stp->io_misc++;
        }
        else {
                tmtab.b_active = SIO;
                tmtab.io_stp->io_ops++;
```

```
                blkacty =| (1<<TM0);
                TMADDR->tmbc = -bp->b_bcount;
                TMADDR->tmba = bp->b_paddr.loword;      /* core address */
                com =| ((bp->b_flags&B_READ)? RCOM|GO:
                    ((tmtab.b_errcnt)? WIRG|GO: WCOM|GO));
        }
        TMADDR->tmcs = com;
}

tmintr()
{
        register struct buf *bp;
        register int unit, state;
        struct device tmregs[0];

        if ((bp = tmtab.b_actf)==0) {
                logstray(TMADDR);
                return;
        }
        blkacty =& ~(1<<TM0);
        state = tmtab.b_active;
        unit = bp->b_dev.d_minor&03;
        if (TMADDR->tmcs < 0 && state != TCMD) {        /* error */
                while(TMADDR->tmrd & GSD) ;     /* wait for gap shutdown */
                if((state&SSEEK) || TMADDR->tmer&HARD) {
                        t_openf[unit] = -1;
                        state = SBAD;
                }
                else if((TMADDR->tmer&EOF) == 0) {      /* soft errors */
                        if (++tmtab.b_errcnt < 10) {
                                tmtab.io_stp = &tmstat[unit];
                                fmtberr(&tmtab,0);
                                t_blkno[unit]++;
                                tmtab.b_active = 0;
                                tmstart();
                                return;
                        }
                        else
                                state = SBAD;
                }
                else if((bp->b_flags&B_HEAD) == 0) {    /* EOF */
                        if(state&SBACK)
                                t_openf[unit] = 1;
                        else
                                t_openf[unit] = -2;
                }
                if(state&SBAD) {
                        bp->b_flags =| B_ERROR;
                        tmtab.io_stp = &tmstat[unit];
                        fmtberr(&tmtab,0);
                }
                else
                        TMADDR->tmbc = -bp->b_bcount;
        }
        if(state&(SIO|TCMD|SBAD)) {
                if (tmtab.io_erec)
```

```
                        logberr(&tmtab,bp->b_flags&B_ERROR);
                tmtab.b_errcnt = 0;
                t_blkno[unit]++;
                tmtab.b_actf = bp->av_forw;
                tmtab.b_active = 0;
                bp->b_resid = -TMADDR->tmbc;
                iodone(bp);
        } else
                t_blkno[unit] = bp->b_blkno;
        tmstart();
}

tmread(dev)
{
        tmphys(dev);
        physio(tmstrategy, dev, B_READ, 0);
}

tmwrite(dev)
{
        tmphys(dev);
        physio(tmstrategy, dev, B_WRITE, 0);
}

tmphys(dev)
{
        register unit, a;

        unit = dev.d_minor&03;
        a = u.u_offset>>BSHIFT;
        t_blkno[unit] = a;
        t_nxrec[unit] = ++a;
}
```

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

#

/*
 * transparent line discipline
 */

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

trswrite(tp)
register struct tstp *tp;
{
        register c;
        extern lbolt;

        spl5();
        while(tp->ts_state&WOPEN && (tp->ts_flags&NOSLEEP) == 0)
                sleep(tp, TTOPRI);
        spl0();
        if((tp->ts_state&CARR_ON) == 0)
                return;
        while((c = cpass()) >= 0) {
                if(tp->ts_outq.c_cc > TTHIWAT) {
                        ttstart(tp);
                        spl5();
                        while (tp->ts_outq.c_cc > TTHIWAT) {
                                tp->ts_state |= ASLEEP;
                                if (tp->ts_chan)
                                        return((caddr_t)&tp->ts_outq);
                                sleep((caddr_t)&tp->ts_outq, PZERO);
                                if (issig()) {
                                        spl0();
                                        goto out;
                                }
                        }
                        spl0();
                }
                while(putc(c, &tp->ts_outq)) {
                        ttstart(tp);
                        sleep((caddr_t)&lbolt, -1);
                }
        }
out:
        ttstart(tp);
        return(0);
}

trsxint(tp, flag)
```

```
register struct tstp *tp;
{
        register c;

        if(flag)
                return;
        if((c = getc(&tp->ts_outq)) < 0)
                return(0);
        if (tp->ts_state&ASLEEP && tp->ts_outq.c_cc <= TTLOWAT) {
                tp->ts_state &= ~ASLEEP;
                if (tp->ts_chan)
                        mcstart(tp->ts_chan, (caddr_t)&tp->ts_outq);
                else
                        wakeup((caddr_t)&tp->ts_outq);
        }
        return(c | CPRES);
}

trsread(atp)
struct tstp *atp;
{
        register struct tstp *tp;
        register c;
        register char q;
        extern trswake();

        tp = atp;
        spl5();
        while(tp->ts_state&WOPEN && (tp->ts_flags&NOSLEEP)==0
            && tp->ts_chan==NULL)
                sleep(tp, TTIPRI);
        spl0();
        if(u.u_count==0)
                return(0);
        tp->ts_tout = 0;
        while(tp->ts_tout==0) {
                if((tp->ts_count = u.u_count) > 128)
                        tp->ts_count = 128;
                spl5();
                if(tp->ts_delct==0 && tp->ts_count>tp->ts_rawq.c_cc) {
                        if((tp->ts_flags&NOSLEEP) || (tp->ts_state&CARR_ON)==0
                          || tp->ts_chan!=NULL)
                        {
                                spl0();
                                return(0);
                        }
                        if(q=tp->ts_quanta)
                                timeout(&trswake, tp, tp->ts_quanta*60);
                        sleep(&tp->ts_rawq, TTIPRI);
                        if(tp->ts_tout==0 && q)
                                kiltout(&trswake, tp);
                }
                spl0();
                while((c=getc(&tp->ts_rawq)) != -1) {
                        if(delchar(tp, c)) {
                                tp->ts_delct--;
```

```
                                passc(c);
                                return(tp->ts_delct);
                        }
                        if(passc(c) == -1)
                                return(tp->ts_delct);
                }
        }
}

trswake(tp)
struct tstp *tp;
{
        tp->ts_tout++;
        wakeup(&tp->ts_rawq);
}

trsinput(ac, atp)
struct tstp *atp;
{
        register struct tstp *tp;
        register c;

        tp = atp;
        c = ac;
        if((tp->ts_state&ISOPEN) == 0)
                return;
        if(tp->ts_rawq.c_cc >= TTYHOG) {
                flushtty(tp);
                return;
        }
        putc(c, &tp->ts_rawq);
        if(delchar(tp, c)) {
                tp->ts_delct++;
            wake:
                wakeup(&tp->ts_rawq);
                return;
        }
        if(tp->ts_rawq.c_cc == tp->ts_count)
                goto wake;
}

delchar(atp, ac)
struct tstp *atp;
char ac;
{
        register struct tstp *tp;
        register char c;

        c = ac;
        tp = atp;

        if(tp->ts_brk0 != 0 || tp->ts_brk1 != 0)
                if(tp->ts_brk1 == -1) {
                        if(tp->ts_brk0 == c)
                                return(1);
                } else if(tp->ts_brk0 == c || tp->ts_brk1 == c)
```

```
                        return(1);
        return(0);
}

trsioctl(com, tp, addr, flag)
register com;
register struct tstp *tp;
register caddr_t *addr;
{
        struct transcb transcb;

        switch(com) {
        case OLDSGTTY:
                if (addr) {                 /* old gtty */
                        addr[0].lobyte = tp->ts_quanta;
                        addr[2].lobyte = tp->ts_brk0;
                        addr[2].hibyte = tp->ts_brk1;
                        return;
                }
                if(flag == LUNSET) {
                        flushtty(tp);
                        return;
                }
                tp->ts_quanta = u.u_arg[0].lobyte;
                tp->ts_brk0 = u.u_arg[2].lobyte;
                tp->ts_brk1 = u.u_arg[2].hibyte;
                flushtty(tp);
                return;

        case TIOCSETD:
                /*
                 * Turn on/off the line discipline
                 */
                if (flag == LUNSET) {
                        flushtty(tp);
                        return;
                }
                tp->ts_quanta = 0;
                tp->ts_brk0 = 0;
                tp->ts_brk1 = 0;
                return;

        case DIOCGETP:
                /*
                 * Send line discipline parameters to user
                 */
                transcb.trs_quanta = tp->ts_quanta;
                transcb.trs_bk0 = tp->ts_brk0;
                transcb.trs_bk1 = tp->ts_brk1;
                if (copyout((caddr_t)&transcb, addr, sizeof(transcb)))
                        u.u_error = EFAULT;
                return;

        case DIOCSETP:
                /*
                 * Get line discipline parameters
```

```
                 */
                if (copyin(addr, (caddr_t)&transcb, sizeof(transcb))) {
                        u.u_error = EFAULT;
                        return;
                }
                tp->ts_quanta = transcb.trs_quanta;
                tp->ts_brk0 = transcb.trs_bk0;
                tp->ts_brk1 = transcb.trs_bk1;
                flushtty(tp);
                return;

        default:
                u.u_error = EINVAL;
                return;
        }
}
```

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

#include "sys/param.h"
#include "sys/tty.h"
#include "sys/ttyx.h"
#include "sys/conf.h"
#include "sys/confx.h"


ttxdma(atp, adp)
struct tty *atp;
struct tty_dma *adp;
{
        extern ttrstrt();
        register struct tty *tp;
        register struct tty_dma *dp;
        register char *cp;
        int cnt, c;

        tp = atp;
        dp = adp;
        cnt = 0;

        dp->dma_xmem = 0;
        if((c = dp->dma_char) & (CTOUT|CBREAK)) {
            tmeout:
                if(c & CTOUT) {
                        timeout(ttrstrt, tp,
                                (c&0177)+chrdelay[tp->t_speeds.hibyte&017]);
                        tp->t_state =| TIMEOUT;
                }
                dp->dma_char = 0;
                return(c);
        }

        while((c = (*linesw[tp->t_ltype].l_xmtd)(tp, 0))) {
                if(c < 0) {
                        if(cnt == 0)
                                if((cp=dp->dma_blk) == &dp->dma_char)
                                        if((cp=cfreelist) == 0) {
                                                dp->dma_char = c;
                                                return(1 | CPRES);
                                        } else {
                                                dp->dma_blk = cp;
                                                cfreelist = cp->c_next;
                                        }

                        *cp++ = c;
                        if(++cnt > sizeof(*cfreelist)-1)
                                break;

                } else {
                        if(cnt == 0)
                                goto tmeout;
                        dp->dma_char = c;
```

```
                        break;
                }
        }

        if(cnt)
                return(cnt | CPRES);

        if((cp=dp->dma_blk) != &dp->dma_char) {
                cp->c_next = cfreelist;
                cfreelist = cp;
                dp->dma_blk = &dp->dma_char;
        }

        (*linesw[tp->t_ltype].l_xmtd)(tp, 1);
        return(0);
}
```

```
/*  (#) @(#)tty.c       2.16.1.1        */

#
/*
 */

/*
 * general TTY subroutines
 */
#include "sys/param.h"
#include "sys/systm.h"
#include "sys/user.h"
#include "sys/userx.h"
#include "sys/ioctl.h"
#include "sys/tty.h"
#include "sys/ttyx.h"
#include "sys/proc.h"
#include "sys/procx.h"
#include "sys/inode.h"
#include "sys/inodex.h"
#include "sys/file.h"
#include "sys/filex.h"
#include "sys/reg.h"
#include "sys/conf.h"
#include "sys/confx.h"
#include "sys/seg.h"
#include "sys/crtctl.h"
#include "sys/sysmes.h"
#include "sys/sysmesx.h"
#include "sys/version.h"

