```
/*      @(#)vtmn.h      2.4     */

#define PR_CLEAR        0               /* clear code for proclin */
#define PR_BLINK        1               /* blink proc name */
#define PR_STATE        2               /* proc state */
#define PR_FLAG         3               /* proc flag */
#define PR_WCHAN        4               /* proc wchan (sleep) */
#define PR_SIG          5               /* proc caught signal entry */
#define PR_PRI          6               /* proc priority */
#define PR_PTIM         7               /* proc time on current medium */
#define PR_CTIM         8               /* proc time in current state */
#define PR_CLOCK        9               /* proc controlling tty entry */
#define PR_GROUP        10              /* process group */
#define PR_PID          11              /* proc id entry */
#define PR_PPD          12              /* proc parent id entry */
#define PR_NAME         13              /* process name */
#define PR_SWH          14              /* proc going to switch */
#define PR_WKP          15              /* proc waking up */
#define PR_NEW          16              /* new process */
#define PR_PSIG         17              /* signal caught */
#define PR_SWP          18              /* swap call */
#define PR_BNKOF        19              /* no one active */
#define PR_SIZE         20              /* proc size */

#define HD_CLEAR        0               /* clear header line */
#define HD_TYPE         1               /* CPU owner entry */
#define HD_TIME         2               /* current time */
#define HD_IDLE         3               /* idle header */
#define HD_SWPON        4               /* swap in progress */
#define HD_SWPOF        5               /* swap done */
#define HD_BUFON        6               /* buffer in use */
#define HD_BUFOF        7               /* buffer freed */
#define HD_SYSA         8               /* system active */

#ifdef VTMON
#define VTTINC()                vttinc()
#define VTPROCENT(one,two)              vtprocent(one,two)
#define VTNEWPROC(one,two,three,four)   vtprocent(one,two,three,four)
#define VTOPEN()                        vtopen()
#define VTMISCENT(one,two,three)        vtmiscent(one,two,three)
#endif

#ifndef VTMON
#define VTTINC()
#define VTNEWPROC(one,two,three,four)
#define VTPROCENT(one,two)
#define VTOPEN()
#define VTMISCENT(one,two,three)        three;
#endif
```

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

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

/*
 * Perform process accounting functions.
 */

#ifdef SYSACCT
sysacct()
{
        extern uchar();
        register struct inode *ip;
        register fmt;

        if (suser()) {
                if (u.u_arg[0]==0) {
                        if (acctp) {
                                plock(acctp);
                                iput(acctp);
                                acctp = NULL;
                        }
                        return;
                }
                if (acctp) {
                        u.u_error = EBUSY;
                        return;
                }
                u.u_dirp = u.u_arg[0];
                if ((ip = namei(&uchar, 0))==NULL)
                        return;
                fmt = ip->i_mode & IFMT;
                if((fmt != IFREG) && (fmt != IFLRG)) {
                        u.u_error = EACCES;
                        iput(ip);
                        return;
                }
                acctp = ip;
                prele(ip);
        }
}

/*
 * On exit, write a record on the accounting file.
 */
acct()
{
        register struct inode *ip;
        register i;
        off_t siz;
	if ((ip=acctp)==NULL)
		return;
	plock(ip);
	for (i=0; i<DIRSIZ; i++)
		acctbuf.ac_comm[i] = u.u_comm[i];
	acctbuf.ac_flag = u.u_acflag;
	acctbuf.ac_uid = u.u_ruid;
	acctbuf.ac_date = u.u_start;
	acctbuf.ac_etime = time - u.u_start;
	acctbuf.ac_utime = u.u_utime;
	acctbuf.ac_stime = u.u_stime;
	acctbuf.ac_dread = u.u_dread;
	acctbuf.ac_dwrit = u.u_dwrit;
	siz.hiword = ip->i_size0&0377;
	siz.loword = ip->i_size1;
	u.u_offset = siz;
	u.u_base = (caddr_t)&acctbuf;
	u.u_count = sizeof(acctbuf);
	u.u_segflg = 1;
	u.u_error = 0;
	writei(ip);
	if(u.u_error) {
		ip->i_size0 = siz.hiword;
		ip->i_size1 = siz.loword;
	}
	prele(ip);
}
#endif

