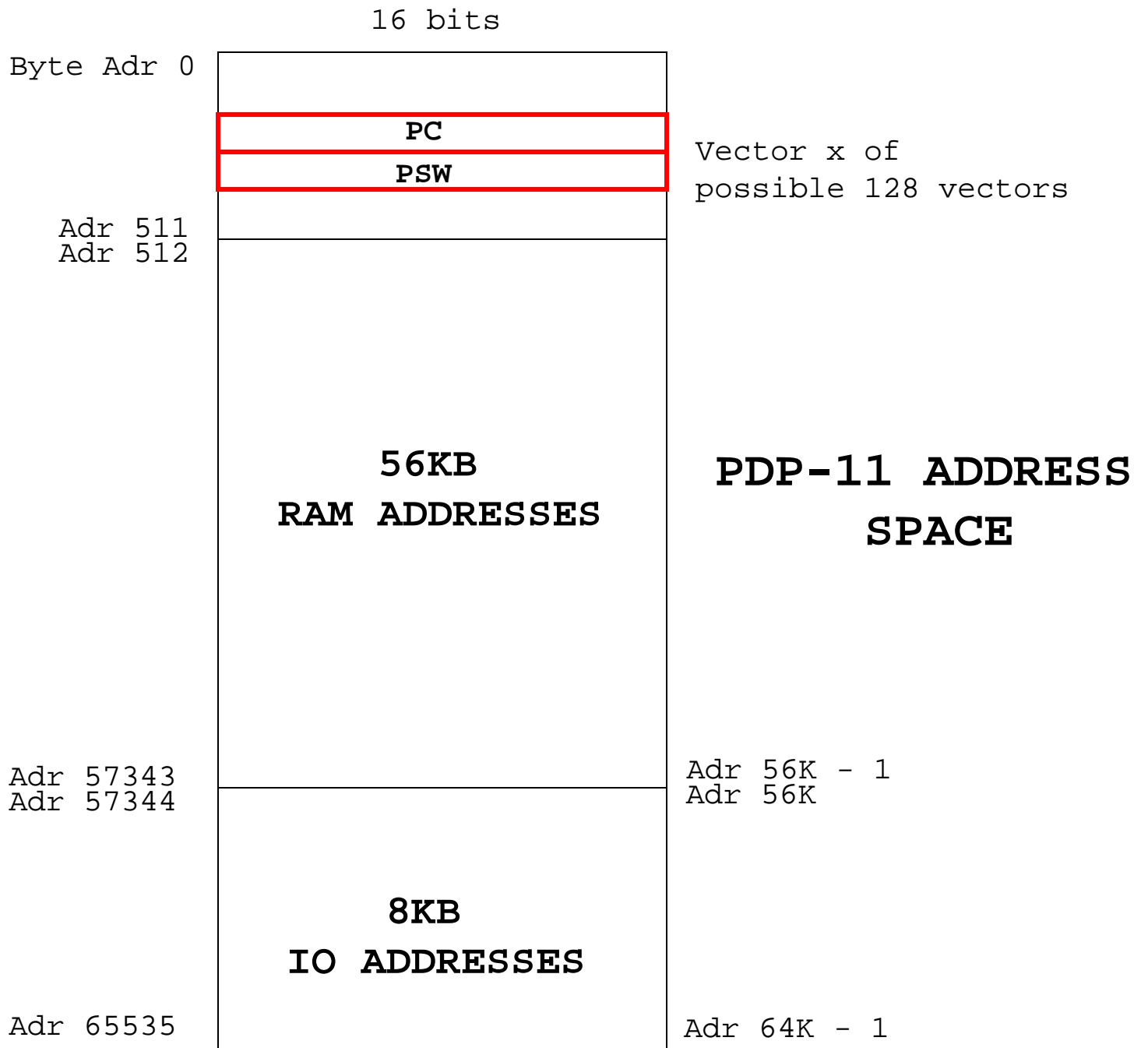
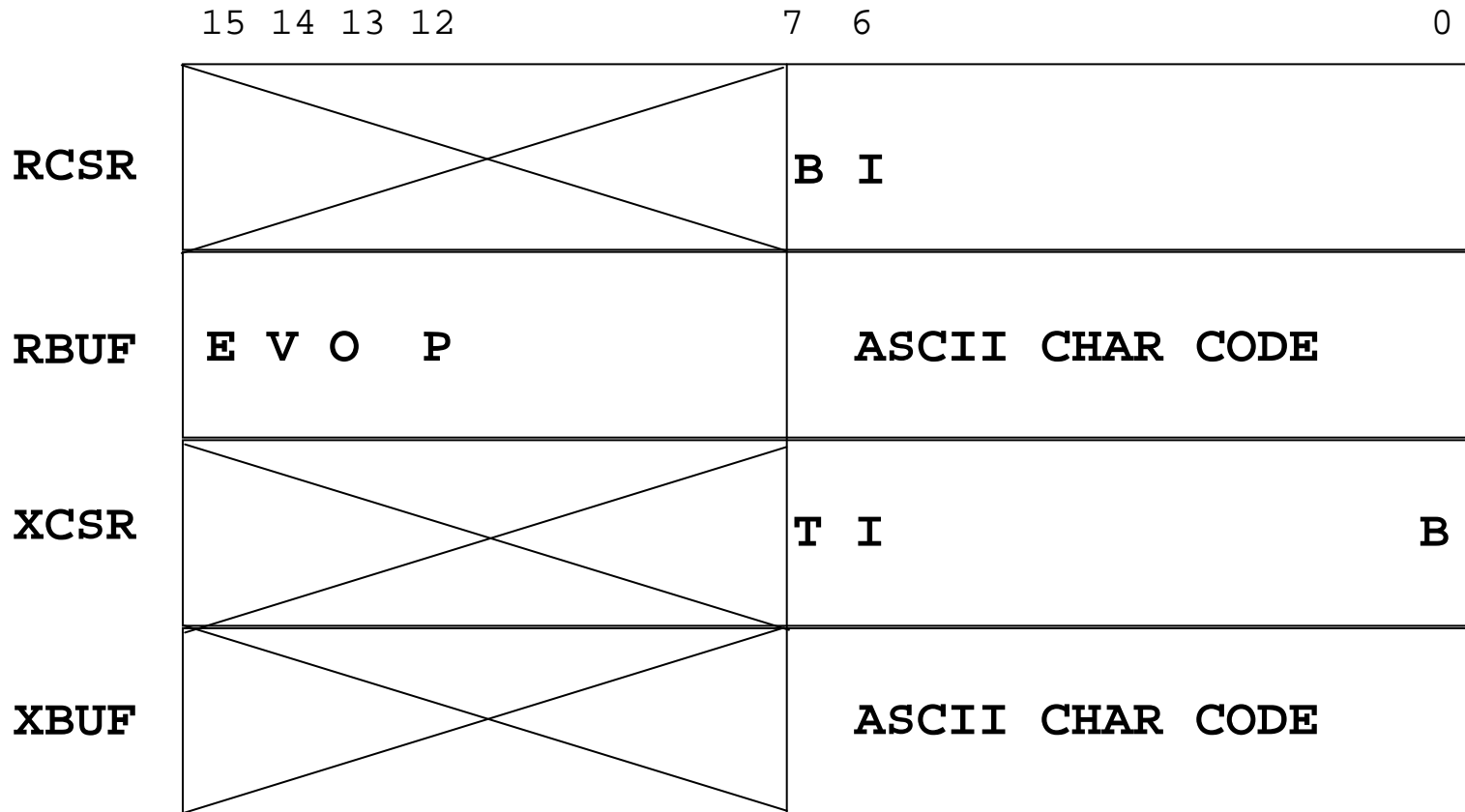


PDP-11 CPU REGISTERS

16 bits

R0
R1
R2
R3
R4
R5
R6 SP
R7 PC
PSW





```

/* q.h - firstid, firstkey, isempty, lastkey, nonempty      */
/* q structure declarations, constants, and inline procedures */

#ifndef NQENT
#define NQENT          NPROC + NSEM + NSEM + 4 /* for ready & sleep */
#endif

struct qent {
    /* one for each process plus two for      */
    /* each list                               */
    short qkey; /* key on which the queue is ordered */
    short qnext; /* pointer to next process or tail */
    short qprev; /* pointer to previous process or head */
};

extern struct qent q[];
extern int nextqueue;

/* inline list manipulation procedures */

#define isempty(list) (q[(list)].qnext >= NPROC)
#define nonempty(list) (q[(list)].qnext < NPROC)
#define firstkey(list) (q[q[(list)].qnext].qkey)
#define lastkey(tail) (q[q[(tail)].qprev].qkey)
#define firstid(list) (q[(list)].qnext)

#define EMPTY -1 /* equivalent of null pointer */

```

```
/* queue.c - dequeue, enqueue */
```

```
#include <conf.h>
```

```
#include <kernel.h>
```

```
#include <q.h>
```

```
/*-----
```

```
* enqueue -- insert an item at the tail of a list
```

```
*-----
```

```
*/
```

```
int enqueue(item, tail)
```

```
int item; /* item to enqueue on a list */
```

```
int tail; /* index in q of list tail */
```

```
{
```

```
struct qent *tptr; /* points to tail entry */
```

```
struct qent *mptr; /* points to item entry */
```

```
tptr = &q[tail];
```

```
mptr = &q[item];
```

```
mptr->qnext = tail;
```

```
mptr->qprev = tptr->qprev;
```

```
q[tptr->qprev].qnext = item;
```

```
tptr->qprev = item;
```

```
return(item);
```

```
}
```

```

/*-----
 * dequeue  --  remove an item from a list and return it
 *-----
 */
int  dequeue(item)
    int  item;
{
    struct qent  *mptr;  /* pointer to q entry for item */

    mptr = &q[item];
    q[mptr->qprev].qnext = mptr->qnext;
    q[mptr->qnext].qprev = mptr->qprev;
    return(item);
}