/*
 * Input mapping table-- if an entry is non-zero, when the
 * corresponding character is typed preceded by "\" the escape
 * sequence is replaced by the table value.  Mostly used for
 * upper-case only terminals.
 */

char    maptab[]
{
        000,000,000,000,CEOT,000,000,000,
        000,000,000,000,000,000,000,000,
        000,000,000,000,000,000,000,000,
        000,000,000,000,000,000,000,000,
        000,'|',000,'#',000,000,000,'`',
        '{','}',000,000,000,000,000,000,
        000,000,000,000,000,000,000,000,
        000,000,000,000,000,000,000,000,
        '@',000,000,000,000,000,000,000,
        000,000,000,000,000,000,000,000,
        000,000,000,000,000,000,000,000,
        000,000,000,000,000,000,'~',000,
        000,'A','B','C','D','E','F','G',
        'H','I','J','K','L','M','N','O',
        'P','Q','R','S','T','U','V','W',
```

```
        'X','Y','Z',000,000,000,000,000,
};

#ifdef SPY
struct  tp      *spyontp, *spytp;
#endif

struct  clist tempq;
char    colsave,colpres;
char    rowsave,rowpres;

/* The character lists-- space for (2^n)*NCLIST characters */
struct cblock cfree[NCLIST];

/* Counters maintained by getc and putc. */
#ifdef  NXCLIST
        int     xcfrecnt        NXCLIST-1;
#endif
        int     cfrecnt         NCLIST;

/*
 * structure of device registers for KL, DL, and DC
 * interfaces-- more particularly, those for which the
 * SSTART bit is off and can be treated by general routines
 * (that is, not DH).
 */
struct {
        int ttrcsr;
        int ttrbuf;
        int tttcsr;
        int tttbuf;
};

/*
 * routine called on first teletype open.
 * establishes a process group for distribution
 * of quits and interrupts from the tty.
 */
ttyopen(tp)
register struct tty *tp;
{
        register struct proc *pp;

        pp = u.u_procp;
        if (tp->t_flags&XCLUDE) {
                u.u_error = EBUSY;
                return;
        }
        if(pp->p_pgrp == 0) {
                if(tp->t_pgrp)
                        pp->p_pgrp = tp->t_pgrp;
                else {
                        pp->p_pgrp = pp->p_pid;
                        tp->t_pgrp = pp->p_pid;
                }
                u.u_ttyp = tp;
```

```
                u.u_ttyd = tp->t_dev;
        }
        spl5();
        if ((tp->t_state&CARR_ON) == 0)
                tp->t_state =| WOPEN;
        spl0();
        if ((tp->t_state&ISOPEN) == 0) {
                tp->t_erase = CERASE;
                tp->t_kill = CKILL;
        }
        tp->t_state =| ISOPEN|EVEROPEN;
}

/*
 * Close a tty line.
 */
ttyclose(dev, tp)
register struct tty *tp;
{

        wflushtty(tp);
        tp->t_flags =& ~(XCLUDE|STDTTY|TANDEMI);
        tp->t_state =& (EVEROPEN|CARR_ON|SSTART);
        tp->t_pgrp = 0;
}

/*
 * The routine implementing the gtty system call.
 * Just call lower level routine and pass back values.
 */
gtty()
{
        int v[3];
        register *up, *vp;

        if (VERSION(UV_CBR2)) {
                up = u.u_arg[0];
                v[0] = fuword(up);
                v[1] = fuword(up+1);
                v[2] = fuword(up+2);
                vp = v;
                sgtty(vp);
                if (u.u_error)
                        return;
                suword(up, *vp++);
                suword(++up, *vp++);
                suword(++up, *vp++);
                return;
        }
        u.u_arg[2] = u.u_arg[0];
        u.u_arg[1] = TIOCGETP;
        u.u_arg[0] = u.u_ar0[R0];
        ioctl();
}

/*
```

```
 * The routine implementing the stty system call.
 * Read in values and call lower level.
 */
stty()
{
        register int *up;

        if (VERSION(UV_CBR2)) {
                up = u.u_arg[0];
                u.u_arg[3] = up;
                u.u_arg[0] = fuword(up);
                u.u_arg[1] = fuword(++up);
                u.u_arg[2] = fuword(++up);
                sgtty(0);
                return;
        }
        u.u_arg[2] = u.u_arg[0];
        u.u_arg[1] = TIOCSETP;
        u.u_arg[0] = u.u_ar0[R0];
        ioctl();
}

/*
 * Stuff common to stty and gtty.
 * Check legality and switch out to individual
 * device routine.
 * v  is 0 for stty; the parameters are taken from u.u_arg[].
 * v  is non-zero for gtty and is the place in which the device
 * routines place their information.
 */
sgtty(v)
int *v;
{
        register struct file *fp;
        register struct inode *ip;
        register fmt;

        if ((fp = getf(u.u_ar0[R0])) == NULL)
                return;
        ip = fp->f_inode;
        fmt = ip->i_mode & IFMT;
        if (fmt != IFCHR && fmt != IFMPC) {
                u.u_error = ENOTTY;
                return;
        }
        (*cdevsw[ip->i_un.i_rdev.d_major].d_ioctl)(ip->i_un.i_rdev,OLDSGTTY,v);
}

lsgtty(av, tp)
register struct tty *tp;
int *av;
{
        register *v;
        register i;
        extern nodev();
```

```
        if(v = av) {    /* gtty case */
                switch(v[1].lobyte) {

                case 0:
                case 1:
                        v[0] = tp->t_speeds;
                        v[2] = tp->t_flags;
                        break;

                case 2:
                        v[1].hibyte = tp->t_ltype;
                        (*linesw[tp->t_ltype].l_ioctl)(OLDSGTTY, tp, v);
                        break;

                case 3:
                        v[0].lobyte = tp->t_tmflgs;
                        v[1].hibyte = tp->t_term;
                        break;

                case 4:
                        v[0].lobyte = tp->t_row;
                        v[0].hibyte = tp->t_col;
                        v[1].hibyte = tp->t_vrow;
                        v[2].hibyte = tp->t_lrow;
                        break;

                default:
                        goto err;
                }
                return(0);
        }

        if(u.u_arg[1].lobyte != 1)
                wflushtty(tp);
        i = u.u_arg[1].hibyte;
        switch (u.u_arg[1].lobyte) {    /* stty case */

        case 0:
        case 1:
                if ((unsigned)u.u_arg[0].lobyte > 15) {
                        u.u_error = EINVAL;
                        return(0);
                }
                tp->t_speeds = u.u_arg[0];
                i = tp->t_flags;
                tp->t_flags = u.u_arg[2];
                if (tp->t_flags&STDTTY) {
                        tp->t_erase = SCERASE;
                        tp->t_kill = SCKILL;
                } else if (i&STDTTY) {
                        tp->t_erase = CERASE;
                        tp->t_kill = CKILL;
                }
                if(tp->t_flags&RAW && tp->t_state&XMTSTOP) {
                        tp->t_state =& ~XMTSTOP;
                        ttstart(tp);
```

```
                }
                return(1);

        case 2:
                if ((unsigned)i >= nldisc || *linesw[i].l_ioctl == &nodev)
                        goto err;
                if(i == tp->t_ltype)
                        (*linesw[i].l_ioctl)(OLDSGTTY, tp, 0, LRESET);
                else {
                        (*linesw[tp->t_ltype].l_ioctl)(OLDSGTTY, tp, 0, LUNSET)
*               }
                        (*linesw[tp->t_ltype=i].l_ioctl)(OLDSGTTY, tp, 0, LSET)
*               }
                }
                break;

        case 3:
                if ((unsigned)i >= nttype)
                        goto err;
                tp->t_vrow = 0;
                if (i) {
                        (*termsw[i].t_ioctl)(tp, LSET, 0);
                        if(u.u_error)
                                break;
                        tp->t_term = i;
                        if(u.u_arg[0].lobyte&TM_SET)
                                tp->t_tmflgs = u.u_arg[0].lobyte & ~TM_SET;
                } else {
                        tp->t_term = 0;
                }
                tp->t_state =& ~INESC;
                break;

        case 4:
                (*termsw[tp->t_term].t_ioctl)(tp, LRESET, i);
                if (u.u_error)
                        break;
                tp->t_vrow = i;
                break;

        default:
        err:
                u.u_error = ENXIO;
        }
        return(0);
}

/*
 * Line discipline zero ioctl routine
 */
ttyioctl(com, tp, addr, flag)
register com;
register struct tty *tp;
caddr_t addr;
{
        struct termcb termblk;
```

```
        switch(com) {
        case OLDSGTTY:
                if (addr)       /* old gtty case */
                        return;

        case TIOCSETD:
                if (flag == LSET) {
                        tp->t_term = 0;
                        tp->t_erase = CERASE;
                        tp->t_kill = CKILL;
                }
                else if (flag == LUNSET)
                        flushtty(tp);
                break;

        case DIOCGETT:
                termblk.st_flgs = tp->t_tmflgs;
                termblk.st_termt = tp->t_term;
                termblk.st_crow = tp->t_row;
                termblk.st_ccol = tp->t_col;
                termblk.st_vrow = tp->t_vrow;
                termblk.st_lrow = tp->t_lrow;
                if (copyout((caddr_t)&termblk, addr, sizeof(termblk)))
                        u.u_error = EFAULT;
                break;

        case DIOCSETT:
                if (copyin(addr, (caddr_t)&termblk, sizeof(termblk))) {
                        u.u_error = EFAULT;
                        break;
                }
                if ((unsigned)termblk.st_termt >= nttype) {
                        u.u_error = ENXIO;
                        break;
                }
                if (termblk.st_termt) {
                        (*termsw[tp->t_term].t_ioctl)
                            (tp,
                            tp->t_term==termblk.st_termt ? LRESET : LSET,
                            termblk.st_vrow);
                        if (u.u_error)
                                break;
                        tp->t_vrow = termblk.st_vrow;
                        tp->t_term = termblk.st_termt;
                        if(termblk.st_flgs&TM_SET)
                                tp->t_tmflgs = termblk.st_flgs & ~TM_SET;
                } else {
                        tp->t_term = 0;
                }
                tp->t_state &= ~INESC;
                break;

#ifdef SPY
        case DIOCSETS:
                if (!suser())
```

termblk.st-term

```
                        break;
                if (addr) {
                        if(spyontp) {
                                u.u_error = EBUSY;
                                break;
                        }
                        if (u.u_ttyp == (int)tp) {
                                u.u_error = EINVAL;
                                break;
                        }
                        spyontp = tp;
                        spytp = u.u_ttyp;
                } else
                        spyontp = 0;
                break;
#endif

        default:
                u.u_error = ENXIO;
        }
}
/*
 * Wait for output to drain, then flush input waiting.
 */
wflushtty(tp)
register struct tty *tp;
{

        if(tp->t_flags & NOSLEEP)
                return;
        spl5();
        while(tp->t_outq.c_cc) {
                tp->t_state |= ASLEEP;
                sleep(&tp->t_outq, TTOPRI);
        }
        flushtty(tp);
        spl0();
}

/*
 * Initialize clist by freeing all character blocks, then count
 * number of character devices. (Once-only routine)
 */
cinit()
{
        register int ccp;
        register struct cblock *cp;
        register struct cdevsw *cdp;

        ccp = cfree;
        for (cp=(ccp+sizeof(*cfreelist)-1)&~(sizeof(*cfreelist)-1);
                cp <= &cfree[NCLIST-1]; cp++) {
                cp->c_next = cfreelist;
                cfreelist = cp;
        }
```

tp->t_state &BUSY

```
#ifdef NXCLIST
        UISA->r[0] = xclistbase;
        UISD->r[0] = 077406;    /* 4k RW */
        for(ccp=0; ccp < (NXCLIST * sizeof(*cp));) {
                suiword(&((ccp&017777)->c_next), xcfreelist);
                xcfreelist = ccp;
                ccp =+ sizeof(*cp);
                if((ccp&017777)==0)
                        UISA->r[0] =+ 128;
        }
#endif
}