#ifndef SYSACCT
sysacct() { nodev(); }

acct() { }
#endif
```

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

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


/*
 * iinit is called once (from main)
 * very early in initialization.
 * It reads the root's super block
 * and initializes the current date
 * from the last modified date.
 *
 * panic: iinit -- cannot read the super
 * block. Usually because of an IO error.
 */
iinit()
{
        register *cp, *bp;

        (*bdevsw[rootdev.d_major].d_open)(rootdev, 1);
        (*bdevsw[swapdev.d_major].d_open)(swapdev, 1);
        bp = bread(rootdev, 1);
        cp = malloc(swapmap, 1);
        cp = agetblk(swapdev, cp);
        if(u.u_error)
                panic("iinit");
        copyio(paddr(bp), (caddr_t)cp->b_paddr, BSIZE, U_RKD);
        brelse(bp);
        mount[0].m_bufp = cp;
        mount[0].m_dev = rootdev;
        mount[0].m_flags = M_INCOR;
        suincnt++;
        cp = (caddr_t)cp->b_paddr;
        cp->s_flock = 0;
        cp->s_ilock = 0;
        cp->s_ronly = 0;
        time = cp->s_time;
}

/*
 * alloc will obtain the next available
 * free disk block from the free list of
 * the specified device.
 * The super block has up to 100 remembered
 * free blocks; the last of these is read to
 * obtain 100 more . . .
 *
 * no space on dev x/y -- when
 * the free list is exhausted.
 */
alloc(dev)
{
        register bno;
        register *bp, *fp;

        fp = getfs(dev);
        while(fp->s_flock)
                sleep(&fp->s_flock, PINOD);
        do {
                if(fp->s_nfree <= 0)
                        goto nospace;
                bno = fp->s_free[--fp->s_nfree];
                if(bno == 0)
                        goto nospace;
        } while (badblock(fp, bno, dev));
        if(fp->s_nfree <= 0) {
                fp->s_flock++;
                bp = bread(dev, bno);
                fp->s_nfree = xget(paddr(bp));
                copyio(paddr(bp)+2, fp->s_free, 200, U_RKD);
                brelse(bp);
                fp->s_flock = 0;
                wakeup(&fp->s_flock);
        }
        bp = getblk(dev, bno);
        clear(paddr(bp), BSIZE);
        fp->s_fmod = 1;
        putfs(fp);
        return(bp);

nospace:
        fp->s_nfree = 0;
        putfs(fp);
        prdev(E_FSNS, dev);
        u.u_error = ENOSPC;
        return(NULL);
}

/*
 * place the specified disk block
 * back on the free list of the
 * specified device.
 */
free(dev, bno)
register bno;
{
        register *fp, *bp;

        fp = getfs(dev);
        fp->s_fmod = 1;
	while(fp->s_flock)
		sleep(&fp->s_flock, PINOD);
	if (badblock(fp, bno, dev))
		goto ret;
	if(fp->s_nfree <= 0) {
		fp->s_nfree = 1;
		fp->s_free[0] = 0;
	}
	if(fp->s_nfree >= 100) {
		fp->s_flock++;
		bp = getblk(dev, bno);
		xput(paddr(bp), fp->s_nfree);
		copyio(paddr(bp)+2, fp->s_free, 200, U_WKD);
		fp->s_nfree = 0;
		bwrite(bp);
		fp->s_flock = 0;
		wakeup(&fp->s_flock);
	}
	fp->s_free[fp->s_nfree++] = bno;
	fp->s_fmod = 1;
ret:
	putfs(fp);
}

/*
 * Check that a block number is in the
 * range between the I list and the size
 * of the device.
 * This is used mainly to check that a
 * garbage file system has not been mounted.
 *
 * bad block on dev x/y -- not in range
 */
badblock(afp, abn, dev)
{
	register struct filsys *fp;
	register char *bn;

	fp = afp;
	bn = abn;
	if (bn < fp->s_isize+2 || bn >= fp->s_fsize) {
		prdev(E_FSBB, dev);
		return(1);
	}
	return(0);
}

/*
 * Allocate an unused I node
 * on the specified device.
 * Used with file creation.
 * The algorithm keeps up to
 * 100 spare I nodes in the
 * super block. When this runs out,
 * a linear search through the
 * I list is instituted to pick
 * up 100 more.
 */
ialloc(dev)
{
        register *fp, *bp, *ip;
        int i, j, k, ino;

        fp = getfs(dev);
        while(fp->s_ilock)
                sleep(&fp->s_ilock, PINOD);
loop:
        if(fp->s_ninode > 0) {
                ino = fp->s_inode[--fp->s_ninode];
                ip = iget(dev, ino);
                if (ip==NULL)
                        goto ret;
                if(ip->i_mode == 0) {
                        for(bp = &ip->i_mode; bp < &ip->i_un.i_addr[8];)
                                *bp++ = 0;
                        fp->s_fmod = 1;
                        putfs(fp);
                        return(ip);
                }
                /*
                 * Inode was allocated after all.
                 * Look some more.
                 */
                iput(ip);
                goto loop;
        }
        fp->s_ilock++;
        ino = 0;
        for(i=0; i<fp->s_isize; i++) {
                bp = abread(dev, i+2);
                ip = (caddr_t)bp->b_paddr;
                for(j=0; j<256; j=+16) {
                        ino++;
                        if(ip[j] != 0)
                                continue;
                        for(k=0; k<NINODE; k++)
                        if(dev==inode[k].i_dev && ino==inode[k].i_number)
                                goto cont;
                        fp->s_inode[fp->s_ninode++] = ino;
                        if(fp->s_ninode >= 100)
                                break;
                cont:;
                }
                brelse(bp);
                if(fp->s_ninode >= 100)
                        break;
        }
        fp->s_ilock = 0;
        wakeup(&fp->s_ilock);
        if (fp->s_ninode > 0)
                goto loop;
        prdev(E_FSOI, dev);
        u.u_error = ENOSPC;
ret:
        putfs(fp);
        return(NULL);
}

/*
 * Free the specified I node
 * on the specified device.
 * The algorithm stores up
 * to 100 I nodes in the super
 * block and throws away any more.
 */
ifree(dev, ino)
{
        register *fp;

        fp = getfs(dev);
        if(fp->s_ilock)
                goto ret;
        if(fp->s_ninode >= 100)
                goto ret;
        fp->s_inode[fp->s_ninode++] = ino;
        fp->s_fmod = 1;
ret:
        putfs(fp);
}

/*
 * getfs maps a device number into
 * a pointer to the incore super
 * block.
 * If the super block is not in core, getfs will bring it in
 * from the swapdevice.
 * The algorithm is a linear
 * search through the mount table.
 * A consistency check of the
 * in core free-block and i-node
 * counts.
 *
 * bad count on dev x/y -- the count
 *      check failed. At this point, all
 *      the counts are zeroed which will
 *      almost certainly lead to "no space"
 *      diagnostic
 * panic: no fs -- the device is not mounted.
 *      this "cannot happen"
 */
getfs(dev)
{
        register struct mount *p;
        register char *n1, *n2;

        for(p = &mount[0]; p < &mount[NMOUNT]; p++)
loop:
        if(p->m_bufp != NULL && p->m_dev == dev) {
```

```
		if((p->m_flags&M_INCOR)==0) {
			spl6();
			if(p->m_flags & M_GET) {
				p->m_flags =| M_WANT;
				sleep(p, PRIBIO);
				spl0();
				goto loop;
			}
			p->m_flags =| M_GET;
			spl0();
			if (suincnt >= NBUF-NFBUF)
				update();
			p->m_bufp = abread(swapdev, p->m_bufp);
			p->m_flags =| M_INCOR;
			p->m_flags =& ~M_GET;
			if(p->m_flags & M_WANT) {
				p->m_flags =& ~M_WANT;
				wakeup(p);
			}
			suincnt++;
		}
		p->m_refc++;
		p = (caddr_t)p->m_bufp->b_paddr;
		n1 = p->s_nfree;
		n2 = p->s_ninode;
		if(n1 > 100 || n2 > 100) {
			prdev(E_FSBC, dev);
			p->s_nfree = 0;
			p->s_ninode = 0;
		}
		return(p);
	}
	panic("no fs");
}

/*
 * putfs decrements the reference count for the superblock
 * returned by getfs. Thus any call to getfs must be followed by
 * a call to putfs. Update will not flush out to swapdev any
 * superblocks with a nonzero reference count.
 */
putfs(bp)
register char *bp;
{
	register struct mount *mp;

	for(mp = &mount[0]; mp < &mount[NMOUNT]; mp++)
		if ( (mp->m_bufp != NULL)
		  && (mp->m_flags&M_INCOR)
		  && ((caddr_t)mp->m_bufp->b_paddr == bp) ) {
			mp->m_refc--;
			return;
		}
	panic("putfs");
}
```

```
/*
 * rofs is used to determine if a file system is read only.
 * It returns a nonzero value if this is the case, zero otherwise.
 */
rofs(dev)
{
        register rdev;
        register struct mount *p;

        rdev = dev;
        for(p = &mount[0]; p < &mount[NMOUNT]; p++)
                if(p->m_bufp != NULL && p->m_dev == rdev)
                        return(p->m_flags&M_RONLY);
        panic("no fs");
}
/*
 * update is the internal name of
 * 'sync'. It goes through the disk
 * queues to initiate sandbagged IO;
 * goes through the I nodes to write
 * modified nodes; goes through
 * the mount table to initiate modified
 * super blocks; and flushes stale superblocks to
 * the swap device.
 */
update()
{
        register struct inode *ip;
        register struct mount *mp;
        register *bp;

        if(updlock)
                return;
        updlock++;
        bflush(NODEV);
        for(mp = &mount[0]; mp < &mount[NMOUNT]; mp++)
                if(mp->m_bufp != NULL && mp->m_flags&M_INCOR) {
                        ip = (caddr_t)mp->m_bufp->b_paddr;
                        if(ip->s_ilock!=0 || ip->s_flock!=0)
                                continue;
                        if(ip->s_fmod==0) {
                                if(mp->m_refc != 0)
                                        continue;
                                bp = mp->m_bufp->b_blkno;
                                bdwrite(mp->m_bufp);
                                mp->m_bufp = bp;
                                mp->m_flags =& ~M_INCOR;
                                suincnt--;
                                continue;
                        }
                        bp = getblk(mp->m_dev, 1);
                        ip->s_fmod = 0;
                        ip->s_time = time;
                        copyio(paddr(bp), ip, BSIZE, U_WKD);
                        bwrite(bp);
                }
	for(ip = &inode[0]; ip < &inode[NINODE]; ip++)
		if((ip->i_flag&ILOCK)==0 && ip->i_count) {
			ip->i_flag =| ILOCK;
			ip->i_count++;
			iupdat(ip, &time);
			iput(ip);
		}
	updlock = 0;
}
```

```
/*      @(#)clock.c     2.17    */

#
/*
 *      Copyright 1973 Bell Telephone Laboratories Inc
 */

#include "sys/param.h"
#include "sys/systm.h"
#include "sys/user.h"
#include "sys/userx.h"
#include "sys/proc.h"
#include "sys/procx.h"
#include "sys/vtmn.h"
#include "sys/sysmes.h"
#include "sys/sysmesx.h"
#ifdef SPROF
#include "sys/sprof.h"
#include "sys/sprofx.h"
#include "sys/seg.h"
#endif

#define UMODE   0170000
#define CSW     0177570

/*
 * clock is called straight from
 * the real time clock interrupt.
 *
 * Functions:
 *      reprime clock
 *      copy *switches to display
 *      implement callouts
 *      maintain user/system times
 *      maintain date
 *      profile
 *      tout wakeup (sys sleep)
 *      lighting bolt wakeup (every 4 sec)
 *      jab the scheduler
 *      clk_fn may be temporarily changed by power fail stuff
 */
extern char waitloc;
extern char idle;

extern  int     clock();
int     (*clk_fn)()     &clock;
clock(dev, sp, r1, nps, r0, pc, ps)
{
        register struct callo *p2;
        register struct proc *pp;
        register a;
        extern *pirr;
        int  rqlen, sqlen;

        /*
```

```
	 * restart clock
	 */

	*lks = 0115;

	/*
	 * display register
	 */
 18(lbolt & 011)
	display();

	/*
	 * callouts
	 * if none, just return
	 * else update first non-zero time
	 */

	if(callout[0].c_func == 0)
		goto out;
	p2 = &callout[0];
	while(p2->c_time<=0 && p2->c_func!=0)
		p2++;
	p2->c_time--;

	/*
	 * Check for callouts.
	 * Process if ps is not high,
	 * otherwise set up programmed interrupt
	 */

	if (callout[0].c_time <= 0)                 88 Func != 0
		if ((ps&0340) != 0)
			*pirr =| 1 << (1+8);	/* PIRQ 1 */
		else
			pir(0);

	/*
	 * lightning bolt time-out
	 * and time of day
	 */

out:
	a = dk_busy&(~(-1<<NIOSTAT));

#ifdef SPROF
#ifndef IPROFCLK
#ifndef IPROFCLB
	if (sysprof.pid)
		sincupc(dev,sp,r1,nps,r0,pc,ps);
#endif
#endif
#endif

	if((ps&UMODE) == UMODE) {
		u.u_utime++;
		if(u.u_prof[3])
```

```
                        incupc(pc, u.u_prof);
                if(u.u_procp->p_nice > 0)
                        a =+ (1<<NIOSTAT);
        } else {
                a =+ (2<<NIOSTAT);
                if(pc == &waitloc)
                        a =+ (1<<NIOSTAT);
                if((ps&0340) == 0)
                        u.u_stime++;
        }
        dk_time[a] =+ 1;
        pp = u.u_procp;
        if(++pp->p_cpu == 0)
                pp->p_cpu--;
        if(++lbolt >= HZ) {
                if((ps&0340) != 0)
                        return;
                lbolt =- HZ;
                ++time;
                VTTINC();
                spl1();
                if((time.loword&03) == 0)
                        wakeup(&lbolt);

                rqlen = 0;
                sqlen = 0;
                for(pp = &proc[0]; pp < procend; pp++)
                if(pp->p_stat) {
                        if(pp->p_time != 127) {
                                pp->p_time++;
                                VTPROCENT(pp,PR_PTIM);
                        }

                        if(pp->p_ctime != 127) {
                                pp->p_ctime++;
                                VTPROCENT(pp,PR_CTIM);
                        }

                        if(pp->p_clktim)  {
                                if(--pp->p_clktim == 0) {
                                                psignal(pp,SIGCLK);
                                }
                                VTPROCENT(pp,PR_CLOCK);
                        }

                        a = (pp->p_cpu & 0377)*8/10 + pp->p_nice;
                        if(a < 0)
                                a = 0;
                        if(a > 255)
                                a = 255;
                        pp->p_cpu = a;
                        if(pp->p_pri >= PUSER)
                                setpri(pp);

                        /* Calculate queue lengths */
                        if(pp->p_stat == SRUN)
```

```
                                if(pp->p_flag & SLOAD)
                                        rqlen++;
                                else
                                        sqlen++;
                }

                /* Update global queue counters */
                if(dqlen) {
                        meas.m_dqlen =+ dqlen;
                        meas.m_dqocc++;
                }
                if(rqlen) {
                        meas.m_rqlen =+ rqlen;
                        meas.m_rqocc++;
                }
                if(sqlen) {
                        meas.m_sqlen =+ sqlen;
                        meas.m_sqocc++;
                }

                if(runin!=0) {
                        runin = 0;
                        wakeup(&runin);
                }
                if((ps&UMODE) == UMODE) {
                        u.u_ar0 = &r0;
                        if(issig())
                                psig();
                        setpri(u.u_procp);
                }
                runrun++;
        }
}

/*
 * timeout is called to arrange that
 * fun(arg) is called in tim/HZ seconds.
 * An entry is sorted into the callout
 * structure. The time in each structure
 * entry is the number of HZ's more
 * than the previous entry.
 * In this way, decrementing the
 * first entry has the effect of
 * updating all entries.
 */
timeout(fun, arg, tim)
{
        register struct callo *p1, *p2;
        register t;
        int s;

        t = tim;
        p1 = &callout[0];
        s = spl7();
        while(p1->c_func != 0 && p1->c_time <= t) {
                t =- p1->c_time;
```

```
		pl++;
	}
	pl->c_time =- t;
	p2 = pl;
	while(p2->c_func != 0)
		p2++;
	if (p2 >= &callout[NCALL-2])	/* kiltout assumes last entry is 0 */
		panic("Timeout table overflow");
	while(p2 >= pl) {
		(p2+1)->c_time = p2->c_time;
		(p2+1)->c_func = p2->c_func;
		(p2+1)->c_arg = p2->c_arg;
		p2--;
	}
	pl->c_time = t;
	pl->c_func = fun;
	pl->c_arg = arg;
	splx(s);
}

/*
 * Programmed interrupt request; Process callouts
 * pir_fn may be temporarily changed by power fail stuff
 */

int	(*pir_fn)()	&pir;
pir(pri)
{
	register struct callo *pl, *p2;
	extern *pirr;

	if (pri)
		pirr->hibyte =& ~(1 << ((*pirr & 016) >> 1));
	spl5();
	pl = &callout[0];
	while(pl->c_time <= 0 && pl->c_func) {
		(*pl->c_func) (pl->c_arg);

		p2 = pl;
		while(p2->c_func = (++pl)->c_func) {
			p2->c_time = pl->c_time;
			p2->c_arg = pl->c_arg;
			p2++;
		}
	}
}

/*
 *	kiltout() is a function which removes
 *	timeout entries from the callout table.  This
 *	function removes entries made by timeout().
 *	the parameters are:
 *		fun - address of the wakeup function
 *		arg - the argument for the function
 *	if the wakeup call is made to the function, the
 *	entry is removed by pir().
```

?