```

```

/* insert.c - insert */

#include <conf.h>
#include <kernel.h>
#include <q.h>

/*-----
 * insert.c -- insert an process into a q list in key order
 *-----
 */
int      insert(proc, head, key)
    int      proc;          /* process to insert          */
    int      head;         /* q index of head of list   */
    int      key;          /* key to use for this process */
{
    int      next;         /* runs through list        */
    int      prev;

    next = q[head].qnext;
    while (q[next].qkey < key)      /* tail has MAXINT as key   */
        next = q[next].qnext;
    q[proc].qnext = next;
    q[proc].qprev = prev = q[next].qprev;
    q[proc].qkey = key;
    q[prev].qnext = proc;
    q[next].qprev = proc;
    return(OK);
}

```

```

/* getitem.c - getfirst, getlast */

#include <conf.h>
#include <kernel.h>
#include <q.h>

/*-----
 * getfirst -- remove and return the first process on a list
 *-----
 */
int getfirst(head)
int head; /* q index of head of list */
{
int proc; /* first process on the list */

if ((proc=q[head].qnext) < NPROC)
return( dequeue(proc) );
else
return(EMPTY);
}

```

```

/*-----
 * getlast  --  remove and return the last process from a list
 *-----
 */
int  getlast(tail)
    int  tail;          /* q index of tail of list */
{
    int  proc;         /* last process on the list */

    if ((proc=q[tail].qprev) < NPROC)
        return( dequeue(proc) );
    else
        return(EMPTY);
}

```

```

/* newqueue.c - newqueue */

#include <conf.h>
#include <kernel.h>
#include <q.h>

/*-----
 * newqueue -- initialize a new list in the q structure
 *-----
 */
int newqueue()
{
    struct qent *hptr;          /* address of new list head */
    struct qent *tptr;          /* address of new list tail */
    int hindex, tindex;        /* head and tail indexes */

    hptr = &q[ hindex=nextqueue++ ]; /* nextqueue is global variable */
    tptr = &q[ tindex=nextqueue++ ]; /* giving next used q pos. */
    hptr->qnext = tindex;
    hptr->qprev = EMPTY;
    hptr->qkey = MININT;
    tptr->qnext = EMPTY;
    tptr->qprev = hindex;
    tptr->qkey = MAXINT;
    return(hindex);
}

```

```

/* proc.h - isbadpid */

/* process table declarations and defined constants */

#ifndef NPROC /* set the number of processes */
#define NPROC 10 /* allowed if not already done */
#endif

/* process state constants */

#define PRCURR '\001' /* process is currently running */
#define PRFREE '\002' /* process slot is free */
#define PRREADY '\003' /* process is on ready queue */
#define PRRECV '\004' /* process waiting for message */
#define PRSLEEP '\005' /* process is sleeping */
#define PRSUSP '\006' /* process is suspended */
#define PRWAIT '\007' /* process is on semaphore queue*/

/* miscellaneous process definitions */

#define PNREGS 9 /* size of saved register area */
#define PNMLEN 8 /* length of process "name" */
#define NULLPROC 0 /* id of the null process; it
/* is always eligible to run */
#define BADPID -1 /* used when invalid pid needed */

#define isbadpid(x) (x<=0 || x>=NPROC)

```

```

/* process table entry */

struct pentry {
    char    pstate;          /* process state: PRCURR, etc. */
    short   pprio;          /* process priority */
    short   pregs[PNREGS];  /* saved regs. R0-R5,SP,PC,PS */
    short   psem;          /* semaphore if process waiting */
    short   pmsg;          /* message sent to this process */
    Bool    phasmsg;       /* True iff pmsg is valid */
    short   pbase;         /* base of run time stack */
    short   pstklen;       /* stack length */
    short   plimit;        /* lowest extent of stack */
    char    pname[PNMLEN]; /* process name */
    short   pargs;         /* initial number of arguments */
    short   paddr;         /* initial code address */
};

extern struct pentry proctab[];
extern int numproc;      /* currently active processes */
extern int nextproc;    /* search point for free slot */
extern int currpids;    /* currently executing process */

```

```

/* resched.c - resched */

#include <conf.h>
#include <kernel.h>
#include <proc.h>
#include <q.h>

/*-----
 * resched -- reschedule processor to highest priority ready process
 *
 * Notes:      Upon entry, currpid gives current process id.
 *             Proctab[currpid].pstate gives correct NEXT state for
 *             current process if other than PRCURR.
 *-----
 */
int resched()
{
    register struct pentry *optr; /* pointer to old process entry */
    register struct pentry *nptr; /* pointer to new process entry */

    /* no switch needed if current process priority higher than next*/

    if ( ( (optr= &proctab[currpid])->pstate == PRCURR) &&
        (lastkey(rdytail)<optr->pprio))
        return(OK);
}

```

```

/* force context switch */

if (optr->pstate == PRCURR) {
    optr->pstate = PRREADY;
    insert(currpid,rdyhead,optr->pprio);
}

/* remove highest priority process at end of ready list */

nptr = &proctab[ (currpid = getlast(rdytail)) ];
nptr->pstate = PRCURR;          /* mark it currently running */
#ifdef RTCLOCK
preempt = QUANTUM;            /* reset preemption counter */
#endif
ctxsw(optr->pregs,nptr->pregs);

/* The OLD process returns here when resumed. */
return(OK);
}

```

```

/* ctxsw.s - ctxsw */

/*-----
/* ctxsw -- actually perform context switch, saving/loading registers
/*-----
/ The stack contains three items upon entry to this routine:
/
/   SP+4 => address of 9 word save area with new registers + PS
/   SP+2 => address of 9 word save area for old registers + PS
/   SP   => return address
/
/ The saved state consists of: the values of R0-R5 upon entry, SP+2,
/ PC equal to the return address, and the PS (i.e., the PC and SP are
/ saved as if the calling process had returned to its caller).

        .globl  _ctxsw          / declare the routine name global
_ctxsw:          / entry point to context switch
mov        r0,*2(sp)        / Save old R0 in old register area
mov        2(sp),r0        / Get address of old register area
add        $2,r0          /   in R0; increment to saved pos. of R1
mov        r1,(r0)+        / Save registers R1-R5 in successive
mov        r2,(r0)+        /   locations of the old process
mov        r3,(r0)+        /   register save area. (r0)+ denotes
mov        r4,(r0)+        /   indirect reference and, as a side
mov        r5,(r0)+        /   effect, incrementing r0 to next word.
add        $2,sp          / move sp beyond the return address,
                        /   as if a return had occurred.

```

```

mov     sp,(r0)+      / save stack pointer
mov     -(sp),(r0)+   / Save caller's return address as PC
mfps    (r0)          / Save processor status beyond registers
mov     4(sp),r0      / Pick up address of new registers in R0
                               / Ready to load registers for the new
                               /   process and abandon the old stack.
mov     2(r0),r1      / Load R1-R5 and SP from the saved area
mov     4(r0),r2      /   for the new process.
mov     6.(r0),r3     / NOTE: dot following a number makes it
mov     8.(r0),r4     /   decimal; all others are octal
mov     10.(r0),r5
mov     12.(r0),sp    / Have now actually switched stacks
mov     16.(r0),-(sp) / Push new process PS on new process stack
mov     14.(r0),-(sp) / Push new process PC on new process stack
mov     (r0),r0       / Finally, load R0 from new area
rtt      / Load PC, PS, and reset SP all at once

```

```

/* ready.c - ready */

#include <conf.h>
#include <kernel.h>
#include <proc.h>
#include <q.h>

/*-----
 * ready -- make a process eligible for CPU service
 *-----
 */
int ready (pid, resch)
    int pid; /* id of process to make ready */
    int resch; /* reschedule afterward? */
{
    register struct pentry *pptr;

    if (isbadpid(pid))
        return(SYSERR);
    pptr = &proctab[pid];
    pptr->pstate = PRREADY;
    insert(pid,rdyhead,pptr->pprio);
    if (resch)
        resched();
    return(OK);
}

```

```

/* resume.c - resume */

#include <conf.h>
#include <kernel.h>
#include <proc.h>

/*-----
 * resume -- unsuspend a process, making it ready; return the priority
 *-----
 */
SYSCALL resume(pid)
    int    pid;
{
    char    ps;                /* saved processor status */
    struct pentry *pptr;      /* pointer to proc. tab. entry */
    int    prio;              /* priority to return */

    disable(ps);
    if (isbadpid(pid) || (pptr = &proctab[pid])->pstate != PRSUSP) {
        restore(ps);
        return(SYSERR);
    }
    prio = pptr->pprio;
    ready(pid, RESCHYES);
    restore(ps);
    return(prio);
}

```

```

/* suspend.c - suspend */
#include <conf.h>
#include <kernel.h>
#include <proc.h>
/*-----
 * suspend -- suspend a process, placing it in hibernation
 *-----
 */
SYSCALL suspend(int pid){
    struct pentry *pptr;
    char ps;
    int prio;
    disable(ps);
    /* id of process to suspend */
    /* pointer to proc. tab. entry */
    /* saved processor status */
    /* priority returned */
    if (isbadpid(pid) || pid==NULLPROC ||
        ((pptr= &proctab[pid])->pstate!=PRCURRE && pptr->pstate!=PRREADY)) {
        restore(ps);
        return(SYSERR);
    }
    if (pptr->pstate == PRREADY) {
        dequeue(pid);
        pptr->pstate = PRSUSP;
    } else {
        pptr->pstate = PRSUSP;
        resched();
    }
    prio = pptr->pprio;
    restore(ps);
    return(prio);
}