/*
 * flush all TTY queues
 */
flushtty(tp)
register struct tty *tp;
{
        register int sps;

        while (getc(&tp->t_canq) >= 0);
        while (getc(&tp->t_outq) >= 0);
        wakeup(&tp->t_rawq);
        wakeup(&tp->t_outq);
        sps = spl5();
        while (getc(&tp->t_rawq) >= 0);
        tp->t_delct = 0;
        splx(sps);
}

/*
 * transfer raw input list to canonical list,
 * doing erase-kill processing and handling escapes.
 * It waits until a full line has been typed in cooked mode,
 * or until any character has been typed in raw mode.
 */
canon(tp)
register struct tty *tp;
{
        register char *bp;
        char *bp1;
        register int c;
        int mc;

        spl5();
        while ((tp->t_flags&RAW)==0 && tp->t_delct==0
                || (tp->t_flags&RAW)!=0 && tp->t_rawq.c_cc==0) {
                if((tp->t_flags&NOSLEEP) || tp->t_chan!=NULL)
                        return(0);
                sleep(&tp->t_rawq, TTIPRI);
        }
        spl0();
loop:
        bp = &canonb[2];
        while ((c=getc(&tp->t_rawq)) >= 0) {
```

```
                if ((tp->t_flags&RAW)==0) {
                        if (c==0377) {
                                tp->t_delct--;
                                break;
                        }
                        if (bp[-1]!='\\') {
                                if (c==tp->t_erase) {
                                        if (bp > &canonb[2])
                                                bp--;
                                        continue;
                                }
                                if (c==tp->t_kill)
                                        goto loop;
                                if (c==CEOT)
                                        continue;
                        } else {
                                mc = maptab[c];
                                if(mc && (mc==c || (tp->t_flags&LCASE))){
                                if (bp[-2] != '\\')
                                        c = mc;
                                bp--;
                                }
                        }
                }
                *bp++ = c;
                if (bp>=canonb+CANBSIZ)
                        break;
        }
        bp1 = bp;
        bp = &canonb[2];
        c = &tp->t_canq;
        while (bp<bp1)
                putc(*bp++, c);

        if (tp->t_state&TBLOCK && tp->t_rawq.c_cc < TTYHOG/5) {
                if (putc(XON, &tp->t_outq)==0) {
                        tp->t_state &= ~TBLOCK;
                        ttstart(tp);
                }
        }

        return(1);
}

/*
 * Place a character on raw TTY input queue, putting in delimiters
 * and waking up top half as needed.
 * Also echo if required.
 * The arguments are the character and the appropriate
 * tty structure.
 */
ttyinput(ac, tp)
register struct tty *tp;
{
        register int t_flags, c;
```

```
        meas.m_termin++;
        c = ac;
        if ((tp->t_state&ISOPEN) == 0)
                return;
        t_flags = tp->t_flags&(EVENP|ODDP);
        if(t_flags==0 || (t_flags!=(EVENP|ODDP) && c&PERROR))
                return;
        t_flags = tp->t_flags;
        if (c&FRERROR && (c&0177) == 0) {
                if((t_flags&RAW) == 0)
                        goto xmtesc;
                c = BRK;
                goto esc;
        }

        if((t_flags&RAW) == 0)
                c &= 0177;
        else
                c &= 0377;
        if(tp->t_term) {
                c &= 0177;
                c = (*termsw[tp->t_term].t_input) (c, tp);
                if(c == -1)
                        return;

                if(c < 0) {

                    esc:
                        putc(ESC, &tp->t_rawq);
                        putc(c, &tp->t_rawq);
                        if (tp->t_flags&RAW)
                                wakeup(&tp->t_rawq);
                        goto out;

/*
 * If Character erase was entered, and echo is on,
 * then transmit a sequence to erase the previous
 * input character.
 */

                } else if (((t_flags&(ECHO|RAW)) == ECHO) &&
                    ((tp->t_state&INESC) == 0) && (c == tp->t_erase)) {
                        putc(c, &tp->t_rawq);
                        if (tp->t_col) {
                                putc('\b', &tp->t_outq);
                                        putc(' ', &tp->t_outq);
                                putc('\b', &tp->t_outq);
                                        tp->t_col--;
                        }
                        goto out;
                }

        }

        if (c == '\r' && t_flags&CRMOD)
                c = '\n';
```

```
        if ((t_flags&RAW) == 0) {
                if (c==034 || c==0177
                    || (t_flags&STDTTY && c=='&')) {
                        signal(tp->t_pgrp, c==034 ? SIGQIT:SIGINT);
                        if(tp->t_hqcnt == 0)
                                flushtty(tp);
                        if (c != '&')
                                return;
                        goto echo;
                }
                if (c==XMTESC) {
            xmtesc:
                        tp->t_state =^ XMTSTOP;
                        ttstart(tp);
                        return;
                }
                if ((tp->t_state&INESC) == 0) {
                        if ((c == tp->t_kill) && (t_flags&ECHO)) {
                                putc(c, &tp->t_rawq);
                                ttyoutput(c, tp);
                                c = '\n';
                                goto echo;
                        }
                        if (c == '\\')
                                tp->t_state =| INESC;
                } else
                        tp->t_state =& ~INESC;
        }
        if (tp->t_state&STANDEMO) {
                if (tp->t_rawq.c_cc >= TTYHOG/2 && (tp->t_state&TBLOCK)==0) {
                        if (putc(XOFF, &tp->t_outq)==0) {
                                tp->t_state |= TBLOCK;
                                ttstart(tp);
                        }
                }
        }
        if (tp->t_rawq.c_cc>=TTYHOG) {
                flushtty(tp);
                return;
        }
        if (t_flags&LCASE && c>='A' && c<='Z')
                c =+ 'a'-'A';
        putc(c, &tp->t_rawq);
        if (t_flags&RAW || c=='\n' || c==CEOT
            || (t_flags&STDTTY && (c=='!' || c=='/'))) {
                wakeup(&tp->t_rawq);
                if ((t_flags&RAW)==0 && putc(0377, &tp->t_rawq)==0)
                        tp->t_delct++;
                if (tp->t_chan!=NULL)
                        sdata(tp->t_chan);
        }

    echo:
        if (t_flags&ECHO)
                ttyoutput(c, tp);
    out:
```

```
        tp->t_state =& ~XMTSTOP;
        ttstart(tp);
}

/*
 * put character on TTY output queue, adding delays,
 * expanding tabs, and handling the CR/NL bit.
 * It is called both from the top half for output, and from
 * interrupt level for echoing.
 * The arguments are the character and the tty structure.
 */
ttyoutput(ac, tp)
register struct tty *tp;
{
        register c;

        c = ac;
        if(tp->t_state&OLOCKI) {
                putc(c, &tempq);
                return;
        }
        ttyout1(c, tp);
        return;
}

ttyout1(ac, tp)
register struct tty *tp;
{
        register int c;
        register char *colp;
        int *qp;
        int ctype;
        int csave;

        meas.m_termop++;
        if ((tp->t_flags&RAW) && tp->t_term==0)
                c = ac&0377;
        else
                c = ac&0177;
        /*
         * Turn tabs to spaces as required
         */
        if (c=='\t' && tp->t_flags&XTABS) {
                do
                        ttyout1(' ', tp);
                while(tp->t_col&07);
                goto ret;
        }
        /*
         * for upper-case-only terminals,
         * generate escapes.
         */
        if (tp->t_flags&LCASE) {
                colp = "({)}!|^~'`";
                while(*colp++)
                        if(c == *colp++) {
```

```
                        ttyoutl('\\', tp);
                        c = colp[-2];
                        break;
                }
        if ('a'<=c && c<='z')
                c =+ 'A' - 'a';
}
/*
 * turn <nl> to <cr><lf> if desired.
 */
top:
if(c == '\n') {
        if(tp->t_term) {
                if(tp->t_vrow && tp->t_row>=tp->t_lrow) {
                        ttyctl(UVSCN, tp);
                        goto ret;
                }
                if(tp->t_tmflgs&SNL) {
                        ttyctl(NL, tp);
                        goto ret;
                }
        }
        if(tp->t_flags&CRMOD)
                ttyoutl('\r', tp);
        if(tp->t_row < tp->t_lrow)
                tp->t_row++;
}

/*
 * Calculate delays.
 * The numbers here represent clock ticks
 * and are not necessarily optimal for all terminals.
 * The delays are indicated by characters above 0200,
 * thus (unfortunately) restricting the transmission
 * path to 7 bits.
 */
colp = &tp->t_col;
csave = c;
ctype = partab[c&0177];

c = 0;
switch (ctype&077) {

/* ordinary */
case 0:
        (*colp)++;

/* non-printing */
case 1:
        break;

/* backspace */
case 2:
        if (*colp)
                (*colp)--;
        break;
```

```
        /* newline */
        case 3:
                if ((tp->t_flags&NLDELAY) == 0)
                        c = cdelay(colp);
                break;

        /* tab */
        case 4:
                if ((tp->t_flags&NTDELAY)==0)
                        c = 1 - (*colp | ~07);
                *colp =| 07;
                (*colp)++;
                break;

        /* vertical motion */
        case 5:
                if ((tp->t_flags&(NLDELAY|NTDELAY|NCDELAY))
                    != (NLDELAY|NTDELAY|NCDELAY))
                        c = 0177;
                break;

        /* carriage return */
        case 6:
                if ((tp->t_flags&NCDELAY) == 0)
                        c = cdelay(colp);
                *colp = 0;
        }
        if(*colp >= 80 && tp->t_term &&
                        tp->t_row >= tp->t_lrow && tp->t_tmflgs&LCF) {
                ttyctl(VHOME, tp);
                ttyctl(DL, tp);
                ttyctl(LCA, tp, 79, tp->t_lrow-1);
                (*colp)++;
        }

        qp = &tp->t_outq;
        if ((tp->t_flags&RAW) && tp->t_term==0) {
                putc(csave, qp);
                goto ret;
        }
        qputc(csave, qp);
        if(c) {
                putc(QESC, qp);
                putc(c|0200, qp);
        }
        if(*colp >= 80 && tp->t_term && tp->t_tmflgs&ANL)
                if(tp->t_tmflgs&LCF)
                        ttyctl(LCA, tp, 0, tp->t_row+1);
                else {
                        if((tp->t_flags&CRMOD) == 0)
                                ttyoutl('\r', tp);
                        c = '\n';
                        goto top;
                }
ret:
```

```
        return;
}

cdelay(colp)
char *colp;
{
        if (*colp)
                return(max((*colp>>4) + 3, 12));
        else
                return(6);
}

/* simulate Up Variable SCreeN as common routine */
ttuvscn(atp)
struct tty *atp;
{
        register struct tty *tp;
        tp = atp;

        ttyctl(VHOME, tp);
        ttyctl(DL, tp);
        ttyctl(LCA, tp, 0, tp->t_lrow);
}

/* simulate Down Variable SCreeN as common routine */
ttdvscn(atp)
struct tty *atp;
{
        register struct tty *tp;
        tp = atp;

        ttyctl(VHOME, tp);
        ttyctl(IL, tp);
}

ttyctl(ac, tp, acol, arow)
register struct tty *tp;
{
        register char *colp;
        register c;
        int sps;

        c = ac;
        colp = &tp->t_col;
        sps = spl5();

        colpres = *colp;
        rowpres = tp->t_row;
        switch(c) {

        case CUP:
        case DSCRL:
                if(tp->t_row == 0)
                        goto out;
                tp->t_row--;
                break;
```

```
        case CDN:
        case USCRL:
                if(tp->t_row >= tp->t_lrow)
                        goto out;
                tp->t_row++;
                break;

        case UVSCN:
                *colp = 0;
                tp->t_row = tp->t_lrow;
                break;

        case DVSCN:
                *colp = 0;
                tp->t_row = tp->t_vrow;
                break;

        case CRI:
        case STB:
        case SPB:
                if(*colp >= 79)
                        goto out;
                (*colp)++;
                break;

        case CLE:
                if(*colp == 0)
                        goto out;
                (*colp)--;
                break;

        case HOME:
        case CS:
        case CM:
                tp->t_row = 0;
        case DL:
        case IL:
                *colp = 0;
                break;

        case VHOME:
                *colp = 0;
                tp->t_row = tp->t_vrow;
                c = LCA;
                break;

        case LCA:
                *colp = acol;
                tp->t_row = arow;
                break;

        case ASEG:
                tp->t_row = (tp->t_row+24)%(tp->t_lrow+1);
                break;
```

