SE 390: Series 300 HP-UX Internals

April 1993

Introduction

Introductions

Expectations

- This class must be practical.
- We are *very* interested in constructive criticism and suggestions. As much as possible, put your comments in writing on the evaluation that was page 0.
- This class contains sensitive information - please be *very* careful who you share it with.
- Please work in pairs, and work on the same machine all week long.
  - if you trash your disk, you need to fix it
  - we will be doing detailed work, which goes faster with two people
- Expect to work hard. This is not a class for the faint-of-heart or people that want to be spoon-fed.
- Realize that your instructor doesn't know everything - if he doesn't say, "I don't know" from time to time, get suspicious :-)
- HP-UX source will not be a part of the class.
- Feel free to ask questions, but please defer them until lab time if appropriate.
- "I hear and I forget. I see and I remember. I do and I understand."

Overview of the Class

Background of HP-UX

The "Big Picture" of the Kernel

SE 390: Series 300 HP-UX Internals

Introduction

HP-UX Origins and Compatibility

- Original UNIX(tm) came from Bell Labs in the late 1960s.

- Over time it was refined, and AT&T released version 7. It has been said that V7 was better than either its predecessors or its successors.

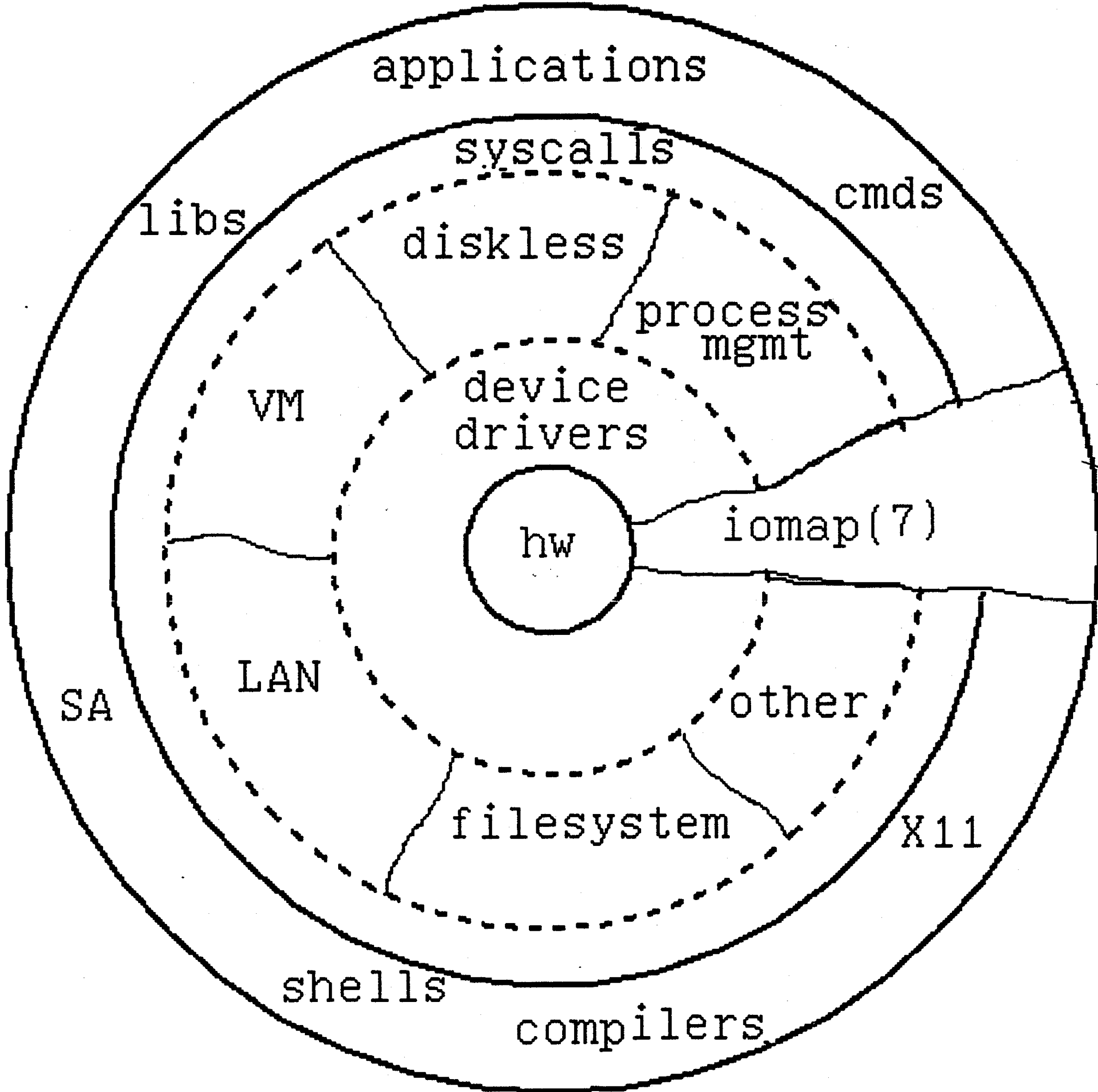
- AT&T released System III and System V, and System V has become a standard that many people accept.

- UC Berkeley took V7 or something similar and started going another way. They have since released 4.1-4.3, and BSD is a standard that another set of people accept.

- HP-UX on both the S300 and S800 is a port of BSD4.2, with a System V call interface on top of it. It passes the SVID (for V.2 as of May 1988), but has many of the smart things that Berkeley did (demand-paged VM, HFS filesystem, etc).

  In 8.0, there is a totally different VM system, based largely on System V.3.

HP-UX Structure Overview



SE 390: Series 300 HP-UX Internals

Introduction

The "Big Picture" of the Kernel

- What is it there for?
    - manage resources
    - make life easier for the programmer
- What are the major components?
    - kernel processes: swapper, pager, [init], [CSPs]...
    - device drivers
    - privileged library routines that deal with:
        - processes
        - memory
        - the file system
        - the I/O system
        - diskless nodes
    - Timeout routines - not really a process, but they act like it in the sense that they are responsible for monitoring free memory, CPU scheduling, etc. If they were part of a process, it would be process 0, but they operate independently of it.

      (These can be thought of as "internal at(1) jobs". Inside the kernel, one can call a routine named timeout() and tell it to call a particular function N clock ticks from now.)

SE 390: Series 300 HP-UX Internals

Introduction

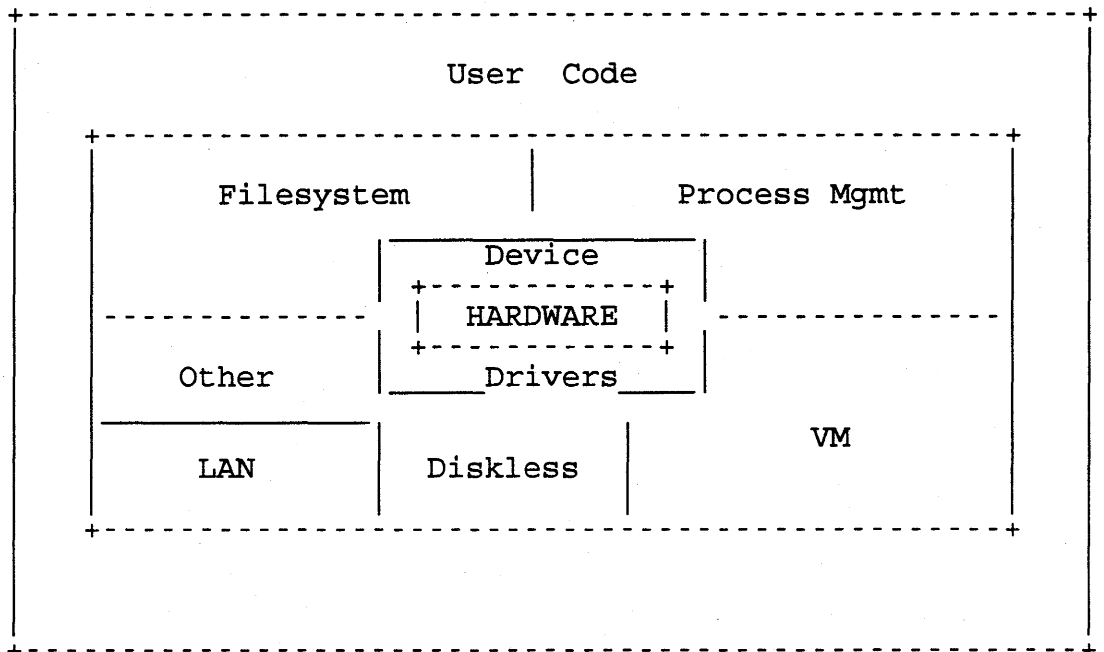
The Kernel in One Page :-)

- PROCESSES are running programs; they have their own private address space, they (hopefully) get to use the CPU from time to time, and the kernel keeps information about them in structures called the "u area" and "proc table entry".

- The I/O system is largely composed of device drivers, each of which specializes in a particular kind of device interface. There are also general principles of how interrupt-driven devices talk to the system and how we decide which driver should be called for a given task.

- The FILESYSTEM is responsible for organizing non-volatile data on the disk. HP-UX uses the Berkeley filesystem, which can be thought of as many small Bell filesystems stuck together on the disk. The filesystem also has provisions in-core to handle other kinds of filesystems, such as NFS or CDFS. It does this through an abstraction called a "vnode".

- MEMORY is managed by the kernel in such a way that each process gets some private address space, and the sum of the amounts of memory used by each process can be much greater than the amount of RAM in the machine.



SE 390: Series 300 HP-UX Internals

Introduction

Access to the Kernel

- System calls.
    - front ends in libc
        - 68K
            - move system call number into d0 (680x0 register)
            - change modes with "trap 0", which kernel catches
            - trap handler calls syscall()
        - PA
            - each process has something called a "gateway" page mapped into its address space
            - in this page there is a "gate" instruction, which "promotes" the privilege level of the process and calls the kernel routine syscall()
    - actual system call code is called indirectly, using the system call number as an index into sysent[]
- The assembly-level debugger, adb(1).
- The kernel debugger, SYSDEBUG (68K) or DDB (PA). This is most useful for people in the lab's kernel group or people writing drivers - not very useful without source (and DDB requires a 300 or 400 to run on - it is not a standalone debugger)
- Calls to nlist(3) & /dev/kmem
    - YOU ARE ON YOUR OWN
    - call nlist(3) to get address of symbol from "a.out" file (/hp-ux in this case)
    - open /dev/kmem and seek to address
    - read information
    - YOU ARE ON YOUR OWN - KERNEL DATA STRUCTURES CHANGE FROM RELEASE TO RELEASE!

```
/* @(#) $Revision: 1.8.62.17 $ */
#ifndef _MSYS_SPACE_INCLUDED /* allows multiple inclusion */
#define _MSYS_SPACE_INCLUDED

#include "../ufs/fsdir.h"

#include "../h/user.h"
#include "../h/proc.h"
#include "../h/sem_beta.h"

#include "../h/vnode.h"
#include "../ufs/inode.h"

#include "../cdfs/cdfsdir.h"
#include "../cdfs/cdnode.h"
#include "../cdfs/cdfs.h"
#ifdef SIXR
#include "../machine/sna_space.h" /*  for SNAP */
#endif

#include "../h/callout.h"
#include "../h/kernel.h"
#include "../h/map.h"
#include "../h/buf.h"
#include "../h/pty.h"
#include "../h/nvs.h"
#include "../machine/iobuf.h";
#include "../machine/dilio.h";
#include "../dux/rmswap.h"
#include "../dux/dm.h"
#include "../dux/protocol.h"
#include "../dux/nsp.h"

#include "../machine/lnatypes.h"
#include "../machine/intrpt.h"
#include "../machine/hpibio.h"
#include "../machine/drvhw.h"
#include "../h/devices.h"
#include "../h/dnlc.h"
#include "../h/file.h"

/*
 * System parameter formulae.
 */

struct  timezone tz = { TIMEZONE, DST };

short rootlink[3] = { 0xffff, 0xffff, 0xffff };
char *bootlink = 0;
int lanselectcode = -1;

int num_cnodes             = NUM_CNODES;

/*
** Size the using/serving arrays. USING_ARRAY_SIZE and SERVING_ARRAY_SIZE
** are configurable parameters.
```

```
*/
int using_array_size = USING_ARRAY_SIZE;
struct using_entry using_array[ USING_ARRAY_SIZE ];

int serving_array_size = (SERVING_ARRAY_SIZE > MAX_SERVING_ARRAY) ? MAX_SERV
struct serving_entry serving_array[ (SERVING_ARRAY_SIZE > MAX_SERVING_ARRAY)

int dskless_fsbufs = (DSKLESS_FSBUFS > MAX_SERVING_ARRAY) ? MAX_SERVING_ARRA

/*
** Define timeout periods for selftest and crash detection.  SELFTEST_PERIOD
** SEND_ALIVE_PERIOD and CHECK_ALIVE_PERIOD are configurable parameters.
*/

/* If selftest period is 0 then no selftest, otherwise lowerbound of 90 secs
int selftest_period = ((SELFTEST_PERIOD == 0) ? SELFTEST_PERIOD : ((SELFTEST

int check_alive_period = CHECK_ALIVE_PERIOD;
int retry_alive_period = RETRY_ALIVE_PERIOD;

int ngcsp = NGCSP;
int ncsp = NGCSP + 1;                       /* always one for limited CSP */
struct nsp nsp[NGCSP+1];                    /* always one for limited CSP */
struct nsp *nspNCSP = &nsp[NGCSP+1];


/* semaphore to prevent regular LAN init to reinitialize the network. */
/* USEFUL ??? */
int     DUX_init = 1;
/* dskless subsystem initialization flag */
int     dskless_initialized = 0;

#ifdef UIPC    /* UIPC is the umbrella subsystem for networking */

/*
 * Networking
 */

#include "../h/mbuf.h"
#define PRUREQUESTS
#include "../h/protosw.h"
#include "../h/socket.h"

#ifdef INET
#include "../net/if.h"
#include "../net/route.h"
#include "../net/raw_cb.h"
#include "../netinet/in.h"
#include "../netinet/if_ether.h"
#include "../h/mib.h"
#include "../netinet/mib_kern.h"
#include "../net/if_ni.h"
/* ni */
int ni_max = NNI;
struct ni_cb ni_cb[NNI];
```

```
/*
 * Internet Domain
 */
#define TCPSTATES
#include "../netinet/tcp_fsm.h"
struct ifqueue ipintrq;

/*
 * (X)NS Domain
 */
struct ifqueue nsintrq;

#endif /* INET */
#endif /* UIPC */

/*
 * Netisr
 */
int netisr_priority = NETISR_PRIORITY;
int     netmemmax       = NETMEMMAX;


#ifdef NSDIAG
#include        "../sio/nsdiag0.h"
#define NSDIAG_MAX_QUEUE        500
int nsdiag0_high_water = NSDIAG_MAX_QUEUE;
nsdiag_event_msg_type *nsdiag0_msg_queue;       /* msg queue */
#endif /* NSDIAG */

#ifdef LAN01
#include "../sio/lanc.h"
#include "../machine/drvhw_ift.h"

#if ((NUM_LAN_CARDS > 0) && (MAX_LAN_CARDS > NUM_LAN_CARDS))
int num_lan_cards = NUM_LAN_CARDS;
#else
#if (NUM_LAN_CARDS > MAX_LAN_CARDS)
int num_lan_cards = MAX_LAN_CARDS;    /* exceed MAX_LAN_CARDS */
#else  /* We force it to defatul */
int num_lan_cards = 2;
#endif  /* NUM_LAN_CARDS > MAX_LAN_CARDS */
#endif

lan_ift * lan_dio_ift_ptr[10];
#endif /* LAN01 */


/*
 * Streams subsystem
 */

#ifdef HPSTREAMS

int strmsgsz = STRMSGSZ;
int strctlsz = STRCTLSZ;
int nstrevent = NSTREVENT;
```

```
int nstrpush  = NSTRPUSH;

#include "../streams/str_hpux.h"
#include "../streams/str_stream.h"

#endif /* HPSTREAMS */


#define NETSLOP 20

#ifdef  NOSWAP
#define NOSWAP  1
#else
#define NOSWAP  0
#endif

#define NCLIST  (100+16*MAXUSERS)
int     nclist = NCLIST;

int     nproc = NPROC;
int     ninode = NINODE;

/*
 * maxfiles is the system default soft limit for the maximum number of
 * open files per process.  maxfiles defaults to 60 if not configured.
 * maxfiles_lim is the system default hard limit for the maximum number of
 * open files per process.  maxfiles_lim defaults to 1024 if not configured.
 */

int     maxfiles = MAXFILES;
int     maxfiles_lim = MAXFILES_LIM;

/*The NCDNODE should be defined in master for configurability. Before we
can actually do it, this is what we can do now.*/
#define NCDNODE 150
int     ncdnode = NCDNODE;
int     ncallout = NCALLOUT;
long    unlockable_mem = UNLOCKABLE_MEM;
int     nfile = NFILE + FILE_PAD;
int     file_pad = FILE_PAD;
int     nbuf = NBUF;
int     nflocks = NFLOCKS;
int     npty = NPTY;
int     ndilbuffers = NDILBUFFERS;
int     ncsize = NINODE;
struct ncache ncache[NINODE];

/*
 *      Hash table of open devices.
 */
dtaddr_t devhash[DEVHSZ];

int     maxuprc = MAXUPRC;
int     maxdsiz = MAXDSIZ/NBPG; /* unit: page size */
int     maxssiz = MAXSSIZ/NBPG; /* unit: page size */
int     maxtsiz = MAXTSIZ/NBPG; /* unit: page size */
```

```
int     parity_option = PARITY_OPTION;
int     reboot_option = REBOOT_OPTION;
int     noswap  = NOSWAP;
int     install = NOSWAP;
int     timeslice = TIMESLICE;          /* unit: 20ms tick */
int     acctsuspend = ACCTSUSPEND;      /* unit: percent of filesystem free
int     acctresume = ACCTRESUME;        /* unit: percent of filesystem free
int     dos_mem_byte = DOS_MEM_BYTE;    /* mem. reserved for dos in bytes

int     mem_no = 3;     /* major device number of memory special file */
int     ieee802_no  = 18;
int     ethernet_no = 19;
uint    dos_mem_start;                  /* physical addr. of dos mem. */
int     scroll_lines = SCROLL_LINES;    /* number of lines of ITE buffer */

/*
   The tty stuff that needs to be declared somewhere.
*/
#define NPCI    16
short   npci = NPCI;
struct tty *tty_line[NPCI];
struct tty *cons_tty;

/*
 * These have to be allocated somewhere; allocating
 * them here forces loader errors if this file is omitted.
 */
struct  proc *proc, *procNPROC, *cur_proc;
struct  inode *inode, *inodeNINODE;
struct  callout *callout;
struct  file *file, *fileNFILE, *file_reserve;
struct locklist locklist[NFLOCKS];      /* The lock table itself */
struct  tty  pt_tty[NPTY];
struct  tty  *pt_line[NPTY];
struct  pty_info  pty_info[NPTY];
struct  nvsj            nvsj[NPTY];
struct  buf     dil_bufs[NDILBUFFERS];
struct  iobuf  dil_iobufs[NDILBUFFERS];
struct  dil_info  dil_info[NDILBUFFERS];
int (*fhs_timeout_proc)() = NULL;

/* declarations for stub routines for non-configurable portions of EISA bus
extern nop();
int (*eisa_init_routine)() = nop;
int (*eisa_nmi_routine)() = nop;
int (*eisa_eoi_routine)() = nop;

/* declarations for stub routines for non-configurable portions of MTV (VME)
int (*vme_init_routine)() = nop;

/*
**      The following supports savecore on the s300
*/

long    dumplo;         /* offset into dumpdev */
int     dumpsize;       /* amount of NBPG phys mem to save - dep on swap */
```

```
int     dumpmag;        /* magic number for savecore, 0x8fca0101 */
/* dumpdev is now generated into conf.c by config */

struct  cblock *cfree;
struct  buf *buf, *swbuf;
short   *swsize;
int     *swpf;
char    *buffers;

struct  bufqhead bfreelist[BQUEUES];    /* heads of available lists */
struct  buf     bswlist;                /* head of free swap header list */

char    runin;                          /* scheduling flag */
char    runout;                         /* scheduling flag */
int     runrun;                         /* scheduling flag */
#ifdef  RTPRIO
u_char  curpri;                         /* more scheduling */
#else   /* RTPRIO */
char    curpri;                         /* more scheduling */
#endif  /* RTPRIO */

int     maxmem;                         /* actual max memory per process */
int     physmem;                        /* physical memory on this CPU */
int     hand;                           /* current index into coremap used b
int     wantin;
int     selwait;
/*
 * The following is for the shared memory subsystem (if configured)
 */

#if MESG==1
#include        "../h/ipc.h"
#include        "../h/msg.h"

struct ipcmap   msgmap[MSGMAP];
struct msqid_ds msgque[MSGMNI];
struct msg      msgh[MSGTQL];
struct msginfo  msginfo = {
        MSGMAP,
        MSGMAX,
        MSGMNB,
        MSGMNI,
        MSGSSZ,
        MSGTQL,
        MSGSEG
};
int     messages_present = 1;
#else
int     messages_present = 0;
#endif

#if SEMA==1
#       ifndef IPC_ALLOC
#       include "../h/ipc.h"
#       endif
#include        "../h/sem.h"
```

```
struct semid_ds sema[SEMMNI];
struct sem      sem[SEMMNS];
struct map      semmap[SEMMAP];
struct  sem_undo        *sem_undo[NPROC];
#define SEMUSZ  (sizeof(struct sem_undo)+sizeof(struct undo)*SEMUME)
int     semu[((SEMUSZ*SEMMNU)+NBPW-1)/NBPW];
union {
        short           semvals[SEMMSL];
        struct semid_ds ds;
        struct sembuf   semops[SEMOPM];
}       semtmp;

struct  seminfo seminfo = {
        SEMMAP,
        SEMMNI,
        SEMMNS,
        SEMMNU,
        SEMMSL,
        SEMOPM,
        SEMUME,
        SEMUSZ,
        SEMVMX,
        SEMAEM
};
int     semaphores_present = 1;
#else
int     semaphores_present = 0;
#endif

#if SHMEM == 1
#       ifndef  IPC_ALLOC
#       include "../h/ipc.h"
#       endif
#include        "../h/shm.h"
struct  shmid_ds        shmem[SHMMNI];
struct  shminfo shminfo = {
        SHMMAX,
        SHMMIN,
        SHMMNI,
        SHMSEG
};
int     shared_memory_present = 1;
#else
#       ifndef  IPC_ALLOC
#       include "../h/ipc.h"
#       endif
#include        "../h/shm.h"
struct  shmid_ds shmem[1];
int     shared_memory_present = 0;
#endif

/* The parser is currently not configurable, but when it is, modify the
 * assignment of (*pn_getcomponent)() = to your choice of parser.
 * right now its pn_getcomponent_n_computer() (8bit).
 */
/* two-byte characters in file names. */
```

```
/* extern int pn_getcomponent_chinese_t(); not supported yet */
extern int pn_getcomponent_n_computer();
#ifndef PARSER
#define PARSER pn_getcomponent_n_computer
#endif
int (*pn_getcomponent)() = PARSER;

struct pidchunk
{
        int start;
        int end;
} mypidchunks[NPROC];

/* The following are configuration flags for networking */
int rel1nsc_1_flag = 1;
int rel1nsc_2_flag = 1;
int rel1nsc_3_flag = 1;

int     swapspc_cnt;    /* pages of available swap space */
int     swapmem_max;    /* total pages of system available swap space */
int     swapmem_cnt;    /* pages of available memory for "swap" */
int     maxfs_pri;      /* highest available device priority */
int     maxdev_pri;     /* highest available swap prioirity*/
int     sys_mem;        /* pages of memory not available for "swap" */

int minswapchunks = MINSWAPCHUNKS;

#ifdef X25
#if (defined(NUM_PDN0) && (NUM_PDN0 >= 0))
#ifndef IPPROTO_ICMP
#include "../netinet/in.h"
#endif /* NOT IPPROTO_ICMP */
#ifndef IFF_UP
#include "../net/if.h"
#endif /* IFF_UP not defined */
#include "../x25/x25gen.h"
#endif /* NUM_PDN0 */
#endif /* X25 */

/*
 * Double Stuff data structures/configuration; a -1 value means that the
 * parameter will be calculated from available memory at boot time.
 */
#define VHNDFRAC        -1
#define MAXPMEM         -1

#include "../h/sysinfo.h"
#include "../h/pfdat.h"
#include "../h/swap.h"

int desperate;
struct minfo minfo;
struct pfdat **phash;
struct pfdat *pfdat;
int phashmask;  /* Page hash mask */
struct pfdat phead;
```

```
long phread, phwrite;

int swchunk = SWCHUNK;

int nswapfs = NSWAPFS;
struct fswdevt fswdevt[NSWAPFS];

int nswapdev = NSWAPDEV;

struct swap_stats swap_stats[NSWAPDEV+NSWAPFS+1];

int swapmem_on = SWAPMEM_ON;
int sysmem_max = SYSMEMMAX;
int maxswapchunks = MAXSWAPCHUNKS;

struct devpri swdev_pri[NSWPRI];
struct fspri swfs_pri[NSWPRI];

struct swaptab swaptab[MAXSWAPCHUNKS];

vm_sema_t swap_lock;
int nextswap;
int swapwant;
int mpid;                /* For generating unique process IDs */

#include "../h/var.h"
struct var v = {
        VHNDFRAC,
};
int ticks_since_boot;

/*
 * Variables used for sar
 */
#include "../h/sar.h"

long sar_swapin;
long sar_swapout;
long sar_bswapin;
long sar_bswapout;
struct syswait syswait;

int procovf = 0;
int istackptr = 0;       /* True if running on istack */
int freemem_cnt = 0;

#ifdef GENESIS

/* Set by graphics_make_entry(), used in main() to decide whether or */
/* not to start vdmad.                                               */

int vdma_present = 0;
#endif


/*
```

```
 * A bunch of stuff was allocated in proc.h.  I've moved it here.
 */
short   freeproc_list;          /* Header of free proc table slots */

struct  prochd qs[NQS];
int     whichqs[NQELS];         /* Bit mask summarizing non-empty qs's */

struct  map *sysmap;            /* Map of vaddr pool for system */
/*
 * HACK ATTACK
 *
 * Dux had defined this variable in cluster.c.  Including this module,
 * however leads to many more dux modules having to be compiled and linked
 * into the kernel.  Rather than deal with configurability now, we simply
 * hack around the problem, knowing full well that this isnt' used for
 * anything outside of a discless environment anyway.
 */
#include "../dux/duxparam.h"
#include "../dux/cct.h"
struct cct clustab[MAXSITE];    /* incore cluster configuration table */



/* File system async flag. If set file system data structures
   are written asychronously.                  */

int fs_async = FS_ASYNC;

/*
 * flag to control creation of "fast" symbolic links.
 */
int create_fastlinks = CREATE_FASTLINKS;

/*
 * flag to turn off new AES conformance behavior for hp-ux system calls.
 */
int hpux_aes_override = AES_OVERRIDE;

/* hash table size scale with number of items hashed */

/* lpow2 returns largest power of 2 less than arg, min value 16, max 8192 */
#define lpow2(arg) \
        (arg) < 32? 16: \
        (arg) < 64? 32: \
        (arg) < 128? 64: \
        (arg) < 256? 128: \
        (arg) < 512? 256: \
        (arg) < 1024? 512: \
        (arg) < 2048? 1024: \
        (arg) < 4096? 2048: \
        (arg) < 8192? 4096: \
        8192

#define hashsize(length, items, default) \
        (lpow2((items)/(length)))
```

```
/* proc table */

#define PIDHSZ hashsize(4, NPROC, 64)
int     PIDHMASK = PIDHSZ - 1;
short pidhash[PIDHSZ];

#define PGRPHSZ hashsize(4, NPROC, 64)
int     PGRPHMASK = PGRPHSZ - 1;
short pgrphash[PGRPHSZ];

#define UIDHSZ hashsize(4, NPROC, 64)
int     UIDHMASK = UIDHSZ - 1;
short uidhash[UIDHSZ];

#define SIDHSZ hashsize(4, NPROC, 64)
int     SIDHMASK = SIDHSZ - 1;
short sidhash[SIDHSZ];


/* sleep table */

#define SQSIZEDEF hashsize(4, NPROC, 64)
int     SQSIZE  = SQSIZEDEF;
int     SQMASK  = SQSIZEDEF-1;

struct proc *slpque[SQSIZEDEF];
struct proc *slptl[SQSIZEDEF];  /* For FIFO sleep queues */

/* buffer table */

/* average buf hash chain length desired -- see machdep.c */
int bufhash_chain_length = 4;
struct  bufhd *bufhash; /* buffer hash table */
int     BUFHSZ, BUFMASK;/* size and mask for accessing bufhash */

/* inode table */

#define INOHSZDEF hashsize(6, NINODE, 64)
int     INOHSZ  = INOHSZDEF;
int     INOMASK = INOHSZDEF-1;
union ihead {                           /* inode LRU cache, Chris Maltby */
        union  ihead *ih_head[2];
        struct inode *ih_chain[2];
} ihead[INOHSZDEF];


/* spinlocks */

#define SPINSIZEDEF (B_SEMA_HTBL_SIZE + SQSIZEDEF + 50)
int MAX_SPINLOCKS = SPINSIZEDEF;

lock_t  spin_alloc_base[SPINSIZEDEF] = { 0 };
lock_t  *spin_alloc_end = spin_alloc_base + SPINSIZEDEF;

int ddb_boot = DDBBOOT;
```

```
/* @(#) $Revision: 1.11.61.2 $ */

#define MSG_MAGIC       0x063060
#define MSG_BSIZE       (4096 - 2 * sizeof (long))

struct  msgbuf {
        long    msg_magic;
        long    msg_bufx;
        char    msg_bufc[MSG_BSIZE];
};

#ifdef  __hp9000s800
extern  struct  msgbuf msgbuf;
#else
#ifdef  __hp9000s300
struct  msgbuf Msgbuf;
#else
struct  msgbuf msgbuf;
#endif  /* else not __hp9000s300 */
#endif  /* else not __hp9000s800 */
```