```

```

/* kill.c - kill */

#include <conf.h>
#include <kernel.h>
#include <proc.h>
#include <sem.h>
#include <mem.h>
#include <io.h>

/*-----
 * kill  --  kill a process and remove it from the system
 *-----
 */
SYSCALL kill(pid)
    int      pid;          /* process to kill          */
{
    struct pentry *pptr;   /* points to proc. table for pid*/
    char      ps;

    disable(ps);
    if (isbadpid(pid) || (pptr= &proctab[pid])->pstate==PRFREE) {
        restore(ps);
        return(SYSERR);
    }
}

```

```

freestk(pptr->pbase, pptr->pstklen);
switch (pptr->pstate) {

    case PRCURR:  pptr->pstate = PRFREE; /* suicide */
                  resched();

    case PRWAIT:  semaph[pptr->psem].semcnt++;
                  /* fall through */

    case PRSLEEP:
    case PRREADY: dequeue(pid);

    default:      pptr->pstate = PRFREE;
}
restore(ps);
return(OK);
}

```

```
/* kernel.h - disable, enable, halt, restore, isodd, min */
```

```
/* Symbolic constants used throughout Xinu */
```

```
typedef char          Bool;          /* Boolean type          */
#define FALSE        0              /* Boolean constants    */
#define TRUE         1
#define NULL         (char *)0      /* Null pointer for linked lists*/
#define NULLCH       '\0'          /* The null character   */
#define NULLSTR      ""            /* Pointer to empty string */
#define SYSCALL      int           /* System call declaration */
#define LOCAL        static        /* Local procedure declaration */
#define INTPROC      int           /* Interrupt procedure  */
#define PROCESS      int           /* Process declaration  */
#define RESCHYES     1             /* tell ready to reschedule */
#define RESCHNO     0             /* tell ready not to resch. */
#define MININT       0100000      /* minimum integer (-32768) */
#define MAXINT       0077777      /* maximum integer      */
#define LOWBYTE      0377         /* mask for low-order 8 bits */
#define HIBYTE       0177400      /* mask for high 8 of 16 bits */
#define LOW16        0177777      /* mask for low-order 16 bits */
#define SP           6            /* reg. 6 is stack pointer */
#define PC           7            /* reg. 7 is program counter */
#define PS           8            /* proc. status in 8th reg. loc */
#define MINSTK       40           /* minimum process stack size */
#define NULLSTK      300          /* process 0 stack size  */
#define DISABLE      0340         /* PS to disable interrupts */
```

```

/* Universal return constants */

#define OK          1          /* system call ok          */
#define SYSERR     -1          /* system call failed      */
/*      (usu. defined as ^B)  */

/* Initialization constants */

#define INITARGC    1          /* initial process argc    */
#define INITSTK    500        /* initial process stack   */
#define INITPRIO   20         /* initial process priority */
#define INITNAME   "main"     /* initial process name    */
#define INITRET    userret    /* processes return address */
#define INITPS     0          /* initial process PS      */
#define INITREG    0          /* initial register contents */
#define QUANTUM    10         /* clock ticks until preemption */

/* Miscellaneous utility inline functions */

#define isodd(x)    (01&(int)(x))
#define min(a,b)   ( (a) < (b) ? (a) : (b) )
#define disable(ps)  asm("mfps ~ps");asm("mtps $0340")
#define restore(ps)  asm("mtps ~ps") /* restore interrupt status */
#define enable()     asm("mtps $000") /* enable interrupts      */
#define pause()      asm("wait")    /* machine "wait for interr." */
#define halt()       asm("halt")    /* machine halt instruction */

extern int    rdyhead, rdytail;
extern int    preempt;

```

```

/* create.c - create, newpid */
#include <conf.h>
#include <kernel.h>
#include <proc.h>
#include <mem.h>
#include <io.h>
/*-----
 * create - create a process to start running a procedure
 *-----
 */
SYSCALL create(procaddr, ssize, priority, name, nargs, args)
    int      *procaddr;          /* procedure address          */
    int      ssize;              /* stack size in words       */
    int      priority;           /* process priority > 0      */
    char     *name;              /* name (for debugging)      */
    int      nargs;             /* number of args that follow */
    int      args;              /* arguments (treated like an
                                /* array in the code)        */

{
    int      pid;                /* stores new process id     */
    struct   pentry *pptr;       /* pointer to proc. table entry */
    int      i;
    int      *a;                 /* points to list of args    */
    int      *saddr;             /* stack address              */
    char     ps;                 /* saved processor status    */
    int      INITRET();
    disable(ps);
    ssize = roundew(ssize);

```

```

if ( ssize < MINSTK || ((saddr=getstk(ssize)) == SYSERR ) ||
    (pid=newpid()) == SYSERR || isodd(procaddr) ||
    priority < 1 ) {
    restore(ps);
    return(SYSERR);
}
numproc++;
pptr = &proctab[pid];
pptr->pstate = PRSUSP;
for (i=0 ; i<PNMLEN && (pptr->pname[i]=name[i])!=0 ; i++);
pptr->pprio = priority;
pptr->pbase = (short)saddr;
pptr->pstklen = ssize;
pptr->psem = 0;
pptr->phasmsg = FALSE;
pptr->plimit = (short)(saddr - ssize + 1);
pptr->pargs = nargs;
for (i=0 ; i<PNREGS ; i++) pptr->pregs[i]=INITREG;
pptr->pregs[PC] = pptr->paddr = (short)procaddr;
pptr->pregs[PS] = INITPS;
a = (&nargs) + (nargs-1);          /* point to last argument      */
for ( ; nargs > 0 ; nargs--)      /* machine dependent; copy args */
    *saddr-- = *a--;              /* onto created process' stack */
*saddr = (int)INITRET;           /* push on return address      */
pptr->pregs[SP] = (int)saddr;
restore(ps);
return(pid);
}

```

```

/*-----
 * newpid -- obtain a new (free) process id
 *-----
 */
LOCAL newpid()
{
    int pid; /* process id to return */
    int i;

    for (i=0 ; i<NPROC ; i++) { /* check all NPROC slots */
        if ( (pid=nextproc--) <= 0)
            nextproc = NPROC-1;
        if (proctab[pid].pstate == PRFREE)
            return(pid);
    }
    return(SYSERR);
}

```

```
/* userret.c - userret */
```

```
#include <conf.h>
```

```
#include <kernel.h>
```

```
/*-----
```

```
* userret -- entered when a process exits by return
```

```
*-----
```

```
*/
```

```
userret()
```

```
{
```

```
    kill( getpid() );
```

```
}
```

```

/* chprio.c - chprio */
#include <conf.h>
#include <kernel.h>
#include <proc.h>
/*-----
 * chprio -- change the scheduling priority of a process
 *-----
 */
SYSCALL chprio(pid,newprio)
    int    pid;
    int    newprio;                /* newprio > 0          */
{
    int    oldprio;
    struct pentry *pptr;
    char    ps;

    disable(ps);
    if (isbadpid(pid) || newprio<=0 ||
        (pptr = &proctab[pid])->pstate == PRFREE) {
        restore(ps);
        return(SYSERR);
    }
    oldprio = pptr->pprio;
    pptr->pprio = newprio;
    restore(ps);
    return(oldprio);
}

```

```
/* getpid.c - getpid */

#include <conf.h>
#include <kernel.h>
#include <proc.h>

/*-----
 * getpid -- get the process id of currently executing process
 *-----
 */
SYSCALL getpid()
{
    return(currpid);
}
```

```

/* sem.h - isbadsem */

#ifndef NSEM
#define NSEM          45      /* number of semaphores, if not defined */
#endif

#define SFREE        '\01'   /* this semaphore is free          */
#define SUSED       '\02'   /* this semaphore is used          */

struct sentry {             /* semaphore table entry          */
    char    sstate;         /* the state SFREE or SUSED       */
    short   semcnt;        /* count for this semaphore       */
    short   sqhead;        /* q index of head of list        */
    short   sqtail;        /* q index of tail of list        */
};

extern struct sentry semaph[];
extern int    nextsem;

#define isbadsem(s)        (s<0 || s>=NSEM)

```

```

/* wait.c - wait */
#include <conf.h>
#include <kernel.h>
#include <proc.h>
#include <q.h>
#include <sem.h>
/*-----
 * wait -- make current process wait on a semaphore
 *-----*/
SYSCALL wait(sem)
    int      sem;
{
    char      ps;
    register struct sentry *sptr;
    register struct pentry *pptr;
    disable(ps);
    if (isbadsem(sem) || (sptr= &semaph[sem])->sstate==SFREE) {
        restore(ps);
        return(SYSERR);
    }
    if (--(sptr->semcnt) < 0) {
        (pptr = &proctab[currpid])->pstate = PRWAIT;
        pptr->psem = sem;
        enqueue(currpid, sptr->sqtail);
        resched();
    }
    restore(ps);
    return(OK);
}

```

```

/* signal.c - signal */

#include <conf.h>
#include <kernel.h>
#include <proc.h>
#include <q.h>
#include <sem.h>

/*-----
 * signal -- signal a semaphore, releasing one waiting process
 *-----
 */
SYSCALL signal(sem)
register int sem;
{
    register struct sentry *sptr;
    char ps;

    disable(ps);
    if (isbadsem(sem) || (sptr= &semaph[sem])->sstate==SFREE) {
        restore(ps);
        return(SYSERR);
    }
    if ((sptr->semcnt++) < 0)
        ready(getfirst(sptr->sqhead), RESCHYES);
    restore(ps);
    return(OK);
}

```

```

/* screate.c - screate, newsem */

#include <conf.h>
#include <kernel.h>
#include <proc.h>
#include <q.h>
#include <sem.h>
/*-----
 * screate -- create and initialize a semaphore, returning its id
 *-----
 */
SYSCALL screate(count)
    int      count;                /* initial count (>=0) */
{
    char     ps;
    int      sem;

    disable(ps);
    if ( count<0 || (sem=newsem())==SYSERR ) {
        restore(ps);
        return(SYSERR);
    }
    semaph[sem].semcnt = count;
    /* sqhead and sqtail were initialized at system startup */
    restore(ps);
    return(sem);
}