What is timeout is called from spl7 interrupt routine?

```
 */

kiltout(afun,aarg)
int (*afun)();
{

        register struct callo *p1;
        register int (*fun)();
        register int arg;
        int s;

        p1 = &callout[0];
        fun = afun;
        arg = aarg;
        s = spl7();

        /*
         * Look through callout table for a match...
         */

        while(p1->c_func != fun || p1->c_arg != arg) {
                if(p1->c_func == 0) {
                        splx(s);
                        return(-1);
                }
                p1++;
        }

        /*
         * Now reset table without entry...
         */

        (p1+1)->c_time =+ p1->c_time;
        while((p1+1)->c_func != 0) {
                p1->c_time = (p1+1)->c_time;
                p1->c_func = (p1+1)->c_func;
                p1->c_arg  = (p1+1)->c_arg;
                p1++;
        }
        p1->c_func = 0;
        splx(s);
        return(0);
}

#ifdef SPROF
#define KDSD    0172320
sincupc (dev,sp,r1,nps,r0,pc,ps)

/*      sincupc is called directly if an independent profiling
 *      clock has been generated in the system.
 *      otherwise, it is called from the normal system clock routine
 *      if anyone is profiling.
 */

{
        register struct sysprof *tp;
	register int t;

	tp = &sysprof;

#ifdef IPROFCLK
	if (tp->pid==0) {
		KW11K->kw11ks = 0;	/* turn off independent k clock */
		return;
	}
#else
#ifdef IPROFCLB
	if (tp->pid==0) {
		*TCU100 = 0;	/* turn off indep battery clock int */
		return;
	}
#endif
#endif

	tp->oldpg.par = KDSA->r[5];
	tp->oldpg.pdr = KDSD->r[5];
	KDSA->r[5] = tp->newpg.par;
	KDSD->r[5] = tp->newpg.pdr;

	/*
	 *	use kernel D-space register 5 for
	 *	accessing beginning of block where
	 *	SPCNT resides.
	 *	this dangerous trick requires the
	 *	clock routine not to use any unitialized
	 *	variables that might normally be accessed
	 *	using kernel register 5
	 */

	if ((ps&UMODE)==UMODE) {
		tp->base->b_urhits++;
		goto ret;
	}
	if (pc == &waitloc) {
		tp->base->b_idhits++;
		goto ret;
	}
	tp->base->b_syhits++;
	if (tp->intsize==0) {	/* -r or routine option */
		t = bisrch((pc>>1)&077777,tp->base->u_ct.ropt,tp->numcnts);
		tp->base->u_ct.ropt[t].nhits++;
	} else {		/* -i or word intervals option */
		t = (pc >> 1) & 077777; /* word address */
		t -= tp->lowpc;
		if (t<0) goto ret;	/* below code focusing on */
		t = t/tp->intsize;	/* get interval no. */
		if(t > tp->numcnts-1)
			goto ret;	/* above code focusing on */
		tp->base->u_ct.iopt[t]++;
	}
ret:
	KDSA->r[5] = tp->oldpg.par;
```

```
        KDSD->r[5] = tp->oldpg.pdr;

#ifdef IPROFCLK
        KW11K->kw11ks = 0507;   /* restart clock */
#else
#ifdef IPROFCLB
        /* TCU-100 is re-enabled automatically when pdp-11
           receives interrupt */
#endif
#endif

}

bisrch(x,buf,siz)
int x,siz;
struct NHIT *buf;
{
        register int low,high,mid;

        low = 0;
        high = siz -1;
        while (low <= high) {
                mid = (low+high)/2;
                if (x < buf[mid].nloc)
                        high = mid -1;
                else if( x > buf[mid].nloc)
                        low  = mid +1;
                else            /* found the location */
                        return(mid);
        }
        return(high);
}
#endif
```

```
/*      @(#)conf.70.c   2.16    */

#include "sys/param.h"
#include "sys/buf.h"
#include "sys/elog.h"
#include "sys/iobuf.h"
#include "sys/conf.h"
#include "sys/utsname.h"

/*
 *      Copyright 1974 Bell Telephone Laboratories Inc
 */

/*
 *      Frozen Configuration Table - New devices may be
 *      added to the end of the tables only!
 *      block order - rx,rp,rf,tm,tc,hp,ht,hs,0
 *      character order - kl,dz,lp,dc,dh,dp,dj,dn,mem,rk,rf,rp,tm,
                          hp,ht,hs,0
 */

extern nodev(), nulldev();
extern fmcntrl();       /* fake modem control routine */

extern rxopen(), rxstrategy();
extern struct iobuf rxtab;
extern hpopen(), hpread(), hpwrite(), hpstrategy();
extern struct iobuf hptab;
extern htopen(), htclose(), htread(), htwrite(), htstrategy();
extern struct iobuf httab;
extern klopen(), klclose(), klread(), klwrite(), klioctl(), kl11;
extern dhopen(), dhclose(), dhread(), dhwrite(), dhioctl(), dmcntrl(), dh11;
extern vtopen(), vtclose(), vtwrite();
extern dnopen(), dnclose(), dnwrite(), dnint();
extern dzopen(), dzclose(), dzread(), dzwrite(), dzioctl(), dz11;
extern mmread(), mmwrite();
extern syopen(), syread(), sywrite(), syioctl();
extern npopen(), npclose(), npread(), npwrite(), npioctl();
extern vpopen(), vpclose(), vpwrite(), vpioctl();
extern mxopen(), mxclose(), mxread(), mxwrite(), mxioctl();
extern erropen(), errclose(), errread();

struct  bdevsw  bdevsw[] =
{
        &rxopen,        &nulldev,       &rxstrategy,    &rxtab,         /*rx*/
        &nodev,         &nodev,         &nodev,         0,              /*rp*/
        &nodev,         &nodev,         &nodev,         0,              /*rf*/
        &nodev,         &nodev,         &nodev,         0,              /*tm*/
        &nodev,         &nodev,         &nodev,         0,              /*tc*/
        &hpopen,        &nulldev,       &hpstrategy,    &hptab,         /*hp*/
        &htopen,        &htclose,       &htstrategy,    &httab,         /*ht*/
};

struct  cdevsw  cdevsw[] =
{
```

```
/* 0*/  &klopen,        &klclose,       &klread,        &klwrite,       /*kl*/
                &klioctl,       &fmcntrl,       &kl11,
        &dzopen,        &dzclose,       &dzread,        &dzwrite,       /*dz*/
                &dzioctl,       &fmcntrl,       &dz11,
        &vpopen,        &vpclose,       &nodev,         &vpwrite,       /*lp*/
                &vpioctl,       &nulldev,       0,
        &nodev,         &nodev,         &nodev,         &nodev,         /*dc*/
                &nodev,         &nulldev,       0,
        &dhopen,        &dhclose,       &dhread,        &dhwrite,       /*dh*/
                &dhioctl,       &dmcntrl,       &dh11,

/* 5*/  &vtopen,        &vtclose,       &nodev,         &vtwrite,       /*dp*/
                &nodev,         &nulldev,       0,
        &nodev,         &nodev,         &nodev,         &nodev,         /*dj*/
                &nodev,         &nulldev,       0,
        &dnopen,        &dnclose,       &nodev,         &dnwrite,       /*dn*/
                &nodev,         &nulldev,       0,
        &nulldev,       &nulldev,       &mmread,        &mmwrite,       /*mm*/
                &nodev,         &nulldev,       0,
        &nodev,         &nodev,         &nodev,         &nodev,         /*rk*/
                &nodev,         &nulldev,       0,

/*10*/  &nodev,         &nodev,         &nodev,         &nodev,         /*rf*/
                &nodev,         &nulldev,       0,
        &nodev,         &nodev,         &nodev,         &nodev,         /*rp*/
                &nodev,         &nulldev,       0,
        &nodev,         &nodev,         &nodev,         &nodev,         /*tm*/
                &nodev,         &nulldev,       0,
        &hpopen,        &nulldev,       &hpread,        &hpwrite,       /*hp*/
                &nodev,         &nulldev,       0,
        &htopen,        &htclose,       &htread,        &htwrite,       /*ht*/
                &nodev,         &nulldev,       0,

/*15*/  &nodev,         &nodev,         &nodev,         &nodev,         /*hs*/
                &nodev,         &nulldev,       0,
        &syopen,        &nulldev,       &syread,        &sywrite,       /*sy*/
                &syioctl,       &nulldev,       0,
        &nodev,         &nodev,         &nodev,         &nodev,
                &nodev,         &nulldev,       0,
        &npopen,        &npclose,       &npread,        &npwrite,       /*np*/
                &npioctl,       &nulldev,       0,
        &mxopen,        &mxclose,       &mxread,        &mxwrite,       /*mx*/
                &mxioctl,       &nulldev,       0,

/*20*/  &erropen,       &errclose,      &errread,       &nodev,         /*er*/
                &nodev,         &nulldev,       0,
};

extern ttread(), ttyinput(), ttwrite(), ttxint(), ttyioctl(), ttydst();
extern ttyopen(), ttyclose(), ttxdma();
extern vsread(), vsinput(), vswrite(), vsxint(), vsioctl(), vsdst(), vsopen();
extern mcread(), mcwrite();

struct  linesw  linesw[] =
{
        &ttread,        &ttyinput,      &ttwrite,       &ttxint,
                &ttyioctl,      &ttydst,        &ttyopen,       &ttyclose,
                &ttxdma,
        &nodev,         &nodev,         &nodev,         &nodev,
                &nodev,         &nulldev,       &nodev,         &nodev,
                &nodev,
        &nodev,         &nodev,         &nodev,         &nodev,
                &nodev,         &nulldev,       &nodev,         &nodev,
                &nodev,
        &nodev,         &nodev,         &nodev,         &nodev,
                &nodev,         &nulldev,       &nodev,         &nodev,
                &nodev,
        &nodev,         &nodev,         &nodev,         &nodev,
                &nodev,         &nulldev,       &nodev,         &nodev,
                &nodev,
        &vsread,        &vsinput,       &vswrite,       &vsxint,
                vsioctl,        &vsdst,         &vsopen,        &ttyclose,
                &ttxdma,
        &mcread,        &nulldev,       &mcwrite,       &nulldev,
                &nulldev,       &nulldev,       &nulldev,       &nulldev,
                &nulldev,
};

extern hp45input(), hp45output(), hp45ioctl();
extern vt100input(), vt100output(), vt100ioctl();

struct  termsw  termsw[] =
{
        &nulldev,       &nulldev,       &nulldev,       /*none (tty)*/
        &nulldev,       &nulldev,       &nodev,         /*TEC*/
        &nulldev,       &nulldev,       &nodev,         /*VT61*/
        &vt100input,    &vt100output,   &vt100ioctl,    /*VT100*/
        &nulldev,       &nulldev,       &nodev,         /*Tektronix 4023*/
        &nulldev,       &nulldev,       &nodev,         /*TTY 40/1*/
        &hp45input,     &hp45output,    &hp45ioctl,     /*HP 45*/
};

int     nblkdev = sizeof(bdevsw)/sizeof(struct bdevsw);
int     nchrdev = sizeof(cdevsw)/sizeof(struct cdevsw);
/*
 *      nldisc must not include mc line disc, hence the -1 below.
 *      the -1 should be removed if the mc line disc is not present.
 */
int     nldisc = (sizeof(linesw)/sizeof(struct linesw))-1;
int     nttype = sizeof(termsw)/sizeof(struct termsw);

dev_t   rootdev = makedev(5, 15);
dev_t   swapdev = makedev(5, 39);
daddr_t swplo   1;
int     nswap   8359;

struct  utsname utsname = {
        "CBUNIX",
        "cbosg",
        "CB2.0",
        "0820"
};
```

```
/*      @(#)conf.tu.c   2.1.1.1 */

/*      @(#)conf.util.c 2.2     */
/*      based on conf.70.c 2.12 */

#include "sys/param.h"
#include "sys/buf.h"
#include "sys/elog.h"
#include "sys/iobuf.h"
#include "sys/conf.h"
#include "sys/utsname.h"

/*
 *      UTIL MAKEFILE -- Stripped for hp, ht, and kl only.
 *      Copyright 1974 Bell Telephone Laboratories Inc
 */

/*
 *      Frozen Configuration Table - New devices may be
 *      added to the end of the tables only!
 *      block order - rk,rp,rf,tm,tc,hp,ht,hs,0
 *      character order - kl,pc,lp,dc,dh,dp,dj,dn,mem,rk,rf,rp,tm,
 *                        hp,ht,hs,0
 */

extern nodev(), nulldev();
extern hpopen(), hpread(), hpwrite(), hpstrategy();
extern struct iobuf hptab;
extern htopen(), htclose(), htread(), htwrite(), htstrategy();
extern struct iobuf httab;
extern klopen(), klclose(), klread(), klwrite(), klioctl(), kl11;
extern fmcntrl();       /* fake modem control routine */
extern mmread(), mmwrite();
extern syopen(), syread(), sywrite(), syioctl();
extern erropen(), errclose(), errread();

struct  bdevsw  bdevsw[] =
{
        nodev,          nodev,          nodev,          0,      /*rk*/
        nodev,          nodev,          nodev,          0,      /*rp*/
        nodev,          nodev,          nodev,          0,      /*rf*/
        nodev,          nodev,          nodev,          0,      /*tm*/
        nodev,          nodev,          nodev,          0,      /*tc*/
        hpopen,         nulldev,        hpstrategy,     &hptab, /*hp*/
        htopen,         htclose,        htstrategy,     &httab, /*hs*/
};

struct  cdevsw  cdevsw[] =
{
/* 0*/  klopen,         klclose,        klread,         klwrite,
                klioctl,        fmcntrl,        &kl11,          /*kl*/
        nodev,          nodev,          nodev,          nodev,
                nodev,          nulldev,        0,              /*pc*/
        nodev,          nodev,          nodev,          nodev,
                nodev,          nulldev,        0,              /*lp*/
        nodev,          nodev,          nodev,          nodev,
                nodev,          nulldev,        0,              /*dc*/
        nodev,          nodev,          nodev,          nodev,
                nodev,          nulldev,        0,              /*dh*/

/* 5*/  nodev,          nodev,          nodev,          nodev,
                nodev,          nulldev,        0,              /*dp*/
        nodev,          nodev,          nodev,          nodev,
                nodev,          nulldev,        0,              /*dj*/
        nodev,          nodev,          nodev,          nodev,
                nodev,          nulldev,        0,              /*dn*/
        nulldev,        nulldev,        mmread,         mmwrite,
                nodev,          nulldev,        0,              /*mm*/
        nodev,          nodev,          nodev,          nodev,
                nodev,          nulldev,        0,              /*rk*/

/*10*/  nodev,          nodev,          nodev,          nodev,
                nodev,          nulldev,        0,              /*rf*/
        nodev,          nodev,          nodev,          nodev,
                nodev,          nulldev,        0,              /*rp*/
        nodev,          nodev,          nodev,          nodev,
                nodev,          nulldev,        0,              /*tm*/
        hpopen,         nulldev,        hpread,         hpwrite,
                nodev,          nulldev,        0,              /*hp*/
        htopen,         htclose,        htread,         htwrite,
                nodev,          nulldev,        0,              /*ht*/

/*15*/  nodev,          nodev,          nodev,          nodev,
                nodev,          nulldev,        0,              /*hs*/
        syopen,         nulldev,        syread,         sywrite,
                syioctl,        nulldev,        0,              /*sy*/
        nodev,          nodev,          nodev,          nodev,
                nodev,          nulldev,        0,
        nodev,          nodev,          nodev,          nodev,
                nodev,          nulldev,        0,
        nodev,          nodev,          nodev,          nodev,
                nodev,          nulldev,        0,

/*20*/ erropen,         errclose,       errread,        nodev,
                nodev,          nulldev,        0,
};

extern ttread(), ttyinput(), ttwrite(), ttxint(), ttyioctl(), ttydst();
extern ttyopen(), ttyclose(), ttxdma();

struct  linesw  linesw[] =
{
        ttread,         ttyinput,       ttwrite,        ttxint,
                ttyioctl,       ttydst,         ttyopen,        ttyclose,
                ttxdma,
};

struct  termsw  termsw[] =
{
        nulldev,        nulldev,        nulldev,
};
```

```
int      nblkdev = sizeof(bdevsw)/sizeof(struct bdevsw);
int      nchrdev = sizeof(cdevsw)/sizeof(struct cdevsw);
/*
 *       nldisc must not include mc line disc, hence the -1 below.
 *       the -1 should be removed if the mc line disc is not present.
 */
int      nldisc = (sizeof(linesw)/sizeof(struct linesw));
int      nttype = sizeof(termsw)/sizeof(struct termsw);

dev_t    rootdev = makedev(5, 0);
dev_t    swapdev = makedev(5, 0);
daddr_t  swplo   = 4599;
int      nswap   = 417;

         /* Paramters for UNAME system call       */
struct   utsname utsname = {
         "CBUNIX",                /*Operating System Name        */
         "cbosg",                 /*UUCP Network Name (change for local system)*/
         "CB2.1",                 /*Operating System Release     */
         "1031"                   /*Operating System Date        */
};
```

```
/*      @(#)conf.util.c 2.3     */
/*      based on conf.70.c 2.12 */

#include "sys/param.h"
#include "sys/buf.h"
#include "sys/elog.h"
#include "sys/iobuf.h"
#include "sys/conf.h"
#include "sys/utsname.h"

/*
 *      UTIL MAKEFILE -- Stripped for hp, ht, and kl only.
 *      Copyright 1974 Bell Telephone Laboratories Inc
 */

/*
 *      Frozen Configuration Table - New devices may be
 *      added to the end of the tables only!
 *      block order - rk,rp,rf,tm,tc,hp,ht,hs,0
 *      character order - kl,pc,lp,dc,dh,dp,dj,dn,mem,rk,rf,rp,tm,
 *                        hp,ht,hs,0
 */

extern nodev(), nulldev();
extern hpopen(), hpread(), hpwrite(), hpstrategy();
extern struct iobuf hptab;
extern htopen(), htclose(), htread(), htwrite(), htstrategy();
extern struct iobuf httab;
extern klopen(), klclose(), klread(), klwrite(), klioctl(), kl11;
extern fmcntrl();       /* fake modem control routine */
extern mmread(), mmwrite();
extern syopen(), syread(), sywrite(), syioctl();
extern erropen(), errclose(), errread();

struct  bdevsw  bdevsw[] =
{
        nodev,          nodev,          nodev,          0,      /*rk*/
        nodev,          nodev,          nodev,          0,      /*rp*/
        nodev,          nodev,          nodev,          0,      /*rf*/
        nodev,          nodev,          nodev,          0,      /*tm*/
        nodev,          nodev,          nodev,          0,      /*tc*/
        hpopen,         nulldev,        hpstrategy,     &hptab, /*hp*/
        htopen,         htclose,        htstrategy,     &httab, /*hs*/
};

struct  cdevsw  cdevsw[] =
{
/* 0*/  klopen,         klclose,         klread,        klwrite,
                klioctl,        fmcntrl,        &kl11,          /*kl*/
        nodev,          nodev,          nodev,          nodev,
                nodev,          nulldev,        0,              /*pc*/
        nodev,          nodev,          nodev,          nodev,
                nodev,          nulldev,        0,              /*lp*/
        nodev,          nodev,          nodev,          nodev,
                nodev,          nulldev,        0,              /*dc*/
        nodev,          nodev,          nodev,          nodev,
                nodev,          nulldev,        0,              /*dh*/

/* 5*/  nodev,          nodev,          nodev,          nodev,
                nodev,          nulldev,        0,              /*dp*/
        nodev,          nodev,          nodev,          nodev,
                nodev,          nulldev,        0,              /*dj*/
        nodev,          nodev,          nodev,          nodev,
                nodev,          nulldev,        0,              /*dn*/
        nulldev,        nulldev,        mmread,         mmwrite,
                nodev,          nulldev,        0,              /*mm*/
        nodev,          nodev,          nodev,          nodev,
                nodev,          nulldev,        0,              /*rk*/

/*10*/  nodev,          nodev,          nodev,          nodev,
                nodev,          nulldev,        0,              /*rf*/
        nodev,          nodev,          nodev,          nodev,
                nodev,          nulldev,        0,              /*rp*/
        nodev,          nodev,          nodev,          nodev,
                nodev,          nulldev,        0,              /*tm*/
        hpopen,         nulldev,        hpread,         hpwrite,
                nodev,          nulldev,        0,              /*hp*/
        htopen,         htclose,        htread,         htwrite,
                nodev,          nulldev,        0,              /*ht*/

/*15*/  nodev,          nodev,          nodev,          nodev,
                nodev,          nulldev,        0,              /*hs*/
        syopen,         nulldev,        syread,         sywrite,
                syioctl,        nulldev,        0,              /*sy*/
        nodev,          nodev,          nodev,          nodev,
                nodev,          nulldev,        0,
        nodev,          nodev,          nodev,          nodev,
                nodev,          nulldev,        0,
        nodev,          nodev,          nodev,          nodev,
                nodev,          nulldev,        0,

/*20*/ erropen,         errclose,       errread,        nodev,
                nodev,          nulldev,        0,
};

extern ttread(), ttyinput(), ttwrite(), ttxint(), ttyioctl(), ttydst();
extern ttyopen(), ttyclose(), ttxdma();

struct  linesw  linesw[] =
{
        ttread,         ttyinput,       ttwrite,        ttxint,
                ttyioctl,       ttydst,         ttyopen,        ttyclose,
                ttxdma,
};

struct  termsw  termsw[] =
{
        nulldev,        nulldev,        nulldev,
};

int     nblkdev = sizeof(bdevsw)/sizeof(struct bdevsw);
int     nchrdev = sizeof(cdevsw)/sizeof(struct cdevsw);
/*
 *      nldisc must not include mc line disc, hence the -1 below.
 *      the -1 should be removed if the mc line disc is not present.
 */
int     nldisc = (sizeof(linesw)/sizeof(struct linesw));
int     nttype = sizeof(termsw)/sizeof(struct termsw);

dev_t   rootdev = makedev(5, 8);
dev_t   swapdev = makedev(5, 9);
daddr_t swplo   = 1;
int     nswap   = 4179;

        /* Paramters for UNAME system call      */
struct  utsname utsname = {
        "CBUNIX",               /*Operating System Name         */
        "cbosg",                /*UUCP Network Name (change for local system)*/
        "CB2.0",                /*Operating System Release      */
        "0820"                  /*Operating System Date         */
};
```