SE 390: Series 300 HP-UX Internals

Process Management

The Big Picture

- How does HP-UX share system resources among competing processes?

The Little Picture(s)

- The context of a process.
- Signal handling & job control.
- Process creation/deletion.
- Fork - duplicate current process.
- Exec - replace current program with another.
- Context switching.
- Tunable parameters.

The Problem

```
$ ps -ef
fork failed - too many processes
```

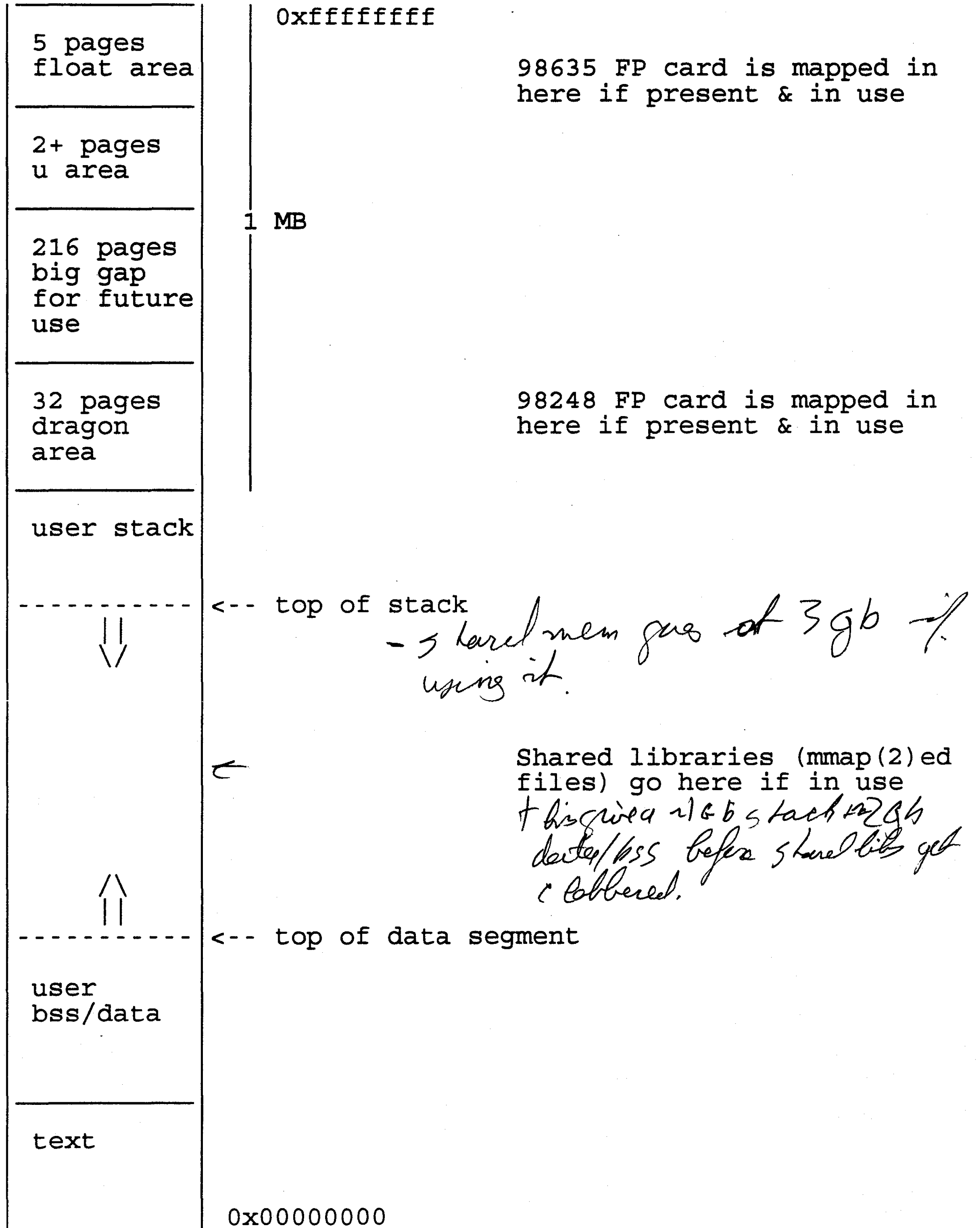
What's going on here?

Process Management

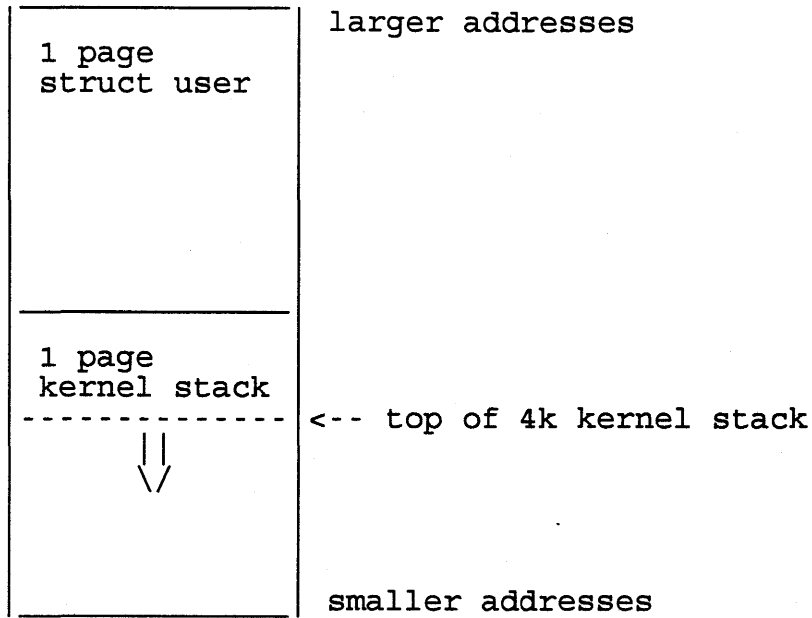
The Context of a Process (running program)

- Stack, text, and data areas.
- Registers, stack pointer, program counter, etc.
- Segment and page tables.
- The u area - defined in /usr/include/sys/user.h.
    - available when process is in memory - won't be paged out, but can be swapped with the process
    - has stuff like arguments to system calls, a place to save registers, the command that was typed, etc. These are things we don't need to have available when the process is swapped out.
    - the kernel stack is part of the u area, but is not defined in user.h - it is actually in a different page and is not part of the "user structure".
- the proc table entry - defined in /usr/include/sys/proc.h
    - stuff that needs to always be available - priority, PID, signal masks, etc.
- State
    - running - we are the currently executing process.
    - runnable - we are ready to run, and are waiting for the processor.
        - in a run queue based on our priority
    - stopped - we were running, but were stopped by ptrace(2) or we received a SIGTSTP (BSD Job Control).
    - sleeping - we are waiting for a resource.
        - in a sleep queue based on temporary priority (interruptible if sleep will *NOT* end quickly; comatose if it will :-)
    - zombie - we've exited, but parent hasn't done a wait(2) on us yet; *all* resources are freed up except the proc table entry (& u area in 8.0).

68K process logical address space:



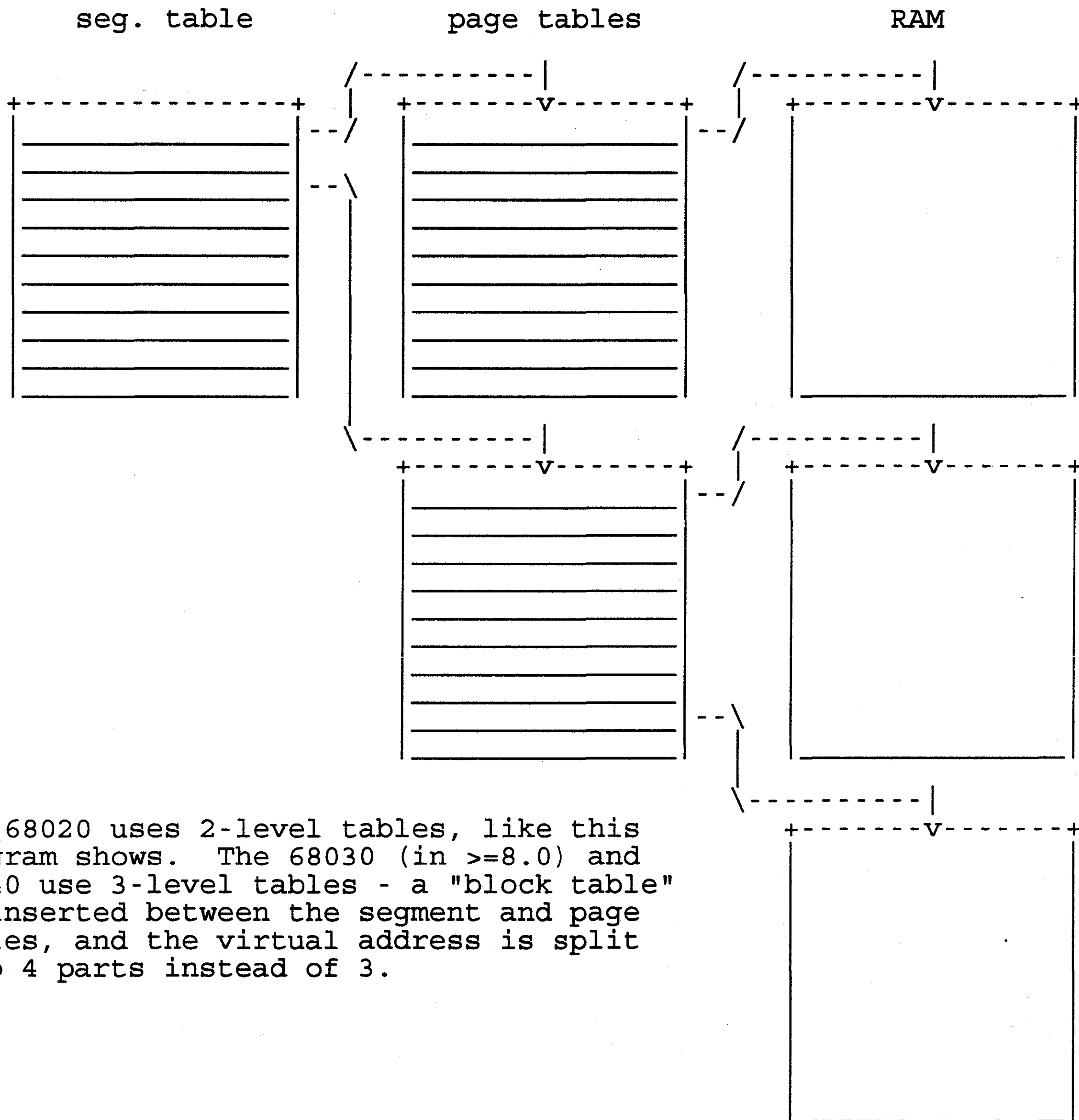
68K u area looks like this:



In 8.0 and later releases, kernel stacks are actually allowed to use 4 pages (rather than just 1), but this is not often done (most kernel functions do *not* use much stack space).

Process-eye View of Memory Management (68K)

- The segment table pointer is the root of all address translation.

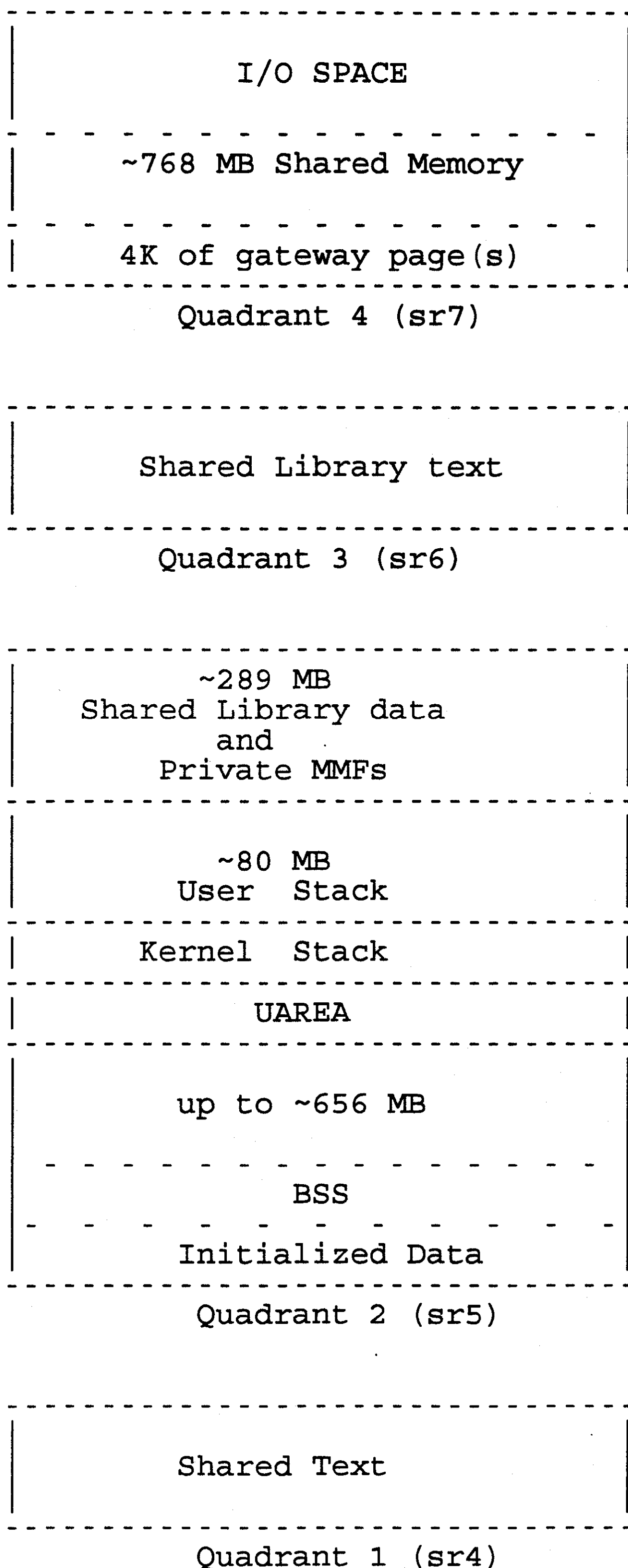


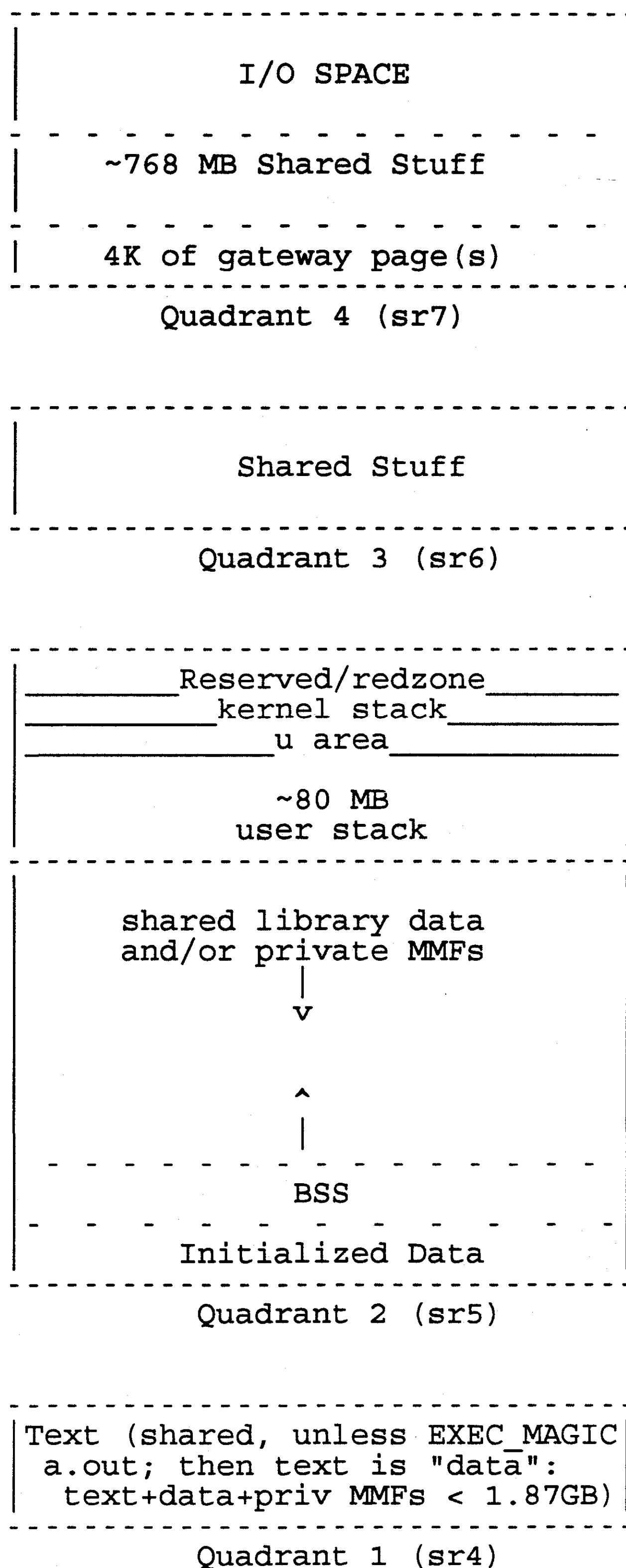
The 68020 uses 2-level tables, like this diagram shows. The 68030 (in >=8.0) and 68040 use 3-level tables - a "block table" is inserted between the segment and page tables, and the virtual address is split into 4 parts instead of 3.