```

```

/*-----
 * newsem -- allocate an unused semaphore and return its index
 *-----
 */
LOCAL newsem( )
{
    int    sem;
    int    i;

    for (i=0 ; i<NSEM ; i++) {
        sem=nextsem--;
        if (nextsem < 0)
            nextsem = NSEM-1;
        if (semaph[sem].sstate==SFREE) {
            semaph[sem].sstate = SUSED;
            return(sem);
        }
    }
    return(SYSERR);
}

```

```

/* sdelete.c - sdelete */
#include <conf.h>
#include <kernel.h>
#include <proc.h>
#include <q.h>
#include <sem.h>
/*-----
 * sdelete -- delete a semaphore by releasing its table entry
 *----- */
SYSCALL sdelete(sem)
    int    sem;
{
    char    ps;
    int     pid;
    struct  sentry *sptr;          /* address of sem to free */
    disable(ps);
    if (isbadsem(sem) || semaph[sem].sstate==SFREE) {
        restore(ps);
        return(SYSERR);
    }
    sptr = &semaph[sem];
    sptr->sstate = SFREE;
    if (nonempty(sptr->sqhead)) { /* free waiting processes */
        while( (pid=getfirst(sptr->sqhead)) != EMPTY)
            ready(pid,RESCHNO);
        resched();
    }
    restore(ps);
    return(OK);
}

```

Setting Async Input Behavior

```
int      ch_status_flag1, ch_status_flag2;  
  
ch_status_flag1 = fcntl(DEVCONIN, F_GETFL, 0);  
ch_status_flag2 = ch_status_flag1;  
  
ch_status_flag2 |= O_NONBLOCK;  
  
if(fcntl(DEVCONIN, F_SETFL, ch_status_flag2)  
    == -1){  
    perror("fcntl f_setfl error");  
    exit(1);  
}
```

```

/* send.c - send */
#include <conf.h>
#include <kernel.h>
#include <proc.h>
/*-----
 * send -- send a message to another process
 *-----
 */
SYSCALL send(pid, msg)
int pid;
int msg;
{
    struct pentry *pptr;          /* receiver's proc. table addr. */
    char ps;

    disable(ps);
    if (isbadpid(pid) || ( (pptr= &proctab[pid])->pstate == PRFREE)
        || pptr->phasmsg) {
        restore(ps);
        return(SYSERR);
    }
    pptr->pmsg = msg;             /* deposit message */
    pptr->phasmsg = TRUE;
    if (pptr->pstate == PRRCV)   /* if receiver waits, start it */
        ready(pid, RESCHYES);
    restore(ps);
    return(OK);
}

```

```

/* receive.c - receive */
#include <conf.h>
#include <kernel.h>
#include <proc.h>
/*-----
 * receive - wait for a message and return it
 *-----
*/
SYSCALL receive()
{
    struct pentry *pptr;
    int msg;
    char ps;

    disable(ps);
    pptr = &proctab[currpid];
    if ( !pptr->phasmsg ) { /* if no message, wait for one */
        pptr->pstate = PRRECV;
        resched();
    }
    msg = pptr->pmsg; /* retrieve message */
    pptr->phasmsg = FALSE;
    restore(ps);
    return(msg);
}

```

```

/* recvclr.c - recvclr */

#include <conf.h>
#include <kernel.h>
#include <proc.h>

/*-----
 *  recvclr  --  clear messages, returning waiting message (if any)
 *-----
 */
SYSCALL recvclr()
{
    char    ps;
    int     msg;

    disable(ps);
    if ( proctab[currpid].phasmsg ) {          /* existing message? */
        proctab[currpid].phasmsg = FALSE;
        msg = proctab[currpid].pmsg;
    } else
        msg = OK;
    restore(ps);
    return(msg);
}

```

```

/* mem.h - freestk, roundew, truncew */

/*-----
 * roundew, truncew - round or truncate address to next even word
 *-----
 */
#define roundew(x)      (int *) ( (3 + (int)(x)) & (~3) )
#define truncew(x)     (int *) ( ((int)(x)) & (~3) )

/*-----
 * freestk -- free stack memory allocated by getstk
 *-----
 */
#define freestk(p,len)  freemem((unsigned)(p)                \
                               - (unsigned)(roundew(len))    \
                               + (unsigned)sizeof(int),      \
                               roundew(len) )

struct mblock {
    struct mblock *mnext;
    unsigned mlen;
};

extern struct mblock memlist; /* head of free memory list */
extern int *maxaddr; /* max memory address */
extern int end;

```

```

/* getmem.c - getmem */

#include <conf.h>
#include <kernel.h>
#include <mem.h>

/*-----
 * getmem -- allocate heap storage, returning lowest integer address
 *-----
 */
SYSCALL *getmem(nbytes)
    unsigned nbytes;
{
    char    ps;
    struct  mblock *p, *q, *leftover;

    disable(ps);
    if (nbytes==0 || memlist.mnext==NULL) {
        restore(ps);
        return( (int *)SYSERR);
    }
}

```

```

nbytes = (unsigned) roundew(nbytes);
for (q= &memlist,p=memlist.mnext ; p!=NULL ; q=p,p=p->mnext)
    if ( p->mten == nbytes) {
        q->mnext = p->mnext;
        restore(ps);
        return( (int *)p );
    } else if ( p->mten > nbytes ) {
        leftover = (struct mblock *)(( unsigned)p + nbytes );
        q->mnext = leftover;
        leftover->mnext = p->mnext;
        leftover->mten = p->mten - nbytes;
        restore(ps);
        return( (int *)p );
    }
restore(ps);
return( (int *)SYSERR );
}

```

```

/* getstk.c - getstk */

#include <conf.h>
#include <kernel.h>
#include <mem.h>

/*-----
 * getstk -- allocate stack memory, returning address of topmost int
 *-----
 */
SYSCALL *getstk(nbytes)
    unsigned int nbytes;
{
    char    ps;
    struct mblock *p, *q; /* q follows p along memlis */
    struct mblock *fits, *fitsq;
    unsigned len;

    disable(ps);
    if (nbytes == 0) {
        restore(ps);
        return( (int *)SYSERR );
    }
}

```

```

nbytes = (unsigned)roundew(nbytes);
fits = NULL;
q = &memlist;
for (p = q->mnext ; p != NULL ; q = p,p = p->mnext)
    if ( p->mmlen >= nbytes) {
        fitsq = q;
        fits = p;
    }
if (fits == NULL) {
    restore(ps);
    return( (int *)SYSERR );
}
if (nbytes == (len = fits->mmlen) ) {
    fitsq->mnext = fits->mnext;
} else {
    fits->mmlen -= nbytes;
}
fits = ((unsigned)fits) + len - sizeof(int);
*((unsigned *) fits) = nbytes; /* put size at base */
restore(ps);
return( (int *)fits );
}

```

```

/* freemem.c - freemem */

#include <conf.h>
#include <kernel.h>
#include <mem.h>

/*-----
 * freemem -- free a memory block, returning it to memlist
 *-----
 */
SYSCALL freemem(block, size)
    struct mblock *block;
    unsigned size;
{
    char ps;
    struct mblock *p, *q;
    unsigned top;

    if (size==0 || (unsigned)block>(unsigned)maxaddr
        || ((unsigned)block)<((unsigned)&end))
        return(SYSERR);
    size = (unsigned)roundew(size);
    disable(ps);
    for( p=memlist.mnext,q= &memlist ; (char *)p!=NULL && p<block ;
        q=p,p=p->mnext )
        ;
}

```

```

if ((top=q->mlem+(unsigned)q)>(unsigned)block && q!= &memlist ||
    (char *)p!=NULL && (size+(unsigned)block) > (unsigned)p) {
    restore(ps);
    return(SYSERR);
}
if ( q!= &memlist && top == (unsigned)block )
    q->mlem += size;
else {
    block->mlem = size;
    block->mnext = p;
    q->mnext = block;
    q = block;
}
if ( (unsigned)( q->mlem + (unsigned)q ) == (unsigned)p) {
    q->mlem += p->mlem;
    q->mnext = p->mnext;
}
restore(ps);
return(OK);
}

```

```

/* io.h - fgetc, fputc, getchar, isbaddev, putchar */
#define INTVECI  inint          /* input interrupt dispatch routine */
#define INTVECO  outint        /* output interrupt dispatch routine */
extern  int      INTVECI();
extern  int      INTVECO();

struct  intmap  {              /* device-to-interrupt routine mapping */
    int      (*iin)();         /* address of input interrupt routine */
    int      icode;           /* argument passed to input routine */
    int      (*iout)();        /* address of output interrupt routine */
    int      ocode;           /* argument passed to output routine */
};

#ifdef  NDEVS
extern  struct  intmap  intmap[NDEVS];
#define  isbaddev(f)      ( (f)<0 || (f)>=NDEVS )
#endif

#define  BADDEV          -1
/* In-line I/O procedures */
#define  getchar()       getc(CONSOLE)
#define  putchar(ch)     putc(CONSOLE,(ch))
#define  fgetc(unit)     getc((unit))
#define  fputc(unit,ch)  putc((unit),(ch))

struct  vector  {
    char      *vproc;         /* address of interrupt procedure */
    int      vps;            /* saved process status word */
};