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

#include "sys/param.h"
#include "sys/systm.h"
#include "sys/buf.h"
#include "sys/conf.h"
#include "sys/confx.h"
#include "sys/elog.h"
#include "sys/err.h"
#include "sys/iobuf.h"

struct  err     err = {
        NESLOT,
};

errinit()
{
        if(err.e_nslot)
                mfree(err.e_map,err.e_nslot,1);
        err.e_org = err.e_ptrs;
        err.e_nxt = err.e_ptrs;
}
err_t *
geteslot(size)
{
        register ns, *p;
        register err_t *eup;
        int n, sps;

        ns = (size+sizeof(struct errslot)-1)/sizeof(struct errslot);
        sps = spl7();
        n = malloc(err.e_map,ns);
        splx(sps);
        if(n == 0)
                return(NULL);
        eup = (err_t *)(&err.e_slot[--n]);
        ns *= sizeof(struct errslot)/sizeof(int);
        p = (int *)eup;
        do {
                *p++ = 0;
        } while(--ns);
        eup->e_hdr.e_len = size;
        return(eup);
}

freeslot(eup)
register err_t *eup;
{
        register ns, sps;

        ns = (eup->e_hdr.e_len+sizeof(struct errslot)-1)/sizeof(struct errslot)
*  ;
        sps = spl7();
        mfree(err.e_map,ns,(((struct errslot *)eup)-err.e_slot)+1);
        splx(sps);
```