- The 680x0/MMU have stack and segment table pointers for both user and supervisor modes. Whenever a process gets to use the CPU, its segment table pointer and stack pointers are put into the appropriate hardware registers. In the table below, each item marked with Xs is changed at context-switch time.

	segment table	stack
user	XXXXX	XXXXX
supervisor		XXXXX

700 Per-process Virtual Address Space





Signal Handling

- Signal sending
    - crude form of IPC
    - accomplished with kill(2), which is the heart of kill(1), as in

      $ kill -1 2344

    - SIGUSR[12] are available for cooperating processes

- Signal receiving or "catching"
    - Read signal(5) for an overview of the various signal families.
    - can be controlled somewhat with sig*(2)
        - can specify a procedure to call when a given signal comes in
        - can specify an alternate signal stack
    - if a non-default handler is specified, it will be called in such a way that it appears to be a normal procedure call
    - SIGKILL (as in "kill -9") can NOT be caught or ignored
        - special case for init(1m) - kill(2) will refuse to send SIGKILL to PID 1!

Signal Implementation

- Signal sending

  - set a bit in the proc table entry of the receiving process

  - mark receiving process as runnable, *as long as it isn't sleeping at a priority of PZERO or less* - this is important to remember, but shouldn't often be an issue

- Signal receiving

  - check to see if we have signal(s) pending whenever we're about to return to user mode from kernel mode and whenever we block in the kernel (by calling sleep()).

  - if we do, handle them or core dump or exit or whatever....

  - if we were in the middle of a system call, we may restart it or we may return an error - depends on what programmer asked for.

SE 390: Series 300 HP-UX Internals

Process Management

Process Creation/Deletion

- Created by fork(2).
    - most things are exactly duplicated
    - things like pid, ppid, etc. are different
    - stdio buffers are duplicated
    - vfork(2) is a fast version - it does NOT copy the stack and data - it trusts the child to do an exec
        - in 8.0, copy-on-write has made normal fork(2) fast as well
- Currently-running program replaced by exec(2).
    - things like file descriptors are preserved
    - things like "when this signal comes in, call this routine" are NOT preserved
- Deleted by exit(2) (voluntary), or most signals (involuntary).

    >> - note that unless parent process does a wait(2), there <<
    >> will be a zombie sitting around... <<

- A process gets created whenever fork is called.

Process Management

What Happens When Fork(2) Is Called

- The general idea is to "xerox" the calling process, changing only the things that must be unique (PID, resource usage, etc)
- Specifics:
  - child will share
    - text (code - including shared libs, if used), shared memory; in general, any SHARED regions
    - references to open files, current/root dirs
  - child must have its own
    - proc table entry and u area
    - page tables (68K)
  - if this is a real fork and not a vfork, child will have its own
    - data
    - stack
    - swap area for the above
- vfork(2) is a fast, cheap alternative to fork(2) - useful when all we want to do is exec(2) something; the basic idea is to borrow the parent's resources rather than making copies of them that are immediately thrown away
- in 8.0, fork(2) is implemented with copy-on-write
  - parent and child have the same physical pages mapped
  - pages are marked readonly
  - when parent *or* child modifies a page, it gets a private copy of that page
  - most of the time, very few pages are modified before the child exits or execs; this winds up being a significant performance win
  - vfork(2) was initially implemented this way (in 8.x), but this caused *serious* problems:
    - the child had to have swap space allocated
    - programs that used it as cheap shared memory broke

SE 390: Series 300 HP-UX Internals

Process Management

What Happens When Exec(2) Is Called

- Check modes: execute bits, set[ug]id bits, etc.
- Read in first few bytes to see what kind of file it is.
- If it is non-shared, lump the data and text together as data.
- If it is a "#!" script, loop to get the real executable file.
- Be sure the file is as big as the header claims, but not too big.
- Copy arguments to a buffer.
- Be sure the file is big enough to have text, data, etc.
- Be sure text isn't busy: ptrace(2), open for write, etc.
- Get *swap* space.
- Release any locked memory.
- If we are a "vfork child", give memory back to the parent; otherwise, release memory.
- Get virtual memory (actually just initialize page tables to the appropriate thing - usually zero-fill-on-demand).
- Read data (and text if non-shared) in.
- Attach to text, reading it in if necessary.
- Set uid/gid.
- Copy arguments from buffer to new stack.
- Set registers (mostly clear them, but one is used to tell if we have a floating point card and one is used to indicate processor type).
- Reset caught signals - there's nothing to catch them anymore!
- Close close-on-exec files.

Context Switching - Priorities

- Our fundamental goal is to be running the most important process at any given time; for a typical process, its "importance" is determined by its recent CPU usage and its nice value.

- Every time the clock ticks (50 times/sec = every 20 ms for the 300/400, 100 times/sec = every 10 ms for the 700), the process that was running when the clock interrupted is charged with a "tick" of CPU time (i.e. its p_cpu gets incremented).

- The system keeps a rough count of the number of processes that are either runnable or will/could be very soon in an array called "avenrun"; this is often referred to as the "load average" and is what things like xload/top/uptime/monitor print.

- p_cpu is decayed once per second, and all process priorities are recalculated:

    - p_cpu = p_cpu*(2*load_ave)/(2*load_ave + 1) + nice_value

    - p_usrpri = PUSER + p_cpu/4 + 2*nice_value

- If process has been rtprio()'ed, forget the 2nd part....

- Process priorities are recalculated every second for all processes on the system (via the two equations above), and every four clock ticks for the current process.

- When some process becomes more important than the current one, a context switch is requested. The switch won't actually happen until we are ready to go back into user mode.

- A switch will automatically be requested every timeslice/HZ of a second. Since timeslice is normally HZ/10, we will default to requesting a switch every 1/10th of a second.
  300/400: HZ = 50 700: HZ = 100

Context Switching - Mechanics

- Can only happen when
  - process blocks by calling sleep() (in the kernel);
  - process is about to return to user mode from kernel mode; this could be a return from an interrupt or exception handler or a system call.
- Save current context into u area, which is mapped into the top of the process' address space on the 68K and quadrant 2 on PA systems
- Restore other process' context from its u area.
- Resume execution.

Context Switching - Being Nice :-)

- Before 9.0, the "nice value" was used in the equations for calculating process priority and had a *small* influence on the swapper. It affected how much a process could use the CPU, but did not really affect how much of the system's throughput a process could consume.
- In 9.0, a process' nice value will have more effect on how much it can do - the pager and swapper pay *much* more attention to the nice value than they used to. This can be used in positive and negative ways - to preserve interactive performance, one could negatively nice the X server and positively nice the chip simulator running in the background.
- nice(1) is a command wrapper around nice(2), which will change the nice value of the current process (must be root to improve it :-)
- renice(1) uses setpriority(2), which allows an appropriate user to change the priority of other processes (not just the current one). Top (version 2.5 or greater) also uses this.

Tunable Parameters

- maxfiles - default number of files a single process can open
    - defaults to 60
- maxfiles_lim - number of files a process can open if it does a setrlimit(2) call
    - defaults to 1024
- maxuprc - number of processes a single user (UID) can have
    - setting it high allows a single user to take lots of the system's resources
    - setting it low can cause users to get angry
- nproc - maximum number of processes on the system at any given tim
    - used to size a static array, the proc table
    - it is also used to size other kernel data structures that relate to the number of processes on the system
- timeslice - length of timeslice for round-robin CPU scheduling
    - normally 100ms (timeslice of "5" on 68K, "10" on PA)
    - setting it too low makes us spend more of our time switching, less of it working
    - setting it too high means interactive response is bad

Kernel Variables Of Interest

- _nproc, _timeslice from above; both are integers
- _proc - pointer to proc table; defined in proc.h
- u area - see getu.c

SE 390: Series 300 HP-UX Internals

Process Management

Summary

- A process is a running program, and consists of text, data, and stack areas as well as a u area and proc-table entry. Most processes also use shared libraries, and some use shared memory.

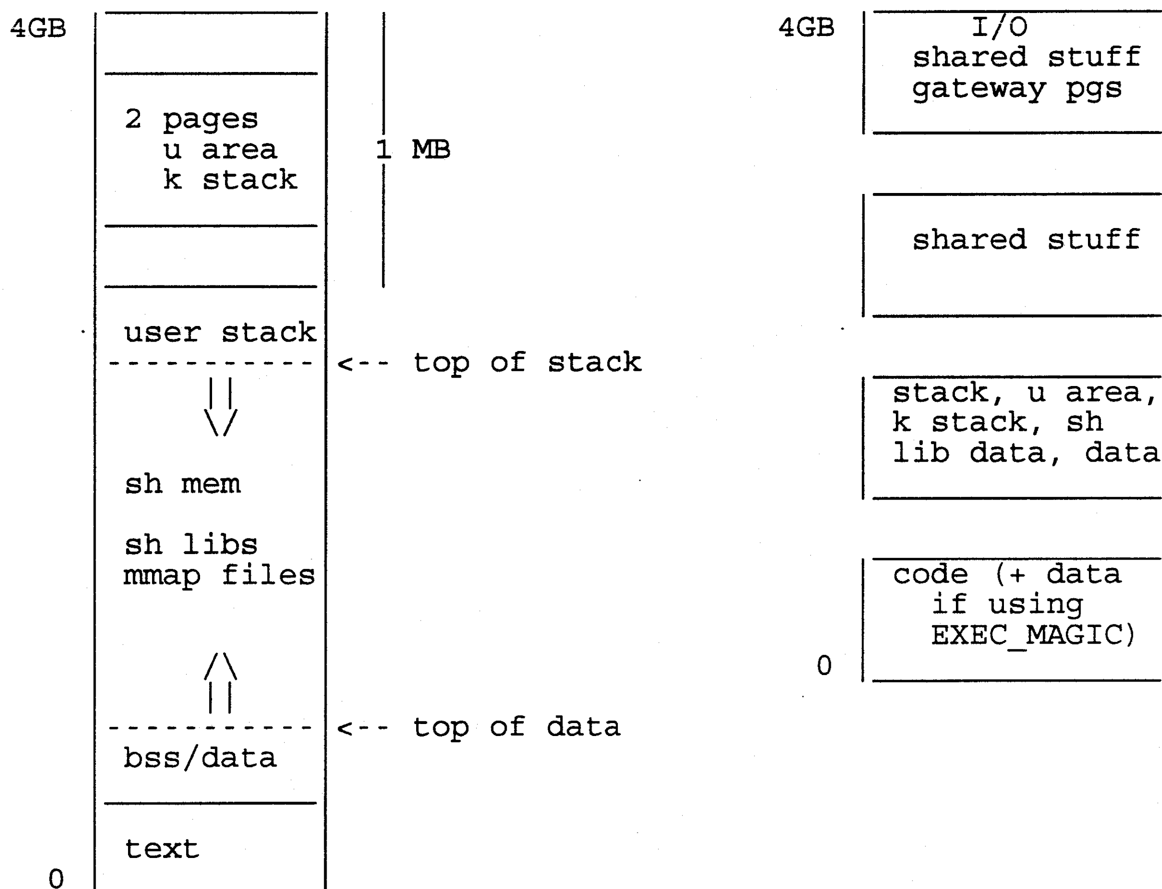
- Context switching refers to the kernel's efforts to be sure we are running the "right" process at any given time. Processes "lose" priority by using up CPU time, and the kernel sees if it should switch processes any time the CPU is going from kernel mode to user mode.

- Each process gets a slot in the "proc table", and this table is sized by "nproc" (a tunable parameter). This parameter is also used to size other things, so it is a good one to bump up if there are general resource problems on the system.

- The proc-table entry is unique in that it will never be swapped out for as long as the process exists. This is important, and has much to do with the next point....

- To send a signal, all we do is set a bit in the proc-table entry of the receiving process, and (possibly) mark it runnable.