```

```

/* ioint.s - inint, outint */
/* I/O interrupts trap here.  Original PC and PS are on top of the      */
/* stack upon entry.  Low order 4 bits of the current PS contain the    */
/* device descriptor.  Interrupts are disabled.                          */

        .globl  _inint,_outint,_intmap
_outint:
        mfps    -(sp)           / Save device descriptor from PS
        mov     r0,-(sp)        / Save r0 (csv does not)
        mov     $_intmap+4,r0   / point r0 to output in intmap
        br     ioint           / Go do common part of code
_inint:
        mfps    -(sp)           / Save device code from PS
        mov     r0,-(sp)        / Save r0 (csv does not)
        mov     $_intmap,r0     / point r0 to input in intmap
ioint:
        mov     r1,-(sp)        / Save r1 (csv does not)
        mov     4(sp),r1        / Get saved PS in r1
        bic     $177760,r1      / Mask off device descriptor
        ash     $3,r1           / pick correct entry in intmap
        add     r1,r0           / Form pointer to intmap entry
        mov     2(r0),-(sp)     / Push "code" from intmap as arg
        jsr    pc,*(r0)        / Call interrupt routine
        mov     2(sp),r1        / Restore r1 and R0 from stack
        mov     4(sp),r0
        add     $8,sp           / Pop arg, saved r0, r1, and PS
        rtt

```

```

/* insertd.c - insertd */
#include <conf.h>
#include <kernel.h>
#include <q.h>
/*-----
 * insertd -- insert process pid in delta list "head", given its key
 *-----
 */
insertd(pid, head, key)
    int    pid;
    int    head;
    int    key;
{
    int    next;          /* runs through list          */
    int    prev;         /* follows next through list */
    for(prev=head,next=q[head].qnext ;
        q[next].qkey < key ; prev=next,next=q[next].qnext)
        key -= q[next].qkey;
    q[pid].qnext = next;
    q[pid].qprev = prev;
    q[pid].qkey  = key;
    q[prev].qnext = pid;
    q[next].qprev = pid;
    if (next < NPROC)
        q[next].qkey -= key;
    return(OK);
}

```

```

/* sleep10.c - sleep10 */
#include <conf.h>
#include <kernel.h>
#include <proc.h>
#include <q.h>
#include <sleep.h>
/*-----
 * sleep10 -- delay the caller for a time specified in tenths of seconds
 *-----*/
SYSCALL sleep10(n)
    int n;
{
    char ps;

    if (n < 0 || clkruns==0) return(SYSERR);
    if (n == 0) {
        resched();
        return (OK);    /* sleep10(0) -> end time slice */
    }
    disable(ps);
    insertd(currpid,clockq,n);
    slnempty = TRUE;
    sltop = (int *) & q[q[clockq].qnext].qkey;
    proctab[currpid].pstate = PRSLEEP;
    resched();
    restore(ps);
    return(OK);
}

```

```

/* sleep.h */

#define CVECTOR 0100          /* location of clock interrupt vector */

extern  int    clkruns;      /* 1 iff clock exists; 0 otherwise */
                                /* Set at system startup. */
extern  int    clockq;      /* q index of sleeping process list */
extern  int    count6;      /* used to ignore 5 of 6 interrupts */
extern  int    *sltop;      /* address of first key on clockq */
extern  int    slnempty;    /* 1 iff clockq is nonempty */

extern  int    defclk;      /* >0 iff clock interrupts are deferred */
extern  int    clkdiff;     /* number of clock ticks deferred */
extern  int    clkint();    /* clock interrupt handler */

```

```

/* setclkr.s - setclkr */
CVECTPC =      100          / clock interrupt vector address
CVECTPS =      102          / " " " "
DISABLE =      340          / PS that disables interrupts
ENABLE =       000          / PS that enables interrupts
COUNT =      32700.        / Times to loop (in decimal)
/*-----
/* setclkr -- set cklruns to 1 iff real-time clock exists, 0 otherwise
/*-----
        .globl  _setclkr
_setclkr:
        mov     r1,-(sp)      / save register used
        clr     _cklrns      / initialize for no clock
        mov     *$CVECTPS,-(sp) / save clock interrupt vector
        mov     *$CVECTPC,-(sp) / on caller's stack
        mov     $DISABLE,*$CVECTPS / set up new interrupt vector
        mov     $setint,*$CVECTPC
        mov     $COUNT,r1   / initialize counter for loop
        reset   / clear other interrupts, if any
        mtps    $ENABLE      / allow interrupts
setloop:  dec     r1          / loop COUNT times waiting for
        bpl    setloop      / a clock interrupt
        mtps    $DISABLE     / no interrupt occurred, so quit
        br     setdone
setint:   inc     _cklrns    / clock interrupt jumps here
        add    $4,sp        / pop pc/ps pushed by interrupt
setdone:  mov     (sp)+,*$CVECTPC / restore old interrupt vector
        mov    (sp)+,*$CVECTPS
        mov    (sp)+,r1     / restore register
        rts    pc          / return to caller

```

```

/* sleep.c - sleep */
#include <conf.h>
#include <kernel.h>
#include <proc.h>
#include <q.h>
#include <sleep.h>
/*-----
 * sleep  --  delay the calling process n seconds
 *-----
 */
SYSCALL sleep(n)
    int    n;
{
    char    ps;

    if (n<0 || clkruns==0)  return(SYSERR);
    if (n == 0) {
        resched();
        return(OK);
    }
    while (n >= 1000) {
        sleep10(10000);
        n -= 1000;
    }
    if (n > 0)
        sleep10(10*n);
    return(OK);
}

```

```

/* wakeup.c - wakeup */

#include <conf.h>
#include <kernel.h>
#include <proc.h>
#include <q.h>
#include <sleep.h>

/*-----
 * wakeup  --  called by clock interrupt dispatcher to awaken processes
 *-----
 */
INTPROC wakeup()
{
    while (nonempty(clockq) && firstkey(clockq) <= 0)
        ready(getfirst(clockq), RESCHNO);
    if ( slnempty = nonempty(clockq) )
        sltop = (int *) & q[q[clockq].qnext].qkey;
    resched();
}

```

```
/* ssclock.c - stopclk, strtclk */
```

```
#include <conf.h>  
#include <kernel.h>  
#include <proc.h>  
#include <q.h>  
#include <sleep.h>
```

```
/*-----  
 * stopclk -- put the clock in defer mode  
 *-----  
 */  
stopclk()  
{  
    defclk++;  
}
```

```

/*-----
 * strtclk  --  take the clock out of defer mode
 *-----*/
strtclk()
{
    char ps;
    int makeup;
    int next;
    disable(ps);
    if ( defclk<=0 || --defclk>0 ) {
        restore(ps);
        return;
    }
    makeup = clkdiff;
    preempt -= makeup;
    clkdiff = 0;
    if ( slnempty ) {
        for (next=firstid(clockq) ;
            next < NPROC && q[next].qkey < makeup ;
            next=q[next].qnext) {
            makeup -= q[next].qkey;
            q[next].qkey = 0;
        }
        if (next < NPROC)
            q[next].qkey -= makeup;
        wakeup();
    }
    if ( preempt <= 0 )
        resched();
    restore(ps);
}

```

```

/* clkint.s - clkint */

/*-----
/* clkint -- real-time clock interrupt service routine
/*-----
        .globl  _clkint
_clkint:
        dec     _count6           / Is this the 6th interrupt?
        bpl     clret             / no => return
        mov     $6,_count6       / yes=> reset counter&continue
        tst     _defclk          / Are clock ticks deferred?
        beq     notdef           / no => go process this tick
        inc     _clkdiff         / yes=> count in clkdiff and
        rtt                          / return quickly

```

```

notdef:
    tst     _slnempty           / Is sleep queue nonempty?
    beq     clpreem            / no => go process preemption
    dec     *_sltop            / yes=> decrement delta key
    bpl     clpreem            /      on first process,
    mov     r0,-(sp)           /      calling wakeup if
    mov     r1,-(sp)           /      it reaches zero
    jsr     pc,_wakeup         /      (interrupt routine
    mov     (sp)+,r1           /      saves & restores r0
    mov     (sp)+,r0           /      and r1; C doesn't)

clpreem:
    dec     _preempt           / Decrement preemption counter
    bpl     clret              / and call resched if it
    mov     r0,-(sp)           / reaches zero
    mov     r1,-(sp)           /      (As before, interrupt
    jsr     pc,_resched        /      routine must save &
    mov     (sp)+,r1           /      restore r0 and r1
    mov     (sp)+,r0           /      because C doesn't)

clret:
    rtt                        / Return from interrupt

```

```

/* conf.h (GENERATED FILE; DO NOT EDIT) */

#define NULLPTR (char *)0

/* Device table declarations */
struct devsw {
    int     dvnum;
    int     (*dvinit)();
    int     (*dvopen)();
    int     (*dvclose)();
    int     (*dvread)();
    int     (*dvwrite)();
    int     (*dvseek)();
    int     (*dvgetc)();
    int     (*dvputc)();
    int     (*dvcntl)();
    int     dvcsr;
    int     dvivec;
    int     dvovec;
    int     (*dviint)();
    int     (*dvoint)();
    char    *dvioblk;
    int     dvminor;
};

extern struct devsw devtab[];
/* one entry per device */