```
        case NL:
                if(tp->t_row < tp->t_lrow)
                        tp->t_row++;

        case CRTN:
                *colp = 0;
                break;

        case SVID:
                tp->t_dstat |= acol;
                c = DVID;
                break;

        case CVID:
                tp->t_dstat &= ~acol;
                c = DVID;
                break;

        case DVID:
                tp->t_dstat = acol;
                break;

        }

        (*termsw[tp->t_term].t_output)(c, tp);

    out:
        splx(sps);
}

/*
 * Restart typewriter output following a delay
 * timeout.
 * The name of the routine is passed to the timeout
 * subroutine and it is called during a clock interrupt.
 */
ttrstrt(tp)
register struct tty *tp;
{

        tp->t_state =& ~TIMEOUT;
        ttstart(tp);
}

/*
 * Start output on the typewriter. It is used from the top half
 * after some characters have been put on the output queue,
 * from the interrupt routine to transmit the next
 * character, and after a timeout has finished.
 * If the SSTART bit is off for the tty the work is done here,
 * using the protocol of the single-line interfaces (KL, DL, DC);
 * otherwise the address word of the tty structure is
 * taken to be the name of the device-dependent startup routine.
 */
ttstart(tp)
register struct tty *tp;
```

```
{
        register int *addr, c;
        int sps;
        struct { int (*func)(); };

        addr = tp->t_addr;
        sps = spl5();
        if (tp->t_state&SSTART) {
                (*addr.func)(tp);
                splx(sps);
                return;
        }
        if ((addr->tttcsr&DONE) == 0)
                return;
        c = (*linesw[tp->t_ltype].l_xmtd)(tp, 0);
        if(c < 0)
                addr->tttbuf = c;
        else if(c&CTOUT) {
                timeout(ttrstrt, tp, c&0177);
                tp->t_state =| TIMEOUT;
        } else if(c == 0) {
                (*linesw[tp->t_ltype].l_xmtd)(tp, 1);
        }
        splx(sps);
}

/*
 * Called from device's read routine after it has
 * calculated the tty-structure given as argument.
 * The pc is backed up for the duration of this call.
 * In case of a caught interrupt, an RTI will re-execute.
 */
ttread(tp)
register struct tty *tp;
{

        spl5();
        while(tp->t_state&WOPEN && (tp->t_flags&NOSLEEP) == 0)
                sleep(tp, TTIPRI);
        spl0();
        if (tp->t_canq.c_cc || (tp->t_state&CARR_ON && canon(tp)))
                while (tp->t_canq.c_cc && passc(getc(&tp->t_canq))>=0);
        return(tp->t_canq.c_cc);
}

/*
 * Called from the device's write routine after it has
 * calculated the tty-structure given as argument.
 */
ttwrite(tp)
register struct tty *tp;
{
        register c;
        register hqflag;
        int col;
        int row;
```

```
qwait:
        spl5();
        while(tp->t_state&QLOCKB ||
              tp->t_state&WOPEN && (tp->t_flags&NOSLEEP)==0) {
                tp->t_state =| QWANT;
                sleep(tp, TTOPRI);
        }
        spl0();
        if((tp->t_state&CARR_ON) == 0)
                return;

        hqflag = 0;
        while((c = cpass()) >= 0) {
                if (c == ESC && tp->t_term) {
                        switch(c = cpass()) {
                        case -1:
                                continue;
                        case ESC:
                                goto norm;
                        case BRK:
                                putc(0177, &tp->t_outq);
                                putc(0177, &tp->t_outq);
                                putc(QESC, &tp->t_outq);
                                putc(QBRK, &tp->t_outq);
                                goto nxt;
                        case HIQ:
                                if (hqflag++)
                                        continue;
                                tp->t_state =| QLOCKB|QLOCKI;
                                tp->t_hqcnt++;
                                colsave = tp->t_col;
                                rowsave = tp->t_row;
                                continue;
                        case LCA:
                        case SVID:
                        case DVID:
                        case CVID:
                                col = cpass();
                                if (c == LCA)
                                        row = cpass() & 077;
                        default:
                                ttyctl(c, tp, col, row);
                        }
                } else
                norm:
                        ttyoutl(c, tp);

        nxt:
                if (tp->t_outq.c_cc > TTHIWAT) {
                        if(hqflag) {
                                col = tp->t_col;
                                row = tp->t_row;
                                hqrelse(tp);
                        }
                        ttstart(tp);
```

state = 1;

col &= 077;

state = 0;

```
                        spl5();
                        while(tp->t_outq.c_cc > TTHIWAT) {
                                tp->t_state |= ASLEEP;
                                if (tp->t_chan && hqflag==0)
                                /* missing spl0() here is intentional */
                                        return((caddr_t)&tp->t_outq);
                                sleep((caddr_t)&tp->t_outq, hqflag?-1:PZERO);
                                if (hqflag==0 && issig()) {
                                        spl0();
                                        goto out;
                                }
                        }
                        spl0();
                        if (hqflag) {
                                tp->t_state |= QLOCKI;
                                colsave = tp->t_col;
                                rowsave = tp->t_row;
                                ttyctl(LCA, tp, col, row);
                                continue;
                        }
                        if (tp->t_state&QLOCKB)
                                goto qwait;
                }
        }

        if(hqflag) {                                    -> state = 0;
                hqrelse(tp);
                putc(QESC, &tp->t_outq);
                putc(HQEND, &tp->t_outq);
                if(tp->t_state&QWANT)
                        wakeup(tp);
        }
out:
        tp->t_state =& ~(QWANT|QLOCKB|QLOCKI);
        ttstart(tp);
        return(0);
}

/*
 * release the high priority queue for interrupts.
 * copy over any received characters while
 * queue was locked.
 */
hqrelse(tp)
register struct tty *tp;
{
        register c;

        ttyctl(LCA, tp, colsave, rowsave);
        spl5();
        while((c = getc(&tempq)) >= 0)
                ttyoutl(c, tp);
        tp->t_state =& ~QLOCKI;
        spl0();
}
```

```
/*
 * put a character on the output queue,
 * checking first to see if it is a QESC.
 */
qputc(ac, aqp)
{
        register c;
        register qp;

        c = ac;
        qp = aqp;
        if(c == QESC)
                putc(c, qp);
        putc(c, qp);
}

/*
 * get a character to be transmitted
 * by a character device
 */
ttxint(tp, flag)
register struct tty *tp;
{
        register c;                                    c = 0;

        if(tp->t_state&(TIMEOUT|XMTSTOP) || tp->t_outq.c_cc==0 || flag)
                return(0);                  goto out;

    loop:
        c = getc(&tp->t_outq);
#ifdef SPY
        if(tp == spyontp) {
                if(spytp->t_outq.c_cc >= TTYHOG)
                        flushtty(spytp);
                putc(c, &spytp->t_outq);
                ttstart(spytp);
        }
#endif
        if ((tp->t_flags&RAW) && tp->t_term == 0) {
                c = (c&0377) | CPRES;
                goto out;
        }
        if(c == QESC) {
                switch(c = getc(&tp->t_outq)) {

                case QESC:
                        break;

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

                case HQEND:
                        tp->t_hqcnt--;
                        if (tp->t_outq.c_cc == 0) {
                                c = 0;
```

```
                                goto out;
                        }
                        goto loop;

                default:
                        if(c > 0177) {
                                c = CTOUT | (c & 0177);
                                goto out;
                        }
                }
        }

        c =& 0177;
        c =| CPRES |
         (((tp->t_flags&(ODDP|EVENP))==ODDP) ? ~partab[c] : partab[c])&0200;

    out:
        if (tp->t_outq.c_cc<=TTLOWAT && tp->t_state&ASLEEP) {
                tp->t_state &= ~ASLEEP;
                if (tp->t_chan)
                        mcstart(tp->t_chan, (caddr_t)&tp->t_outq);
                else
                        wakeup((caddr_t)&tp->t_outq);
        }
        return(c);
}

ttydst(tp, acsr, stat)
register struct tty *tp;
{

        if(stat&CARRIER) {
                tp->t_state =| CARR_ON;
                tp->t_state =& ~WOPEN;
                wakeup(tp);
        } else {
                if(tp->t_state&CARR_ON)
                        signal(tp->t_pgrp, SIGHUP);
                flushtty(tp);
                tp->t_state =& ~(CARR_ON| BUSY)
        }
}
```

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

VERSION = util
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)(hpmap.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
```

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

VERSION = util
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)(hpmap.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
```

```
/*      @(#)vp.c        2.8.1.1 */
/*
 *  Versatec matrix printer/plotter dma interface driver
 */

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

#define NVP     1
#define VPPRI   (PZERO-9)

struct device {
        int     plbcr, pbae, prbcr, pbaddr;
        int     plcsr, plbuf, prcsr, prbuf;
};

struct vp {
        int     vp_state;
        char    *vp_buf;
        int     vp_offset;
};

struct vp vp_vp[NVP];

struct device *vp_addr[NVP] = { 0177500 };

/* status bits */
#define ERROR   0100000
#define READY   0200
#define IENABLE 0100
#define FFCOM   020
#define RESET   02
#define SPP     01

/*states */
#define PRINT   0100
#define PLOT    0200
#define PPLOT   0400

vpopen(dev)
register dev;
{
        register struct vp *vp;
        register *vpaddr;

        dev = minor(dev);
        if (dev>=NVP || (vp = &vp_vp[dev])->vp_buf) {
                u.u_error = ENXIO;
                return;
        }
        vp->vp_buf++;
        vpaddr = &(vp_addr[dev]->plcsr);
        *vpaddr = IENABLE|RESET;
```

```
        vp->vp_state = PRINT;
        if (vpready(dev)<0) {
                *vpaddr = 0;
                vp->vp_buf = 0;
                u.u_error = EIO;
                return;
        }
        *vpaddr = IENABLE|RESET;
        vp->vp_buf = getablk(1);
        vp->vp_offset = 0;
}

vpclose(dev)
{
        register struct vp *vp;
        register *vpaddr;

        dev = minor(dev);
        vp = &vp_vp[dev];
        vpaddr = &(vp_addr[dev]->plcsr);
        vpready(dev);
        *vpaddr = 0;
        brelse(vp->vp_buf);
        vp->vp_buf = 0;
}

vpwrite(dev)
{
        register struct vp *vp;
        register *vpaddr;
        register count;

        dev = minor(dev);
        vp = &vp_vp[dev];
        vpaddr = vp_addr[dev];
        while (u.u_count && u.u_error==0) {
                count = min(256,u.u_count);
                pimove(vp->vp_buf->b_paddr + vp->vp_offset, count, B_WRITE);
                if (vpready(dev)<0) {
                        u.u_error = EIO;
                        vpaddr->plcsr = IENABLE|RESET;
                        break;
                }
                vpaddr->pbaddr = vp->vp_buf->b_paddr.loword + vp->vp_offset;
                if (vp->vp_state&PLOT)
                        vpaddr->plbcr = count;
                else
                        vpaddr->prbcr = count;
                vp->vp_offset = 256 - vp->vp_offset;
                if (vp->vp_state&PPLOT)
                        vp->vp_state = PLOT;
        }
}

vpready(dev)
{
```

```
        register struct vp *vp;
        register *vpaddr;

        vp = &vp_vp[dev];
        vpaddr = (vp->vp_state&PLOT)?&(vp_addr[dev]->plcsr):
                &(vp_addr[dev]->prcsr);
        spl4();
        while ((*vpaddr&(READY|ERROR))==0)
                sleep(vp, VPPRI);
        spl0();
        return(*vpaddr);
}

vpioctl(dev, cmd, addr, flag)
register caddr_t addr;
{
        int m;
        register struct vp *vp;
        register *vpaddr;
        register bit;

        dev = minor(dev);
        vp = &vp_vp[dev];
        vpaddr = &(vp_addr[dev]->plcsr);
        switch(cmd) {