- process logical address space:



```
 1  /*
 2   * @(#)proc.h: $Revision: 1.65.61.12 $ $Date: 92/06/29 10:44:30 $
 3   *
 4   */
 5
 6  #ifdef __hp9000s300
 7  #include <machine/pte.h>
 8  #endif /* __hp9000s300 */
 9
10  #ifdef __hp9000s800
11  #include <sys/fss.h>
12  #endif /* __hp9000s800 */
13
14  #include <sys/vas.h>
15  #include <sys/pregion.h>
16  #include <sys/time.h>
17  #include <sys/mman.h>
18
19  /* Values for vfork_state field in struct vforkinfo */
20
21  #define VFORK_INIT      0
22  #define VFORK_PARENT    1
23  #define VFORK_CHILDRUN  2
24  #define VFORK_CHILDEXIT 3
25  #define VFORK_BAD       4
26
27  /*
28   * The following structure is used by vfork to hold state while a
29   * vfork is in progress.
30   */
31
32  struct vforkinfo {
33      int vfork_state;
34      struct proc *pprocp;
35      struct proc *cprocp;
36      unsigned long buffer_pages;
37      unsigned long u_and_stack_len;
38  #ifdef __hp9000s300
39      unsigned char *u_and_stack_addr;
40  #endif
41  #ifdef __hp9000s800
42      unsigned long saved_rp_ptr;
43      unsigned long saved_rp;
44  #endif
45      unsigned char *u_and_stack_buf;
46      struct vforkinfo *prev;
47  };
48
49  /*
50   * One structure allocated per active
51   * process. It contains all data needed
52   * about the process while the
53   * process may be swapped out.
54   * Other per process data (user.h)
55   * is swapped with the process.
56   */
```

```
 57  typedef struct  proc {
 58          struct  proc *p_link;   /* linked list of running processes */
 59          struct  proc *p_rlink;
 60          u_char  p_usrpri;       /* user-priority based on p_cpu and p_nice */
 61          u_char  p_pri;          /* priority, lower numbers are higher pri */
 62          u_char  p_rtpri;        /* real time priority */
 63          char    p_cpu;          /* cpu usage for scheduling */
 64          char    p_stat;
 65          char    p_nice;         /* nice for cpu usage */
 66          char    p_cursig;
 67          int     p_sig;          /* signals pending to this process */
 68          int     p_sigmask;      /* current signal mask */
 69          int     p_sigignore;    /* signals being ignored */
 70          int     p_sigcatch;     /* signals being caught by user */
 71          int     p_flag;         /* see flag defines below */
 72          int     p_flag2;        /* more flags; see below */
 73          int     p_coreflags;    /* core file options; see core.h */
 74  #ifdef _CLASSIC_ID_TYPES
 75          u_short p_filler_uid;
 76          u_short p_uid;          /* user id, used to direct tty signals */
 77  #else
 78          uid_t   p_uid;          /* user id, used to direct tty signals */
 79  #endif
 80  #ifdef _CLASSIC_ID_TYPES
 81          u_short p_filler_suid;
 82          u_short p_suid;         /* set (effective) uid */
 83  #else
 84          uid_t   p_suid;         /* set (effective) uid */
 85  #endif
 86  #ifdef _CLASSIC_ID_TYPES
 87          u_short p_filler_pgrp;
 88          short   p_pgrp;         /* name of process group leader */
 89  #else
 90          gid_t   p_pgrp;         /* name of process group leader */
 91  #endif
 92  #ifdef _CLASSIC_ID_TYPES
 93          u_short p_filler_pid;
 94          short   p_pid;          /* unique process id */
 95  #else
 96          pid_t   p_pid;          /* unique process id */
 97  #endif
 98  #ifdef _CLASSIC_ID_TYPES
 99          u_short p_filler_ppid;
100          short   p_ppid;         /* process id of parent */
101  #else
102          pid_t   p_ppid;         /* process id of parent */
103  #endif
104          caddr_t p_wchan;        /* event process is awaiting */
105          size_t  p_maxrss;       /* copy of u.u_limit[MAXRSS] */
106          u_short p_cpticks;      /* ticks of cpu time */
107          long    p_cptickstotal; /* total for life of process */
108          float   p_pctcpu;       /* %cpu for this process during p_time */
109          short   p_idhash;       /* hashed based on p_pid for kill+exit+... */
110          short   p_pgrphx;       /* pgrp hash index */
111          short   p_uidhx;        /* uid hash index */
112          short   p_fandx;        /* free/active proc structure index */
```

```
113         short   p_pandx;          /* previous active proc structure index */
114         struct  proc *p_pptr;     /* pointer to process structure of parent */
115         struct  proc *p_cptr;     /* pointer to youngest living child */
116         struct  proc *p_osptr;    /* pointer to older sibling processes */
117         struct  proc *p_ysptr;    /* pointer to younger siblings */
118         struct  proc *p_dptr;     /* pointer to debugger, if not parent */
119         vas_t   *p_vas;           /* Virtual address space for process */
120         preg_t  *p_upreg;         /* Pointer to pregion containing U area */
121         ushort  p_mpgneed;        /* number of memory pages needed */
122         struct  proc *p_mlink;            /* link list of processes    */
123                                           /* sleeping on memwant or    */
124                                           /* swapwant.                 */
125         short   p_memresv;        /* # pages reserved by this proc */
126         short   p_swpresv;        /* # pages reserved by swapper this proc */
127         u_short p_xstat;          /* exit stauts */
128         char    p_time;           /* resident time for scheduling */
129         char    p_slptime;        /* time since last block */
130         short   p_ndx;
131         struct  itimerval p_realtimer;
132         sid_t   p_sid;            /* session ID */
133         short   p_sidhx;          /* session ID hash index */
134         short   p_idwrite;        /* process ident write flag for auditing */
135         struct fss *p_fss;        /* fair share group pointer */
136         struct dbipc *p_dbipcp;   /* dbipc pointer */
137         u_char  p_wakeup_pri;     /* priority when proc awakens on semaphore */
138         u_char  p_reglocks;       /* num reglock()'s held (see vm_sched.c) */
139                                   /* VASSERTS in region.h know this is 1 byte */
140         caddr_t p_filelock;       /* address of file lock region process is
141                                      either blocked on or about to block on.*/
142         /* Doubly linked list of processes sharing the same controlling tty.
143          * Head of list is u.u_procp->p_ttyp->t_cttyhp.
144          */
145         struct proc *p_cttyfp;    /* forward ptr */
146         struct proc *p_cttybp;    /* backward ptr */
147         caddr_t p_dlchan;         /* Process deadlock channel              */
148         site_t  p_faddr;          /* Process forwarding address            */
149         /* Fields used by the pstat system call. */
150         struct timeval
151                 p_utime,
152                 p_stime;
153         dev_t           p_ttyd;
154         time_t          p_start;
155
156         struct tty *p_ttyp;       /* controlling tty pointer */
157         int     p_wakeup_cnt;     /* generic counter, wakeup when goes to 0 */
158 #ifdef MP
159 #ifdef SYNC_SEMA_RECOVERY
160         sema_t  *p_recover_sema; /* Semaphore to recover on exit from sleep */
161 #endif
162         int     p_descnt;         /* proc desire age */
163         int     p_desproc;        /* processor desired  */
164         int     p_mpflag;         /* mp flag */
165         int     p_procnum;        /* Processor it ran on, just for user info */
166 #endif  /* MP */
167         struct proc *p_wait_list;         /* Forward link for wait list */
168         struct proc *p_rwait_list;        /* Backward link for wait list */
```

```
169
170             struct sema    *p_sleep_sema;    /* semaphore process is blocked on */
171             struct sema    *p_sema;  /* alpha: head of per-process semaphore list *
172
173             /* These fields have been moved from user.h because you can no
174              * longer retreive this information from a uarea which has been
175              * swapped out.
176              */
177             int      p_maxof;                /* max number of open files allowed */
178             struct  vnode *p_cdir;           /* current directory */
179             struct  vnode *p_rdir;           /* root directory of current process *
180             struct  ofile_t **p_ofilep;      /* pointers to file descriptor chunks
181                                                 to be allocated as needed. */
182
183
184             struct vforkinfo *p_vforkbuf;   /* Vfork state information pointer */
185             struct msem_procinfo *p_msem_info; /* Pointer to msemaphore info struc
186
187     /* All workstation specific fields */
188     #ifdef _WSIO
189             /* support for dil interrupts */
190             struct buf *p_dil_event_f; /* head of list of pending dil interrupts *
191             struct buf *p_dil_event_l; /* tail of list of pending dil interrupts *
192             struct pte *p_addr;      /* u-area kernel map address */
193             struct ste *p_segptr;    /* physical segment table pointer */
194             int     p_stackpages;    /* Number of private kernel stack pages */
195             u_char  p_dil_signal;    /* which signal to use for DIL interrupts */
196     #endif /* _WSIO */
197
198
199     #ifdef __hp9000s300
200     /* Only the 300 uses these time fields in this manner */
201     #define p_uticks        p_utime.tv_sec
202     #define p_sticks        p_stime.tv_sec
203
204     #endif /* __hp9000s300 */
205
206     /* All 800 specific fields */
207     #ifdef __hp9000s800
208             u_short p_pindx;         /* index of this proc table entry */
209     #       ifdef _WSIO
210                 caddr_t graf_ss;     /* graphics per-process (mostly coproc) data *
211     #       endif
212     #endif /* __hp9000s800 */
213     } proc_t;
214
215                                                 /* chain */
216     extern  struct  proc *pfind();
217     extern  struct  proc *proc, *procNPROC;  /* the proc table itself */
218     extern  int     nproc;
219
220     #ifdef __hp9000s800
221     #define NQS     160              /* 160 run queues = 128 RT + 32 TS */
222     #define NQSPEL   16              /* Number of run queues per whichqs element*/
223     #define NQSPELLG 4               /* log2(NQSPEL)*/
224     #define NQELS    (NQS/NQSPEL)      /*  10 elements to hold bitmask(whichqs) */
```

```
225  #define TSQ     128                /* First time-sharing queue */
226  #define TSPRI_TO_RUNQ(pri)         (TSQ + (((pri)-PTIMESHARE) >> 2))
227  #else  /* not __hp9000s800 */
228  #define NQS     256                /* 256 run queues 128 RT + 128 TS */
229  #define NQSPEL   32                /* Number of run queues per whichqs element */
230  #define NQELS   (NQS/NQSPEL)       /* 8 32-bit elements to hold bitmask(whichqs)
231  #define TSPRI_TO_RUNQ(pri)         (pri)   /* Don't use anywhere but schedcpu! */
232  #endif /* not __hp9000s800 */
233
234  struct  prochd {
235          struct  proc *ph_link;     /* linked list of running processes */
236          struct  proc *ph_rlink;
237  };
238
239  extern  struct  prochd qs[NQS];
240  extern  int     whichqs[NQELS]; /* bit mask summarizing non-empty qs's */
241  #endif  /* _KERNEL */
242
243  /* stat codes */
244  #define SSLEEP  1                  /* awaiting an event */
245  #define SWAIT   2                  /* (abandoned state) */
246  #define SRUN    3                  /* running */
247  #define SIDL    4                  /* intermediate state in process creation */
248  #define SZOMB   5                  /* intermediate state in process termination *
249  #define SSTOP   6                  /* process being traced */
250
251  /* flag codes (p_flag) */
252  #define SLOAD   0x00000001         /* in core */
253  #define SSYS    0x00000002         /* swapper or pager process */
254  #define SLOCK   0x00000004         /* process being swapped out */
255  #define STRC    0x00000008         /* process is being traced */
256  #define SWTED   0x00000010         /* another tracing flag */
257  #define SKEEP   0x00000040         /* another flag to prevent swap out */
258  #define SOMASK  0x00000080         /* restore old mask after taking signal */
259  #define SWEXIT  0x00000100         /* working on exiting */
260  #define SPHYSIO 0x00000200         /* doing physical i/o (bio.c) */
261  #define SVFORK  0x00000400         /* Vfork in process */
262  #define SSEQL   0x00000800         /* user warned of sequential vm behavior */
263  #define SUANOM  0x00001000         /* user warned of random vm behavior */
264  #define SOUSIG  0x00002000         /* using old signal mechanism */
265  #define SOWEUPC 0x00004000         /* owe process an addupc() call at next ast */
266  #define SSEL    0x00008000         /* selecting; wakeup/waiting danger */
267  #define SRTPROC 0x00010000         /* real time processes */
268  #define SSIGABL 0x00020000         /* signalable process */
269  #define SPRIV   0x00040000         /* compute privilege mask */
270  #define SPREEMPT 0x00080000        /* Preemption flag */
271  #ifdef HPNSE
272  #define SPOLL   0x00100000         /* process is polling */
273  #endif
274
275  #ifdef _WSIO
276  /* more p_flag bits, used for process deactivation */
277  #define SSTOPFAULTING 0x00200000
278  #define SSWAPPED 0x00400000
279  #define SFAULTING 0x00800000
280
```

```
281  /* used to track number of faulting processes (not a p_flag bit) */
282  #define FAULTCNTPERPROC 8
283  #endif /* _WSIO */
284
285  /* flags for p_flag2 */
286  #define S2CLDSTOP        0x00000001      /* send SIGCLD for stopped processes *
287  #define S2EXEC           0x00000002      /* if bit set, process has completed
288                                              an exec(OS) call */
289  #define SGRAPHICS        0x00000004      /* The process is a graphics process *
290  #define SADOPTIVE        0x00000008      /* process adopted using ptrace */
291  #ifdef __hp9000s800
292  #define SSAVED           0x00000010      /* registers saved for ptrace */
293  #define SCHANGED         0x00000020      /* registers changed by ptrace */
294  #define SPURGE_SIDS      0x00000100      /* purge cr12 and cr13 in resume() */
295  #endif /* __hp9000s800 */
296  #ifdef __hp9000s300
297  #define S2DATA_WT        0x00000010      /* Process's data segment is write thr
298  #define S2STACK_WT       0x00000020      /* Process's stack segment is write th
299  #endif /* __hp9000s300 */
300  #define SANYPAGE         0x00000040      /* Doing any kind of pageing */
301  #define SPA_ON           0x00000080      /* Under consideration for
302                                              activation control */
303  #define S2POSIX_NO_TRUNC  0x00001000  /* no truncate flag for pathname lookup*
304  #define POSIX_NO_TRUNC  S2POSIX_NO_TRUNC /* until dux_sdo.c is fixed */
305
306  #ifdef _WSIO
307  #define S2SENDDILSIG     0x00000200 /* whether to send DIL interrupt (cleared o
308  #endif /* _WSIO */
309  #define SLKDONE          0x00000400    /* Process has done lockf() or fcntl()
310  #define SISNFSLM         0x00000800    /* Process is NFS lock manager. */
311                                         /* See nfs_fcntl() in nfs_server.c */
312
313  #define S2TRANSIENT 0x00002000  /* transient flag (fair share scheduler) */
314
315  #ifdef MP
316  /* These are p_mpflag values */
317  #define SLPT             0x00000001      /* a Lower Priv Transfer trap brought
318  #define SRUNPROC         0x00000002      /* Running on a processor */
319  #define SMPLOCK          0x00000004      /* Locked  */
320  #define SMP_SEMA_WAKE    0x00000008      /* proc awakened by V operation,
321                                              not signal */
322  #define SMP_STOP         0x00000010      /* Process entering stopped state. */
323  #define SMP_SEMA_BLOCK   0x00000020      /* Process blocked on semaphore */
324  #define SMP_SEMA_NOSWAP  0x00000040      /* Do not swap this process */
325  #endif /* MP */
326
327  #ifdef __hp9000s300
328  #define PROCFLAGS2 (SADOPTIVE|S2EXEC|S2SENDDILSIG)
329  #endif
330  #ifdef __hp9000s800
331  #define PROCFLAGS2 (SADOPTIVE|S2EXEC|SCHANGED|SSAVED|S2TRANSIENT)
332  #endif
333
334  /* Constants which are used to call newproc */
335  #define FORK_PROCESS     1
336  #define FORK_VFORK       2
```

```
337  #define FORK_DAEMON     3
338
339  /* Return values for newproc/procdup */
340  #define FORKRTN_PARENT  0
341  #define FORKRTN_CHILD   1
342  #define FORKRTN_ERROR   -1
343
344  /* Constants which can be used to index proc table for kernel daemons*/
345  #define S_SWAPPER       0
346  #define S_INIT          1
347  #define S_PAGEOUT       2
348  #define S_STAT          3
349  #define S_DONTCARE      -1
350
351
352  /* Constants which can be used for pid argument to newproc() */
353  /* Note: proc table slot and pid may be different for some processes   */
354
355  #define PID_SWAPPER     0
356  #define PID_INIT        1
357  #define PID_PAGEOUT     2
358  #define PID_STAT        3
359  #define PID_LCSP        4
360  #define PID_NETISR      5
361  #define PID_SOCKREGD    6
362  #define PID_VDMAD       7
363  #define PID_MAXSYS      7  /* Used in dux/getpid.c */
364
```

```
 1  /* @(#) $Revision: 1.65.61.10 $ */
 2
 3  #include <machine/pcb.h>
 4  #include <sys/time.h>
 5  #include <sys/resource.h>
 6  #include <sys/privgrp.h>
 7  #include <errno.h>              /* u_error codes */
 8  #include <sys/signal.h>         /* SIGARRAYSIZE */
 9  #include <sys/proc.h>
10  #ifdef __hp9000s300
11  #include <a.out.h>
12  #endif /* __hp9000s300 */
13  #ifdef __hp9000s800
14  #include <sys/vmmac.h>
15  #include <machine/save_state.h>
16  #include <machine/som.h>
17  #endif /* __hp9000s800 */
18
19  /*
20   * NFDCHUNKS = number of file descriptor chunks of size SFDCHUNK available
21   * per process.  SFDCHUNK must be NBTSPW = number of bits per int for
22   * select to work.
23   */
24  #define SFDCHUNK        NBTSPW
25  #define NWORDS(n)       ((((n) & (SFDCHUNK - 1)) == 0) ? (n >> 5) :     \
26                                                          ((n >> 5) + 1))
27  /* NWORDS is the number of words necessary for n file descriptors to allow
28     for one bit per file descriptor. */
29
30  #define NFDCHUNKS(n)    NWORDS(n)
31
32  /*
33   * Some constants for fast multiplying, dividing, and mod-ing (%) by SFDCHUNK
34   */
35
36  #define SFDMASK 0x1f
37  #define SFDSHIFT 5
38
39  struct ofile_t {
40          struct file *ofile[SFDCHUNK];   /* file descriptor slots */
41          char pofile[SFDCHUNK];          /* per process open file flags */
42  };
43
44  /*
45   * since fuser() needs this information, we move it to the proc structure
46   * since uareas can be swapped out.  In previous releases, fuser() was
47   * able to scan through the logical swap device to retrieve this information
48   * however, that capability is no longer supported.
49   */
50  #define u_maxof u_procp->p_maxof        /* max # of open files allowed */
51  #define u_rdir u_procp->p_rdir          /* root directory of current process *
52  #define u_cdir u_procp->p_cdir          /* current directory */
53  #define u_ofilep u_procp->p_ofilep      /* pointers to file descriptor chunks
54                                             to be allocated as needed. */
55
56  /*
```

```
 57   * maxfiles is maximum number of open files per process.
 58   * This is also the "soft limit" for the maximum number of open files per
 59   * process.  maxfiles_lim is the "hard limit" for the maximum number of open
 60   * files per process.
 61   */
 62  extern  int maxfiles;
 63  extern  int maxfiles_lim;
 64
 65  #define LOCK_TRACK_MAX          10              /* for qfs lock tracking */
 66
 67  /*
 68   * Per process structure containing data that
 69   * isn't needed in core when the process is swapped out.
 70   */
 71
 72  #define SHSIZE          32
 73
 74  typedef struct user {
 75  #ifdef __hp9000s800
 76          struct  pcb u_pcb;
 77  #endif
 78          struct  proc *u_procp;          /* pointer to proc structure */
 79  #ifdef __hp9000s800
 80          struct  save_state *u_sstatep;  /* pointer to a saved state */
 81  #endif
 82  #ifdef __hp9000s300
 83          int     *u_ar0;                 /* address of users saved R0 */
 84  #endif /* __hp9000s300 */
 85          char    u_comm[MAXCOMLEN + 1];
 86
 87  /* syscall parameters, results and catches */
 88          int     u_arg[10];              /* arguments to current system call */
 89          int     *u_ap;                  /* pointer to arglist */
 90          label_t u_qsave;                /* for non-local gotos on interrupts *
 91          u_short u_spare_short;          /* Replaces top half of u_error */
 92          u_short u_error;                /* return error code */
 93
 94          union {                         /* syscall return values */
 95                  struct  {
 96                          int     R_val1;
 97                          int     R_val2;
 98                  } u_rv;
 99  #define r_val1  u_rv.R_val1
100  #define r_val2  u_rv.R_val2
101
102  /* Bell-to-Berkeley translations */
103  #define u_rval1         u_r.r_val1
104  #define u_rval2         u_r.r_val2
105
106                  off_t   r_off;
107                  time_t  r_time;
108          } u_r;
109          char    u_eosys;                /* special action on end of syscall */
110          u_short u_syscall;              /* syscall # passed to signal handler
111
112  /* 1.1 - processes and protection */
```

```
113             struct  ucred *u_cred;          /* user credentials (uid, gid, etc) *
114     #define u_uid   u_cred->cr_uid
115     #define u_gid   u_cred->cr_gid
116     #define u_groups u_cred->cr_groups      /* groups, NOGROUP terminated */
117     #define u_ruid  u_cred->cr_ruid
118     #define u_rgid  u_cred->cr_rgid
119             aid_t   u_aid;                  /* audit id */
120             short   u_audproc;              /* audit process flag */
121             short   u_audsusp;              /* audit suspend flag */
122             struct  audit_filename *u_audpath;      /* ptr to audit pathname info
123             struct  audit_string *u_audstr; /* ptr to string data for auditing */
124             struct  audit_sock *u_audsock;  /* ptr to sockaddr data for auditing *
125             char    *u_audxparam;           /* generic loc. to attach audit data *
126     #ifdef __hp9000s800
127             u_int   u_spare1[5];            /* spares for backward compatibility *
128     #endif /* __hp9000s800 */
129     #ifdef _CLASSIC_ID_TYPES
130             unsigned short u_filler_sgid;
131             unsigned short u_sgid;          /* set (effective) gid */
132     #else
133             gid_t   u_sgid;                 /* set (effective) gid */
134     #endif
135             u_int   u_priv[PRIV_MASKSIZ];   /* privlege mask */
136
137     /* 1.2 - memory management */
138             label_t u_ssave;                /* label variable for swapping */
139     #ifdef __hp9000s800
140             tlabel_t u_psave;               /* trap recovery vector - machine dep
141     #endif /* __hp9000s800 */
142     #ifdef __hp9000s300
143             label_t u_rsave;                /* for exchanging stacks */
144             label_t u_psave;                /* for probe simulation */
145     #endif /* __hp9000s300 */
146             time_t  u_outime;               /* user time at last sample */
147             short   u_flag;                 /* See u_flag values */
148     #define UF_MEMSIGL  0x00000001          /* Signal upon memory allocation
149                                              *  and process locked
150
151     /* 1.3 - signal management */
152                     /* same for users and the kernel; see signal.h            */
153             void    (*u_signal[SIGARRAYSIZE])();    /* disposition of signals */
154             int     u_sigmask[SIGARRAYSIZE];        /* signals to be blocked */
155             int     u_sigonstack;           /* signals to take on sigstack */
156             int     u_oldmask;              /* saved mask from before sigpause */
157             int     u_code;                 /* ''code'' to trap */
158             struct  sigstack u_sigstack;    /* sp & on stack state variable */
159     #define u_onstack       u_sigstack.ss_onstack
160     #define u_sigsp         u_sigstack.ss_sp
161     #ifdef __hp9000s800
162             void    (*u_sigreturn)();       /* handler return address */
163     #define PA83_CONTEXT    0x1
164     #define PA89_CONTEXT    0x2
165             int     u_sigcontexttype;       /* to tell PA83 from PA89 contexts */
166     #endif /* __hp9000s800 */
167     #ifdef __hp9000s300
168             int     u_sigcode[6];           /* signal "trampoline" code */
```

```
169  #endif /* __hp9000s300 */
170          int     u_sigreset;            /* reset handler after catching */
171  #ifdef __hp9000s300
172          size_t  u_lockovh;             /* locked proc overhead size (clicks)
173                                         /* belongs with u_locksdsize */
174  #endif /* __hp9000s300 */
175
176  /* 1.4 - descriptor management */
177
178  #define UF_EXCLOSE      0x1            /* auto-close on exec */
179  #define UF_MAPPED       0x2            /* mapped from device */
180          int     u_highestfd;           /* highest file descriptor currently
181                                            opened by this process. */
182  #ifdef _WSIO
183          struct  file *u_fp;            /* current file pointer */
184  #endif /* _WSIO */
185  #define UF_FDLOCK       0x4            /* lockf was done,see vno_lockrelease
186          int     u_spare2[1];           /* spare */
187  #ifdef  HPNSE
188          dev_t   u_ttyd;                /* controlling tty dev */
189  #endif
190          short   u_cmask;               /* mask for file creation */
191
192  /* 1.5 - timing and statistics */
193          /* The user accumulated seconds and system accumulated seconds fields
194           * of the following structure are maintained in the proc structure.
195           * This should be taken into account in computations.
196           */
197          struct  rusage u_ru;           /* stats for this proc */
198          struct  rusage u_cru;          /* sum of stats for reaped children */
199          struct  itimerval u_timer[3];
200          int     u_XXX[2];
201          time_t  u_ticks;
202          short   u_acflag;
203
204  /* 1.6 - resource controls */
205          struct  rlimit u_rlimit[RLIM_NLIMITS];
206
207  /* BEGIN TRASH */
208          char    u_segflg;              /* 0:user D; 1:system; 2:user I */
209          caddr_t u_base;                /* base address for IO */
210          unsigned int u_count;          /* bytes remaining for IO */
211          off_t   u_offset;              /* offset in file for IO */
212
213  #ifdef __hp9000s800
214             /* The magic number, auxillary SOM header and spares */
215             struct{
216                  int       u_magic;
217                  struct som_exec_auxhdr som_aux;
218             } u_exdata;
219  #endif /* __hp9000s800 */
220  #ifdef __hp9000s300
221          union {
222             struct exec  Ux_A;
223             char ux_shell[SHSIZE];     /* #! and name of interpreter */
224          } u_exdata;
```

```
225  #endif /* __hp9000s300 */
226  #ifdef __hp9000s800
227              int     u_spare[9];
228
229  #define ux_mag          u_magic
230  #define ux_tsize        som_aux.exec_tsize
231  #define ux_dsize        som_aux.exec_dsize
232  #define ux_bsize        som_aux.exec_bsize
233  #define ux_entloc       som_aux.exec_entry
234  #define ux_tloc         som_aux.exec_tfile
235  #define ux_dloc         som_aux.exec_dfile
236  #define ux_tmem         som_aux.exec_tmem
237  #define ux_dmem         som_aux.exec_dmem
238  #define ux_flags        som_aux.exec_flags
239  #define Z_EXEC_FLAG     0x1
240  #endif /* __hp9000s800 */
241
242  #ifdef __hp9000s300
243  #define ux_mag          Ux_A.a_magic.file_type
244  #define ux_system_id    Ux_A.a_magic.system_id
245  #define ux_miscinfo     Ux_A.a_miscinfo
246  #define ux_tsize        Ux_A.a_text
247  #define ux_dsize        Ux_A.a_data
248  #define ux_bsize        Ux_A.a_bss
249  #define ux_entloc       Ux_A.a_entry
250  #endif  /* __hp9000s300 */
251
252          caddr_t u_dirp;                 /* pathname pointer */
253  /* END TRASH */
254
255          struct TrHeaderT *u_trptr;      /* QFS transaction header */
256          int     u_lcount;               /* stack size of lock keys */
257          int     u_ldebug;               /* for debug */
258          int     u_lck_keys[LOCK_TRACK_MAX]; /* stack of lock keys */
259
260          dev_t   u_devsused;             /* count of locked devices */
261  #ifdef __hp9000s800
262          u_int   u_spare3[8];            /* spares for backward compatibility *
263          int u_sstep;                    /* process single stepping flags */
264  #define ULINK   0x01f                   /* link register */
265  #define USSTEP  0x020                   /* process is single stepping */
266  #define UPCQM   0x040                   /* pc queue modified */
267  #define UBL     0x080                   /* branch and link at pcq head */
268  #define UBE     0x100                   /* branch external at pcq head */
269          unsigned u_pcsq_head;           /* pc space and offset queue */
270          unsigned u_pcoq_head;           /*   values for single stepping */
271          unsigned u_pcsq_tail;
272          unsigned u_pcoq_tail;
273          unsigned u_ipsw;                /* ipsw for single stepping */
274          int u_gr1;                      /* value for general register 1 */
275          int u_gr2;                      /* value for general register 2 */
276  #endif /* __hp9000s800 */
277          struct uprof {                  /* profile arguments */
278                  short   *pr_base;       /* buffer base */
279                  unsigned pr_size;       /* buffer size */
280                  unsigned pr_off;        /* pc offset */
```

```
281                 unsigned pr_scale;        /* pc scaling */
282         } u_prof;
283 #ifdef __hp9000s800
284         u_int u_kpreemptcnt;              /* kernel preemption counter:  */
285                                           /* read with GETKPREEMPTCNT()  */
286                                           /* clear with CLRKPREEMPTCNT() */
287                                           /* incremented in kpreempt()   */
288 #endif /* __hp9000s800 */
289         dm_message u_request;             /* request message*/
290         struct nsp *u_nsp;                /* nsp performing service*/
291         site_t u_site;                    /* site for which nsp executing */
292         int     u_duxflags;               /* see defines below */
293         char    **u_cntxp;                /* context pointer */
294         struct locklist *u_prelock;       /* preallocated lock for lockadd() */
295
296         struct ki_timeval u_syscall_time; /* system call timestamp */
297         dev_t   u_dev_t;                  /* device location of this process */
298         ino_t   u_inode;                  /* inode number of this process */
299         int     *ki_clk_tos_ptr;
300
301 #define KI_CLK_STACK_SIZE 20
302         int ki_clk_stack[KI_CLK_STACK_SIZE];
303
304         caddr_t u_vapor_mlist;            /* linked list of vapor_malloc mem */
305         int u_ord_blk;                    /* last ordered write block */
306 #ifdef  __hp9000s300
307         struct  pcb     u_pcb;            /* should be last except u_stack */
308 #endif  /* __hp9000s300 */
309
310         union {                           /* double word aligned stack */
311          double s_dummy;
312          int    s_stack[1];
313         } u_s;                            /* must be last thing in user_t */
314 #define u_stack u_s.s_stack
315 } user_t;
316
317 /*
318  * These two defines are moved (logically) from param.h.  Need to have them
319  * here to be able to get at sizeof(user_t)
320  */
321 #ifdef __hp9000s800
322 #define KSTACKBYTES     8192              /* size of kernel stack */
323 #define UPAGES          btorp(sizeof(user_t) + KSTACKBYTES)
324 #endif
325
326 struct ucred {
327 #ifdef _CLASSIC_ID_TYPES
328         unsigned short cr_filler_uid;
329         unsigned short cr_uid;            /* effective user id */
330 #else
331         uid_t cr_uid;                     /* effective user id */
332 #endif
333 #ifdef _CLASSIC_ID_TYPES
334         unsigned short cr_filler_gid;
335         unsigned short cr_gid;            /* effective group id */
336 #else
```

```
337             gid_t cr_gid;                       /* effective group id */
338     #endif
339     #ifdef _CLASSIC_ID_TYPES
340             int         cr_groups[NGROUPS];     /* groups, 0 terminated */
341     #else
342             gid_t       cr_groups[NGROUPS];     /* groups, 0 terminated */
343     #endif
344     #ifdef _CLASSIC_ID_TYPES
345             unsigned short cr_filler_ruid;
346             unsigned short cr_ruid;             /* real user id */
347     #else
348             uid_t   cr_ruid;                    /* real user id */
349     #endif
350     #ifdef _CLASSIC_ID_TYPES
351             unsigned short cr_filler_rgid;
352             unsigned short cr_rgid;             /* real group id */
353     #else
354             gid_t   cr_rgid;                    /* real group id */
355     #endif
356             short   cr_ref;                     /* reference count */
357     };
358
359     #ifdef _KERNEL
360     #define crhold(cr)      {SPINLOCK(cred_lock);(cr)->cr_ref++;SPINUNLOCK(cred_lo
361     struct ucred *crget();
362     struct ucred *crcopy();
363     struct ucred *crdup();
364     #endif /* _KERNEL */
365
366
367     /* u_eosys values */
368     #define EOSYS_NOTSYSCALL        0       /* not in kernel via syscall() */
369     #define EOSYS_NORMAL            1       /* in syscall but nothing notable */
370     #define EOSYS_INTERRUPTED       2       /* signal is not yet fully processed *
371     #define EOSYS_RESTART           3       /* user has requested restart */
372     #define EOSYS_NORESTART         4       /* user has requested error return */
373     #define RESTARTSYS              EOSYS_INTERRUPTED /* temporary!!! */
374
375     /*
376      * defines for u_duxflags
377      */
378     #define DUX_UNSP        4       /* process is a user NSP */
379
380     /* u_error codes */
381     #include <errno.h>              /* Traditional */
382
383     #if defined(__hp9000s800) && defined(_KERNEL)
384     /* WARNING: NEVER, NEVER, NEVER use u as a local variable
385      * name or as a structure element in I/O system or elsewhere in the
386      * kernel.
387      */
388     #define u (*uptr)
389     #define udot (*uptr)
390     #endif /* __hp9000s800 && _KERNEL */
391
```