```

```
/* Device name definitions */
```

```
#define CONSOLE      0          /* type tty      */
#define OTHER        1          /* type tty      */
#define RING0IN      2          /* type dlc      */
#define RING0OUT     3          /* type dlc      */
#define DISK0        4          /* type dsk      */
#define FILE1        5          /* type df       */
#define FILE2        6          /* type df       */
#define FILE3        7          /* type df       */
#define FILE4        8          /* type df       */
```

```
/* Control block sizes */
```

```
#define Ntty         2
#define Ndlc         2
#define Ndsk         1
#define Ndf          4

#define NDEVS        9
```

```
/* Declarations of I/O routines referenced */

extern int    ttyinit();
extern int    ttyopen();
extern int    ionull();
extern int    ttyread();
extern int    ttywrite();
extern int    ioerr();
extern int    ttycntl();
extern int    ttygetc();
extern int    ttyputc();
extern int    ttyiin();
extern int    ttyoin();
extern int    dlcinit();
extern int    dlcread();
extern int    dlcwrite();
extern int    dlccntl();
extern int    dlcputc();
extern int    dlciin();
extern int    dlcoin();
extern int    dsinit();
extern int    dsopen();
extern int    dsread();
extern int    dswrite();
extern int    dsseek();
extern int    dscntl();
extern int    dsinter();
```

```
extern int    lfinit();
extern int    lfclose();
extern int    lfred();
extern int    lfwrite();
extern int    lfseek();
extern int    lfgetc();
extern int    lfputc();
```

```
/* Configuration and Size Constants */
```

```
#define MEMMARK          /* define if memory marking used*/
#define NNETS           1  /* number of XINU ring networks */
                          /* (remove if there are 0)      */
#define NPROC           10 /* number of user processes      */
#define NSEM            50 /* number of semaphores          */
#define VERSION         "6.09b (3/1/83)" /* label printed at startup      */
```

```

/* read.c - read */

#include <conf.h>
#include <kernel.h>
#include <io.h>

/*-----
 * read - read one or more bytes from a device
 *-----
 */
SYSCALL read(descrp, buff, count)
int descrp, count;
char *buff;
{
    struct devsw *devptr;

    if (isbaddev(descrp) )
        return(SYSERR);
    devptr = &devtab[descrp];
    return( (*devptr->dvread)(devptr,buff,count) );
}

```

```

/* control.c - control */

#include <conf.h>
#include <kernel.h>
#include <io.h>

/*-----
 * control - control a device (e.g., set the mode)
 *-----
 */
SYSCALL control(descrp, func, addr, addr2)
int descrp, func;
char *addr,*addr2;
{
    struct devsw *devptr;

    if (isbaddev(descrp) )
        return(SYSERR);
    devptr = &devtab[descrp];
    return( (*devptr->dvcntl)(devptr, func, addr, addr2) );
}

```

```

/* getc.c - getc */

#include <conf.h>
#include <kernel.h>
#include <io.h>

/*-----
 * getc - get one character from a device
 *-----
 */
SYSCALL getc(descrp)
int descrp;
{
    struct devsw *devptr;

    if (isbaddev(descrp) )
        return(SYSERR);
    devptr = &devtab[descrp];
    return( (*devptr->dvgetc)(devptr) );
}

```

```

/* init.c - init */

#include <conf.h>
#include <kernel.h>
#include <io.h>

/*-----
 *  init  -  initialize a device
 *-----
 */
init(descrp)
int descrp;
{
    struct devsw  *devptr;

    if (isbaddev(descrp) )
        return(SYSERR);
    devptr = &devtab[descrp];
    return( (*devptr->dvinit)(devptr) );
}

```

```

/* putc.c - putc */

#include <conf.h>
#include <kernel.h>
#include <io.h>

/*-----
 *   putc   -   write a single character to a device
 *-----
 */
SYSCALL putc(descrp, ch)
int descrp;
char ch;
{
    struct devsw   *devptr;

    if (isbaddev (descrp) )
        return(SYSERR);
    devptr = &devtab[descrp];
    return( (*devptr->dvputc)(devptr,ch) );
}

```

```

/* seek.c seek */

#include <conf.h>
#include <kernel.h>
#include <io.h>

/*-----
 * seek -- position a device (very common special case of control)
 *-----
 */
SYSCALL seek(descrp, pos)
int descrp;
long pos;
{
    struct devsw *devptr;

    if (isbaddev(descrp) )
        return(SYSERR);
    devptr = &devtab[descrp];
    return( (*devptr->dvseek)(devptr, pos) );
}

```

```

/* write.c - write */

#include <conf.h>
#include <kernel.h>
#include <io.h>

/*-----
 * write - write 1 or more bytes to a device
 *-----
 */
SYSCALL write(descrp, buff, count)
    int descrp, count;
    char *buff;
{
    struct devsw *devptr;

    if (isbaddev(descrp) )
        return(SYSEERR);
    devptr = &devtab[descrp];
    return( (*devptr->dvwrite)(devptr,buff,count) );
}

```

```

/* close.c - close */

#include <conf.h>
#include <kernel.h>
#include <io.h>

/*-----
 * close - close a device
 *-----
 */
SYSCALL close(descrp)
int descrp;
{
    struct devsw *devptr;

    if (isbaddev(descrp) )
        return(SYSERR);
    devptr = &devtab[descrp];
    return( (*devptr->dvclose)(devptr) );
}

```

```

/* open.c - open */

#include <conf.h>
#include <kernel.h>
#include <io.h>

/*-----
 * open - open a connection to a device/file (parms 2 &3 are optional)
 *-----
 */
SYSCALL open(descrp, nam, mode)
int    descrp;
char   *nam;
char   *mode;
{
    struct devsw *devptr;

    if ( isbaddev(descrp) )
        return(SYSERR);
    devptr = &devtab[descrp];
    return( (*devptr->dlopen)(devptr, nam, mode) );
}

```

```
/* ioerr.c - ioerr */

#include <conf.h>
#include <kernel.h>

/*-----
 * ioerr - return an error (used for "error" entries in devtab)
 *-----
 */
ioerr()
{
    return(SYSERR);
}
```

```
/* ionull.c - ionull */

#include <conf.h>
#include <kernel.h>

/*-----
 * ionull - do nothing (used for "don't care" entries in devtab)
 *-----
 */
ionull()
{
    return(OK);
}
```

```

/* conf.c (GENERATED FILE; DO NOT EDIT) */

#include <conf.h>

/* device independent I/O switch */

struct devsw  devtab[NDEVS] = {

/* Format of entries is:
device-number,
init, open, close,
read, write, seek,
getc, putc, cntl,
device-csr-address, input-vector, output-vector,
iint-handler, oint-handler, control-block, minor-device,
*/

/* CONSOLE is tty */

0,
ttyinit, ttyopen, ionull,
ttyread, ttywrite, ioerr,
ttygetc, ttyputc, ttycntl,
0177560, 0060, 0064,
ttyiin, ttyoin, NULLPTR, 0,

```

```

/* OTHER is tty */
1,
ttyinit, ttyopen, ionull,
ttyread, ttywrite, ioerr,
ttygetc, ttyputc, ttycntl,
0176500, 0300, 0304,
ttyiin, ttyoin, NULLPTR, 1,

/* RING0IN is dlc */
2,
dlcinit, ioerr, ioerr,
dlcread, dlctime, ioerr,
ioerr, dlctime, dlccntl,
0176510, 0310, 0314,
dlciin, dlcoin, NULLPTR, 0,

/* RING0OUT is dlc */
3,
dlcinit, ioerr, ioerr,
dlcread, dlctime, ioerr,
ioerr, dlctime, dlccntl,
0176520, 0320, 0324,
dlciin, dlcoin, NULLPTR, 1,

/* DISK0 is ds */
4,
dsinit, dsopen, ioerr,
dsread, dswrite, dsseek,
ioerr, ioerr, dscntl,
0177460, 0134, 0134,
dsinter, dsinter, NULLPTR, 0,

```

```

/* FILE1 is lf */
5,
lfinit, ioerr, lfclose,
lfread, lfwrite, lfseek,
lfgetc, lfputc, ioerr,
0000000, 0000, 0000,
ioerr, ioerr, NULLPTR, 0,

/* FILE2 is lf */
6,
lfinit, ioerr, lfclose,
lfread, lfwrite, lfseek,
lfgetc, lfputc, ioerr,
0000000, 0000, 0000,
ioerr, ioerr, NULLPTR, 1,
/* FILE3 is lf */
7,
lfinit, ioerr, lfclose,
lfread, lfwrite, lfseek,
lfgetc, lfputc, ioerr,
0000000, 0000, 0000,
ioerr, ioerr, NULLPTR, 2,

/* FILE4 is lf */
8,
lfinit, ioerr, lfclose,
lfread, lfwrite, lfseek,
lfgetc, lfputc, ioerr,
0000000, 0000, 0000,
ioerr, ioerr, NULLPTR, 3,
};

```

```

/* ioinit.c - ioinit, iosetattr */

#include <conf.h>
#include <kernel.h>
#include <io.h>

/*-----
 * ioinit -- standard interrupt vector and dispatch initialization
 *-----
 */
ioinit(descrp)
int      descrp;
{
    int      minor;

    if (isbaddev(descrp) )
        return(SYSERR);
    minor = devtab[descrp].dvminor;
    iosetattr(descrp, minor, minor);
    return(OK);
}

```

```

/*-----
 *  iosetvec  -  fill in interrupt vectors and dispatch table entries
 *-----*/
iosetvec(descrp, incode, outcode)
int      descrp;
int      incode;
int      outcode;
{
    struct devsw      *devptr;
    struct intmap     *map;
    struct vector     *vptr;

    if (isbaddev(descrp))
        return(SYSERR);
    devptr = &devtab[descrp];
    map = &intmap[devptr->dvnum]; /* fill in interrupt dispatch */
    map->iin = devptr->dviint; /* map with addresses of high-*/
    map->icode = incode; /* level input and output */
    map->iout = devptr->dvooint; /* interrupt handlers and */
    map->ocode = outcode; /* minor device numbers */
    vptr = (struct vector *)devptr->dvivec;
    vptr->vproc = (char *)INTVECI; /* fill in input interrupt */
    vptr->vps = descrp | DISABLE; /* vector PC and PS values */
    vptr = (struct vector *)devptr->dvovec;
    vptr->vproc = (char *)INTVECO; /* fill in output interrupt */
    vptr->vps = descrp | DISABLE; /* vector PC and PS values */
    return(OK);
}