```
}

err_t *
geterec()
{
        register sps;
        register err_t *eup;

        sps = spl7();
        while(*err.e_org == NULL)
                sleep(&err.e_org,PZERO+1);
        eup = *err.e_org;
        *err.e_org++ = NULL;
        if(err.e_org >= &err.e_ptrs[err.e_nslot])
                err.e_org = err.e_ptrs;
        splx(sps);
        return(eup);
}

puterec(eup,type)
register err_t *eup;
{
        register sps;

        eup->e_hdr.e_type = type;
        eup->e_hdr.e_time = time;
        sps = spl7();
        *err.e_nxt++ = eup;
        if(err.e_nxt >= &err.e_ptrs[err.e_nslot])
                err.e_nxt = err.e_ptrs;
        splx(sps);
        wakeup(&err.e_org);
}

logstart()
{
        register err_t *eup;
        register struct bdevsw *bdp;
        struct {
                unsigned losize;
                unsigned hisize;
        };
        extern int nblkdev;

        if ((eup = geteslot(sizeof(struct estart))) == NULL)
                return;
        eup->e_cpu = cputype;
        if(cputype == 70 || cputype == 45) {
                eup->e_mmr3 = MMR3->r[0];
                if(cputype == 70) {
                        eup->e_syssize = (((long)(SYSSIZE->hisize))<<16) +
                                        (long)(SYSSIZE->losize) + 1;
                        eup->e_syssize = ctob((eup->e_syssize));
                }
        }
        for(bdp = &bdevsw[nblkdev-1]; bdp >= bdevsw; bdp--)
```

```
                eup->e_bconf |= 1<<major((bdp->d_tab->b_dev));
        puterec(eup,E_GOCB);
}

logtchg(nt)
time_t nt;
{
        register err_t *eup;

        if((eup = geteslot(sizeof(struct etimchg))) != NULL) {
                eup->e_ntime = nt;
                puterec(eup,E_TCHG);
        }
}

logstray(addr)
physadr addr;
{
        register err_t *eup;

        if((eup = geteslot(sizeof(struct estray))) != NULL) {
                eup->e_saddr = addr;
                eup->e_sbacty = blkacty;
                puterec(eup,E_STRAY);
        }
}

logparity(addr)
register physadr addr;
{
        register err_t *eup;
        register n;

        if((eup = geteslot(sizeof(struct eparity))) != NULL) {
                for(n = 0; n < 4; n++)
                        eup->e_parreg[n] = addr->r[n];
                puterec(eup,E_PRTY);
        }
}

logprdev(type, dev)
register short type;
register dev_t dev;
{
        register struct eprdev *eup;
        static int missed;
        static time_t lasttime;
        static short lasttype;
        static dev_t lastdev;

        if (dev != lastdev || type != lasttype || (time > (lasttime + 60))) {
                if ((eup = geteslot(sizeof(struct eprdev))) == NULL)
                        return;
                eup->e_missed = missed;
                eup->e_fserr = type;
                eup->e_fsdev = dev;
```

```
		puterec(eup, E_PRDV);
		missed = 0;
		lasttime = time;
		lasttype = type;
		lastdev = dev;
		return;
	}
	missed++;
}

logpower()
{
	register err_t *eup;

	if ((eup = geteslot(sizeof(struct epower))) != NULL)
		puterec(eup, E_POWER);
}

logovfl(type)
{
	register err_t *eup;
	static int missed;
	static time_t lasttime;
	static lasttype;

	if (type != lasttype || (time > (lasttime + 60))) {
		if ((eup = geteslot(sizeof(struct eovfl))) == NULL)
			return;
		eup->e_missed = missed;
		eup->e_tabt = type;
		puterec(eup, E_OVFL);
		missed = 0;
		lasttime = time;
		lasttype = type;
		return;
	}
	missed++;
}

fmtberr(dp,cyl)
register struct iobuf *dp;
{
	register err_t *eup;
	register struct buf *bp;
	register *p, n;
	struct br {
		struct eblock	eb;
		int		cregs[];
	};
	struct	iostat	*iosp;
	physadr addr;

	if(dp->io_erec != NULL) {
		dp->io_erec->e_rtry++;
		return;
	}
```

```
        iosp = dp->io_stp;
        if((eup = geteslot(sizeof(struct eblock)+(dp->io_nreg*sizeof(int)))) ==
*   NULL) {
                iosp->io_unlog++;
                return;
        }
        n = major(dp->b_dev);
        bp = dp->b_actf;
        eup->e_dev = makedev(n,(bp==NULL)?minor(dp->b_dev):minor(bp->b_dev));
        eup->e_regloc = addr = dp->io_addr;
        eup->e_bacty = blkacty;
        eup->e_stats.io_ops = iosp->io_ops;
        eup->e_stats.io_misc = iosp->io_misc;
        eup->e_stats.io_unlog = iosp->io_unlog;
        if(bp != NULL) {
                eup->e_bflags = (bp->b_flags&B_READ) ? E_READ : E_WRITE;
                if(bp->b_flags & B_PHYS)
                        eup->e_bflags |= E_PHYS;
                if(bp->b_flags & B_MAP)
                        eup->e_bflags |= E_MAP;
                eup->e_bnum = bp->b_blkno;
                eup->e_bytes = bp->b_bcount;
                eup->e_memadd = paddr(bp);
        }
        else
                eup->e_bflags = E_NOIO;
        eup->e_cyloff = cyl;
        eup->e_nreg = dp->io_nreg;
        p = (int *)(&(struct br *)eup->cregs[0]);
        switch(n) {
                case HP0:
                case HT0:
                case HS0:
                        for(n = 9; --n >= 0; addr++)    /* skip db on rh */
                                *p++ = addr->r[0];
                        p++;
                        addr++;
                        n = dp->io_nreg - 10;
                        break;
                default:
                        n = dp->io_nreg;
        }
        while(--n >= 0) {
                *p++ = addr->r[0];
                addr++;
        }
        dp->io_erec = eup;
}

logberr(dp,err)
register struct buf *dp;
{
        register err_t *eup;

        if((eup = dp->io_erec) == NULL)
                return;
        if(err)
                eup->e_bflags |= E_ERROR;
        puterec(eup,E_BLK);
        dp->io_erec = NULL;
}
```

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

/*
 * Fake error logging functions for 11/40s
 */

errinit() {}

logstart() {}

logtchg() {}

logstray() {}

logparity() {}

logprdev() {}

logpower() {}

logovfl() {}

logberr() {}

fmtberr() {}
```

```
/*      @(#)fio.c       2.8     */

#
#include "sys/param.h"
#include "sys/user.h"
#include "sys/userx.h"
#include "sys/filsys.h"
#include "sys/file.h"
#include "sys/filex.h"
#include "sys/conf.h"
#include "sys/confx.h"
#include "sys/inode.h"
#include "sys/inodex.h"
#include "sys/reg.h"
#include "sys/systm.h"
#include "sys/sysmes.h"
#include "sys/sysmesx.h"
#include "sys/elog.h"

/*
 * Convert a user supplied
 * file descriptor into a pointer
 * to a file structure.
 * Only task is to check range
 * of the descriptor.
 */
struct file *
getf(f)
register int f;
{
        register struct file *fp;

        if (0<=f && f<NOFILE) {
                fp = u.u_ofile[f];
                if(fp != NULL)
                        return(fp);
        }
        u.u_error = EBADF;
        return(NULL);
}

/*
 * Internal form of close.
 * Decrement reference count on
 * file structure.
 * Also make sure the pipe protocol
 * does not constipate.
 *
 * Decrement reference count on the inode following
 * removal to the referencing file structure.
 * On the last close switch out the the device handler for
 * special files.  Note that the handler is called
 * on every open but only the last close.
 */
closef(fp)
```

```
register struct file *fp;
{
        register struct inode *ip;
        int flag, mode;
        dev_t dev;
        register int (*cfunc)();
        struct chan *cp;

        if(fp == NULL)
                return;
        if ((unsigned)fp->f_count > 1) {
                fp->f_count--;
                return;
        }
        ip = fp->f_inode;
        flag = fp->f_flag;
        cp = fp->f_un.f_chan;
        dev = (dev_t)ip->i_un.i_rdev;
        mode = ip->i_mode;

        plock(ip);
        fp->f_count = 0;
        if(flag & FPIPE) {
                ip->i_mode &= ~(IREAD|IWRITE);
                wakeup((caddr_t)ip+1);
                wakeup((caddr_t)ip+2);
        }
        iput(ip);

        switch(mode&IFMT) {

        case IFCHR:
        case IFMPC:
                cfunc = cdevsw[major(dev)].d_close;
                break;

        case IFBLK:
        case IFMPB:
                cfunc = bdevsw[major(dev)].d_close;
                break;
        default:
                return;
        }

        if (flag & FMP)
                goto call;

        for(fp=file; fp < &file[NFILE]; fp++)
                if (fp->f_count && fp->f_inode==ip)
                        return;

call:
        (*cfunc)(dev, flag, cp);
}
/*
```

```
 * openi called to allow handler
 * of special files to initialize and
 * validate before actual IO.
 * Called on all sorts of opens
 * and also on mount.
 */
openi(ip, rw)
register struct inode *ip;
{
	dev_t dev;
	register unsigned int maj;

	dev = (dev_t)ip->i_un.i_rdev;
	maj = major(dev);
	switch(ip->i_mode&IFMT) {

	case IFCHR:
	case IFMPC:
		if(maj >= nchrdev)
			goto bad;
		(*cdevsw[maj].d_open)(dev, rw);
		break;

	case IFBLK:
	case IFMPB:
		if(maj >= nblkdev)
			goto bad;
		(*bdevsw[maj].d_open)(dev, rw);
	}
	return;

bad:
	u.u_error = ENXIO;
}

/*
 * Check mode permission on inode pointer.
 * Mode is READ, WRITE or EXEC.
 * In the case of WRITE, the
 * read-only status of the file
 * system is checked.
 * Also in WRITE, prototype text
 * segments cannot be written.
 * The mode is shifted to select
 * the owner/group/other fields.
 * The super user is granted all
 * permissions.
 */
access(ip, mode)
register struct inode *ip;
{
	register m;

	m = mode;
	if(m == IWRITE) {
		if(rofs(ip->i_dev)) {
```

```
                        u.u_error = EROFS;
                        return(1);
                }
                if(ip->i_flag & ITEXT) {
                        u.u_error = ETXTBSY;
                        return(1);
                }
        }
        if(u.u_uid == 0)
                return(0);
        if(u.u_uid != ip->i_uid) {
                m =>> 3;
                if(u.u_gid == 0)        /* super group */
                        m = m | m>>3;
                else
                if(u.u_gid != ip->i_gid)
                        m =>> 3;
        }
        if((ip->i_mode&m) != 0)
                return(0);

        u.u_error = EACCES;
        return(1);
}

/*
 * Look up a pathname and test if
 * the resultant inode is owned by the
 * current user.
 * If not, try for super-user.
 * If permission is granted,
 * return inode pointer.
 */
owner()
{
        register struct inode *ip;
        extern uchar();

        if ((ip = namei(uchar, 0)) == NULL)
                return(NULL);
        if(u.u_uid == ip->i_uid)
                return(ip);
        if (suser())
                return(ip);
        iput(ip);
        return(NULL);
}

/*
 * Test if the current user is the
 * super user.
 */
suser()
{

        if(u.u_uid == 0)
```

```
                return(1);
        u.u_error = EPERM;
        return(0);
}

/*
 * Allocate a user file descriptor.
 */
ufalloc(i)
register i;
{

        for (; i<NOFILE; i++)
                if (u.u_ofile[i] == NULL) {
                        u.u_ar0[R0] = i;
                        u.u_pofile[i] = 0;
                        return(i);
                }
        u.u_error = EMFILE;
        return(-1);
}

/*
 * Allocate a user file descriptor
 * and a file structure.
 * Initialize the descriptor
 * to point at the file structure.
 *
 * no file -- if there are no available
 *      file structures.
 */
struct file *
falloc()
{
        register struct file *fp;
        register i;

        if ((i = ufalloc(0)) < 0)
                return(NULL);
        for (fp = &file[0]; fp < &file[NFILE]; fp++)
                if (fp->f_count==0) {
                        u.u_ofile[i] = fp;
                        fp->f_count++;
                        fp->f_un.f_offset = 0;
                        return(fp);
                }
        logovfl(E_FILEO);
        printf("no file\n");
        meas.m_fovfl++;
        u.u_error = ENFILE;
        return(NULL);
}
```