        /* get mode */
        case VIOCGETD:
                suword(addr, vp->vp_state);
                return;

        /* set mode */
        case VIOCSETD:
                m = fuword(addr);
                if (m == -1) {
                        u.u_error = EFAULT;
                        return;
                }
                vpready(dev);
                vp->vp_state = m;
                if (vp->vp_state&PPLOT)
                        *vpaddr = IENABLE|SPP;
                else
                        *vpaddr = IENABLE;
                for (bit=2; bit<IENABLE; bit=<<1)
                        if (vp->vp_state&bit) {
                                vpready(dev);
                                *vpaddr |= bit;
                                vp->vp_state &= ~bit;
                        }
                return;

        default:
                u.u_error = ENOTTY;
                return;
        }
```

```
}

vpintr(dev)
{
        wakeup(&vp_vp[dev]);
}
```

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

/*
 * Votrax line discipline
 */

#include "sys/param.h"
#include "sys/proc.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"
#include "sys/conf.h"
#include "sys/confx.h"
#include "sys/votrax.h"

#define MAGIC_MAP 0377

/*
 * dstat bits
 */
#define CLS             04

vsopen(atp)
struct tty *atp;
{
        register struct proc *pp;
        register struct tty *tp;

        pp = u.u_procp;
        tp = atp;
        if(pp->p_pgrp == 0) {
                if(tp->t_pgrp)
                        pp->p_pgrp = tp->t_pgrp;
                else {
                        pp->p_pgrp = pp->p_pid;
                        tp->t_pgrp = pp->p_pid;
                }
                u.u_ttyp = tp;
                u.u_ttyd = tp->t_dev;
        }
        spl5();
        if ((tp->t_state&CARR_ON) == 0)
                tp->t_state |= WOPEN;
        spl0();
        (*cdevsw[major(tp->t_dev)].d_mctl)(tp, 'c', ROSEND);
        tp->t_state |= ISOPEN|EVEROPEN;
}

vswrite(tp)
register struct vstp *tp;
{
```

```
        register c;
        register count;

        spl5();
        while(tp->t_state&WOPEN && (tp->t_flags&NOSLEEP) == 0)
                sleep(tp, TTOPRI);
        spl0();
        if((tp->t_state&CARR_ON) == 0)
                return;
        count = 0;
        while((c = cpass()) >= 0) {
                while(tp->t_outq.c_cc > TTHIWAT) {
                        ttstart(tp);
                        sleep(&tp->t_outq, TTOPRI);
                }
                if(--count <= 0) {
                        count = 60;
                        putc(0, &tp->t_outq);
                }
                putc(c, &tp->t_outq);
        }
        putc(0, &tp->t_outq);
        ttstart(tp);
}

vsxint(tp, flag)
register struct vstp *tp;
{
        register c;

        if(flag)
                return(0);
        if((tp->t_dstat&CLS) == 0)
                return(0);
        if((c = getc(&tp->t_outq)) < 0)
                return(0);
        c =^ MAGIC_MAP;
        if(tp->t_outq.c_cc == TTLOWAT || tp->t_outq.c_cc == 0)
                wakeup(&tp->t_outq);
        return(c | CPRES);
}

vsread(tp)
register struct vstp *tp;
{
        register c;

        spl5();
        while(tp->t_state&WOPEN && (tp->t_flags&NOSLEEP) == 0)
                sleep(tp, TTIPRI);
        spl0();
        if(u.u_count == 0)
                return;
        spl5();
        while(tp->t_delct == 0) {
                if((tp->t_flags&NOSLEEP) || ((tp->t_state&CARR_ON) == 0)) {
```

```
                        spl0();
                        return;
                }
                sleep(&tp->t_rawq, TTIPRI);
        }
        spl0();
        while((c=getc(&tp->t_rawq)) != -1) {
                if(c == '*' || c == '#') {
                        tp->t_delct--;
                        passc(c);
                        return;
                }
                if(passc(c) == -1)
                        return;
        }
}

vsinput(c, tp)
register c;
register struct vstp *tp;
{

        if((tp->t_state&ISOPEN) == 0)
                return;
        if(tp->t_rawq.c_cc >= TTYHOG) {
                flushtty(tp);
                return;
        }
        c = "d0*#b546a213c879"[c&017];
        putc(c, &tp->t_rawq);
        if(c == '*' || c == '#') {
                tp->t_delct++;
                wakeup(&tp->t_rawq);
        }
}

vsioctl(com, tp, addr, flag)
register com;
register struct vstp *tp;
register caddr_t addr;
{
        struct vscb vscb;
        int lsr;

        switch(com) {
        case TIOCSETD:
                /*
                 * Turn on/off the line discipline
                 */
                flushtty(tp);
                if (flag == LUNSET)
                        return;
                spl5();
                lsr = (*cdevsw[major(tp->t_dev)].d_mctl)(tp, 'c', ROSEND);
                if(lsr&CLSEND)
                        tp->t_dstat |= CLS;
```

```
                else
                        tp->t_dstat &= ~CLS;
                spl0();
                return;

        case DIOCSETP:
                /*
                 * Set line discipline parameters
                 */
                if (copyin(addr, (caddr_t)&vscb, sizeof(vscb))) {
                        u.u_error = EFAULT;
                        return;
                }
                spl5();
                if (vscb.vot_tab)       /* Set tone answer back */
                        lsr = (*cdevsw[major(tp->t_dev)].d_mctl)(tp,'s',SUPTD);
                else                    /* Clear tone answer back */
                        lsr = (*cdevsw[major(tp->t_dev)].d_mctl)(tp,'c',SUPTD);
                if (lsr&CLSEND)
                        tp->t_dstat |= CLS;
                else
                        tp->t_dstat &= ~CLS;
                spl0();
                return;

        default:
                u.u_error = EINVAL;
                return;
        }
}

vsdst(tp, csr, lsr)
register struct vstp *tp;
register csr;
register lsr;
{

        if(csr&CTRANS) {
                if(lsr&CARRIER) {
                        tp->t_state |= CARR_ON;
                        tp->t_state &= ~WOPEN;
                        wakeup(tp);
                } else {
                        signal(tp->t_pgrp, SIGHUP);
                        flushtty(tp);
                        tp->t_state &= ~CARR_ON;
                }
        }
        if(csr&CSTRANS) {
                if(lsr&CLSEND) {
                        tp->t_dstat |= CLS;
                        ttstart(tp);
                } else {
                        tp->t_dstat &= ~CLS;
                }
        }
```

)

```
/*      @(#)vt100.c     2.7.1.1 */

/*.p'vt100'VT100 Driver Defines'*/



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


struct CRTLANG {
        char    code;
        char    inchar;
        char    outchar;
        char    tpindx;
} mapvt100[] {
        LCA,    0,      0,      2,
        CUP,    'A',    'A',    0,
        CDN,    'B',    'B',    0,
        CRI,    'C',    'C',    0,
        CLE,    'D',    'D',    0,
        HOME,   'H',    'H',    0,
        STB,    0,      '5',    3,
        SPB,    0,      '0',    4,
        DVID,   0,      0,      015,
        CS,     0,      0,      5,
        CM,     0,      'c',    013,
        IL,     0200|'S',       0,      012,
        DL,     0200|'R',       0,      6,
        UVSCN,  0,      0,      7,
        DVSCN,  0,      0,      010,
        EEOL,   0,      'K',    0,
        EEOP,   0,      'J',    0,
        SVSCN,  0,      0,      011,
        CUP,    0200|'P',       0,      0,
        CDN,    0200|'Q',       0,      0,
        0
};

/*.s'vt100'Internal Action Tables'*/
/*
 *      table of internal sequences for commands
 *        the following codes are used to construct
 *        these sequences:
 *              0200    output character from map table
 *              0201    output ESC then map table character
 *              0202    output ESC BRAC then map table char
 *              0203    output t_row in proper form
 *              0204    output t_lrow (last row)
 *              0205    output t_vrow (variable home row)
```

```
 *              0206    output t_col (column)
 *              0207    insert delay if required at this speed
 *              0210    process video attributes
 */
char *indx_vt100[] {
        /*  0: normal ESC BRAC char sequence */
        "\202",
        /*  1: ESC char sequence */
        "\201",
        /*  2: load cursor address */
        "\202\203;\206H",
        /*  3: start video attribute sequence */
        " \202m",
        /*  4: end video attribute sequence */
        "\202m ",
        /*  5: clear screen sequence */
        "\202H\202J",
        /*  6: delete line  */
        "\314\307\311",
        /*  7: scroll bottom of screen up */
        "\202\204H\012",
        /*  010: scroll screen down from variable home */
        "\202\205H\201M",
        /*  011: define scrolling boundaries of screen */
        "\202\205r\302",
        /* 012: insert line     */
        "\314\201M\311",
        /* 013: CRT reset with timing   */
        "\201\207",
        /* 014: Temporary Set Scrolling Region to Current Line */
        "\202\203r\302",
        /* 015: Define Video Attributes */
        "\202m\202\210m"
};

#define BRAC    '['
#define BRAC_BIT        0100
#define OH      'O'
#define OH_BIT          040
#define SEMI    ';'

#define TDELAY  0377

char vid_chars[6] { '4', '5', '7', 0, '1', 0 };

/*.s'vt100output'Process Output to VT100'*/
vt100output(ac, atp)
struct tty *atp;
{
        register struct CRTLANG *cp;
        register c;

        c = ac;

        for (cp=mapvt100; cp->code; cp++)
                if (c == cp->code) {
```

```
                        vttrans(atp, cp->tpindx, cp->outchar);
                        return;
                }
}
/*.s'vttrans'Translate to VT100 Language'*/
vttrans(atp, ac, aoc)
struct tty *atp;
char ac, aoc;
{
        register c;
        register char *str;
        register struct clist *qp;
        char *ap;

        qp = &atp->t_outq;

        for( str = indx_vt100[ ac ]; *str; str++) {
                c = *str;
                if ( c & 0200 ) {
                        if( c & 0100) {
                                vttrans(atp, c & 077, 0);
                                continue;
                        } else {
                                switch (c & 077) {
                                case 0:
                                        c = aoc;
                                        break;
                                case 1:
                                        putc(ESC, qp);
                                        c = aoc;
                                        break;
                                case 2:
                                        putc(ESC, qp);
                                        qputc(BRAC, qp);
                                        c = aoc;
                                        break;
                                case 3:
                                        cvtdec(atp->t_row, qp);
                                        continue;
                                case 4:
                                        cvtdec(atp->t_lrow, qp);
                                        continue;
                                case 5:
                                        cvtdec(atp->t_vrow, qp);
                                        continue;
                                case 6:
                                        cvtdec(atp->t_col, qp);
                                        continue;
                                case 7:
                                        putc(QESC, qp);
                                        c = TDELAY;
                                        qputc( c, qp);
                                        putc(QESC, qp);
                                        break;
                                case 010:
                                        c = atp->t_dstat & 077;
```

```
                        ap = vid_chars;
                        while(c) {
                                if((c & 01) && *ap) {
                                        putc(SEMI, qp);
                                        putc(*ap, qp);
                                }
                                ap++;
                                c >>= 1;
                        }
                        continue;
                                }
                        }
                }
                if (c) {
                        qputc(c, qp);
                }
        }
}
/*.s'vt100input'Process VT100 Input'*/
vt100input(ac, atp)
struct tty *atp;
{
        register struct tty *tp;
        register c;
        register struct CRTLANG *cp;

        c = ac;
        tp = atp;

        if (tp->t_tmflgs&BRAC_BIT) {
                if (tp->t_tmflgs&OH_BIT)
                        c =| 0200;
                tp->t_tmflgs =& ~(BRAC_BIT | OH_BIT);
                for (cp=mapvt100; cp->code; cp++)
                        if( (c&0377) == (cp->inchar &0377))
                                return(cp->code | CPRES);
                c =& 0177;
        } else { if (tp->t_tmflgs&TERM_BIT) {
                        tp->t_tmflgs =& ~TERM_BIT;
                        if( c == BRAC ) {
                                tp->t_tmflgs =| BRAC_BIT;
                                return(-1);
                        }
                        if( c == OH) {
                                tp->t_tmflgs =| BRAC_BIT | OH_BIT;
                                return(-1);
                        }
                } else if (c == ESC) {
                        tp->t_tmflgs =| TERM_BIT;
                        if ((tp->t_flags&RAW) == 0)
                                tp->t_state =| XMTSTOP;

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