SE 390: Series 300 HP-UX Internals

I/O Overview

- Memory mapped I/O
- How I/O flows out of the system
  - uses the filesystem
  - uses the major number to go through the bdevsw/cdevsw tables to get to the appropriate driver
  - most of the work is done by the driver
- How I/O flows into the system
  - interrupt comes in from I/O card and is handled by the appropriate driver's interrupt service routine
  - the driver may wake up sleeping processes, send out a new command, or do whatever is appropriate
- Device drivers
  - provide the window to interface to the outside world
  - provide the hardware specific routines
  - provide a common interface to the kernel
- I/O Performance

I/O Overview

How I/O Flows Out of the System

- Background: we create a device file something like this:

  ```
  $ mknod /dev/tty03 c 1 0x0f0204
  ```

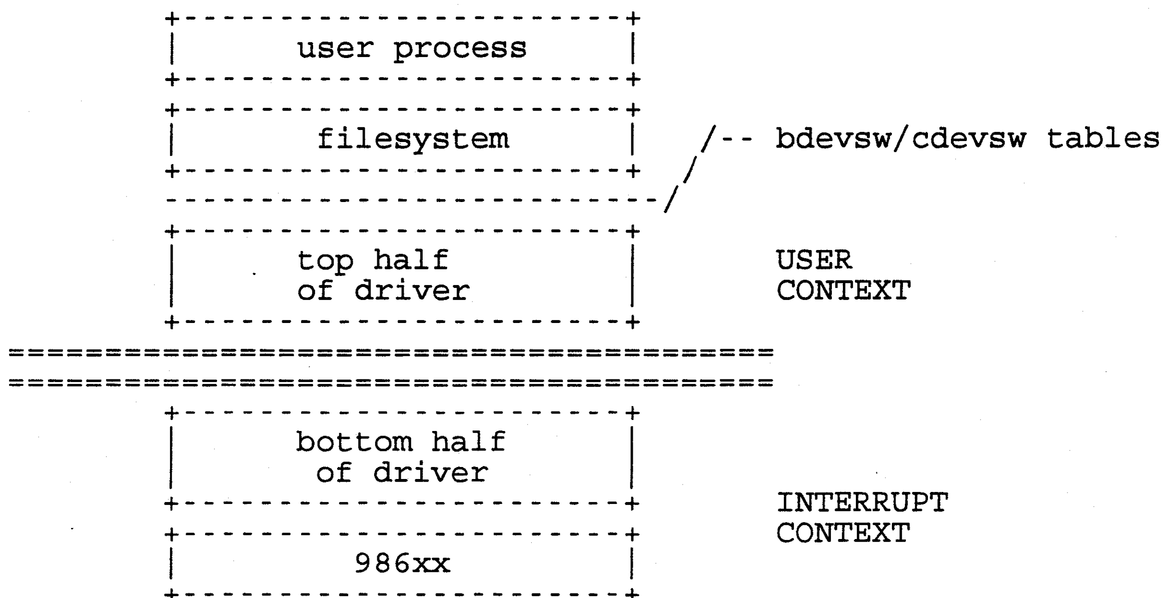
  This creates a special file for port #2 on a mux card, and says that it is hardwired.

- I/O to/from devices is handled using the same semantics as normal files in the file system. Because of this, programs can pretend that devices are just like regular files. However, the filesystem does not know anything about particular devices; it must use the relevant drivers to access them...

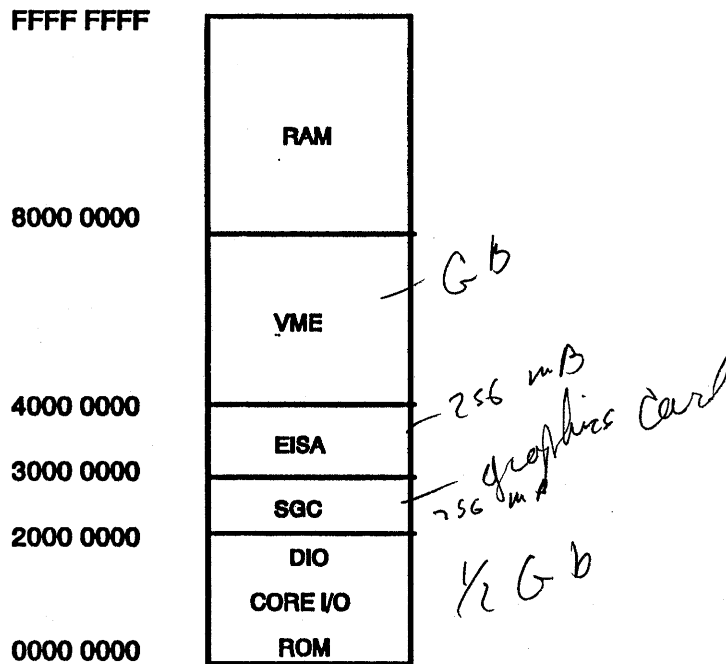
- All I/O starts with accessing the filesystem (during the open). The "open" system call reads the device file's inode and keeps the information for later use. The kernel will look at the major number and type (char vs block) fields in the inode to decide which driver to go through. It will also give the driver a chance to do any necessary device dependent operations (e.g. enable interrupts).

- To get to the right driver, the filesystem will use the type to choose a switch table (bdevsw or cdevsw), and the major number as an index into the chosen table. The operation it is performing (open, read, write, etc) tells it which element of the struct to use once it is there.

I/O Structure Overview



HP 9000 Series 400 and 700 Memory Maps



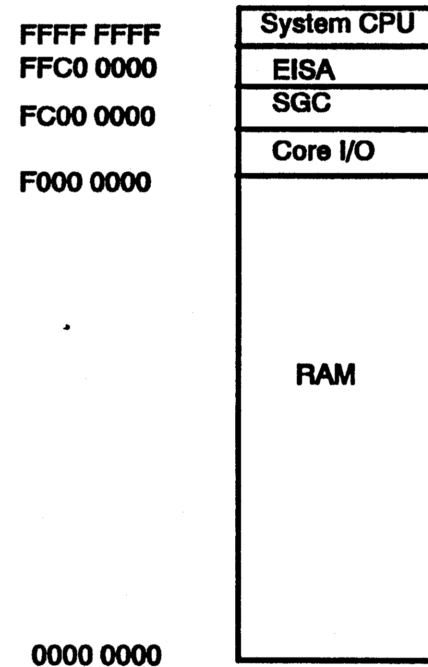


Figure 4. HP 9000 Series 400 and 700 Memory Maps

SE 390: Series 300 HP-UX Internals

I/O Overview

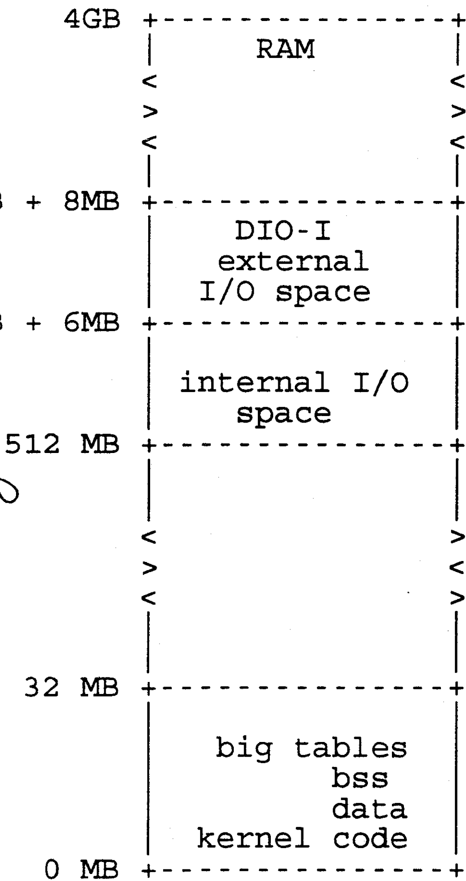
68K I/O Address Space

- The PAS from 0x600000 to 0x800000 is "external I/O space", and is where DIO-I cards are mapped. To figure out where a card will be mapped, multiply its select code by 64K and add that to 0x600000. The 64K starting at that address is available for the card to use.

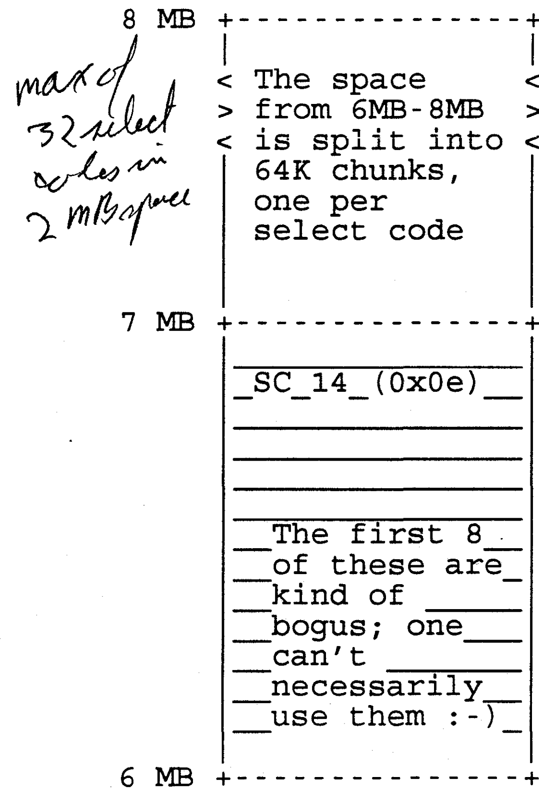
  I/O space is scanned at boot time to see what devices are present. The boot rom does some of this, and prints out the list of cards it finds. The kernel does it again, in preparation for doing I/O later. Essentially all the kernel has to do is try to read from a particular address. If it gets a bus error, that means nothing is there. If it gets some data back, it will try to interpret that and figure out which card is there based on the value returned (the "ID byte" that cards are required to provide).

- When iomap(4) is used, it uses the minor number to calculate the appropriate address, and then calls System V shared memory routines to attach the user process's virtual address space to the space for the card.

Kernel Virtual Address Space



DIO-I External I/O Space

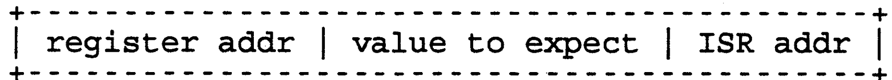


SE 390: Series 300 HP-UX Internals

I/O Overview

68K Interrupt Handling

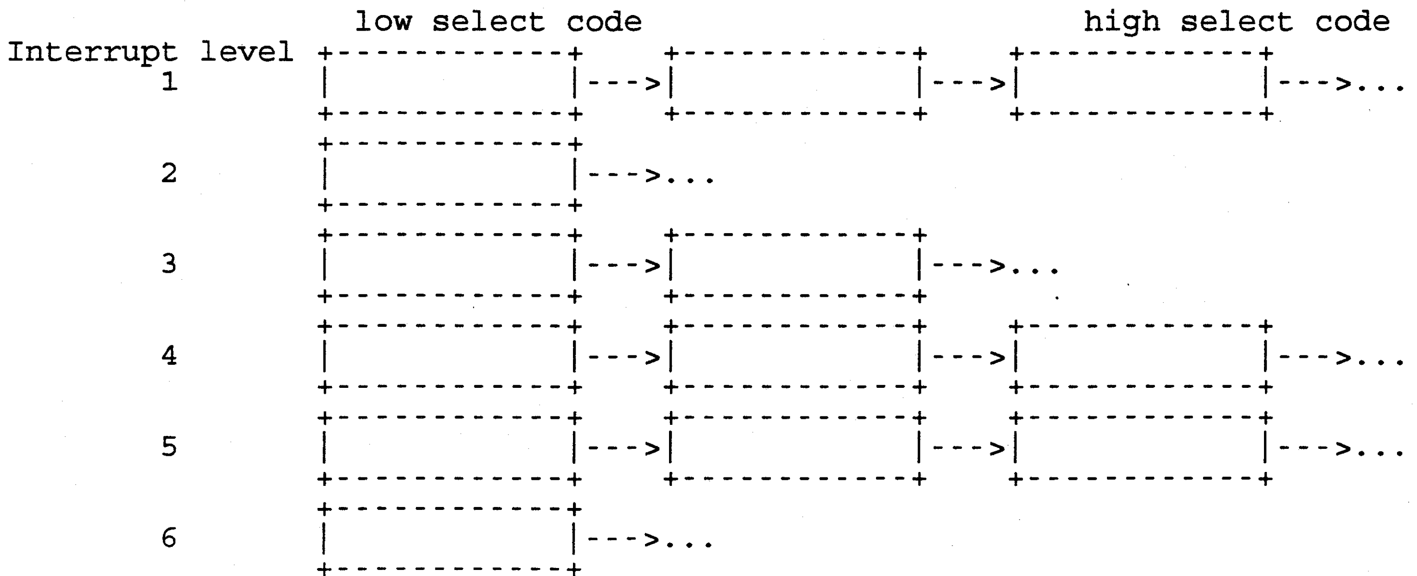
- interrupt comes in from I/O card at card's IL
- IL indexes into _rupttable
- Each entry in _rupttable is the head of a linked list of structures, one per card. They are in increasing order by select code, and look something like this:



look at lev 6 int in locore.s

- The kernel's interrupt handler walks the list, asking each card if it was the one that interrupted. This is done by reading a register on the card and comparing the value with what the driver said would be there if the card interrupted.
- When the right card is identified, its device driver is called to process the interrupt (sending out a new command grabbing the data off of the card, etc).

_rupttable



I/O Overview

700 Specifics

- The top 256MB of physical address space is where PA-RISC thinks I/O space should be. Some interesting pieces of this space:

```
0xf0820000 --> 0xf0ffffff    Core I/O (LAN, SCSI, HIL, etc)

0xf4000000 --> 0xf7ffffff    SGC slot 1
0xf8000000 --> 0xfbffffff    SGC slot 2 (720 uses this one)

0xfc000000 --> 0xffbfffff    EISA
```

- When an interface needs to interrupt, its bit in a dedicated register is set, and the CPU will notice this; note that there is no need to *figure out* who interrupted since each interface has a dedicated bit.

- Devices have no settable "interrupt priorities"; it is up to the software to decide what to service first. Here's the order the software uses as of 8.05:

  bus errors (shouldn't happen)
  EISA
  graphics (doesn't often happen)
  SCSI
  LAN
  parallel
  serial
  HIL (people are slow peripherals :-)

- The cards/adapters tend to have "smart DMA" on them:

  - SCSI uses NCR chip that has a script processor; this maximizes disk throughput and minimizes the need for CPU intervention because the driver can build a whole chain of commands and then point the script processor at them

- The LAN interface has a 128-byte inbound buffer and 64-byte outbound one. Each of the 2 RS232s has a 16-byte buffer for inbound and another for outbound traffic.

- EISA converter is basically a window between EISA cards and the rest of the box

I/O Overview

Types of Drivers

- block mode
    - usually associated with the filesystem, and deals with blocks of data of the same size
    - used with random access devices
    - almost *always* use DMA
    - shields user from hardware details (like disk sector size; a disk doesn't want any requests that aren't a multiple of its hardware sector size)
- character mode
    - usually sequential devices (e.g. printers, tapes)
    - deals with "variable" lengths of data
    - character mode does not mean it deals only with "characters"
    - may use DMA transfers, or may be solely CPU (interrupt) transfers
    - may be *very* similar to block-mode driver (e.g. "raw" and "block" CS80 share about 90% of their code)
- Device drivers don't have to have hardware associated with them; they are a general mechanism for extending the kernel.

How Is A Driver Configured?

- Note: the config(1m) and master(4) manpages are good references.

- /etc/master contains the information on drivers. There are two types of "driver" entry. There is the upper-level (device) drivers (e.g. cs80, tty, etc) and the lower-level (interface or card) drivers (e.g. parallel). Some drivers may combine both, as in the SCSI driver.

- The driver information in /etc/master tells "config" what entries to put in the conf.c file (which will in turn make the linker do most of the work). Here are some lines from /etc/master:

```
* name          handle  type   mask   block   char
*
cs80            cs80    3      3FB    0       4
tape            tp      1      FA     -1      5
ramdisc         ram     3      FB     4       20
98624           ti9914  10     100    -1      -1
98625           simon   10     100    -1      -1
98628           sio628  10     100    -1      -1
98642           sio642  10     100    -1      -1
*
tty             sy      D      FD     -1      2
```

- A description of the fields are:

```
name - the name used in the "dfile" for this driver
handle - the "handle" actually used in the kernel (e.g. the
         tty driver's open routine is sy_open)
type - 5-bit attribute flag indicating "type" of driver:
      4 3 2 1 0
      | | | | \- character device
      | | | \--- block device
      | | \----- required driver
      | \------- specified only once
      \--------- card
mask - 10-bit driver routine flag; tells config what routines to
      include in conf.c for the driver
      9 8 7 6 5 4 3 2 1 0
      | | | | | | | | | \- C_ALLCLOSES flag
      | | | | | | | | \--- seltrue handler (select is always TRUE)
      | | | | | | | \----- select handler
      | | | | | | \------- ioctl handler
      | | | | | \--------- write handler
      | | | | \----------- read handler
      | | | \------------- close handler
      | | \--------------- open handler
      | \----------------- link routine (links interrupt handler;
      |                         found in all interface drivers)
      \------------------- size handler (in disc-type drivers)
block - major number for block device driver
char - major number for character device driver
```

The major (or driver) number indicates the array offset for the routine entries in a device switch table.

Examples from conf.c for the routines "brought in" by the "type" and "mask" values above are as follows:

```
   extern cs80_open(), cs80_close(), cs80_read(), cs80_write(),
       cs80_ioctl(), cs80_size(), cs80_link(), cs80_strategy();
   extern sy_open(), sy_close(), sy_read(), sy_write(), sy_ioctl()
       sy_select();
   extern ti9914_link();
```

Following are exerpts from the bdev/cdev switch tables. It is via these two tables that the proper subroutine calls are made for the appropriate driver. By modifying /etc/master's driver numbers, you can change the "major" numbers :-)

```
  struct bdevsw bdevsw[] = {
  /* 0*/ cs80_open, cs80_close, cs80_strategy, cs80_size, C_ALLCLO
  /* 1*/ nodev, nodev, nodev, nodev, 0,
                 :
                 :
                 :
  };

  struct cdevsw cdevsw[] = {
                 :
                 :
  /* 2*/ sy_open, sy_close, sy_read, sy_write, sy_ioctl, sy_select
         C_ALLCLOSES,
                 :
  /* 4*/ cs80_open, cs80_close, cs80_read, cs80_write, cs80_ioctl,
         seltrue, C_ALLCLOSES,
                 :
                 :
  /*43*/ nodev, nodev, nodev, nodev, nodev, nodev, 0,
                 :
  };
```

This structure is used during the startup to allow for linking of "make_entry" routines for the drivers.