```

```

/* tty.h */
#define IOCHERR          0200          /* bit set on when an error      */
#define OBMINSPP        20            /* min space in buffer before    */
/* processes awakened to write */
#define EBUFLLEN        20            /* size of echo queue           */

/* size constants */

#ifndef Ntty
#define Ntty             1            /* number of serial tty lines   */
#endif
#ifndef IBUFLLEN
#define IBUFLLEN         128          /* num. chars in input queue     */
#endif
#ifndef OBUFLLEN
#define OBUFLLEN         64          /* num. chars in output queue   */
#endif

/* mode constants */

#define IMRAW            'R'          /* raw mode => nothing done     */
#define IMCOOKED        'C'          /* cooked mode => line editing  */
#define IMCBREAK        'K'          /* honor echo, etc, no line edit*/
#define OMRAW           'R'          /* raw mode => normal processing*/

```

```

struct tty {
    int ihead; /* head of input queue */
    int itail; /* tail of input queue */
    char ibuff[IBUFLEN]; /* input buffer for this line */
    int isem; /* input semaphore */
    int ohead; /* head of output queue */
    int otail; /* tail of output queue */
    char obuff[OBUFLEN]; /* output buffer for this line */
    int osem; /* output semaphore */
    int odsend; /* sends delayed for space */
    int ehead; /* head of echo queue */
    int etail; /* tail of echo queue */
    char ebuff[EBUFLEN]; /* echo queue */
    char imode; /* IMRAW, IMCBREAK, IMCOOKED */
    Bool iecho; /* is input echoed? */
    Bool ieback; /* do erasing backspace on echo? */
    Bool evis; /* echo control chars as ^X ? */
    Bool ecrlf; /* echo CR-LF for newline? */
    Bool icrlf; /* map '\r' to '\n' on input? */
    Bool ierase; /* honor erase character? */
    char ierasec; /* erase character (backspace) */
    Bool ikill; /* honor line kill character? */
    char ikillc; /* line kill character */
    int icursor; /* current cursor position */
    Bool oflow; /* honor ostop/ostart? */
    Bool oheld; /* output currently being held? */
    char ostop; /* character that stops output */
    char ostart; /* character that starts output */
    Bool ocrlf; /* echo CR/LF for LF ? */
    char ifullc; /* char to send when input full */
    struct csr *ioaddr; /* device address of this unit */
};
extern struct tty tty[];

```

```

#define BACKSP '\b' /* backspace one character pos. */
#define BELL '\007' /* usually an audible tone */
#define ATSIGN '@'
#define BLANK ' ' /* used to print a "space" */
#define KILLCH '\025' /* line kill character (^U) */
#define NEWLINE '\n' /* line feed */
#define RETURN '\r' /* carriage return */
#define STOPCH '\023' /* control-S stops output */
#define STRTCH '\021' /* control-Q restarts output */
#define UPARROW '^' /* usually for visuals like ^X */

```

```

/* ttycontrol function codes */

```

```

#define TCSETBRK      1 /* turn on BREAK in transmitter */
#define TCRSTBRK     2 /* turn off BREAK " " */
#define TCNEXTC      3 /* look ahead 1 character */
#define TCMODER      4 /* set input mode to raw */
#define TCMODEC      5 /* set input mode to cooked */
#define TCMODEK      6 /* set input mode to cbreak */
#define TCICHARS     8 /* return number of input chars */
#define TCECHO       9 /* turn on echo */
#define TCNOECHO    10 /* turn off echo */
#define TFULLC      BELL /* char to echo when buffer full */

```

```

/* ttygetc.c - ttygetc */

#include <conf.h>
#include <kernel.h>
#include <tty.h>
#include <io.h>

/*-----
 * ttygetc - read one character from a tty device
 *-----
 */
ttygetc(devptr)
struct devsw *devptr;
{
    char ps;
    int ch;
    struct tty *iptr;

    disable(ps);
    iptr = &tty[devptr->dvminor];
    wait(iptr->isem); /* wait for a character in buff */
    ch = LOWBYTE & iptr->ibuff[iptr->itail++];
    if (iptr->itail >= IBUFLEN)
        iptr->itail = 0;
    restore(ps);
    return(ch);
}

```

```

/* ttyread.c - ttyread, readcopy */

#include <conf.h>
#include <kernel.h>
#include <tty.h>
#include <io.h>

/*-----
 * ttyread - read one or more characters from a tty device
 *-----
 */
ttyread(devptr, buff, count)
struct devsw *devptr;
int count;
char *buff;
{
    char ps;
    register struct tty *iptr;
    int avail, nread;

    if (count < 0)
        return(SYSERR);

```

```

disable(ps);
avail=scount((iptr= &tty[devptr->dvminor])->isem);

if ((count = (count==0 ? avail : count)) == 0){
    restore(ps);
    return(0);
}
nread = count;
if (count <= avail) {
    readcopy(buff, iptr, count);
} else {
    if (avail > 0) {
        readcopy(buff, iptr, avail);
        buf    += avail;
        count -= avail;
    }
    for ( ; count > 0 ; count--)
        *buf++ = ttygetc(devptr);
}

restore(ps);
return(nread);
}

```

```

/*-----
 *  readcopy - high speed copy procedure used by ttyread
 *-----
 */
LOCAL readcopy(buff, iptr, count)
register char *buff;
struct tty *iptr;
int count;
{
    register char *qtail, *qend, *uend;    /* copy loop variables */

    qtail = &iptr->ibuff[iptr->itail];
    qend = &iptr->ibuff[IBUFLEN];
    uend = buff + count;
    while ( buff < uend ) {
        *buff++ = *qtail++;
        if ( qtail >= qend )
            qtail = iptr->ibuff;
    }
    iptr->itail = qtail-iptr->ibuff;
    sreset(iptr->isem, scount(iptr->isem)-count);
}

```

```

/* ttyputc.c - ttyputc */
#include <conf.h>
#include <kernel.h>
#include <tty.h>
#include <io.h>
#include <slu.h>
/*-----
 * ttyputc - write one character to a tty device
 *-----*/
ttyputc(devptr, ch )
struct devsw *devptr;
char ch;
{
    struct tty *iptr;
    char ps;

    iptr = &tty[devptr->dvminor];
    if ( ch==NEWLINE && iptr->ocrlf )
        ttyputc(devptr,RETURN);
    wait(iptr->osem); /* wait for space in queue */
    disable(ps);
    iptr->obuf[iptr->ohead++] = ch;
    if (iptr->ohead >= OBUFLLEN)
        iptr->ohead = 0;
    (iptr->ioaddr)->ctstat = SLUENABLE;
    restore(ps);
    return(OK);
}

```

```

/* slu.h */

/* standard serial line unit device constants */

#define SLUENABLE          0100          /* device interrupt enable bit */
#define SLUREADY          0200          /* device ready bit */
#define SLUDISABLE        0000          /* device interrupt disable mask*/
#define SLUTBREAK         0001          /* transmitter break-mode bit */
#define SLUERMASK         0170000      /* mask for error flags on input*/
#define SLUCHMASK         0377         /* mask for input character */

/* SLU device register layout and correspondence to vendor's names */

struct csr
{
    int    crstat;          /* receiver control and status (RCSR) */
    int    crbuf;          /* receiver data buffer (RBUF) */
    int    ctstat;         /* transmitter control & status (XCSR) */
    int    ctbuf;          /* transmitter data buffer (XBUF) */
};

```

```

/* ttywrite.c - ttywrite, writcopy */
#include <conf.h>
#include <kernel.h>
#include <tty.h>
#include <io.h>
#include <slu.h>
/*-----
 * ttywrite - write one or more characters to a tty device
 *-----*/
ttywrite(devptr, buff, count)
struct devsw *devptr;
char *buff;
int count;
{
    register struct tty *ttyp;
    int avail;
    char ps;
    if (count < 0) return(SYSERR);
    if (count == 0) return(OK);
    disable(ps);
    ttyp = &tty[devptr->dvminor];
    if ( (avail=scount(ttyp->osem)) >= count) {
        writcopy(buff, ttyp, count);
        (ttyp->ioaddr)->ctstat = SLUENABLE;
    }else{
        if (avail > 0) {
            writcopy(buff, ttyp, avail);
            buff += avail;
            count -= avail;
        }
        for ( ; count>0 ; count--)
            ttyputc(devptr, *buff++);
    }
    restore(ps);
    return(OK);
}

```

```

/*-----
 * writcopy - high-speed copy from user's buffer into system buffer
 *-----
 */
LOCAL writcopy(buff, ttyp, count)
register char *buff;
struct tty *ttyp;
int count;
{
    register char *qhead, *qend, *uend;

    qhead = &ttyp->obuf[ttyp->ohead];
    qend = &ttyp->obuf[OBUFLLEN];
    uend = buff + count;
    while ( buff < uend ) {
        *qhead++ = *buff++;
        if ( qhead >= qend )
            qhead = ttyp->obuf;
    }
    ttyp->ohead = qhead - ttyp->obuf;
    sreset(ttyp->osem, scount(ttyp->osem)-count);
}

```

```

/* ttyoin.c - ttyoin */

#include <conf.h>
#include <kernel.h>
#include <tty.h>
#include <io.h>
#include <slu.h>

/*-----
 * ttyoin -- lower-half tty device driver for output interrupts
 *-----
 */
INTPROC ttyoin(iptr)
    register struct tty    *iptr;
{
    register struct csr    *cptr;
    int    ct;

    cptr = iptr->ioaddr;
    if (iptr->ehhead != iptr->etail) {
        cptr->ctbuf = iptr->ebuff[iptr->etail++];
        if (iptr->etail >= EBUFLLEN)
            iptr->etail = 0;
        return;
    }
}

```

```

if (iptr->oheld) {
    cptr->ctstat = SLUDISABLE;
    return;
}
if ((ct=scount(iptr->osem)) < OBUFLEN) {
    cptr->ctbuf = iptr->obuff[iptr->otail++];
    if (iptr->otail >= OBUFLEN)
        iptr->otail = 0;
    if (ct > OBMINS)
        signal(iptr->osem);
    else if ( ++(iptr->odsend) == OBMINS) {
        iptr->odsend = 0;
        signaln(iptr->osem, OBMINS);
    }
} else
    cptr->ctstat = SLUDISABLE;
}