```
}

/*.s'vt100ioctl'Special IOCTL for VT100'*/
vt100ioctl(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|EVENP|ODDP|CRMOD|
                        ECHO|NTDELAY|NCDELAY|TANDEMI;
        }
        tp->t_vrow = nvrow;
        vt100output(SVSCN, tp);
        ttstart(tp);
}

/*.s'cvtdev'Convert to Decimal ASCII'*/
cvtdec(value, aqp)
int value;
struct clist *aqp;
{
        register i;
        register struct clist *qp;

        if( value == 0 )
                return;
        qp = aqp;
        i = value + 1;
        if(i>9)
                putc( i/10 + '0', qp);
        putc( i%10 + '0', qp);
}
```

```
/*      @(#)vt11.c      2.11    */

#include "sys/param.h"
#include "sys/user.h"
#include "sys/userx.h"
#include "sys/proc.h"
#include "sys/procx.h"
#include "sys/buf.h"
#include "sys/bufx.h"
#include "sys/vt11.h"
#include "sys/elog.h"
#include "sys/iobuf.h"

/*.n'vt11'Storage allocation'*/
int     vtinuse;                        /* # of system bufs in use by VT11 */
int     vtwait;                         /* # of cycles before user awake */
struct  iobuf vttab;                    /* VT11 header */
struct  vtcycl vt11;                    /* VT11 master frame loop */
struct  vtfrpt vt11f;                   /* VT11 frame replacement values */
struct  vtfrhd vt11h[VTNFRM];           /* VT11 frame headers */

int     lpuser;

/*.s'vtopen'Open VT11 for proper user and start DPU'*/
vtopen(dev,flag)
{
        register struct iobuf  *vp;
        register struct vtbuf   *vtbp;
        register struct vtframp *vtfp;
        int      j;

        vp = &vttab;
#ifdef PWR_FAIL
        if (dev == NODEV) {
                goto go;
        }
#endif
        if(vp->v_flags&VTUSER) {
                u.u_error = ENXIO;
                return;
        }

        /*
         * Make sure display processor has been stopped.
         */

        vp->v_flags =| VTSTOP;
        vt11.vtstop =  VTDSTOP;
        spl4();
        while(vp->v_flags&VTGO)
                sleep(vp,VTSPRI);
        spl0();

        /*
         * Initialize VT11 master loop.
```

```
         */

        vt11.vtsync = VTSTATSA | VTSYNC;
        vtfp = &vt11.vtjyfr;
        for(j=0;j<VTNFRM+3;j++) {
                vtfp->vtjump  = VTDJMP;
                vtfp->vtfradr = vtfp+1;
                vtfp++;
        }
        (--vtfp)->vtfradr = &vt11;

        /*
         * Now set up the display for the proper user.
         */

        if((vp->v_flags&(VTUSER|VTSYS)) == 0) {
                vp->v_flags = VTSYS;
                vt11.vtstop = VTDNOP;
#ifdef VTMON
                vtinit();
#endif
                if(lpuser == 0)
                        vtlpopen(dev,flag);
        } else {
                vp->v_flags = VTUSER;
        }

        /*
         * Start VT11 display processor.
         */

go:
        vp->v_flags =| VTGO;
        VTADDR->vtdpc = &vt11;
}
/*.s'vtclose'Shut of accesses for current user'*/
vtclose(dev)
{
        register struct vtfrhd  *vhp;
        register struct vtframe *frp;
        register int    j;

        if(vttab.v_flags&VTUSER) {
                vhp = vt11h;
                for(j=0;j<VTNFRM;j++) {
                        if(frp=vhp->vfd_frm) {
                                vtfrmfree(frp);
                                vhp->vfd_frm = 0;
                        }
                        vhp++;
                }
        }
        vttab.v_flags =& ~(VTUSER|VTSYS);
        vtopen(dev,2);

}
```

```
/*.t'vtfrmfree'Free all buffers in a frame'*/
vtfrmfree(frp)
register struct vtframe *frp;
{
        register struct iobuf *vp;

        if(frp == 0)
                return;

        vp = &vttab;
        spl4();
        while(vp->v_flags&VTBFREE)
                sleep(&vp->v_frmpc,VTBPRI);

        vp->v_flags =| VTBFREE;
        vp->v_frmpc =  frp;
        spl0();
}
/*.s'vtwrite'Create or replace a frame'*/
vtwrite(dev)
{
        register struct vtframe *lfrmp,*nfrmp;
        register int    *stradr;
        int      frame;

        switch(vttab.v_flags&(VTUSER!VTSYS)) {

                default:
                        u.u_error = ENFILE;
                        return;

                case VTUSER:
                        break;

                case VTSYS:
                        frame  = vt11f.vtfrmn;
                        stradr = vt11f.vtfrad + vt11f.vtfrcn;
                        *stradr++ = VTDJMP;
                        *stradr   = &vt11.vtfrm[frame+1];
                        vt11.vtfrm[frame].vtfradr = vt11f.vtfrad;
                        return;

        }

        if((frame=u.u_offset.loword) < 0 || frame >= VTNFRM) {
                u.u_error = EINVAL;
                return;
        }

        if(u.u_base&1) {
                u.u_error = EROFS;
                return;
        }

        lfrmp = nfrmp = stradr = 0;
```

```
        while(u.u_count) {
                if((nfrmp = vtnxpc(lfrmp,frame)) == 0) {
                        if(stradr) {
                                u.u_error = E2BIG;
                                break;
                        } else {
                                vtfrmfree(stradr);
                                u.u_error = ENOSPC;
                                return;
                        }
                }
                if(stradr == 0) stradr = nfrmp;
                if(vtcopy((caddr_t)((lfrmp=nfrmp)->v_addr),frame) < 0) {
                        vtfrmfree(stradr);
                        u.u_error = EFAULT;
                        return;
                }
        }

        vtreplace(stradr,frame);
        u.u_offset += VTNFRM;
}
/*,s'vtread'Read puts user to sleep for u.u_count syncs'*/
vtread()
{
        if(u.u_count) {
                vtwait = u.u_count&077777;
                spl4();
                while(vtwait--)
                        sleep(&vttab,VTSPRI);
        }
        spl0();
}
/*,s'vtnxpc'Get next VT11 frame piece buffer'*/
/*
 * vtnxpc - Get space to store a 'frame piece'.  A 'frame piece' is VTBUFL
 *          words long and can be linked to other 'frame pieces' to make
 *          a single frame.  If a 'frame piece' pointer is passed to vtnxpc
 *          then vtnxpc will link that 'frame piece' to the new 'frame piece'.
 */

vtnxpc(frmp,frame)
register int frame;
struct vtframe *frmp;
{
        register struct vtframe *vfp;
        struct  vtframe *rvfp;
        register int    *intp;

    try:
        if(vtinuse++ == VTMXBFS) {
                vtinuse--;
                if(vttab.v_flags&VTBFREE) {
                        spl4();
                        while(vttab.v_flags&VTBFREE)
                                sleep(&vttab.v_frmpc,VTBPRI);
```

```
                        spl0();
                        goto try;
                }
                if(vfp=vt11h[frame].vfd_frm) {
                        vtfrmfree(vfp);
                        vt11h[frame].vfd_frm = 0;
                        goto try;
                }
                return(0);
        }

        vfp = getablk(1);
        vfp->v_frmpc = 0;
        rvfp = vfp;
        if(vfp = frmp) {
                intp    = (caddr_t)vfp->v_addr + (VTBUFL-2);
                *intp++ = VTDJMP;
                *intp   = (caddr_t)rvfp->v_addr;
                vfp->v_frmpc = rvfp;
        }
        return(rvfp);
}
/*.s'vtfree'Return system bufs used in frame'*/
/*
 * vtfree - returns all system buffers used as 'frame pieces' within
 *          a single VT11 frame.
 */

vtfree(vfp)
register struct vtframe *vfp;
{
        register struct vtframe *nxp;

        do {
                vtinuse--;
                nxp = vfp->v_frmpc;
                brelse(vfp);
        } while(vfp = nxp);
}
/*.s'vtreplace'Replace a frame'*/
vtreplace(frp,frame)
struct vtframe *frp;
register int frame;
{
        register struct vtframp *vtfrp;
        register int    *jmpadr;

        vtfrp = &vt11.vtfrm[frame];
        if(frp)
                jmpadr = (caddr_t)frp->v_addr;
        else
                jmpadr = vtfrp+1;
        vtfrp->vtfradr = jmpadr;
        vtfrmfree(vt11h[frame].vfd_frm);
        vt11h[frame].vfd_frm = frp;
}
```

```
/*.t'vtcopy'Copy new frame buffer contents'*/
vtcopy(to,frame)
register int *to,frame;
{
        register int    count;

        count = u.u_count > (VTBUFL-2)*2 ? (VTBUFL-2)*2 : u.u_count;
        if(copyin(u.u_base,to,count) < 0)
                return(-1);
        u.u_base        =+ count;
        u.u_count       =- count;
        to =+ count/2;
        *to++ = VTDJMP;
        *to   = &vt11.vtfrm[frame+1];
        return(frame);
}
/*.s'vtint'VT11 interrupt handler'*/
vtint()
{

        register struct iobuf *vp;
        register int    *intp;

        vp = &vttab;
        if(vp->v_flags&VTBFREE) {
                vtfree(vp->v_frmpc);
                vp->v_flags =& ~VTBFREE;
                wakeup(&vp->v_frmpc);
        }
        vtpint();
        if(vp->v_flags&VTGO) {
                if(vp->v_flags&VTSTOP)
                        vp->v_flags =& ~VTGO;
                else
                        VTADDR->vtdpc =| 1;
        }
        if(vtwait || (vp->v_flags&VTSTOP))
                wakeup(vp);
}
```

```
/*      @(#)vt61.c      2.5     */

#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 mapvt61[] {
        0131,   LCA,
        0101,   CUP,
        0102,   CDN,
        0103,   CRI,
        0104,   CLE,
        0110,   HOME,
        0312,   STB,
        0352,   SPB,
        0110,   CS,
        0322,   CM,
        0306,   IL,
        0304,   DL,
        0317,   IC,
        0323,   DC,
        0113,   EEOL,
        0112,   EEOP,
/*      0xxx,   KBL,    */
/*      0xxx,   KBU,    */
        0121,   IL,                     /* Input only */
        0122,   EEOL,                   /* Input only */
        IL,     DVSCN,
        DL,     UVSCN,
        0
};

#define TDELAY  0203

int vt61speed 12;
char vt61str[] {' ', ESC, 'P', 0157, 010, 0};


vt61output(ac, atp)
struct tty *atp;
{
        register struct clist *qp;
        register char *cp;
        register c;

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

        for (cp=mapvt61; *cp++;)
        if (c == *cp++) {
                if (ac == DVSCN ) {
```