The make_entry() routine for each driver is called during startup of the system. For each card found during bootup, the kernel calls the make_entry routine. These routines check to see if the card is theirs. If so, it may perform some initialization and it reports finding the card. If not, the make_entry() routine will call the next driver's make_entry(). There is always a dummy routine at the end of the list that will report no driver found for the card.

```
      int      (*driver_link[])() =
      {
               cs80_link,
               amigo_link,
               scsi_link,
               graphics_link,
               ptys_link,
                   :
                   :
               sio628_link,
               sio642_link,
               ite200_link,
               (int (*)())0
      };
```

```
 1  *dskless
 2  nipc
 3  netman
 4  ni
 5  inet
 6  lla
 7  lan01
 8  cs80
 9  scsi
10  scsitape
11  tape
12  stape
13  printer
14  ptymas
15  ptyslv
16  hpib
17  98624
18  98625
19  98626
20  98628
21  98642
22  uipc
23  nbuf 1024
24  nproc 256
25  ninode 1000
26  nfile 1000
27  swap auto
28  swap scsi f0500 -1
```

10

```
 1  /*
 2   *  Configuration information
 3   */
 4
 5
 6  #define MAXUSERS        8
 7  #define TIMEZONE        420
 8  #define DST     1
 9  #define NPROC   256
10  #define NUM_CNODES      ((5*SERVER_NODE)+DSKLESS_NODE)
11  #define DSKLESS_NODE    0
12  #define SERVER_NODE     0
13  #define NINODE  1000
14  #define NFILE   1000
15  #define FILE_PAD        10
16  #define MAXFILES        60
17  #define MAXFILES_LIM    1024
18  #define NBUF    1024
19  #define FS_ASYNC        0
20  #define DOS_MEM_BYTE    0
21  #define NCALLOUT        (16+NPROC+USING_ARRAY_SIZE+SERVING_ARRAY_SIZE)
22  #define UNLOCKABLE_MEM  102400
23  #define NFLOCKS 200
24  #define NPTY    82
25  #define MAXUPRC 50
26  #define MAXDSIZ 0x01000000
27  #define MAXSSIZ 0x00200000
28  #define MAXTSIZ 0x01000000
29  #define PARITY_OPTION   2
30  #define REBOOT_OPTION   1
31  #define TIMESLICE       0
32  #define ACCTSUSPEND     2
33  #define ACCTRESUME      4
34  #define NDILBUFFERS     30
35  #define FILESIZELIMIT   0x1fffffff
36  #define USING_ARRAY_SIZE        (NPROC)
37  #define SERVING_ARRAY_SIZE      (SERVER_NODE*NUM_CNODES*MAXUSERS+2*MAXUSERS)
38  #define DSKLESS_FSBUFS  (SERVING_ARRAY_SIZE)
39  #define SELFTEST_PERIOD 120
40  #define INDIRECT_PTES   1
41  int     indirect_ptes = INDIRECT_PTES;
42  #define CHECK_ALIVE_PERIOD      4
43  #define RETRY_ALIVE_PERIOD      21
44  #define MAXSWAPCHUNKS   512
45  #define MINSWAPCHUNKS   4
46  #define NSWAPDEV        10
47  #define NSWAPFS 10
48  #define NUM_LAN_CARDS   2
49  #define NETISR_PRIORITY -1
50  #define NGCSP   (8*NUM_CNODES)
51  #define NNI     1
52  #define SCROLL_LINES    100
53  #define NUM_PDN0        -1
54  #define MESG    1
55  #define MSGMAP  (MSGTQL+2)
56  #define MSGMAX  8192
```

```
 57  #define MSGMNB   16384
 58  #define MSGMNI   50
 59  #define MSGSSZ   1
 60  #define MSGTQL   40
 61  #define MSGSEG   16384
 62  #define SEMA     1
 63  #define SEMMAP   (SEMMNI+2)
 64  #define SEMMNI   64
 65  #define SEMMNS   128
 66  #define SEMMNU   30
 67  #define SEMUME   10
 68  #define SEMVMX   32767
 69  #define SEMAEM   16384
 70  #define SHMEM    1
 71  #define SHMMAX   0x00600000
 72  #define SHMMIN   1
 73  #define SHMMNI   30
 74  #define SHMSEG   10
 75  #define FPA      1
 76  #define SWAPMEM_ON       0
 77  #define SWCHUNK 2048
 78
 79  #define UIPC
 80  #define UIPC
 81  #define NIPC
 82  #define INET
 83  #define INET
 84  #define NI
 85  #define LAN01
 86
 87  #include         "/etc/conf/h/param.h"
 88  #include         "/etc/conf/h/systm.h"
 89  #include         "/etc/conf/h/tty.h"
 90  #include         "/etc/conf/h/space.h"
 91  #include         "/etc/conf/h/opt.h"
 92  #include         "/etc/conf/h/conf.h"
 93
 94  #define ieee802_open     lan_open
 95  #define ieee802_close    lan_close
 96  #define ieee802_read     lan_read
 97  #define ieee802_write    lan_write
 98  #define ieee802_link     lan_link
 99  #define ieee802_select   lan_select
100  #define ethernet_open    lan_open
101  #define ethernet_close   lan_close
102  #define ethernet_read    lan_read
103  #define ethernet_write   lan_write
104  #define ethernet_link    lan_link
105  #define ethernet_select  lan_select
106  #define hpib_link        gpio_link
107  #define lla_link         lan_link
108  #define lan01_link       lan_link
109
110  extern nodev(), nulldev();
111  extern seltrue(), notty();
112
```

```
113  extern cs80_open(), cs80_close(), cs80_read(), cs80_write(), cs80_ioctl(), cs8
114  extern swap_strategy();
115  extern swap1_strategy();
116  extern scsi_open(), scsi_close(), scsi_read(), scsi_write(), scsi_ioctl(), scs
117  extern cons_open(), cons_close(), cons_read(), cons_write(), cons_ioctl(), con
118  extern tty_open(), tty_close(), tty_read(), tty_write(), tty_ioctl(), tty_sele
119  extern sy_open(), sy_close(), sy_read(), sy_write(), sy_ioctl(), sy_select();
120  extern mm_read(), mm_write();
121  extern tp_open(), tp_close(), tp_read(), tp_write(), tp_ioctl();
122  extern lp_open(), lp_close(), lp_write(), lp_ioctl();
123  extern swap_read(), swap_write();
124  extern stp_open(), stp_close(), stp_read(), stp_write(), stp_ioctl();
125  extern iomap_open(), iomap_close(), iomap_read(), iomap_write(), iomap_ioctl()
126  extern graphics_open(), graphics_close(), graphics_ioctl(), graphics_link();
127  extern ptym_open(), ptym_close(), ptym_read(), ptym_write(), ptym_ioctl(), pty
128  extern ptys_open(), ptys_close(), ptys_read(), ptys_write(), ptys_ioctl(), pty
129  extern lla_open(), lla_link();
130  extern lla_open();
131  extern hpib_open(), hpib_close(), hpib_read(), hpib_write(), hpib_ioctl();
132  extern r8042_open(), r8042_close(), r8042_ioctl();
133  extern hil_open(), hil_close(), hil_read(), hil_ioctl(), hil_select(), hil_lin
134  extern nimitz_open(), nimitz_close(), nimitz_read(), nimitz_select();
135  extern scsitape_open(), scsitape_close(), scsitape_read(), scsitape_write(), s
136  extern ni_open(), ni_close(), ni_read(), ni_write(), ni_ioctl(), ni_select(),
137  extern audio_open(), audio_close(), audio_read(), audio_write(), audio_ioctl()
138  extern nm_open(), nm_close(), nm_read(), nm_ioctl(), nm_select();
139
140  extern nipc_link();
141  extern inet_link();
142  extern uipc_link();
143  extern scsi_if_link();
144  extern ti9914_link();
145  extern simon_link();
146  extern sio626_link();
147  extern sio628_link();
148  extern sio642_link();
149  extern ite200_link();
150
151  struct bdevsw bdevsw[] = {
152  /* 0*/  {cs80_open, cs80_close, cs80_strategy, cs80_dump, cs80_size, C_ALLCLOS
153  /* 1*/  {nodev, nodev, nodev, nodev, nodev, 0, nodev},
154  /* 2*/  {nodev, nodev, nodev, nodev, nodev, 0, nodev},
155  /* 3*/  {nodev, nodev, swap_strategy, nodev, 0, 0, nodev},
156  /* 4*/  {nodev, nodev, nodev, nodev, nodev, 0, nodev},
157  /* 5*/  {nodev, nodev, swap1_strategy, nodev, 0, 0, nodev},
158  /* 6*/  {nodev, nodev, nodev, nodev, nodev, 0, nodev},
159  /* 7*/  {scsi_open, scsi_close, scsi_strategy, scsi_dump, scsi_size, C_ALLCLOS
160  };
161
162  struct cdevsw cdevsw[] = {
163  /* 0*/  {cons_open, cons_close, cons_read, cons_write, cons_ioctl, cons_select
164  /* 1*/  {tty_open, tty_close, tty_read, tty_write, tty_ioctl, tty_select, C_AL
165  /* 2*/  {sy_open, sy_close, sy_read, sy_write, sy_ioctl, sy_select, C_ALLCLOSE
166  /* 3*/  {nulldev,  nulldev, mm_read, mm_write, notty, seltrue, 0},
167  /* 4*/  {cs80_open, cs80_close, cs80_read, cs80_write, cs80_ioctl, seltrue, C_
168  /* 5*/  {tp_open, tp_close, tp_read, tp_write, tp_ioctl, seltrue, 0},
```

```
169  /* 6*/  {nodev, nodev, nodev, nodev, nodev, nodev, 0},
170  /* 7*/  {lp_open, lp_close, nodev, lp_write, lp_ioctl, seltrue, 0},
171  /* 8*/  {nulldev,  nulldev, swap_read, swap_write, notty, nodev, 0},
172  /* 9*/  {stp_open, stp_close, stp_read, stp_write, stp_ioctl, seltrue, 0},
173  /*10*/  {iomap_open, iomap_close, iomap_read, iomap_write, iomap_ioctl, nodev,
174  /*11*/  {nodev, nodev, nodev, nodev, nodev, nodev, 0},
175  /*12*/  {graphics_open, graphics_close, nodev, nodev, graphics_ioctl, nodev, C
176  /*13*/  {nodev, nodev, nodev, nodev, nodev, nodev, 0},
177  /*14*/  {nodev, nodev, nodev, nodev, nodev, nodev, 0},
178  /*15*/  {nodev, nodev, nodev, nodev, nodev, nodev, 0},
179  /*16*/  {ptym_open, ptym_close, ptym_read, ptym_write, ptym_ioctl, ptym_select
180  /*17*/  {ptys_open, ptys_close, ptys_read, ptys_write, ptys_ioctl, ptys_select
181  /*18*/  {lla_open, nulldev, nodev, nodev, notty, nodev, C_ALLCLOSES},
182  /*19*/  {lla_open, nulldev, nodev, nodev, notty, nodev, C_ALLCLOSES},
183  /*20*/  {nodev, nodev, nodev, nodev, nodev, nodev, 0},
184  /*21*/  {hpib_open, hpib_close, hpib_read, hpib_write, hpib_ioctl, seltrue, C_
185  /*22*/  {nodev, nodev, nodev, nodev, nodev, nodev, 0},
186  /*23*/  {r8042_open, r8042_close, nodev, nodev, r8042_ioctl, nodev, 0},
187  /*24*/  {hil_open, hil_close, hil_read, nodev, hil_ioctl, hil_select, 0},
188  /*25*/  {nimitz_open, nimitz_close, nimitz_read, nodev, notty, nimitz_select,
189  /*26*/  {nodev, nodev, nodev, nodev, nodev, nodev, 0},
190  /*27*/  {nodev, nodev, nodev, nodev, nodev, nodev, 0},
191  /*28*/  {nodev, nodev, nodev, nodev, nodev, nodev, 0},
192  /*29*/  {nodev, nodev, nodev, nodev, nodev, nodev, 0},
193  /*30*/  {nodev, nodev, nodev, nodev, nodev, nodev, 0},
194  /*31*/  {nodev, nodev, nodev, nodev, nodev, nodev, 0},
195  /*32*/  {nodev, nodev, nodev, nodev, nodev, nodev, 0},
196  /*33*/  {nodev, nodev, nodev, nodev, nodev, nodev, 0},
197  /*34*/  {nodev, nodev, nodev, nodev, nodev, nodev, 0},
198  /*35*/  {nodev, nodev, nodev, nodev, nodev, nodev, 0},
199  /*36*/  {nodev, nodev, nodev, nodev, nodev, nodev, 0},
200  /*37*/  {nodev, nodev, nodev, nodev, nodev, nodev, 0},
201  /*38*/  {nodev, nodev, nodev, nodev, nodev, nodev, 0},
202  /*39*/  {nodev, nodev, nodev, nodev, nodev, nodev, 0},
203  /*40*/  {nodev, nodev, nodev, nodev, nodev, nodev, 0},
204  /*41*/  {nodev, nodev, nodev, nodev, nodev, nodev, 0},
205  /*42*/  {nodev, nodev, nodev, nodev, nodev, nodev, 0},
206  /*43*/  {nodev, nodev, nodev, nodev, nodev, nodev, 0},
207  /*44*/  {nodev, nodev, nodev, nodev, nodev, nodev, 0},
208  /*45*/  {nodev, nodev, nodev, nodev, nodev, nodev, 0},
209  /*46*/  {nodev, nodev, nodev, nodev, nodev, nodev, 0},
210  /*47*/  {scsi_open, scsi_close, scsi_read, scsi_write, scsi_ioctl, seltrue, C_
211  /*48*/  {nodev, nodev, nodev, nodev, nodev, nodev, 0},
212  /*49*/  {nodev, nodev, nodev, nodev, nodev, nodev, 0},
213  /*50*/  {nodev, nodev, nodev, nodev, nodev, nodev, 0},
214  /*51*/  {nodev, nodev, nodev, nodev, nodev, nodev, 0},
215  /*52*/  {nodev, nodev, nodev, nodev, nodev, nodev, 0},
216  /*53*/  {nodev, nodev, nodev, nodev, nodev, nodev, 0},
217  /*54*/  {scsitape_open, scsitape_close, scsitape_read, scsitape_write, scsitap
218  /*55*/  {nodev, nodev, nodev, nodev, nodev, nodev, 0},
219  /*56*/  {ni_open, ni_close, ni_read, ni_write, ni_ioctl, ni_select, 0},
220  /*57*/  {audio_open, audio_close, audio_read, audio_write, audio_ioctl, audio_
221  /*58*/  {nodev, nodev, nodev, nodev, nodev, nodev, 0},
222  /*59*/  {nodev, nodev, nodev, nodev, nodev, nodev, 0},
223  /*60*/  {nm_open, nm_close, nm_read, nodev, nm_ioctl, nm_select, 0},
224  };
```

```
225
226  int      nblkdev = sizeof (bdevsw) / sizeof (bdevsw[0]);
227  int      nchrdev = sizeof (cdevsw) / sizeof (cdevsw[0]);
228
229  dev_t    rootdev = makedev(-1,0xFFFFFF);
230
231  /* The following three variables are dependent upon bdevsw and cdevsw. If
232     either changes then these variables must be checked for correctness */
233
234  dev_t    swapdev1 = makedev(5, 0x000000);
235  int      brmtdev = 6;
236  int      crmtdev = 45;
237
238  struct swdevt swdevt[] = {
239           { SWDEF, 0,  -1,  0 },
240           { makedev(7, 0x0f0500), 0,  -1,  0 },
241           { NODEV, 0, 0, 0 },
242           { NODEV, 0, 0, 0 },
243           { NODEV, 0, 0, 0 },
244           { NODEV, 0, 0, 0 },
245           { NODEV, 0, 0, 0 },
246           { NODEV, 0, 0, 0 },
247           { NODEV, 0, 0, 0 },
248           { NODEV, 0, 0, 0 },
249  };
250
251  dev_t    dumpdev = makedev(-1,0xFFFFFF);
252
253  int      (*driver_link[])() =
254  {
255           cs80_link,
256           scsi_link,
257           graphics_link,
258           ptys_link,
259           lla_link,
260           hil_link,
261           ni_link,
262           audio_link,
263           nipc_link,
264           inet_link,
265           uipc_link,
266           scsi_if_link,
267           ti9914_link,
268           simon_link,
269           sio626_link,
270           sio628_link,
271           sio642_link,
272           ite200_link,
273           (int (*)())0
274  };
275  char dfile_data[] = "\
276  nipc\n\
277  netman\n\
278  ni\n\
279  inet\n\
280  lla\n\
```

```
281  lan01\n\
282  cs80\n\
283  scsi\n\
284  scsitape\n\
285  tape\n\
286  stape\n\
287  printer\n\
288  ptymas\n\
289  ptyslv\n\
290  hpib\n\
291  98624\n\
292  98625\n\
293  98626\n\
294  98628\n\
295  98642\n\
296  uipc\n\
297  nbuf 1024\n\
298  nproc 256\n\
299  ninode 1000\n\
300  nfile 1000\n\
301  swap auto\n\
302  swap scsi f0500 -1\n\
303  ";
```

```
 1  ##
 2  ##  HP-UX System Makefile
 3  ##
 4
 5  # .SILENT
 6  STDDEFS=-Dhp9000s200 -D__hp9000s200 -D__hp9000s300 -Dhpux -D_HPUX_SO
 7  IDENT=-D_KERNEL -DKERNEL -Uvax -DHFS -DMC68030 -DPSTAT -DSAVECORE_30
 8          -DREGION -DKVM -DGENESIS -DAUTOCHANGER -DEISA -DWRITE_GUARD_
 9  REALTIME = -DRTPRIO -DPROCESSLOCK -DEISA
10
11  CC = /bin/cc
12  AS = /bin/as
13  LD = /bin/ld
14  SHELL = /bin/sh
15  ROOT = /etc/conf
16
17  LIBS = \
18          $(ROOT)/libuipc.a \
19          $(ROOT)/libnipc.a \
20          $(ROOT)/liblan.a \
21          $(ROOT)/libinet.a \
22          $(ROOT)/libnet.a \
23          $(ROOT)/libkreq.a \
24          $(ROOT)/libdreq.a \
25          $(ROOT)/libpm.a \
26          $(ROOT)/libvm.a \
27          $(ROOT)/libsysV.a \
28          $(ROOT)/libmin.a \
29          $(ROOT)/libdevelop.a \
30          $(ROOT)/libdil_srm.a \
31          $(ROOT)/libkern.a \
32          $(ROOT)/libk.a
33
34  CFLAGS= +M -Wc,-Nd3500,-Ns7000 -Wp,-H250000 -I.
35  COPTS= $(STDDEFS) $(IDENT) $(REALTIME)
36  KREQ1_OBJS= exceptions.o locore.o vers.o
37  KREQ2_OBJS= name.o funcentry.o cdfs_hooks.o
38  DEBUG_OBJS= debug.nms.o
39
40  all:    hp-ux
41
42  hp-ux:  conf.o
43          rm -f hp-ux
44          ar x $(ROOT)/libkreq.a $(KREQ1_OBJS) $(KREQ2_OBJS)
45          @echo 'Loading hp-ux...'
46          $(LD) -n -o hp-ux -e _start -x  \
47                  $(KREQ1_OBJS) conf.o $(KREQ2_OBJS) $(LIBS)
48          rm -f $(KREQ1_OBJS) $(KREQ2_OBJS)
49          chmod 755 hp-ux
50
51  conf.o: conf.c
52          rm -f conf.o
53          @echo 'Compiling conf.c ...'
54          $(CC) $(CFLAGS) $(COPTS) -c conf.c
```

I/O Overview

I/O Performance

- DMA

  - 300 and 400 each have two DMA channels

  - 700 has a DMA channel for most any interface that needs it

  - As of 9.0, the 700 will schedule I/O based on 3 things:
    - how long the request has been waiting
    - disk latency (seek, rotational delay, etc)
    - priority of the requesting process

- Measurement

  - use iostat(1); if it just won't do the job, you can monitor the structures it uses:

    - tk_nin, tk_nout count characters going in and out of the system via ttys

    - dk_*[] arrays - for each of 8 devices,

      dk_time[i] tells how much time this drive has been active

      dk_seek[i] tells how many seeks this drive has done

      dk_xfer[i] tells how many data transfers this drive has done

      dk_wds[i] tells how many 64-byte "words" this drive has read/written

      dk_mspw[i] tells how many milliseconds per "word" it has taken

      there is a bit in dk_busy indicating whether this drive is doing something at the moment

RAMdisk Open

An open routine typically performs some driver specific operations. It may be a driver that supports exclusive open (only one open at a time), so returns an error for any additional opens. It may allocate buffer space (if not already allocated). Also, it may perform card reset (e.g. the gpio card).

The RAM driver will allocate memory if it is the first open (that is, there is presently no memory allocated for it). The open also ensures the requested device is in the range (and size) of the driver. The information on the device (drive number and size) is packed into the minor number. The macros in ram.h are written to pull out the pertinent information. The kernel provides similar type macros for extracting major, minor, selcode, volume, & unit numbers from the "dev" value passed to the driver. The major and minor number are packed into the 32 bit value, with 8 bits for major number and 24 bits for the minor number.

```
/* max ram volumes cannot exceed 16 */
#define RAM_MAXVOLS 16

/* io mapping minor number macros */
/* up to 1048575 - 256 byte sectors */
#define RAM_SIZE(x)      ((x) & 0xfffff)         /* XXX */

/* up 16 disc allowed */
#define RAM_DISC(x)      (((x) >> 20) & 0xf)     /* XXX */
#define RAM_MINOR(x)     ((x) & 0xffffff)        /* XXX */

#define LOG2SECSIZE 8   /*  log2 of the "sector" size (256 bytes)   */

struct ram_descriptor {
        char    *addr;          /* "disc space" in RAM  */
        int     size;           /* size of RAM disc     */
        short   opencount;      /* number of opens      */
        short   flag;
        int     rd1k;           /* Stats for 1k reads   */
        int     rd2k;           /* Stats for 2k reads   */
        int     rd3k;           /* Stats for 3k reads   */
        int     rd4k;           /* Stats for 4k reads   */
        int     rd5k;           /* Stats for 5k reads   */
        int     rd6k;           /* Stats for 6k reads   */
        int     rd7k;           /* Stats for 7k reads   */
        int     rd8k;           /* Stats for 8k reads   */
        int     rdother;        /* Stats for other reads */
        int     wt1k;           /* Stats for 1k writes  */
        int     wt2k;           /* Stats for 2k writes  */
        int     wt3k;           /* Stats for 3k writes  */
        int     wt4k;           /* Stats for 4k writes  */
        int     wt5k;           /* Stats for 5k writes  */
        int     wt6k;           /* Stats for 6k writes  */
        int     wt7k;           /* Stats for 7k writes  */
        int     wt8k;           /* Stats for 8k writes  */
        int     wtother;        /* Stats for other writes */
} ram_device[RAM_MAXVOLS];
```

```
62    /*
63    **  Open the ram device.
64    */
65    ram_open(dev, flag)
66    dev_t dev;
67    int flag;
68    {
69            register unsigned long size;
70            register struct ram_descriptor *ram_des_ptr;
71
72            /* check if this is status open */
73            if (RAM_MINOR(dev) == 0)
74                    return(0);
75
76            /* check if this device is greater than max number of volumes */
77            if ((size = RAM_DISC(dev)) > RAM_MAXVOLS)
78                    return(EINVAL);
79
80            ram_des_ptr = &ram_device[size];
81
82            /* check the size of the ram disc less than 16 sectors */
83            if ((size = RAM_SIZE(dev)) < 16)
84                    return(EINVAL);
85
86            /* check if already allocated */
87            if (ram_des_ptr->addr != NULL) {
88
89                    /* then check if size changed; must be the same size */
90                    if (ram_des_ptr->size != size)
91                            return(EINVAL);
92
93                    /* bump open count */
94                    ram_des_ptr->opencount++;
95            } else {
96                    /* allocate the memory for the ram disc */
97                    if ((ram_des_ptr->addr =
98                            (char *)sys_memall(size<<LOG2SECSIZE)) == NULL) {
99                            return(ENOMEM);
100                   }
101                   /* save size in 256 byte "sectors" */
102                   ram_des_ptr->size = size;
103
104                   /* open count should be zero */
105                   if (ram_des_ptr->opencount++) {
106                           panic("ram_open count wrong\n");
107                   }
108           }
109           return(0);
110   }
```

RAMdisk Read/Write routines

This is a "typical" read & write routine for drivers that have a block driver as well, or that will use a common read/write "strategy" routine and buffer headers. The physio() routine will take the information from the uio and dev variables and construct a buf structure that contains the information necessary for the strategy routine to perform the I/O. Physio() will break up the transfers into small enough transfers for the strategy routine to handle. The parameters to physio() are:

strategy	address of the strategy() routine physio will call
bp	pointer to a buf structure for physio to use; if NULL, physio will get one from the buffer cache
dev	the packed device info obtained when device opened
rw	either B_READ or B_WRITE, indicating transfer type
mincnt	address of mincnt() routine, a routine that determines the max transfer size (usually the kernel-provided minphys() (xfer size = 64k)
uio	uio structure containing info about the user and the I/O request (size & direction of transfer, pointers to user's buffers for the I/O, etc.)