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

#include "sys/param.h"
#include "sys/systm.h"
#include "sys/user.h"
#include "sys/userx.h"
#include "sys/inode.h"
#include "sys/inodex.h"
#include "sys/filsys.h"
#include "sys/conf.h"
#include "sys/confx.h"
#include "sys/buf.h"
#include "sys/bufx.h"
#include "sys/sysmes.h"
#include "sys/sysmesx.h"
#include "sys/elog.h"


/*
 * Look up an inode by device,inumber.
 * If it is in core (in the inode structure),
 * honor the locking protocol.
 * If it is not in core, read it in from the
 * specified device.
 * If the inode is mounted on, perform
 * the indicated indirection.
 * In all cases, a pointer to a locked
 * inode structure is returned.
 *
 * printf warning: no inodes -- if the inode
 *          structure is full
 * panic: no imt -- if the mounted file
 *          system is not in the mount table.
 *          "cannot happen"
 */
struct inode *
iget(dev, ino)
dev_t dev;
ino_t ino;
{
        register struct inode *p;
        register struct mount *ip;

loop:
        ip = NULL;
        for(p = &inode[0]; p < &inode[NINODE]; p++) {
                if(ino == p->i_number && dev == p->i_dev) {
                        if((p->i_flag&ILOCK) != 0) {
                                p->i_flag =| IWANT;
                                sleep(p, PINOD);
                                goto loop;
                        }
                        if((p->i_flag&IMOUNT) != 0) {
                                for(ip = &mount[0]; ip < &mount[NMOUNT]; ip++)
                                if(ip->m_inodp == p) {
                                        dev = ip->m_dev;
                                        ino = ROOTINO;
                                        goto loop;
                                }
                                panic("no imt");
                        }
                        p->i_count++;
                        p->i_flag =| ILOCK;
                        return(p);
                }
                if(ip==NULL && p->i_count==0)
                        ip = p;
        }
        if((p=ip) == NULL) {
                logovfl(E_INODEO);
                printf("Inode table overflow\n");
                meas.m_iovfl++;
                u.u_error = ENFILE;
                return(NULL);
        }
        p->i_dev = dev;
        p->i_number = ino;
        p->i_flag = ILOCK;
        p->i_count++;
        p->i_un.i_lastr = -1;
        ip = bread(dev, ldiv(ino+31,16));
        /*
         * Check I/O errors
         */
        if (ip->b_flags&B_ERROR) {
                brelse(ip);
                iput(p);
                return(NULL);
        }
        copyio((ip->b_paddr + 32*lrem(ino+31, 16)), &p->i_mode,
                (char *)&0->i_un.i_addr[NADDR] - (char *)&0->i_mode, U_RKD);
        brelse(ip);
        return(p);
}

/*
 * Decrement reference count of
 * an inode structure.
 * On the last reference,
 * write the inode out and if necessary,
 * truncate and deallocate the file.
 */
iput(ip)
register struct inode *ip;
{

        if(ip->i_count == 1) {
                ip->i_flag =| ILOCK;
                if(ip->i_nlink <= 0) {
                        itrunc(ip);
                        ip->i_mode = 0;
```

```
                                ip->i_flag =| IUPD;
                                ifree(ip->i_dev, ip->i_number);
                        }
                        iupdat(ip, &time);
                        prele(ip);
                        ip->i_flag = 0;
                        ip->i_number = 0;
                }
                ip->i_count--;
                prele(ip);
}

/*
 * Check accessed and update flags on
 * an inode structure.
 * If either is on, update the inode
 * with the current time.
 */
iupdat(p, tm)
int *p;
time_t *tm;
{
        register size;
        paddr_t addr;
        register *rp;
        struct buf *bp;
        register i;

        rp = p;
        if((rp->i_flag&(IUPD|IACC)) != 0) {
                if(rofs(rp->i_dev))
                        return;
                i = rp->i_number+31;
                bp = bread(rp->i_dev, ldiv(i,16));
                addr = paddr(bp) + 32*lrem(i, 16);
                size = (char *)&0->i_un.i_addr[NADDR] - (char *)&0->i_mode;
                copyio(addr, &rp->i_mode, size, U_WKD);
                addr += size;
                if(rp->i_flag&IACC)
                        xputl(addr, time);
                addr += sizeof(long);
                if(rp->i_flag&IUPD)
                        xputl(addr, *tm);
                rp->i_flag =& ~(IUPD|IACC);
                bdwrite(bp);
        }
}

/*
 * Free all the disk blocks associated
 * with the specified inode structure.
 * The blocks of the file are removed
 * in reverse order. This FILO
 * algorithm will tend to maintain
 * a contiguous free list much longer
 * than FIFO.
 */
itrunc(ip)
int *ip;
{
        register *rp, *bp, *cp;
        int *dp, *ep;
        unsigned fmt;

        rp = ip;
        fmt = rp->i_mode&IFMT;
        if(fmt == IFCHR || fmt == IFBLK || fmt == IFMPC || fmt == IFMPB)
                return;
        for(ip = &rp->i_un.i_addr[NADDR-1]; ip >= &rp->i_un.i_addr[0]; ip--)
        if(*ip) {
                if(fmt == IFLRG || fmt == IFLDR) {
                        bp = abread(rp->i_dev, *ip);
                        for(cp = bp->b_paddr+510;
                            (unsigned)cp >= (unsigned)bp->b_paddr; cp--)
                                if(*cp) {
                                        free(rp->i_dev, *cp);
                                }
                        brelse(bp);
                }
                free(rp->i_dev, *ip);
                *ip = 0;
        }
        if(fmt == IFLRG)
                rp->i_mode = (rp->i_mode & (~IFMT)) | IFREG;
        else if(fmt == IFLDR)
                rp->i_mode = (rp->i_mode & (~IFMT)) | IFDIR;
        rp->i_size0 = 0;
        rp->i_size1 = 0;
        rp->i_flag =| IUPD;
}

/*
 * Make a new file.
 */
maknode(mode)
{
        register *ip;

        ip = ialloc(u.u_pdir->i_dev);
        if (ip==NULL) {
                iput(u.u_pdir);
                return(NULL);
        }
        ip->i_flag =| IACC|IUPD;
        if((mode&IFMT) == 0)
                mode |= IFREG;
        ip->i_mode = mode & ~u.u_mask;
        ip->i_nlink = 1;
        ip->i_uid = u.u_uid;
        ip->i_gid = u.u_gid;
        wdir(ip);
        return(ip);
}

/*
 * Write a directory entry with
 * parameters left as side effects
 * to a call to namei.
 */
wdir(ip)
int *ip;
{
        register char *cp1, *cp2;

        u.u_dent.u_ino = ip->i_number;
        cp1 = &u.u_dent.u_name[0];
        for(cp2 = &u.u_dbuf[0]; cp2 < &u.u_dbuf[DIRSIZ];)
                *cp1++ = *cp2++;
        u.u_count = DIRSIZ+2;
        u.u_segflg = 1;
        u.u_base = &u.u_dent;
        writei(u.u_pdir);
        iput(u.u_pdir);
}
```

```
/       @(#)low.util.s  2.1

/       based on low.70.s       2.8

/ Copyright 1973 Bell Telephone Laboratories Inc
/ low core

SSR0 = 177572
br4 = 200
br5 = 240
br6 = 300
br7 = 340

/////////////////////////////////////////////////////////
/ "Real" copy of interrupt vectors live here, in data
/ space (which also corresponds to physical space).
/////////////////////////////////////////////////////////

.data

.globl  tracet, pfail, tstart, dstart, start, dump, trap

DZERO:
        jmp     dstart
. = DZERO+4
        trap; br7+0.            / bus error
. = DZERO+10
        trap; br7+1.            / illegal instruction
. = DZERO+14
        tracet; br7+2.          / bpt-trace trap
. = DZERO+20
        trap; br7+3.            / iot trap
. = DZERO+24
        pfl; br7                / power fail
. = DZERO+30
        trap; br7+5.            / emulator trap
. = DZERO+34
        trap; br7+6.            / system entry
. = DZERO+40
        .+2;    br .            / Trap catcher
        jmp     dump
. = DZERO+50
        .+2;    br .
        .+2;    br .
. = DZERO+60
        klin; br5+0
        klou; br5+0
        .+2;    br .
        .+2;    br .
. = DZERO+100
        clio; br6
        clio; br6
        .+2;    br .
. = DZERO+114
        trap; br7+10.           /parity error
        .+2;    br .
```

```
. = DZERO+124
        .+2;    br .
        .+2;    br .
        .+2;    br .
        .+2;    br .
        .+2;    br .
        .+2;    br .
        .+2;    br .
        .+2;    br .
        .+2;    br .
        .+2;    br .
. = DZERO+174
        .+2;    br .
        .+2;    br .
        .+2;    br .
        .+2;    br .
        .+2;    br .
        .+2;    br .
. = DZERO+224
        htio; br5
        .+2;    br .
        .+2;    br .
. = DZERO+240
        pir; br7+7.             / programmed interrupt
. = DZERO+244
        trap; br7+8.            / floating point
. = DZERO+250
        trap; br7+9.            / segmentation violation
. = DZERO+254
        hpio; br5
        .+2;    br .
. = DZERO+264
        .+2;    br .
        .+2;    br .
        .+2;    br .