```
                ttdvscn(atp);
                break;
        }
        if (ac == UVSCN ) {
                ttuvscn(atp);
                break;
        }
        if (ac == STB)
                putc(' ', qp);
        putc(ESC, qp);
        if ((c=cp[-2]) < 0)
                if (ac == STB || ac == SPB || ac == CM)
                        putc('O', qp);
                else    putc('P', qp);
        qputc(c&0177, qp);
        switch(ac) {
                case LCA:
                        qputc(atp->t_row+32, qp);
                        qputc(atp->t_col+32, qp);
                        break;

                case IC:
                        for(cp=vt61str; *cp; cp++)
                                putc(*cp, qp);
                        break;

                case SPB:
                        putc(' ', qp);
                        break;

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

                case IL:
                case DL:
                case EEOL:

                        if ((atp->t_speeds.hibyte&017) >= vt61speed)
                                putc(QESC, qp);
                                putc(TDELAY, qp);
                        break;
        }
        break;
    }
}
vt61input(ac, atp)
struct tty *atp;
{
        register struct tty *tp;
        register c;
        register char *cp;

        c = ac;
        tp = atp;
```

```
        if (tp->t_tmflgs&TERM_BIT) {
                tp->t_tmflgs =& ~TERM_BIT;
                for (cp=mapvt61; *cp++;)
                if (c == (cp++)[-1])
                        return(cp[-1] | CPRES);
        } else if (c == ESC) {
                tp->t_tmflgs =| TERM_BIT;
                if ((tp->t_flags&RAW) == 0)
                        tp->t_state =| XMTSTOP;

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

vt61ioctl(tp, flag, nvrow)
register struct tty *tp;
{

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

```
/       @(#)vtast.s 2.3

/ This module contains assembly assist subroutines used by the VT11
/ system monitor package.  The routines are:
/
/       dpsub - subtract single precision from double precision.
/
/       call:  dpsub(add. of double, single prec value);
/
/       ----------
/
/       dpdiv - divide double precision by single precision.
/               result is returned, remainder is available in
/               dprem.
/
/       call:  dpdiv(add. of double, single prec value);
/
/       ----------

.globl _dpsub
_dpsub:
        mov     2(sp),r0                / r0 = address of dbl value
        sub     4(sp),2(r0)             / subtract single from low prec.
        sbc     (r0)                    / subtract borrow
        rts     pc

.globl _dpdiv,  _dprem
_dpdiv:
        mov     2(sp),r0                / r0 = address of dbl value
        mov     2(r0),r1                / r1 = low precision
        mov     (r0),r0                 / r0 = high precision
        div     4(sp),r0                / divide single into double
        mov     r1,_dprem
        rts     pc

.bss
_dprem: .=.+2
```

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

#include "sys/param.h"
#include "sys/vt11.h"
/*.n'vtdbg'Defines and global initializations'*/

#define DMISC   60                      /* # of debug slots to provide */
#define VTCHRHD 20
#define VTCHRWD 14

/*
 * External symbols
 */

extern struct vtfrpt vt11f;

/*
 * The following structure is used by system routines in debugging
 * states to display critical entries.  Vtflagent makes changes to
 * the entries in the displayed debugging information.
 */

struct miscent {
        int ms_point;                   /* VT11 positioning instruction */
        int ms_xpos;                    /* x position */
        int ms_ypos;                    /* y position */
        int ms_mod1;                    /* char mode instruction */
        char ms_name[7];                /* debug label */
        char ms_val[7];                 /* debug value (octal only) */
        int ms_fill[2];                 /* filler for display jump */
};

struct miscent ms_frms[DMISC];
/*.s'vtdbgin'Put debug lines in VT11 frame'*/
vtdbgin(framen)  {

        register char *sp,*ep;

        sp = &ms_frms;  ep = &ms_frms[DMISC];
        vt11f.vtfrmn = framen;
        vt11f.vtfrad = sp;
        vt11f.vtfrcn = (ep-sp) - 4;
        vtwrite();

}                               /* vtdbgin */
/*.s'vtdbgst'Initialize contents of the debug lines'*/
vtdbgst(aypos) {

        register struct miscent *msp;
        register int j,ypos;
        extern char *nulls;

        msp = ms_frms;  ypos = aypos;
        for(j=0;j<DMISC;j++) {
                msp->ms_point = VTPOINT;
```

```
                msp->ms_mod1  = VTCHAR | VTINT7;
                msp->ms_xpos = (j%5) * 14 * VTCHRWD;
                if((j%5) == 0) ypos -= VTCHRHD;
                msp->ms_ypos = ypos;
                vtccopy(nulls,msp->ms_name,7);
                vtccopy(nulls,msp->ms_val ,7);
                msp->ms_fill[0] = msp->ms_fill[1] = VTDNOP;
                msp++;
        }
        return(ypos);

}                               /* vtdbgst */
/*.s'vtmiscent'Insert debug information in debug line'*/
vtmiscent(flagn,name,value)  char *name;  {

        register struct miscent *msp;

        if(flagn < 0 || flagn >= DMISC) return;
        msp = &ms_frms[flagn];
        if(name) vtccopy(name,msp->ms_name,6);
        vtvconv(value,msp->ms_val,6,8);

}                               /* vtmiscent */
```

```
/*      @(#)vtlp.c      2.1     */
#include "sys/param.h"
#include "sys/user.h"
#include "sys/userx.h"
#include "sys/vtll.h"

/*.s'vtll'Light Pen Defines and Global Definitions'*/
struct lphtq {                          /* light pen state queue */
        int lp_state;                   /* state of light pen */
        int lp_xstrt;                   /* starting position of tracking */
        int lp_ystrt;
        int lp_xend;                    /* ending position of tracking */
        int lp_yend;
};

#define LPNQ            20              /* # of q entries */
#define LPSPRI          5               /* priority of light pen sleeps */
#define LPTRKON         1               /* tracking on */
#define LPTRKOF         2               /* tracking off */

struct lphtq vpll[LPNQ];                /* light pen hit queue */
int     vplli,vpllo;                    /* positions in hit queue */
int     lpstate,vprdlst;                /* state of light pen */
int     lpxp,lpxm,lpyp,lpym;            /* hit positions on light pen */
int     lpuser;                         /* lp opened for ?? -- see vtuser */

int lpframe[] {
        0117764,VTDJMP,0,0,0,VTSHORTV,
        024000,050000,VTDNOP,070000,VTDNOP,04120,040040,VTDNOP,
        040140,VTDSTOP,VTDJMP,0,
        014,041004,061010,062110,042114,043014,064020,064120,
        044124,045024,065030,066130,046134,047034,067040,070140,
        050144,051044,071060,074160,054164,055064,VTLONGV,060064,
        0100,060100,020100,040100,020120,040120,0120,060120,
        0140,060140,020140,040140,020140,040140,0140,060140,0140,
        VTDSTOP,0100100,VTDJMP,0
};

#define LPSKIPC         2               /* cross skip point */
#define LPCRX           3               /* x position of cross */
#define LPCRY           4               /* y position of cross */
#define LPXP            7               /* x hit on light pen cross */
#define LPXM            9               /* x hit on light pen cross */
#define LPYP            12              /* y hit on light pen cross */
#define LPYM            14              /* y hit on light pen cross */
#define LPCRND          16              /* stop at end of cross */
#define LPSKIPD         17              /* diamond skip point */
#define LPOFF           60              /* turn light pen off */
#define LPLASTJ         61              /* jump ins. to get out of lpframe */
#define LPLASTA         62              /* address to jump to */

extern struct vtcycl vtll;

/*.s'vtpint'pen interrupt routine, called from vtint'*/
vtpint()
{
```

```
        register int    *intp;

        if(lpstate == LPTRKON) {
                intp = VTADDR->vtdpc;
                switch(intp-lpframe) {

                        case LPCRND:
                                if(lpxp>=0 && lpxm>=0 && lpyp>=0 && lpym>=0) {
                                        lpframe[LPCRX] = (lpxp+lpxm)/2;
                                        lpframe[LPCRY] = (lpyp+lpym)/2;
                                        lpxp = lpxm = lpyp = lpym = -1;
                                }
                                else {
                                        lpframe[LPSKIPD] = &lpframe[LPSKIPD+1];
                                }
                                break;

                        case LPOFF:
                                lpframe[LPSKIPC] = &lpframe[LPLASTJ];
                                lpframe[LPSKIPD] = &lpframe[LPOFF];
                                vtlpqnt(LPTRKOF);
                        }
        }
}
/*.s'vtlp'light pen servicing routines'*/
vtlpopen(dev,flag)
{

        if(dev == NODEV)
                return;
        if(lpuser&VTUSER) {
                u.u_error = ENXIO;
                return;
        }

                                        /* add in special light pen frame */

        lpstate = LPTRKOF;      vplio = vplii = vprdlst = 0;
        lpframe[LPSKIPC] = &lpframe[LPLASTJ];
        lpframe[LPSKIPD] = &lpframe[LPOFF];
        lpframe[LPLASTA] = vtll.vtfrm;
        vtll.vtlpfr.vtfradr = lpframe;
        if((lpuser&(VTSYS|VTUSER)) == 0)
                lpuser = VTSYS;
        else
                lpuser = VTUSER;
}

vtlpclose(dev) {

        if(lpuser&VTUSER) {
                lpuser =& ~(VTUSER|VTSYS);
                vtlpopen(dev,2);
        } else {
                vtll.vtlpfr.vtfradr = vtll.vtfrm;
                lpuser = lpstate = 0;
```

```
        }
}
/*.s'vtlp'Make entry in light pen que'*/
vtlpqnt(flag) {

        register struct lphtq *lpq;
        register int    *stv;

        if(flag == LPTRKON) {
                vplli = (vplli+1) % LPNQ;
                if(vplli == vpllo)
                        vpllo = (vpllo+1) % LPNQ;
                vpll[vplli].lp_state = LPTRKON;
                lpstate              = LPTRKON;
                stv = &vpll[vplli].lp_xstrt;
        }
        else {
                vpll[vplli].lp_state = LPTRKOF;
                lpstate              = LPTRKOF;
                stv = &vpll[vplli].lp_xend;
        }
        *stv++ = lpframe[LPCRX];
        *stv   = lpframe[LPCRY];
        wakeup(vpll);

}
/*.s'vtlpint'Light Pen Interrupt Handling'*/
vtlpint() {

        register int *p;

        switch(lpstate) {

                case LPTRKOF:
                        lpxp = lpxm = lpyp = lpym = -1;
                        lpframe[LPCRX] = VTADDR->vtxsr&VTMAXX;
                        lpframe[LPCRY] = VTADDR->vtysr&VTMAXY;
                        lpframe[LPSKIPC] = &lpframe[LPSKIPC+1];
                        vtlpqnt(LPTRKON);
                        break;

                case LPTRKON:
                        p = VTADDR->vtdpc;
                        switch(p-lpframe) {

                                case LPXP:  case LPXP+1:
                                        lpxp = VTADDR->vtxsr&VTMAXX;
                                        break;

                                case LPXM:  case LPXM+1:
                                        lpxm = VTADDR->vtxsr&VTMAXX;
                                        break;