In the RAM disk driver, the read & write routines have the physio() routine request a buf structure from the file system's buffers. It uses the kernel's minphys() routine, so strategy will break up the transfers to a maximum of 64k transfers.

```
ram_read(dev, uio)
dev_t dev;
struct uio *uio;
{
        return physio(ram_strategy, NULL, dev, B_READ, minphys, uio);
}


ram_write(dev, uio)
dev_t dev;
struct uio *uio;
{
        return physio(ram_strategy, NULL, dev, B_WRITE, minphys, uio);
}
```

```
1
2                        RAMdisk Strategy
3
4
5    This routine will actually perform the "I/O" to the RAM disc.  The buf
6    structure passed to the strategy routine contains the necessary
7    information for the transfer.  This info is filled in by kernel
8    routines; in the case of a character device, physio() does this, and
9    for block devices, the filesystem takes care of filling in the data.
10
11
12
13   ram_strategy(bp)
14   register struct buf *bp;
15   {
16           register block_d7;
17           register char *addr;
18           register struct ram_descriptor *ram_des_ptr;
19
20           /* if this is a status request, return ram_device structure */
21           if (RAM_MINOR(bp->b_dev) == 0) {
22                   if ((bp->b_flags & B_PHYS) &&    /*  must be char dev  */
23                           (bp->b_flags & B_READ) &&
24                           (bp->b_bcount == sizeof(ram_device))) {
25                           bp->b_resid = bp->b_bcount;
26
27                           /* return the "ram_device" structure */
28                           bcopy(&ram_device[0], bp->b_un.b_addr,
29                                   sizeof(ram_device));
30                   } else {
31                           bp->b_error = EIO;
32                           bp->b_flags = B_ERROR;
33                   }
34                   goto done;
35           }
36           /* do the normal reads and writes to ram disc */
37           ram_des_ptr = &ram_device[RAM_DISC(bp->b_dev)];
38
39           /* sanity check if we got the memory */
40           if ((addr = ram_des_ptr->addr) == NULL) {
41                   panic("no memory in ram_strategy\n");
42           }
43           /* make sure the request is within the size of the "disk" */
44           if (bpcheck(bp, ram_des_ptr->size, LOG2SECSIZE, 0))
45                   return;
46
47           /* calculate address to do the transfer */
48           addr += bp->b_un2.b_sectno<<LOG2SECSIZE;
49
50           /* for debugging file system only */
51           block_d7 = bp->b_un2.b_sectno>>2;
```

```
53            if (bp->b_flags & B_READ) {
54                    bcopy(addr, bp->b_un.b_addr, bp->b_bcount);
55                    switch (bp->b_bcount/1024) {
56                    case 1: ram_des_ptr->rd1k++;
57                            break;
58                    case 2: ram_des_ptr->rd2k++;
59                            break;
60                    case 3: ram_des_ptr->rd3k++;
61                            break;
62                    case 4: ram_des_ptr->rd4k++;
63                            break;
64                    case 5: ram_des_ptr->rd5k++;
65                            break;
66                    case 6: ram_des_ptr->rd6k++;
67                            break;
68                    case 7: ram_des_ptr->rd7k++;
69                            break;
70                    case 8: ram_des_ptr->rd8k++;
71                            break;
72                    default: ram_des_ptr->rdother++;
73                    }
74            } else { /* WRITE */
75                    bcopy(bp->b_un.b_addr, addr, bp->b_bcount);
76                    switch (bp->b_bcount/1024) {
77                    case 1: ram_des_ptr->wt1k++;
78                            break;
79                    case 2: ram_des_ptr->wt2k++;
80                            break;
81                    case 3: ram_des_ptr->wt3k++;
82                            break;
83                    case 4: ram_des_ptr->wt4k++;
84                            break;
85                    case 5: ram_des_ptr->wt5k++;
86                            break;
87                    case 6: ram_des_ptr->wt6k++;
88                            break;
89                    case 7: ram_des_ptr->wt7k++;
90                            break;
91                    case 8: ram_des_ptr->wt8k++;
92                            break;
93                    default: ram_des_ptr->wtother++;
94                    }
95            }
96    done:
97            bp->b_resid -= bp->b_bcount;
98            biodone(bp);
99    }
```

```
1
2                                    RAMdisk Ioctl
3
4
5    The ioctl routine:
6            executed via ioctl(2);
7            purpose:
8               handles commands passed to it via ioctl
9            implement the various ioctls by including statements of the
10           following form:
11              #define CMD task(t, n, arg)
12           where:
13              CMD  command name
14              t    arbitrary letter
15              n    sequential number (unique for each ioctl define for a
16                   given ioctl routine)
17              arg  optional arg for command
18           "task" (a macro defined in sys/ioctl.h) is one of
19              _IO    no arg
20              _IOR   user reads info from the driver into arg
21              _IOW   user writes info to driver from data in (or pointed
22                     to by) arg
23              _IOWR  both _IOR and _IOW
24
25   There are two ioctl's defined for the ramdisk driver.  They are:
26
27   /* ioctl to deallocate ram volume */
28   #define RAM_DEALLOCATE  _IOW(R, 1, int)
29
30   /* ioctl to reset the access counter to ram volume */
31   #define RAM_RESETCOUNTS _IOW(R, 2, int)
32
33
34
35   ram_ioctl(dev, cmd, addr, flag)
36   dev_t dev;
37   int cmd;
38   caddr_t addr;
39   int flag;
40   {
41           register struct ram_descriptor *ram_des_ptr;
42           register volume;
43
44           /* check if dev is the status dev */
45           if (RAM_MINOR(dev) != 0)
46                   return(EIO);
47
48           /* check if 0 - 15 disc volume */
49           volume = *(int *)addr;
50           if ((volume % RAM_MAXVOLS) != volume)
51                   return(EIO);
52
53           /* calculate which ram volume it is */
54           ram_des_ptr = &ram_device[volume];
55
56           /* if not allocated, then return error */
57           if (ram_des_ptr->addr == NULL) {
58                   return(ENOMEM);
59           }
```

```
60              switch(cmd) {
61
62              /* mark for memory release on last close */
63              case RAM_DEALLOCATE:
64                      ram_des_ptr->flag = RAM_RETURN;
65                      break;
66
67              /* clear out access counts */
68              case RAM_RESETCOUNTS:
69                      ram_des_ptr->rd8k =     0;
70                      ram_des_ptr->rd7k =     0;
71                      ram_des_ptr->rd6k =     0;
72                      ram_des_ptr->rd5k =     0;
73                      ram_des_ptr->rd4k =     0;
74                      ram_des_ptr->rd3k =     0;
75                      ram_des_ptr->rd2k =     0;
76                      ram_des_ptr->rd1k =     0;
77                      ram_des_ptr->rdother =  0;
78                      ram_des_ptr->wt8k =     0;
79                      ram_des_ptr->wt7k =     0;
80                      ram_des_ptr->wt6k =     0;
81                      ram_des_ptr->wt5k =     0;
82                      ram_des_ptr->wt4k =     0;
83                      ram_des_ptr->wt3k =     0;
84                      ram_des_ptr->wt2k =     0;
85                      ram_des_ptr->wt1k =     0;
86                      ram_des_ptr->wtother =  0;
87                      break;
88              default:
89                      return(EIO);
90              }
91              return(0);
92      }
```

```
1
2                       RAMdisk Close
3
4
5  The close routine may typically perform some driver specific operations.
6  It may flush buffers if the device supports asyncronous I/O (e.g.  tty
7  driver).  It will usually decrement an "open" counter and may release
8  I/O buffers, etc. on close.
9
10 The RAM disk driver just decrements an open count and releases memory on
11 last close iff the RAM_RETURN flag has previously been set (by an ioctl).
12
13
14 #define RAM_RETURN 1
15
16 struct ram_descriptor {
17         char    *addr;
18         int     size;
19         short   opencount;
20         short   flag;
21         int     rd1k;
22                 :
23                 :
24 } ram_device[RAM_MAXVOLS];
25
26 ram_close(dev)
27 dev_t dev;
28 {
29         register struct ram_descriptor *ram_des_ptr;
30         register i;
31
32         /* check if this is status close */
33         if (RAM_MINOR(dev) != 0) {
34                 ram_des_ptr = &ram_device[RAM_DISC(dev)];
35
36                 if (--ram_des_ptr->opencount < 0)
37                         panic("ram_close count less than zero\n");
38         }
39
40         /* free all ram volumes with flag set and open count = 0 */
41         /* RAM_RETURN flag is set by an ioctl call               */
42
43         ram_des_ptr = &ram_device[0];
44         for (i = 0; i < RAM_MAXVOLS; i++, ram_des_ptr++) {
45                 if ((ram_des_ptr->flag & RAM_RETURN) == 0)
46                         continue;
47                 if (ram_des_ptr->opencount != 0)
48                         continue;
49                 /* release the system memory */
50                 sys_memfree(ram_des_ptr->addr, ram_des_ptr->size<<LOG2SECSIZE);
51
52                 /* zero the whole entry */
53                 bzero((char *)ram_des_ptr, sizeof(struct ram_descriptor));
54         }
55 }
```

STEPS TO ADD THE RAMDISK DRIVER TO YOUR KERNEL

STEP 1) # cd /etc/conf

STEP 2) make sure there is a line in /etc/master that looks like this:

```
  ramdisc          ram          3     FB     4     20
```

Note: Major numbers may differ; reflect this in the mknod commands below.

STEP 3) add "ramdisc" to your dfile

STEP 4) compile your source file and either put it in the library that currently has the ramdisk driver in it or else put it in the makefile after step 6

```
# cc -c ramdisk.c
# ar -rv libXXX.a ramdisk.o
```

STEP 5) # config dfile

STEP 6) # make -f config.mk

if you chose not to ar(1) the .o file into the library, edit config.mk (might want to rename it to "makefile" first) to include "ramdisk.o" just before the "LIBS" in the "ld" line:

```
     ld -abcdefg x.o y.o z.o ramdisk.o $ (LIBS)
```

STEP 7) # mv hp-ux /

STEP 8) # reboot

STEP 9)
```
# /etc/mknod /dev/ram  b  4 0xVSSSSS  Where V = volume number (0..0xf)
# /etc/mknod /dev/rram c 20 0xVSSSSS    SSSSS = # of 256 byte sectors

# /etc/mknod /dev/ram128K   b  4 0x000200       (block 128Kb ram volume)
# /etc/mknod /dev/rram128K  c 20 0x000200       (char 128Kb ram volume)

# /etc/mknod /dev/ram1M     b  4 0x101000       (block 1Mb ram volume)
# /etc/mknod /dev/rram1M    c 20 0x101000       (char  1Mb ram volume)

# /etc/mknod /dev/ram4M     b  4 0x404000       (block 4Mb ram volume)
# /etc/mknod /dev/rram4M    c 20 0x404000       (char  4Mb ram volume)
```

STEP 10)
```
# mkfs /dev/ram128K 128 8 8 8192 1024 32 0 60 8192 (mkfs for 128Kb volume)
# mkfs /dev/ram1M   1024          (make file system for 1Mb volume)
# mkfs /dev/ram4M   4096          (make file system for 4Mb volume)
```

STEP 11)
```
# mkdir /ram128K
# mount /dev/ram128K /ram128K                   (mount 128K ram volume)
```

To make the control /dev for "ramstat".

```
# /etc/mknod /dev/ram       c 20 0x0            (status is raw dev only)
```

To release memory of disc #1 (and destroying all files on volume at umount)

```
# ramstat -d 1 /dev/ram
```

To get a status of all memory volumes

```
# ramstat /dev/ram
```

To reset the access counters of a memory volume # 1.

```
# ramstat -r 1 /dev/ram
```

SE 390: Series 300 HP-UX Internals

System Panics

Overview

- Panics happen when the system thinks that "1 == 0" and realizes that thinking this is not a good sign :-)

- The (mounted) disks get sync(2)ed, but are *not* marked clean, which will probably force an fsck(1m) when the system boots.

- If running 7.0 or later, we will consider dumping physical RAM to the swap area (known as "savecore"). This won't happen unless there is local swap of some sort, and it can be disabled in 8.0 and later releases by adb(1)ing the kernel variable do_savecore to 0.

- If the kernel debugger is active, control will be passed to it; otherwise we halt in a tight loop, and the power must be cycled for the system to reboot.

- If you are seeing significant numbers of panics, the most likely possibility is a hardware problem.

- The S700 has "analyze" available, and it is very helpful in extracting useful information from a core dump.

(Hopefully un)Common Kinds of Panics

- Parity error - is a fact of life with parity-checking memory.

- Dup ialloc or freeing free {inode,frag} - usually caused by mounting a corrupt disk. Pay attention when the system tells you to fsck!

- Bus error - often indicates a hardware problem. If it happens to a user, he is sent a signal. It should never happen in the kernel and if it does the system will panic. It could also come from a kernel bug, but most of the ones we've seen have been due to hardware problems.

- I/O Error in Push - generally points to bad interface card, cable, or disk. "Push"ing a page out refers to writing a page to the swap area, and the system will panic if the write() fails.

  In 8.0 this one will say something like "syncpageio detected an error".

If you know of other "legitimate" panics, let me know so I can include them on this list in the future.

Interpreting S300/400 Panic Dumps

- First column consists of stack addresses.
- Numbers in the other columns that are either in the first one or are sandwiched by numbers in the first one are probably frame pointers.
- Find first appropriate address (frame pointer). It is the address of the next one, which is the address of the next one....
- Trace linked list of frame pointers.
- Numbers just to the right of the frame pointers are return addresses.
- Feed return addresses to adb(1) to see who called who.

Reading Series 300 Panic Dumps

When in the course of human events an HP-UX system can't figure out what's going on, it throws up its hands and decides to reboot and try again. When this happens, it is known as a "panic", and the system tries to be helpful by printing out the contents of the kernel stack as it dies. Here is part of one:

```
97bdaa: 00051c90 000ffe01 ffe79405 ffe79401 00000000 00979018 000ec7fa 000ec7fa
97bdca: 0006889a 00000000 0000e000 0006f66c 0097be26 00015314 000ec7fa 00000184
97bdea: 00000000 0000e000 00000000 00000000 03000000 00000000 00000000 00000000
```

The first column consists of stack addresses. The stack grows down in memory, so the top line is the stuff that has been put on the stack most recently. The trace goes from left to right, so the lowest address (most recently pushed) is at the top left; the highest is at the bottom right.

The last eight columns are the actual contents of the stack. There are several kinds of things on it:

- arguments to functions
- return addresses
- frame pointers
- local variables for functions
- saved copies of registers that will be trashed in the called function
- exception information (stuff put there in case of divide by 0, etc)
- junk

It would be nice if the last item didn't have to be there, but it does. This is because not all code uses the conventions established by the HP-UX C compiler. This will be dealt with a bit later.

The second item in the list above is a very important one - it is the key to our ability to trace back through the dump. When a procedure is called, it pushes the frame pointer (register a6 on the 680x0) onto the stack and then copies the stack pointer into the frame pointer. It then subtracts from the stack pointer (remember that the stack grows down) to make room for local variables. The fact that the old frame pointer is pushed each time a procedure is called is what enables us to "walk" or "unwind" the stack.

Since the frame pointers are stack addresses, the basic idea is to look through columns 2-9 for a number that either appears in column 1 or is sandwiched by two numbers in column 1. An important thing to remember is that the addresses may be misaligned by two bytes. An example may help here:

```
98c9da: 00234567 0098c9fa 00034562 ....
98c9fa: .....
```

The "0098c9fa" was properly aligned, but if the line had read

```
98c9da: 00234567 89ab0098 c9fa0003 ....
```

that would have been OK too. Once the first address has been found, others can be found by treating each one as a pointer; i.e., the frame pointers form a linked list.

Surrounding each frame pointer is some interesting information. It is often referred to as an "activation record". The first part of the record will be arguments for the called procedure (keep in mind that these are treated as local variables by the called procedure and thus may have been modified by it). Next, a return address for the calling procedure. Third, the saved frame pointer. Next, space for local variables in the called procedure. Last, space for registers that the called routine wants to use.

Consider the following example. The lines of the dump have been split apart and directional lines have been drawn to show the linked list structure.

```
panic: init died
panic: sleep

97be4a: 0007ff24 00000001 0000800a 0124a6aa 0124a6aa 0097be76 000107ca 0124a6aa
                                                         v
                                       /-----------------/
                                       v
97be6a: 00000094 0124a6aa 00000000 0097be8a 00010062 0124a6aa 00000080 01242000
                                       v
           /---------------------------/
           v
97be8a: 0097beb2 0001450a 0124a6aa 0009ce08 0125f280 0000000a 0000000a 0008022b
           v
           \------------------\
                              v
97beaa: 0097bec2 00024186 0097beca 00016cc8 0009ce08 ffff7dfc 0125f280 01242000
                              v
           /------------------/
           v
97beca: 0097bf02 000099f4 00000000 000ffc01 ffcb0405 ffcb0401 00000001 0000003c
           v
           \------------------------------------------------------\
                                                                  v
97beea: ffff7dfc 0125babc 0000a830 00080221 00000003 00000000 0097bf4a 0000ac8c
                                                                  v
       /----------------------------------------------------------/
       |
97bf0a:|00000080 0097bf52 0007f8fc ffff7dfc 0125babc 00000002 00000001 0097bf46
       |
       |
97bf2a:|0001dd7c 00989fe0 00000003 0125babc 00000003 0000000b 0000003c 00000080
       |
       \---\   /-----------------------------------------------------------\
           v   ^                                                           v
97bf4a: 0097bf66 00004ae4 0007febc 00000004 ffff7dfc 00979018 00000000 0097bf76
                                                                           v
                                       /-----------------------------------/
                                       v
97bf6a: 00004904 00000000 0097bfaa 0097bf9e 0000ebdc 00000031 00000040 ffcab004
                                       v
                                       \-----------------\
                                                         v
97bf8a: fffffa28 0001a1b4 00000000 ffff7f98 00000007 ffff7e00 00000458 0097bfaa
                                                     ^^^^^^^^
   The buck stops here - this address isn't close to what's in the left column.

97bfaa: 00000005 00000001 00000001 00000020 000ffc01 ffcb0405 ffcb0401 00000700

97bfca: 00000031 00000040 00012016 0001a100 ffcab004 fffffa28 0001a1b4 00000000

97bfea: ffff7e00 ffff7df8 00000000 00011acc 0080000f fcb1
```

It is important to remember that much of this is dependent on routines using the normal calling convention. There will be exceptions to this. If someone writes a routine in assembly language and doesn't bother to save the frame pointer, this will mess things up a bit. The frame pointers will be good, but one of the activation records will have a return address that doesn't make too much sense, because there is not a matching frame pointer. The same thing will happen if an exception (such as a bus error) is encountered in kernel mode. Note that either of these things can cause small glitches in the trace, but they don't necessarily mean the end of the hunt.

A third oddity is introduced when a routine is called indirectly. Probably the most common example of this is a kernel routine named syscall(); it calls the actual code for a given system call by jumping indirectly. Indirect calls don't automatically end the trace, but the one in syscall() often does. The reason is that the stack that is dumped out is the *kernel* stack - we can't walk back into user land on the kernel stack. One thing that an indirect call will always do is make things a bit less clear later on when we are trying to figure out who called whom.

Once the stack has been unwound, how do we find out what the numbers mean? The easiest way is probably to use the assembly level debugger, adb(1). If adb(1) is run on the kernel that panicked (or one that is the same version and has been configured IDENTICALLY), it will translate absolute addresses into symbolic ones. By giving each address to adb(1) and doing a bit of interpretation, a symbolic traceback can be constructed. It will usually have things like boot() and panic() at the top and things like read() or setuid() at the bottom. The important stuff will be in the middle.

To start, use a command something like this:

```
$ adb /hp-ux
```

Once adb(1) has started up, you can get it to do things like tie absolute addresses to known symbols or disassemble parts of the code. The fundamental command we will use will be of this form:

```
<address>?<n>i      as in      32cea?20i
```

The address is typically an absolute hexadecimal number, the question mark says to print out what that address is, <n> is the number of times to do it, and "i" tells it to interpret the stuff as instructions. It can safely be said that adb(1) is not one of the friendlier HP-UX utilities. For instance: there is no prompt, and the commands (as seen above) are a bit cryptic. Note that to exit you have two choices: "$q" or the old standby, CTRL-d. And now back to our story....

Since we know that the return address is just to the right in the printout (was pushed just before the frame pointer), we can take this number and feed it to adb(1) to find out what routine made the call. In the 2nd example, the return address was 00034562. To find out what routine that is in, we might use this:

```
34562?i
```

To see a bit of context, we would do something like this:

```
34550?20i
```

There is a catch with this. This is because instructions will sometimes be aligned on even byte (word) boundaries, not on 4 byte (longword) boundaries. Thus, if you tell adb(1) to start disassembling at an address that is halfway through an instruction, you will get a bogus list of instructions. One way of detecting this is to look and see if there is some kind of call instruction in the disassembly listing - if there isn't, chances are *excellent* that the disassembly is misaligned.

For an example, we'll look at the addresses in the stack tracing example above. Just to the right of each frame pointer is the return address for that call. By feeding these to adb(1), we can figure out who called whom. What follows is a logfile of a session with adb(1), with three things done to it: 1) blank lines have been inserted for clarity; 2) most of the tries that yielded misaligned results have been eliminated; 3) comments have been added; they start with "#".

```
$ adb /hp-ux
executable file = /hp-ux
core file = core
ready

107ca?i
_biowait+0x22:          addq.w  &0x8,%a7

107af?10i
_biowait+0x7:           bgt.w   _bmap+0x523
                        eor.b   %d4,%d0
                        ori.b   &0xFFFFEC2D,%a1
                        mov     %sr,???         # not looking good
                        fsun    -(%a0)
                        movq    &0x0,%d4        # should be a call to sleep
                        sub.w   %a0,%d2         # in here somewhere
                        subq.w  &0x2,%a6
                        eor.b   %d4,%d0
                        ori.w   &0x1C50,???

107b0?10i                                       # try again!
_biowait+0x8:           ori.b   &0x4EB9,%a0
                        ori.b   &0x9EC,%d0
                        mov.l   %d0,-0x4(%a6)
                        bra.b   _biowait+0x24
                        pea     0x94.w
                        pea     (%a5)
                        jsr     _sleep          # now we're talking...
                        addq.w  &0x8,%a7        # pop 8 bytes of args off stack
                        mov.l   (%a5),%d0
                        movq    &0x2,%d1

10062?i
_bwrite+0x92:           mov.l   %a5,(%a7)
10050?10i
_bwrite+0x80:           jsr     (%a0)
                        addq.w  &0x4,%a7
                        btst    &0x8,%d7
                        bne.b   _bwrite+0x9E
                        pea     (%a5)
                        jsr     _biowait
                        mov.l   %a5,(%a7)
                        jsr     _brelse
                        addq.w  &0x4,%a7
                        bra.b   _bwrite+0xAE

1450a?i
_sbupdate+0x4C:         mov.l   0x34(%a5),(%a7)
144f0?10i
_sbupdate+0x32:         mov.l   %d0,-(%a7)
                        mov.l   0x22(%a4),-(%a7)
                        pea     (%a5)
                        jsr     _bcopy
                        lea     0xC(%a7),%a7
                        pea     (%a4)
                        jsr     _bwrite
                        mov.l   0x34(%a5),(%a7)
                        mov.l   0x34(%a5),%d0
                        subq.l  &0x1,%d0

16cc8?i
_update+0xD4:           addq.w  &0x4,%a7
16cb0?10i
_update+0xBC:           clr.b   0xD0(%a0)
                        mov.l   -0x4(%a6),%a0
                        mov.l   _time,0x20(%a0)
```

```
                        pea      (%a4)
                        jsr      _sbupdate
                        addq.w   &0x4,%a7
                        lea      0x18(%a4),%a4
                        cmp.l    %a4,&0x9CFE8
                        bcs.w    _update+0x42
                        mov.l    _inode,%a5

99f4?i
_boot+0x8A:             addq.w   &0x4,%a7
99e6?10i
_boot+0x7C:             beq.w    _boot+0x90
                        pea      0x0.w
                        jsr      _update          # this is the one
                        addq.w   &0x4,%a7
                        bra.w    _boot+0x9C
                        pea      0x1.w
                        jsr      _update
                        addq.w   &0x4,%a7
                        pea      _reboot_after_panic+0x1E0
                        jsr      _printf

ac8c?i
_panic+0xC4:            addq.w   &0x8,%a7
ac7c?6i
_panic+0xB4:            ???      (68881)
                        pea      0x8(%a6)
                        mov.l    -0x4(%a6),-(%a7)
                        jsr      _boot
                        addq.w   &0x8,%a7
                        bra.w    _panic+0xC6

4ae4?i
_exit+0x1D8:            addq.w   &0x4,%a7
4ad0?10i
_exit+0x1C4:            or.l     %d4,%d6
                        cmp.w    %d0,0x2A(%a5)
                        bne.b    _exit+0x1DA
                        pea      _nsysent+0x88
                        jsr      _panic
                        addq.w   &0x4,%a7
                        mov.w    0xA(%a6),0x52(%a5)
                        mov.l    _u+0x84E,0x9C(%a5)
                        mov.l    _u+0x84A,0x98(%a5)
                        mov.l    _u+0x846,0x94(%a5)

4904?i
_rexit+0x20:            addq.w   &0x4,%a7
48f4?10i
_rexit+0x10:            andi.l   &0xFF,%d0
                        asl.l    &0x8,%d0
                        mov.l    %d0,-(%a7)
                        jsr      _exit
                        addq.w   &0x4,%a7
                        mov.l    (%a7),%a5
                        unlk     %a6
                        rts
                        link.w   %a6,&0xFFFFFFF0
                        movm.l   &<%d7,%a4,%a5>,(%a7)

ebdc?i
_syscall+0x15E:         lea      _u+0x78,%a0
ebc8?10i
_syscall+0x14A:         sub.l    %d2,%d0
                        mov.b    &0x1,(%a0)
                        lea      _u+0x9FA,%a0
```

```
clr.w   (%a0)
mov.l   0x4(%a3),%a0
jsr     (%a0)           # note indirect call
lea     _u+0x78,%a0
tst.b   (%a0)
beq.b   _syscall+0x186
lea     _u+0x9FA,%a0
```

```
$q
```

By looking at this bottom-up, we can see that the order of calls was like this:

```
syscall()
rexit()
exit()
panic()
boot()
update()
sbupdate()
bwrite()
biowait()
sleep()
```

Note that we didn't see a "jsr _rexit" in syscall(); we just looked at where we had been before.