```

```

/* ttyiin.c ttyiin, erasel, eputc, echoch */

#include <conf.h>
#include <kernel.h>
#include <tty.h>
#include <io.h>
#include <slu.h>

/*-----
 * ttyiin -- lower-half tty device driver for input interrupts
 *-----
 */
INTPROC ttyiin(iptr)
    register struct tty      *iptr; /* pointer to tty block          */
{
    register struct csr *cptr;
    register int      ch;
    int      ct;

    cptr = iptr->ioaddr;
    if (iptr->imode == IMRAW) {
        if (scount(iptr->isem) >= IBUFLEN) {
            ch = cptr->crbuf;
            return; /* discard if no space */
        }
    }
}

```

```

if ( (ch=cptr->crbuf) & SLUERMASK) /* character error */
    iptr->ibuff[iptr->ihead++] = (ch & SLUCHMASK) | IOCHERR;
else /* normal read complete */
    iptr->ibuff[iptr->ihead++] = ch & SLUCHMASK);
if (iptr->ihead >= IBUFLEN) /* wrap buffer pointer */
    iptr->ihead = 0;
signal(iptr->isem);
} else { /* cbreak | cooked mode */
    cerr = ((ch=cptr->crbuf) & SLUERMASK) ? IOCHERR : 0;
    ch &= SLUCHMASK;
    if ( ch == RETURN && iptr->icrlf )
        ch = NEWLINE;
    }
    if (iptr->oflow) {
        if (ch == iptr->ostart) {
            iptr->oheld = FALSE;
            cptr->ctstat = SLUENABLE;
            return;
        }
        if (ch == iptr->ostop) {
            iptr->oheld = TRUE;
            return;
        }
    }
}
iptr->oheld = FALSE;

```

```

if (iptr->imode == IMCBREAK) { /* cbreak mode */
    if (scount(iptr->isem) >= IBUFLEN) {
        eputc(iptr->ifullc, iptr, cptr);
        return;
    }
    iptr->ibuff[iptr->ihead++] = ch;
    if (iptr->ihead >= IBUFLEN)
        iptr->ihead = 0;
    if (iptr->iecho)
        echoch(ch, iptr, cptr);
    if (scount(iptr->isem) < IBUFLEN)
        signal(iptr->isem);
} else { /* cooked mode */
    if (ch == iptr->ikillc && iptr->ikill) {
        iptr->ihead -= iptr->icursor;
        if (iptr->ihead < 0)
            iptr->ihead += IBUFLEN;
        iptr->icursor = 0;
        eputc(RETURN, iptr, cptr);
        eputc(NEWLINE, iptr, cptr);
        return;
    }
    if (ch == iptr->ierasec && iptr->ierase) {
        if (iptr->icursor > 0) {
            iptr->icursor--;
            erasel(iptr, cptr);
        }
        return;
    }
}

```

```

if (ch == NEWLINE || ch == RETURN) {
    if (iptr->iecho) {
        echoch(ch, iptr, cptr);
    }
    iptr->ibuff[iptr->ihead++] = ch | cerr;
    if (iptr->ihead >= IBUFLEN)
        iptr->ihead = 0;
    ct = iptr->icursor+1; /* +1 for \n or \r*/
    iptr->icursor = 0;
    signaln(iptr->isem, ct);
    return;
}
ct = scout(iptr->isem);
ct = ct < 0 ? 0 : ct;
if ((ct + iptr->icursor) >= IBUFLEN-1) {
    eputc(iptr->ifullc, iptr, cptr);
    return;
}
if (iptr->iecho)
    echoch(ch, iptr, cptr);
iptr->icursor++;
iptr->ibuff[iptr->ihead++] = ch | cerr;
if (iptr->ihead >= IBUFLEN)
    iptr->ihead = 0;
}
}
}

```

```

/*-----
 * erasel -- erase one character honoring erasing backspace
 *-----*/
LOCAL erasel(iptr,cptr)
    struct tty *iptr;
    struct csr *cptr;
{
    char ch;
    if (--(iptr->ihead) < 0)
        iptr->ihead += IBUFLEN;
    ch = iptr->ibuff[iptr->ihead];
    if (iptr->iecho) {
        if (ch < BLANK || ch == 0177) {
            if (iptr->evis) {
                eputc(BACKSP,iptr,cptr);
                if (iptr->ieback) {
                    eputc(BLANK,iptr,cptr);
                    eputc(BACKSP,iptr,cptr);
                }
            }
            eputc(BACKSP,iptr,cptr);
            if (iptr->ieback) {
                eputc(BLANK,iptr,cptr);
                eputc(BACKSP,iptr,cptr);
            }
        } else {
            eputc(BACKSP,iptr,cptr);
            if (iptr->ieback) {
                eputc(BLANK,iptr,cptr);
                eputc(BACKSP,iptr,cptr);
            }
        }
    } else cptr->ctstat = SLUENABLE;
}

```

```

/*-----
 * echoch  --  echo a character with visual and ocrLf options
 *-----
 */
LOCAL echoch(ch, iptr, cptr)
    char    ch;                /* character to echo                */
    struct  tty    *iptr;      /* pointer to I/O block for this devptr */
    struct  csr    *cptr;      /* csr address for this devptr        */
{
    if ((ch==NEWLINE || ch==RETURN) && iptr->ecrlf) {
        eputc(RETURN, iptr, cptr);
        eputc(NEWLINE, iptr, cptr);
    } else if ((ch<BLANK || ch==0177) && iptr->evis) {
        eputc(UPARROW, iptr, cptr);
        eputc(ch+0100, iptr, cptr);    /* make it printable */
    } else {
        eputc(ch, iptr, cptr);
    }
    cptr->ctstat = SLUENABLE;
}

```

```

/*-----
 *  eputc - put one character in the echo queue
 *-----
 */
LOCAL eputc(ch,iptr,cptr)
    char    ch;
    struct  tty    *iptr;
    struct  csr    *cptr;
{
    iptr->ebuff[iptr->ehead++] = ch;
    if (iptr->ehead >= EBUFLLEN)
        iptr->ehead = 0;
    cptr->ctstat = SLUENABLE;
}

```

```
/* ttyinit.c - ttyinit */
```

```
#include <conf.h>
#include <kernel.h>
#include <proc.h>
#include <tty.h>
#include <io.h>
#include <slu.h>
```

```
/*-----
 *  ttyinit - initialize buffers and modes for a tty line
 *-----
 */
```

```
ttyinit(devptr)
```

```
    struct devsw  *devptr;
```

```
{
    register struct tty *iptr;
    register struct csr *cptr;
    int      junk, isconsole;
```

```
/* set up interrupt vector and interrupt dispatch table */
```

```
iptr = &tty[devptr->dvminor];
iosetvec(devptr->dvnum, (int)iptr, (int)iptr);
```

```
devptr->dvioblk = (char *)iptr;          /* fill tty control blk */
isconsole = (devptr->dvnum == CONSOLE); /* make console cooked */
```

```

iptr->ioaddr = (struct csr *)devptr->dvcsr; /* copy in csr addr. */
iptr->ihead = iptr->itail = 0; /* empty input queue */
iptr->isem = screate(0); /* chars. read so far=0 */
iptr->osem = screate(OBUFLLEN); /* buffer available=all */
iptr->odsend = 0; /* sends delayed so far */
iptr->ohead = iptr->otail = 0; /* output queue empty */
iptr->ehead = iptr->etail = 0; /* echo queue empty */
iptr->imode = (isconsole ? IMCOOKED : IMRAW);
iptr->iecho = iptr->evis = isconsole; /* echo console input */
iptr->ierase = iptr->ieback = isconsole; /* console honors erase */
iptr->ierasec = BACKSP; /* using ^h */
iptr->ecrlf = iptr->icrlf = isconsole; /* map RETURN on input */
iptr->ocrlf = iptr->oflow = isconsole; /* map RETURN on output */
iptr->ikill = isconsole; /* set line kill == @ */
iptr->ikillc = ATSIGN;
iptr->oheld = FALSE;
iptr->ostart = STRTCH;
iptr->ostop = STOPCH;
iptr->icursor = 0;
iptr->ifullc = TFULLC;
cptr = (struct csr *)devptr->dvcsr;
junk = cptr->crbuf; /* clear receiver and */
cptr->crstat = SLUENABLE; /* enable in. interrupts */
cptr->ctstat = SLUDISABLE; /* disable out. " */

```

```

}

```

```

/* ttycntl.c - ttycntl */

#include <conf.h>
#include <kernel.h>
#include <tty.h>
#include <io.h>
#include <slu.h>

/*-----
 * ttycntl - control a tty device by setting modes
 *-----
 */
ttycntl(devptr, func, addr)
struct devsw *devptr;
int func;
char *addr;
{
    register struct tty *ttyp;
    char ch;
    char ps;

    ttyp = &tty[devptr->dvminor];
    switch ( func ) {
    case TCSETBRK:
        ttyp->ioaddr->ctstat |= SLUTBREAK;
        break;
    case TCRSTBRK:
        ttyp->ioaddr->ctstat &= ~SLUTBREAK;
        break;

```

```

case TCNEXTC:
    disable(ps);
    wait(ttyp->isem);
    ch = ttyp->ibuff[ttyp->itail];
    restore(ps);
    signal(ttyp->isem);
    return(ch);
case TCMODER:
    ttyp->imode = IMRAW;
    break;
case TCMODEC:
    ttyp->imode = IMCOOKED;
    break;
case TCMODEK:
    ttyp->imode = IMCBREAK;
    break;
case TCECHO:
    ttyp->iecho = TRUE;
    break;
case TCNOECHO:
    ttyp->iecho = FALSE;
    break;
case TCICCHARS:
    return(scount(ttyp->isem));
default:
    return(SYSERR);
}
return(OK);

```

```

}

```