                                case LPYP:  case LPYP+1:
                                        lpyp = VTADDR->vtysr&VTMAXY;
                                        break;
```

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

#include "sys/param.h"
#include "sys/user.h"
#include "sys/userx.h"
#include "sys/proc.h"
#include "sys/procx.h"
#include "sys/text.h"
#include "sys/textx.h"
#include "sys/vtll.h"
#include "sys/vtmn.h"

/*.n'vtmon'Defines and global initializations'*/
/*
 * External symbols
 */

extern struct vtfrpt vtllf;

/*
 * Defines and Structure definitions
 */

#define UMODE   0170000                 /* ps mode bits for user */
#define DPROCS  38                      /* # of proc entries to display */
#define DSSIZE  DPROCS+1                /* # of proc display frames */
#define DOFLE   NPROC-(DPROCS+1)        /* last element in pr_ofls array */
#define VTCHRHD 20                      /* display height of chars */
#define VTCHRWD 14                      /* display width of chars */

/*
 * The following structure defines the positions on the VTll screen in
 * which the proclin entries (following structure) will appear, i.e.
 * a jump to the proclin occurs from one of these structures and the
 * proclin is filled in with a jump back to the following dsfrms structure.
 */

struct dsfrms {
        int ds_point;                   /* VTll positioning instruction */
        int ds_xpos;                    /* x position of this line */
        int ds_ypos;                    /* y position of this line */
        int ds_djmp;                    /* VTll jump instruction */
        struct dsfrms *ds_djad;         /* jump to proc line entry */
};

/*
 * The following structure is the set-up for the process display area.
 * It contains the header display, a mode set for the process display
 * and a jump to the display frames.
 */

struct prochdr {
        int ps_modl;
        char ps_title[62];              /* process display title */
        int ps_fill[2];                 /* space for return jump */
```

```
};

/*
 * The following defines the structure and control flags for vtprocent
 * which controls the proc frame entries being displayed on the VT11.
 */

struct proclin {
        int pr_mod;                     /* process mode (character) */
        int pr_iton;                    /* Load A italics instruction */
        char pr_state[2];               /* state of proc */
        char pr_flag[3];                /* flag of proc */
        char pr_wchan[7];               /* wait channel of proc (sleep) */
        char pr_sig[3];                 /* next sig to process */
        char pr_pri[5];                 /* priority of proc */
        char pr_ptim[4];                /* resident time */
        char pr_ctim[4];                /* length of time in current state*/
        char pr_clock[6];               /* process clock */
        char pr_group[7];               /* process group */
        char pr_pid[6];                 /* process id */
        char pr_ppd[6];                 /* process parent id */
        char pr_size[5];                /* size of process in 32 word blks */
        char pr_name[15];               /* process name + some args */
        int pr_itof;                    /* italics off instruction */
        int pr_fill[2];                 /* filler for frame jumps */
};

/*
 * The following defines control the actions of vthdent.  vthdent makes
 * changes to the header line of the proc state table.
 */

struct headlin {
        int hd_point;                   /* VT11 positioning instruction */
        int hd_xpos;                    /* x position of header line */
        int hd_ypos;                    /* y position of header line */
        int hd_mod1;                    /* char mode for header line */
        char hd_unix[40];               /* current UNIX version */
        int hd_pnt2;                    /* position time */
        int hd_tmx;                     /* time x position */
        int hd_tmy;                     /* time y position */
        int hd_mod2;                    /* character mode for time */
        char hd_time[10];               /* current system time */
        int hd_fill[2];                 /* filler for frame jump */
};

/*
 * The following structure lays out the system info line.
 */

struct infolin {
        int in_point;                   /* VT11 positioning instruction */
        int in_xpos;
        int in_ypos;
        int in_mod1;
        char in_nproc[3];               /* # of free procs */
```

```
        char in_lproc[6];
        char in_ntext[3];               /* # of free texts */
        char in_ltext[6];               /* text label */
        int in_fill[2];
};

int proflnx,proflls;
struct dsfrms   ds_frms[DSSIZE+1];
struct dsfrms   *pr_ofls[NPROC-DPROCS];
struct proclin  pr_frms[NPROC];
struct prochdr  pr_title;
struct headlin  hd_fram;
struct infolins in_fram[1];

int vlstbnk;                            /* current active proc number */
int vframen;                            /* next available frame */
int vtrestart;                          /* restart flag */

char *nulls "";

char *pr_states[] {
        " ","s","w","r","i","z","t",
};

char *hd_types[] {
        " s ","u  "," i",
};
/*.s'vtinit'start/restart VT11 monitor'*/
vtinit() {

        register struct headlin *hdp;
        register char *sp,*ep;

        vframen = 0;
        if(vtrestart == 0) {
                vtstrt();
                vtrestart = 1;
        }
        hdp = &hd_fram;
        sp = hdp;  ep = hdp+1;
        vt11f.vtfrmn = vframen++;
        vt11f.vtfrad = sp;
        vt11f.vtfrcn = (ep-sp) - 4;
        vtwrite();

        sp = in_fram;  ep = &in_fram[1];
        vt11f.vtfrmn = vframen++;
        vt11f.vtfrad = sp;
        vt11f.vtfrcn = (ep-sp) - 4;
        vtwrite();

        sp = ds_frms;  ep = &ds_frms[DSSIZE+1];
        vt11f.vtfrmn = vframen++;
        vt11f.vtfrad = sp;
        vt11f.vtfrcn = (ep-sp) - 4;
        vtwrite();
```

```
#ifdef VTDBUG
        vtdbgin(vframen++);             /* insert debug lines */
#endif
}
/*.s'vtstrt'initialize all process state display frames'*/
vtstrt()
{
        register int j,ypos;
        register int *txp;

        /*
         * Start at bottom of line 1 (upper left corner).
         */

        ypos = VTMAXY - VTCHRHD;
        txp = &hd_fram;

        /*
         * Put in the UNIX message.
         */

        txp->hd_point = VTPOINT;
        txp->hd_xpos  = 0;
        txp->hd_ypos  = ypos;
        txp->hd_mod1  = VTCHAR | VTINT7;
        vtccopy("CB-UNIX Release 2.1", txp->hd_unix, sizeof(txp->hd_unix));

        /*
         * Put in time message.
         */

        txp->hd_pnt2 = VTPOINT;
        txp->hd_tmx  = 56 * VTCHRWD;
        txp->hd_tmy  = ypos;
        txp->hd_mod2 = VTCHAR | VTINT7;
        vtccopy("0000:00:00", txp->hd_time, sizeof(txp->hd_time));

        /*
         * Initialize UNIX info line(s).
         */

        ypos = vtinfinit(ypos);

        /*
         * Initialize process display.
         */

        ypos = vtprinit(ypos);

        /*
         * Initialize debug display.
         */

#ifdef VTDBUG
        ypos = vtdbgst(ypos-2*VTCHRHD);
```

```
                case PR_CLOCK:
                        vtvconv(prp->p_clktim,pcp->pr_clock,5,10);
                        return;

                case PR_GROUP:
                        vtvconv(prp->p_pgrp,pcp->pr_group,6,10);
                        return;

                case PR_PID:
                        vtvconv(prp->p_pid,pcp->pr_pid,5,10);
                        return;

                case PR_PPD:
                        vtvconv(prp->p_ppid,pcp->pr_ppd,5,10);
                        return;

                case PR_NAME:
                        if (prefix("getty", pcp->pr_name))
                                vtaddpr(pcp);
                        vtccopy(vtlstpath(value,valu2),pcp->pr_name,15);
                        if (prefix("getty", pcp->pr_name))
                                vtrmvpr(pcp);
                        return;

                case PR_SWH:
                        vtccopy(vtflag(prp->p_flag),pcp->pr_flag,2);
                        vtccopy(pr_states[prp->p_stat],pcp->pr_state,1);
                        vtsconv(prp->p_pri,pcp->pr_pri,4,10);
                        vtvconv(prp->p_wchan,pcp->pr_wchan,6,8);
                        vtvconv(prp->p_ctime,pcp->pr_ctim,3,10);
                        return;

                case PR_WKP:
                        vtccopy(pr_states[prp->p_stat],pcp->pr_state,1);
                        vtvconv(prp->p_wchan,pcp->pr_wchan,6,8);
                        vtvconv(prp->p_ctime,pcp->pr_ctim,3,10);
                        return;

                case PR_NEW:
                        vtccopy(pr_states[SRUN],pcp->pr_state,1);
                        vtccopy(vtflag(prp->p_flag),pcp->pr_flag,2);
                        vtvconv(prp->p_pid,pcp->pr_pid,5,10);
                        vtvconv(prp->p_ppid,pcp->pr_ppd,5,10);
                        vtvconv(prp->p_time,pcp->pr_ptim,3,10);
                        vtvconv(prp->p_size,pcp->pr_size,4,8);
                        vtvconv(prp->p_pgrp,pcp->pr_group,6,10);
                        vtccopy(pr_frms[value-proc].pr_name, pcp->pr_name, 15);
                        if(!prefix("getty", pcp->pr_name));
                                vtaddpr(pcp);
                        return;

                case PR_PSIG:
                        vtvconv(fsig(prp),pcp->pr_sig,2,10);
                        vtvconv(prp->p_wchan,pcp->pr_wchan,6,8);
                        vtccopy(pr_states[prp->p_stat],pcp->pr_state,1);
                        return;
```

```
                case PR_SWP:
                        vtvconv(prp->p_time,pcp->pr_ptim,3,10);
                        vtccopy(vtflag(prp->p_flag),pcp->pr_flag,2);
                        return;

                case PR_BNKOF:
                        pr_frms[vlstbnk].pr_iton = VTSTATSA|VTITAL0;
                        return;


                case PR_SIZE:
                        vtvconv(prp->p_size,pcp->pr_size,4,8);
                        return;
        }
}


/*.t'vtflag'Convert process flag to display'*/
vtflag(flag)
register int flag;
{
        static struct {
                char lochar;
                char hichar;
        } a;
        register int two;

        if(flag&SLOCK)
                a.lochar = 'L';
        else if(flag&SLOAD)
                a.lochar = ' ';else
                a.lochar = '*';

        two = 1;
        if(flag&SSYS)   two =+ 0;
        if(flag&SSWAP)  two =+ 1;
        if(flag&STRC)   two =+ 2;
        if(flag&SWTED)  two =+ 4;

        a.hichar = "U DT3W5Q7S9abcdef"[two];

        return(&a);
}
/*.s'vtaddpr'Find a free line to display proc'*/
vtaddpr(pcp)
register struct proclin *pcp;
{
        register struct dsfrms *dsp;

        if(dsp=vtfreelin()) {
                pcp->pr_fill[1] = dsp+1;
                dsp->ds_djad = pcp;
                return;
        }
        pr_ofls[proflnx++] = pcp;
```

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

#
/*.n'vtmon'utility conversion and copy routines'*/
vtccopy(from,to,count)  char *from,*to;  {

        register char *fp,*tp;
        register int  n;

        fp = from;  tp = to;  n = count;

        while(n-- && *fp) *tp++ = *fp++;
        while(n-- >= 0)  *tp++ = ' ';
}                                       /* vtccopy */

vtvconv(val,to,ndig,base)  char *to;  {

        register char *tp;
        register int n;
        register unsigned v;

        v = val;  n = ndig;  tp = to+n-1;
        while(n-- && v) {
                *tp-- = (v%base) + '0';
                v =/ base;
        }
        if(n == ndig-1) {
                n--;
                *tp-- = '0';
        }
        while(n-- >= 0) *tp-- = ' ';
}                                       /* vtvconv */

vtsconv(val,to,ndig,base)  char *to;  {

        register char *tp;
        register int  v,n;
        int sign;

        if((v=val) < 0) {
                sign = 1;  v = -v;
                n = ndig-1;
        }
        else {
                sign = 0;
                n = ndig;
        }
        tp = to + ndig - 1;
        while(n-- && v) {
                *tp-- = v%base + '0';
                v =/ base;
        }
        if(sign) *tp-- = '-';
        while(n-- >= 0) *tp-- = ' ';
}                                       /* vtsconv */
```

```
/*.e*/
vtlstpath(stringp,nchars)  char *stringp;  {

        static char name[15];
        register int  n;
        register char *cp,*lp;

        cp = lp = stringp;  n = nchars;
        while(*cp && n--) if(*cp++ == '/') lp=cp;
        n = nchars - (lp-stringp);
        n = (n > 15) ? 15 : n;  cp = name;
        while(n--) {
                if(*lp < ' ' || *lp > 0176) *cp++ = ' ';
                else                        *cp++ = *lp;
                lp++;
        }
        *cp = 0;
        return(name);
}                                       /* vtlstpath */

/* Is arg1 a prefix of arg2 ? */
prefix(a, b)
register char *a, *b;
{
        while(*a++ == *b++);
        if(*--a == 0)
                return(1);
        return(0);
}
```