/ floating vectors
. = DZERO+300
        .+2;    br .
        .+2;    br .
. = DZERO+310
        .+2;    br .
        .+2;    br .
. = DZERO+320
        .+2;    br .
        .+2;    br .
. = DZERO+330
        .+2;    br .
        .+2;    br .
. = DZERO+340
        .+2;    br .
        .+2;    br .
        .+2;    br .
        .+2;    br .
. = DZERO+360
        .+2;    br .
        .+2;    br .
. = DZERO+370
        .+2;    br .
        .+2;    br .
. = DZERO+400
        .+2;    br .
        .+2;    br .
. = DZERO+410
        .+2;    br .
        .+2;    br .
. = DZERO+420
        .+2;    br .
        .+2;    br .
. = DZERO+430
        .+2;    br .
        .+2;    br .
        .+2;    br .
        .+2;    br .
        .+2;    br .
        .+2;    br .
. = DZERO+460
        .+2;    br .
        .+2;    br .
        .+2;    br .
        .+2;    br .
        .+2;    br .
        .+2;    br .
        .+2;    br .
        .+2;    br .
        .+2;    br .
        .+2;    br .
        .+2;    br .
        .+2;    br .
        .+2;    br .
        .+2;    br .
        .+2;    br .
        .+2;    br .
        .+2;    br .
        .+2;    br .
        .+2;    br .
        .+2;    br .
        .+2;    br .
        .+2;    br .
        .+2;    br .
        .+2;    br .
        .+2;    br .
        .+2;    br .
        .+2;    br .
        .+2;    br .
        .+2;    br .
        .+2;    br .
        .+2;    br .
        .+2;    br .
        .+2;    br .
        .+2;    br .
        .+2;    br .
        .+2;    br .
        .+2;    br .
        .+2;    br .
        .+2;    br .
        .+2;    br .
        .+2;    br .
        .+2;    br .
        .+2;    br .
        .+2;    br .
        .+2;    br .
        .+2;    br .
        .+2;    br .
        .+2;    br .
        .+2;    br .
        .+2;    br .
        .+2;    br .
        .+2;    br .
. = DZERO+1000

pfl:    jmp     dstart
tstart: inc     SSR0
        jmp     dstart

.globl  _reboot, _rebunit, _rebstr, restart
pathadr = 172340
reset   = 5
dreboot:
        reset
        movb    $0355,*$pathadr / put in magic for autoboot
        mov     $_rebstr,r0
        mov     $pathadr+1,r1
1:      movb    (r0),(r1)+
        cmpb    $'\n,(r0)+
        bne     1b
        mov     $040000,sp
        clr     (sp)
        mov     _rebunit,r0
        jmp     *$0173020

//////////////////////////////////////////////////////////////////////
/ Copy of data space vectors live here in text space
/ since the br . is executed in I space for stray trap
//////////////////////////////////////////////////////////////////////

.text

TZERO:
        42;     br .            / 42 is illegal ins to catch stray jmp 0
. = TZERO+4
        trap; br7+0.            / bus error
. = TZERO+10
        trap; br7+1.            / illegal instruction
. = TZERO+14
        tracet; br7+2.          / bpt-trace trap
. = TZERO+20
        trap; br7+3.            / iot trap
```

```
. = TZERO+24
        pfail; br7              / power fail
. = TZERO+30
        trap; br7+5.            / emulator trap
. = TZERO+34
        trap; br7+6.            / system entry
. = TZERO+40
        .+2;    br .            / Trap catcher
        .+2;    br .
. = TZERO+50
        .+2;    br .
        .+2;    br .
. = TZERO+60
        klin; br5
        klou; br5
        .+2;    br .
        .+2;    br .
. = TZERO+100
        clio; br6
        clio; br6
        .+2;    br .
. = TZERO+114
        trap; br7+10.           /parity error
        .+2;    br .
. = TZERO+124
        .+2;    br .
        .+2;    br .
        .+2;    br .
        .+2;    br .
        .+2;    br .
        .+2;    br .
        .+2;    br .
        .+2;    br .
        .+2;    br .
        .+2;    br .
. = TZERO+174
        .+2;    br .
        .+2;    br .
        .+2;    br .
        .+2;    br .
        .+2;    br .
        .+2;    br .
. = TZERO+224
        htio; br5
        .+2;    br .
        .+2;    br .
. = TZERO+240
        pir; br7+7.             / programmed interrupt
. = TZERO+244
        trap; br7+8.            / floating point
. = TZERO+250
        trap; br7+9.            / segmentation violation
. = TZERO+254
        hpio; br5
        .+2;    br .
. = TZERO+264
```

```
        .+2;    br .
        .+2;    br .
        .+2;    br .

/ floating vectors
. = TZERO+300
        .+2;    br .
        .+2;    br .
. = TZERO+310
        .+2;    br .
        .+2;    br .
. = TZERO+320
        .+2;    br .
        .+2;    br .
. = TZERO+330
        .+2;    br .
        .+2;    br .
. = TZERO+340
        .+2;    br .
        .+2;    br .
. = TZERO+350
        .+2;    br .
        .+2;    br .
. = TZERO+360
        .+2;    br .
        .+2;    br .
. = TZERO+370
        .+2;    br .
        .+2;    br .
. = TZERO+400
        .+2;    br .
        .+2;    br .
. = TZERO+410
        .+2;    br .
        .+2;    br .
. = TZERO+420
        .+2;    br .
        .+2;    br .
. = TZERO+430
        .+2;    br .
        .+2;    br .
        .+2;    br .
        .+2;    br .
        .+2;    br .
        .+2;    br .
. = TZERO+460
        .+2;    br .
        .+2;    br .
        .+2;    br .
        .+2;    br .
        .+2;    br .
        .+2;    br .
        .+2;    br .
        .+2;    br .
        .+2;    br .
        .+2;    br .
```

```
	.+2;	br .
	.+2;	br .
	.+2;	br .
	.+2;	br .
	.+2;	br .
	.+2;	br .
	.+2;	br .
	.+2;	br .
	.+2;	br .
	.+2;	br .
	.+2;	br .
	.+2;	br .
	.+2;	br .
	.+2;	br .
	.+2;	br .
	.+2;	br .
	.+2;	br .
	.+2;	br .
	.+2;	br .
	.+2;	br .
	.+2;	br .
	.+2;	br .
	.+2;	br .
	.+2;	br .
	.+2;	br .
	.+2;	br .
	.+2;	br .
	.+2;	br .
	.+2;	br .
	.+2;	br .
	.+2;	br .
	.+2;	br .
	.+2;	br .
	.+2;	br .
	.+2;	br .
	.+2;	br .
	.+2;	br .
	.+2;	br .
	.+2;	br .
	.+2;	br .
	.+2;	br .
	.+2;	br .
. = TZERO+1000

	jmp	pfail
restart:dec	SSR0
	jmp	start
_reboot:
	reset
/	Control goes to dreboot+2 in "data" space.

/////////////////////////////////////////////////////////////
/		interface code to C
/////////////////////////////////////////////////////////////
```

```
.text

.globl  call
.globl  _klrint
klin:   jsr     r0,call; jmp _klrint
.globl  _klxint
klou:   jsr     r0,call; jmp _klxint

.globl  _clock, _clk_fn
clio:   jsr     r0,call; jmp *_clk_fn

.globl  _htintr
htio:   jsr     r0,call; jmp _htintr

.globl  _hpintr
hpio:   jsr     r0,call; jmp _hpintr

.globl  pir, _pir_fn
pir:    jsr     r0,call; jmp *_pir_fn
```

```
/       @(#)low.70.s    2.10

/ Copyright 1973 Bell Telephone Laboratories Inc
/ low core

SSR0 = 177572
br4 = 200
br5 = 240
br6 = 300
br7 = 340

////////////////////////////////////////////////////////
/ "Real" copy of interrupt vectors live here, in data
/ space (which also corresponds to physical space).
////////////////////////////////////////////////////////

.data

.globl  tracet, pfail, tstart, dstart, start, dump, trap

DZERO:
        jmp     dstart
. = DZERO+4
        trap; br7+0.            / bus error
. = DZERO+10
        trap; br7+1.            / illegal instruction
. = DZERO+14
        tracet; br7+2.          / bpt trace trap
. = DZERO+20
        trap; br7+3.            / iot trap
. = DZERO+24
        pfl; br7                / power fail
. = DZERO+30
        trap; br7+5.            / emulator trap
. = DZERO+34
        trap; br7+6.            / system entry
. = DZERO+40
        .+2;    br .            / Trap catcher
        jmp     dump
. = DZERO+50
        .+2;    br .
        .+2;    br .
. = DZERO+60
        klin; br5+0
        klou; br5+0
        .+2;    br .
        .+2;    br .
. = DZERO+100
        clio; br6
        clio; br6
        .+2;    br .
. = DZERO+114
        trap; br7+10.           /parity error
        .+2;    br .
. = DZERO+124
```

```
.text

.globl  call
.globl  _klrint
klin:   jsr     r0,call; jmp _klrint
.globl  _klxint
klou:   jsr     r0,call; jmp _klxint

.globl  _clock, _clk_fn
clio:   jsr     r0,call; jmp *_clk_fn

.globl  _htintr
htio:   jsr     r0,call; jmp _htintr

.globl  _hpintr
hpio:   jsr     r0,call; jmp _hpintr

.globl  pir, _pir_fn
pir:    jsr     r0,call; jmp *_pir_fn
```

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

#
#include "sys/param.h"
#include "sys/buf.h"
#include "sys/bufx.h"
#include "sys/user.h"
#include "sys/userx.h"
#include "sys/systm.h"
#include "sys/proc.h"
#include "sys/procx.h"
#include "sys/text.h"
#include "sys/textx.h"
#include "sys/inode.h"
#include "sys/inodex.h"
#include "sys/seg.h"
#include "sys/maus.h"
#include "sys/conf.h"
#include "sys/confx.h"
#include "sys/vtmn.h"
#include "sys/tty.h"
#ifdef DDC
#include "sys/ddc.h"
#include "sys/ddcx.h"
#endif

int     memsiz  {MAXCORE};

#define CLOCK1  0177546
#define CLOCK2  0172540
/*
 * Icode is the octal bootstrap
 * program executed in user mode
 * to bring up the system.
 */
int     icode[]
{
        0104413,        /* sys exec; init; initp */
        0000014,
        0000010,
        0000777,        /* br . */
        0000014,        /* initp: init; 0 */
        0000000,
        0062457,        /* init: </etc/init\0> */
        0061564,
        0064457,
        0064556,
        0000164,
};

/*
 * Initialization code.
 * Called from m40.s or m45.s as
 * soon as a stack and segmentation
 * have been established.
 * Functions:
 *      clear and free user core
 *      find which clock is configured
 *      hand craft 0th process
 *      call all initialization routines
 *      fork - process 0 to schedule
 *           - process 1 execute bootstrap
 *
 * panic: no clock -- neither clock responds
 * loop at loc 6 in user mode -- /etc/init
 *      cannot be executed.
 */
main()
{
        register i, *p;
        register ssize;

        /*
         * zero and free all of core
         */

        printf("CB-UNIX RELEASE 2.1");
        i = *ka6 + USIZE;
        memsiz =- i;
        UISD->r[0] = 077406;
        for(;maxmem<memsiz;) {
                UISA->r[0] = i;
                if(suiword(0, 0) < 0)
                        break;
                clearseg(i);
                maxmem++;
                mfree(coremap, 1, i);
                i++;
        }

        /*
         * Initialize the low 6 UNIBUSMAP registers to point to
         * D space 0-24K words (this area is the maximum
         * possible size of system data+bss.)
         */

        if(cputype == 70)
        for(i=0; i<=5; i++) {
                ssize = KDSA->r[i];
                UBMAP->r[i*2] = ssize<<6;
                UBMAP->r[(i*2)+1] = ssize>>10;
        }
        mfree(ubmap, 25, 6);

        /*
         * Allocate core for the external buffers
         */
        if ((bufbase = malloc(coremap, btoc(BSIZE)*NXBUF)) == NULL)
                panic("buffers");
        bufbase <<= 6;
```

```
#ifdef DDC
        /* Allocate space for message char strings
         * This area will be accessed by the Supervisor D register
         */

        if((cpool = malloc(coremap,NPOOL)) == NULL)
                panic("ddcmp char pool too large\n");
        maxmem =- NPOOL;
#endif
        /*
         * allocate core for
         * MAUS common regions
         */

#ifdef MAKEMAUS
        {
                int     maussize;
                ssize = 0;
                for(i=0; mausmap[i].boffset != -1; i++) {
                        if((maussize=mausmap[i].boffset+mausmap[i].bsize)>ssize
*   )
                                ssize = maussize;
                }
                if (mauscore = malloc(coremap, ssize)) {
                        mausend = mauscore + ssize;
                        maxmem =- ssize;
                        nmausent = i;
                }
        }
#endif

        maxmem -= msginit();

#ifdef NXCLIST
        /*
         * Allocate core for external character lists.
         */

        i = (((NXCLIST * (PACKETSIZ+2)) + 63) >> 6) & 01777;
        if((xclistbase = malloc(coremap, i)) == 0)
                panic("external clist too big\n");
        maxmem =- i;
#endif

        ssize = malloc(coremap, 0);
        printf("    *** SYSTEM SIZE=%d.%dK **** AVAIL MEM=%d.%dK ***\n",
                ctok(ssize), ctoik(ssize), ctok(maxmem), ctoik(maxmem));
        maxmem = min(maxmem, MAXMEM);
        mfree(swapmap, nswap, swplo);

        /*
         * determine clock
         */

        UISA->r[7] = ka6[1]; /* io segment */
        UISD->r[7] = 077406;
```

```
	lks = CLOCK1;
	if(fuiword(lks) == -1) {
		lks = CLOCK2;
		if(fuiword(lks) == -1)
			panic("no clock");
	}

	/*
	 * set up system process
	 */

	proc[0].p_addr = *ka6;
	proc[0].p_size = USIZE;
	proc[0].p_stat = SRUN;
	proc[0].p_flag =| SLOAD|SSYS;
	u.u_procp = &proc[0];

	/*
	 * Start clock
	 */

	*lks = 0115;

	/*
	 * Start VT11 monitor
	 */

	VTOPEN();
	VTNEWPROC(p=proc,PR_NAME,"UNIX Scheduler",15);
	VTNEWPROC(p,PR_NEW,p,0);
	VTPROCENT(p,PR_BLINK);

	/*
	 * Set up known i-nodes
	 */

	cinit();
	binit();
	errinit();
	iinit();
#ifdef DDC
	ddcinit();
#endif
	rootdir = iget(rootdev, ROOTINO);
	rootdir->i_flag =& ~ILOCK;
	u.u_cdir = u.u_rdir = rootdir;
	rootdir->i_count =+ 2;

	/*
	 * make init process
	 * enter scheduling loop
	 * with system process
	 */

	if(newproc()) {
		expand(USIZE+1);
```

```
                estabur(0, 1, 0, 0, RO);
                copyout(icode, 0, sizeof icode);
                /*
                 * Return goes to loc. 0 of user init
                 * code just copied out.
                 */
                return;
        }
        sched();
}

/*
 * Load the user hardware segmentation
 * registers from the software prototype.
 * The software registers must have
 * been setup prior by estabur.
 */
sureg()
{
        register *udp, *uap, *rdp;
        int *rap, daddr, taddr, *limudp;

        taddr = daddr = u.u_procp->p_addr;
        if (udp=u.u_procp->p_textp)
                taddr = udp->x_caddr;
        limudp = &u.u_uisd[16];
        if (cputype==40)
                limudp = &u.u_uisd[8];
        rap = UISA;
        rdp = UISD;
        uap = &u.u_uisa[0];
        for (udp = &u.u_uisd[0]; udp < limudp;) {
                if(*udp&ABS)
                        *rap++ = *uap++;
                else
                        *rap++ = *uap++ + (*udp&TX? taddr: daddr);
                *rdp++ = *udp++;
        }
}

/*
 * Set up software prototype segmentation
 * registers to implement the 3 pseudo
 * text,data,stack segment sizes passed
 * as arguments.
 * The argument sep specifies if the
 * text and data+stack segments are to
 * be separated.
 * The last argument determines whether the text
 * segment is read-write or read-only.
 */
estabur(nt, nd, ns, sep, xrw)
{
        register a, *ap, *dp;
#ifdef MAKEMAUS
        int *limudp;
        char bitm;
#endif

        if(checkur(nt, nd, ns, sep))
                return(-1);

        a = 0;
        ap = &u.u_uisa[0];
        dp = &u.u_uisd[0];
        while(nt >= 128) {
                *dp++ = (127<<8) | xrw|TX;
                *ap++ = a;
                a =+ 128;
                nt =- 128;
        }
        if(nt) {
                *dp++ = ((nt-1)<<8) | xrw|TX;
                *ap++ = a;
        }
        if(sep)
        while(ap < &u.u_uisa[8]) {
                *ap++ = 0;
                *dp++ = 0;
        }
        a = USIZE;
        while(nd >= 128) {
                *dp++ = (127<<8) | RW;
                *ap++ = a;
                a =+ 128;
                nd =- 128;
        }
        if(nd) {
                *dp++ = ((nd-1)<<8) | RW;
                *ap++ = a;
                a =+ nd;
        }
#ifdef MAKEMAUS
        if(sep) {
                limudp = &u.u_uisa[16];
                bitm = u.u_mbitm >> ((ap - u.u_uisa) - 8);
        } else {
                limudp = &u.u_uisa[8];
                bitm = u.u_mbitm >> (ap - u.u_uisa);
        }
        while(ap < limudp) {
                if(bitm&01) {
                        dp++;
                        ap++;
                } else {
                        *dp++ = 0;
                        *ap++ = 0;
                }
                bitm =>> 1;
        }
#endif
#ifndef MAKEMAUS
```

```
	while(ap < &u.u_uisa[8]) {
		*dp++ = 0;
		*ap++ = 0;
	}
	if(sep)
	while(ap < &u.u_uisa[16]) {
		*dp++ = 0;
		*ap++ = 0;
	}
#endif
	a =+ ns;
	while(ns >= 128) {
		a =- 128;
		ns =- 128;
		*--dp = (127<<8) | RW;
		*--ap = a;
	}
	if(ns) {
		*--dp = ((128-ns)<<8) | RW | ED;
		*--ap = a-128;
	}
	if(!sep) {
		ap = &u.u_uisa[0];
		dp = &u.u_uisa[8];
		while(ap < &u.u_uisa[8])
			*dp++ = *ap++;
		ap = &u.u_uisd[0];
		dp = &u.u_uisd[8];
		while(ap < &u.u_uisd[8])
			*dp++ = *ap++;
	}
	sureg();
	return(0);

}
checkur(nt, nd, ns, sep)
{
#ifdef MAKEMAUS
	int last, first;
	char bitm;

	last = 0;
	first = 8;
	for(bitm=u.u_mbitm; bitm; bitm =<< 1) {
		first--;
		if(bitm < 0 && last == 0)
			last = first;
	}
#endif

	if(sep) {
		if(cputype == 40)
			goto err;
		if(ctos(nt) > 8)
			goto err;
#ifdef MAKEMAUS
```

```
                if(ctos(nd) > first || 8-ctos(ns) <= last)
                        goto err;
#endif
                if(ctos(nd) + ctos(ns) > 8)
                        goto err;
        } else {
#ifdef MAKEMAUS
                if(ctos(nt) + ctos(nd) > first || 8-ctos(ns) <= last)
                        goto err;
#endif
                if(ctos(nt) + ctos(nd) + ctos(ns) > 8)
                        goto err;
        }
        if(nt+nd+ns+USIZE > ((maxmem)(32*32) && sep==0)? (32*32): maxmem))
                goto err;
        return(0);

err;
        u.u_error = ENOMEM;
        return(-1);
}
```

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

#
/*
 * maus system call
 */

#include "sys/param.h"
#include "sys/systm.h"
#include "sys/user.h"
#include "sys/userx.h"
#include "sys/inode.h"
#include "sys/inodex.h"
#include "sys/maus.h"
#include "sys/conf.h"
#include "sys/confx.h"
#include "sys/seg.h"
#include "sys/reg.h"

#define READM   0
#define WRITEM  1
#define RDWRM   2
#define FREEM   3
#define SWITM   4

int nmausent;
int mauscore;
struct mausmap  mausmap[];

maus()
{
        register i, j, *ip;
        int min;
        extern uchar();

        switch(i=u.u_ar0[R0]) {
            case READM:
            case WRITEM:
            case RDWRM:
                if((ip = namei(&uchar, 0)) == NULL)
                        return;
                if(i == READM || i == RDWRM)
                        access(ip, IREAD);
                if(i == WRITEM || i == RDWRM)
                        access(ip, IWRITE);
                min = (minor(ip->i_un.i_rdev) - 8) & 0377;
                if(major(ip->i_un.i_rdev) != MEMDEV || min >= nmausent)
                        u.u_error = EINVAL;
                iput(ip);
                if(u.u_error)
                        return;
                for(j=0; u.u_msav[j].ms_uisd;)
                        if(++j >= NMSAV) {
                                u.u_error = EMFILE;
                                return;
                }
        if(i == READM)
                i = RO;
        else
                i = RW;
        u.u_msav[j].ms_uisd = i | ABS | (mausmap[min].bsize-1)<<8;
        u.u_msav[j].ms_uisa = mausmap[min].boffset + mauscore;
        u.u_ar0[R0] = j;
        return;

    case FREEM:
        if((j=u.u_arg[0]) >= NMSAV || u.u_msav[j].ms_uisd == 0) {
                u.u_error = EINVAL;
                return;
        }
        u.u_msav[j].ms_uisd = 0;
        u.u_msav[j].ms_uisa = 0;
        return;

    case SWITM:
        if((j = u.u_arg[0]) != -1) {
                j = ((j >> 13) & 07) + 8;
                if((u.u_mbitm & (1 << (j-8))) == 0 && u.u_uisd[j]) {
                        u.u_error = EINVAL;     /* vaddr illegal */
                        return;
                }
        } else for(j=8; u.u_uisd[j];)
                if(++j > 14) {
                        u.u_error = ENOMEM;     /* no reg left */
                        return;
                }
        if(u.u_ar0[R1] == -1) {         /* disable case */
                u.u_uisd[j] = 0;
                u.u_uisa[j] = 0;
                if(cputype != 40) {
                        UISD->r[j] = 0;
                        UISA->r[j] = 0;
                }
                j =- 8;
                if(u.u_sep == 0) {
                        UISD->r[j] = 0;
                        UISA->r[j] = 0;
                        u.u_uisd[j] = 0;
                        u.u_uisa[j] = 0;
                }
                u.u_mbitm =& ~(1 << j);
                return;
        }
        if((i = u.u_ar0[R1]) >= NMSAV || u.u_msav[i].ms_uisd == 0) {
                u.u_error = EINVAL;     /* mdes illegal */
                return;
        }
        u.u_uisd[j] = u.u_msav[i].ms_uisd;
        u.u_uisa[j] = u.u_msav[i].ms_uisa;
        if(cputype != 40) {
                UISD->r[j] = u.u_uisd[j];
```

```
                        UISA->r[j] = u.u_uisa[j];
                }
                j =- 8;
                if(u.u_sep == 0) {
                        UISD->r[j] = u.u_uisd[j] = u.u_uisd[j+8];
                        UISA->r[j] = u.u_uisa[j] = u.u_uisa[j+8];
                }
                u.u_mbitm =| 1 << j;
                u.u_ar0[R0] = j << 13;
                return;

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