What can we learn from all of this? That depends. It is conceivable that this kind of information could help track down a kernel bug. It is also possible that it could satisfy a customer's curiosity. One nice thing to know is that as of 6.0, the kernel will construct a sybolic traceback complete with the arguments to the calls - this will be printed on the screen just below the stack dump.

SE 390: Series 300 HP-UX Internals

File System

The Big Picture

How does HP-UX organize disks and access files?

The Little Pictures

- History.
- The vnode layer & pathname lookup.
- Caching: buffers, inodes, cdnodes, and directory names.
- Mounting and unmounting file systems.
- General flow within the kernel.
- The HFS/Berkeley/McKusick file system.
  - History and layout.
  - On-disk data structures.

The Problem

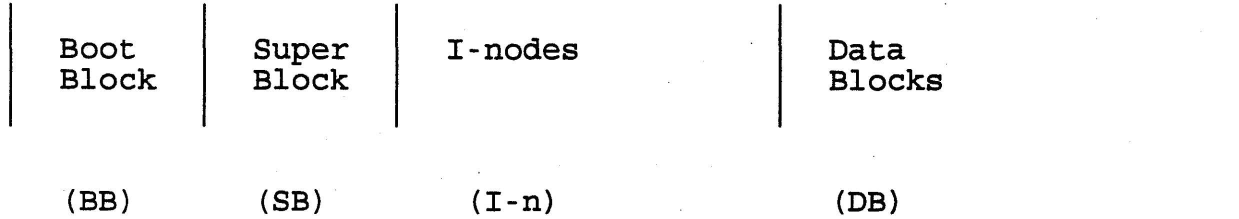
A customer calls and says that he can't boot. You go to help him out, and take a loaner disk. You boot off of the loaner and try to fsck(1m) his disk. It fails, and after a bit of poking around you deduce that someone has tar(1)ed over the first part of his disk. What will you do?

File System

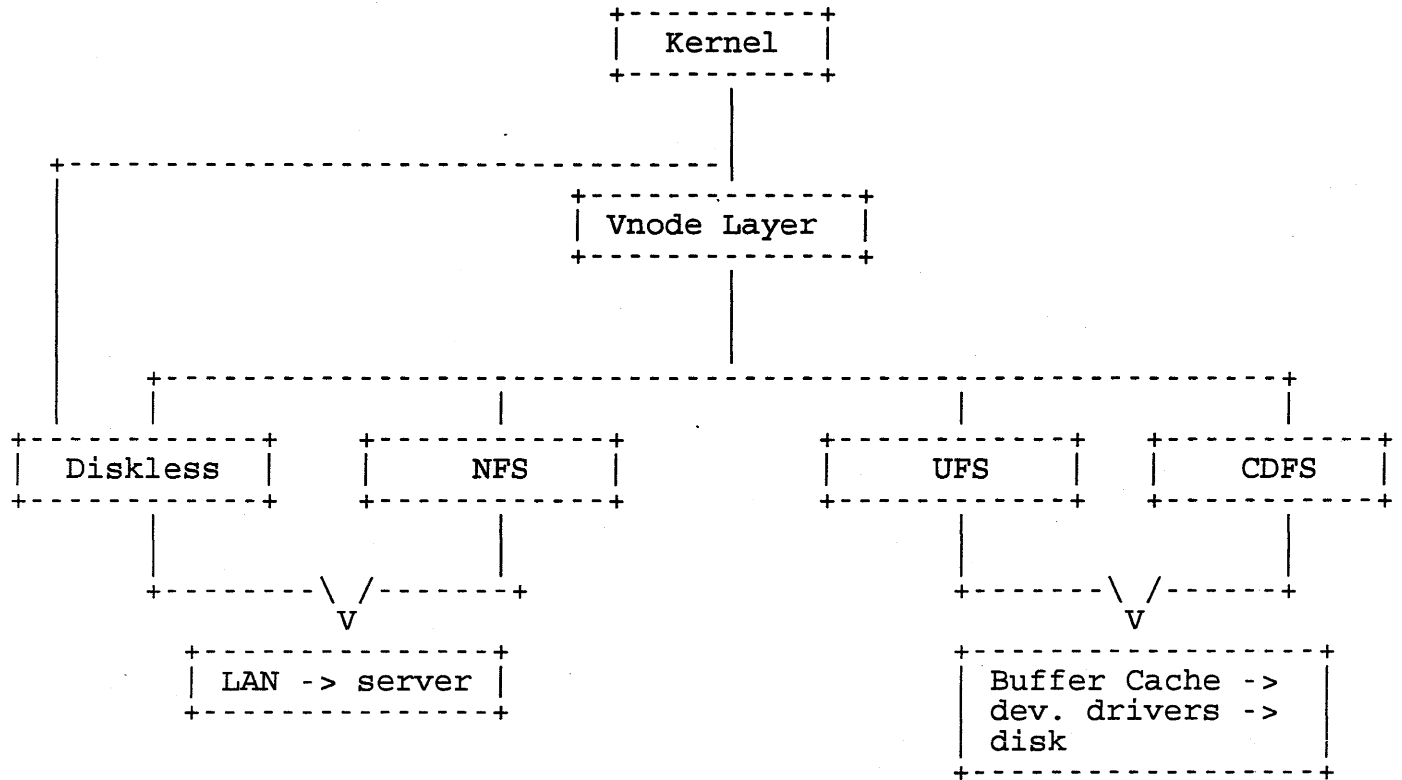
The original UN*X file system

- Superblock (single copy on disc)
- I-nodes (grouped together)
- Data blocks (small size = 512 bytes)
- Advantages:
  * handles large numbers of small files efficiently
  * easy to implement
- Disadvantages:
  * limited file I/O throughput
  * lack of locality on disk
  * lack of robustness
  * designed for "small" systems/disks

Picture of a Bell file system



How The Kernel & File System Fit Together



SE 390: Series 300 HP-UX Internals

File System

The Vnode Layer

- Why?

    - It does for the filesystem what the device driver interface (open, close, read, write, strategy, etc) did for device drivers.

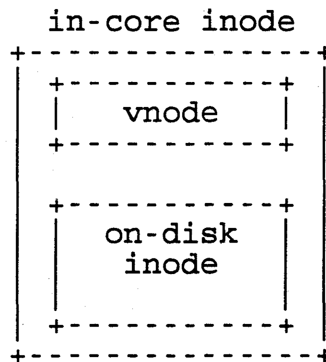
    - To allow the system to access files that are on a remote machine, or that are on a disk that isn't HFS.

    - To be compatible with the industry.

- How?

    - Most file system activity revolves around "vnodes", which are like inodes but are not implementation dependent.

    - vnodes only exist in-core, and are part of in-core inodes or cdnodes or...



At boot time, the vnode in each in-core inode will be initialized to point at HFS routines; if CD-ROM is configured into the system, the vnode in each cdnode would be set up to point at CDFS functions.

    - The vnode layer is object-oriented in the sense that a vnode carries around a list of operations that can be done on it. If the system wants to read from a file represented by (struct vnode *)vp, it will do something like this (this is not actual code):

```
        (*(vp)->v_op->vn_read)(vp, rwflag, buf, size)
```

This will call a routine to read from the file, whether the file is local, remote, on a PC, or whatever. In concept, it is roughly this:

```
switch (vp->v_type)

        case VHFS: hfs_read(vp, rwflag, buf, size)

        case VNFS: nfs_read(vp, rwflag, buf, size)

        case VCDFS: cdfs_read(vp, rwflag, buf, size)
```

SE 390: Series 300 HP-UX Internals

File System

Pathname Lookup

- Many system calls take a character string that is a pathname. Before they can do much, they must figure out where the file is and what type it is. This requires lots of work....

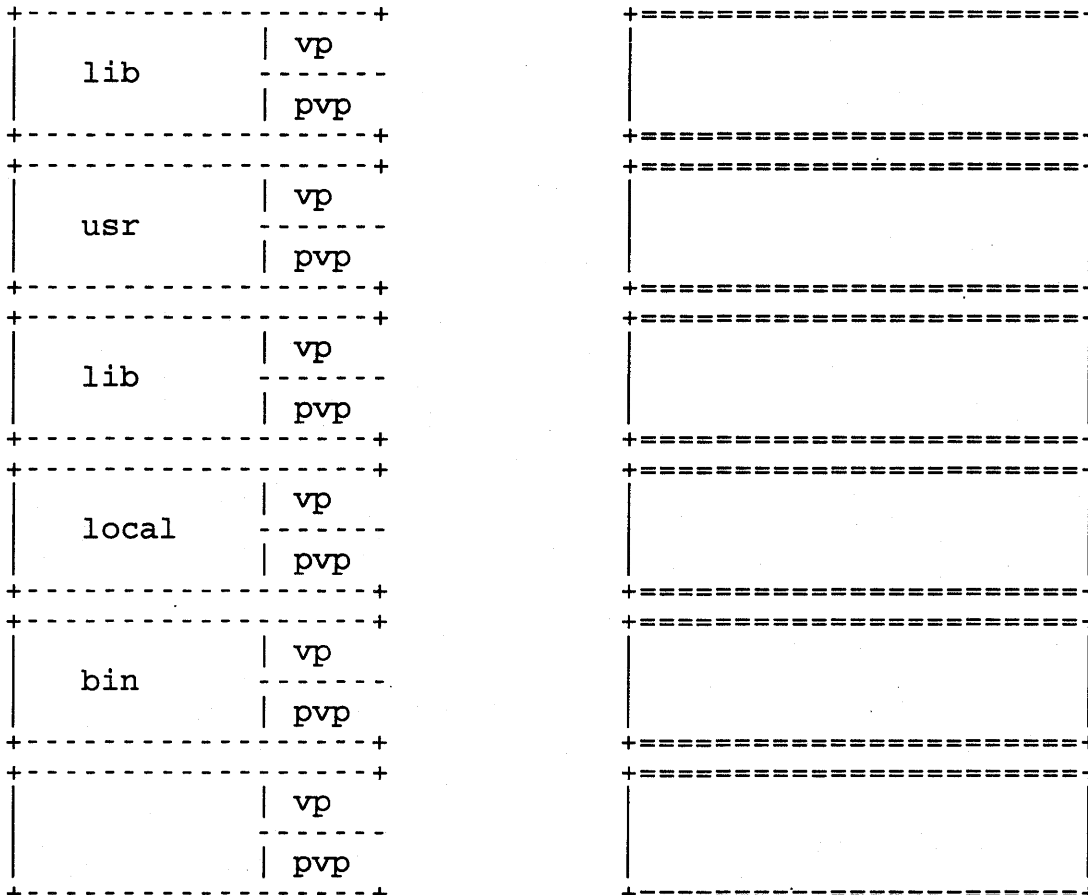
  - The basic plan of attack for lookupname() is to look for the vnode that corresponds to the pathname we're interested in. Here's a greatly simplified view:

```
while there's another element in the path

        if that element is in the dnlc
                use the vp there
        else
                call _lookup for the type of
                fs the current component is in
```

There are some "gotchas" left out here (RFA, Diskless, mount pts), but this is the guts of the algorithm. The "else" clause above is important - it's what allows us to cleanly resolve pathnames even though each element of the path may belong to a different fs type.

DNLC   in-core inode table



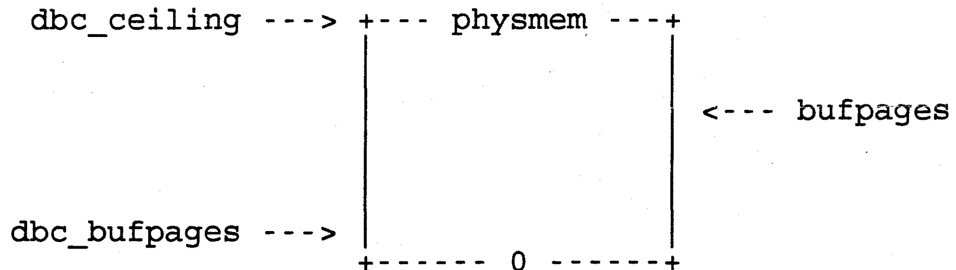
Caching

- The buffer cache - used to avoid reading things that were read "recently" and to keep from having to write stuff out if it's just going to get trashed shortly. Buffers are also available for use as scratch space if drivers need to use them.

  - Prior to 9.0, the buffer cache was sized by nbuf/bufpages; if these were nonzero, the system used them; otherwise, 68K machines would use 10% of the 1st 5MB and 5% of the rest of RAM; PA boxes would use 10% of RAM

  - In 9.0, we have a "dynamic buffer cache" (DBC); it is still possible to set a specific size using the tunables above, but in general it is best to let the system grow/shrink the cache as needed - as the filesystem uses pages, the cache size will grow; if the system runs short of memory (user processes ask for some), the pager will take pages back from the DBC.

    If the DBC is taking too much, either set nbuf/bufpages explicitly or else adb(1) dbc_ceiling to set a limit on the size of the cache.



  - dbc_bufpages is the "floor" - the minimum number of pages the cache will have (default 64)

  - dbc_ceiling is the maximum - (default physmem)

  - bufpages is the current number of pages taken by the cache (if you set bufpages explicitly, it will do at boot time what it used to - hold the cache at that size)

- The inode cache - used to keep track of inodes so that we don't always have to get them off of the disk. Pathname translation boils down to accessing lots of inodes, so the less often we have to get them from disk the better. If a file on a Berkeley FS disk is open, there *must* be a copy of its inode in-core.

- Directory name lookup cache - speeds up pathname translation. It consists of a set of filenames and their respective vnode pointers. The system is frequently asked to open files in /usr/lib; thus it makes sense to have "usr" and "lib" sitting in the cache. This will often save several disk accesses for a single pathname translation. The name is somewhat misleading; there are ordinary filenames in the cache too.

Mounting And Umounting File Systems

- Only block devices need apply :-)

- Mounting a disk with vfsmount(2) makes that disk's file system a part of the present file system; its root "covers" the directory we mount it on.

- Pathname lookup is affected. When lookupname() is resolving a pathname, it checks the vnode for each element to see if it has been "covered". If so, it jumps to the "covering" vnode and continues the search. The "is this thing covered?" question is asked before "where's the directory this vnode corresponds to?"

  There is also a possibility that the current vnode is covering another one and we are moving *up* in the directory hierarchy (what if we are resolving "../.."?); in this case, we must jump to the vnode we are covering and continue on.

- When a disk is mounted, it is added to a list of mounted file systems. This is used for a number of things, not the least of which is when reboot(2) is shutting down the system. In that case, it's important that we not have to rely on /etc/mnttab!

- When a disk is umount(2)ed, the system checks to make sure no files on that disk are open; if they are, the umount(2) will fail with EBUSY. No such checking is done with vfsmount(2) (try it :-)

Stats are kept for hits + misses on the dnlc. Look at script in archive.

SE 390: Series 300 HP-UX Internals

File System

Important Data Structures

Per process:

- u_ofile - semi-static array in each process's u area. A "file descriptor" is just an index into this array, so whenever a process open(2)s a file, a slot in this array is taken up. In >=8.0, this array will be dynamic and will be sized by calls to setrlimit(2), with an upper bound of "maxfiles_lim" (1024).

- u_rdir - vnode pointer for this process' root directory. See sys/user.h

- u_cdir - vnode pointer for this process' current directory See sys/user.h How does this interact with "cd"?

  In 8.0, all of the above move to the proc table entry.

System wide:

- file - the kernel open file table. There is at least one slot in it for each file or socket that is open, and it is sized by the tunable parameter "nfile". See sys/file.h.

- inode - the inode cache. There is a slot in it for each inode that is in core (remember that we do caching, so a given in-core inode isn't necessarily being used), and it is sized by (all together now :-)) "ninode". Every file that is open on a local HFS disk *must* have *one* slot in the inode table. See sys/inode.h and sys/ino.h.

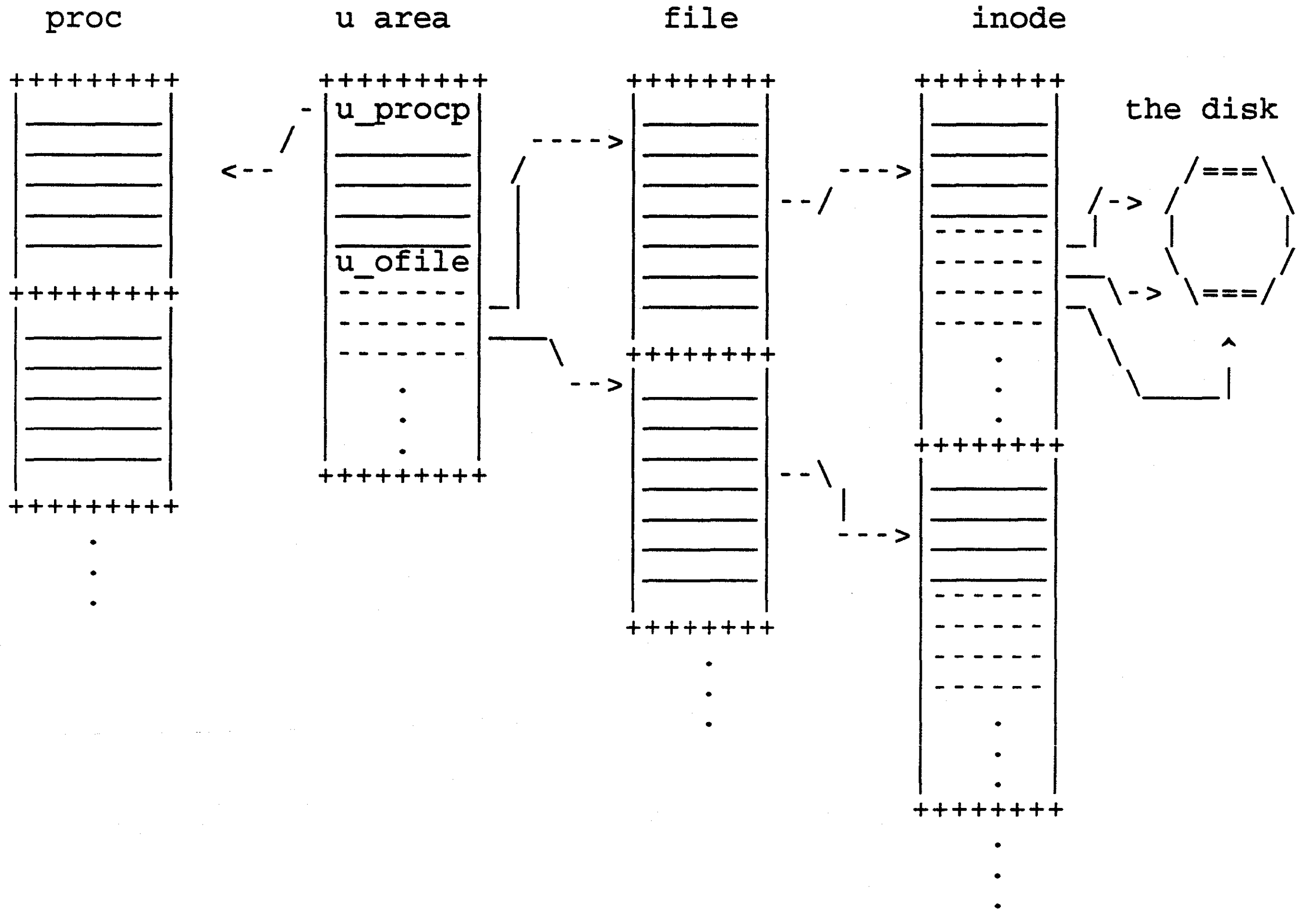
- ncache - the directory name lookup cache, also sized by "ninode".

- In 8.0: fs_async decides whether the filesystem should lean toward reliability or performance. If it is set to 0 (default), the system will write inodes/blocks to disk more often, which give reliability at the expense of performance. If it is 1, the system will delay these writes, yielding a great deal of performance in some situations and very little in others.

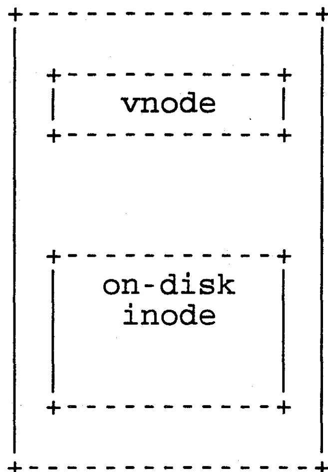
>> Having it set to 1 pretty much guarantees having to do a manual fsck(1m) if the system crashes or loses power. <<

File System

Relevant Kernel Structures



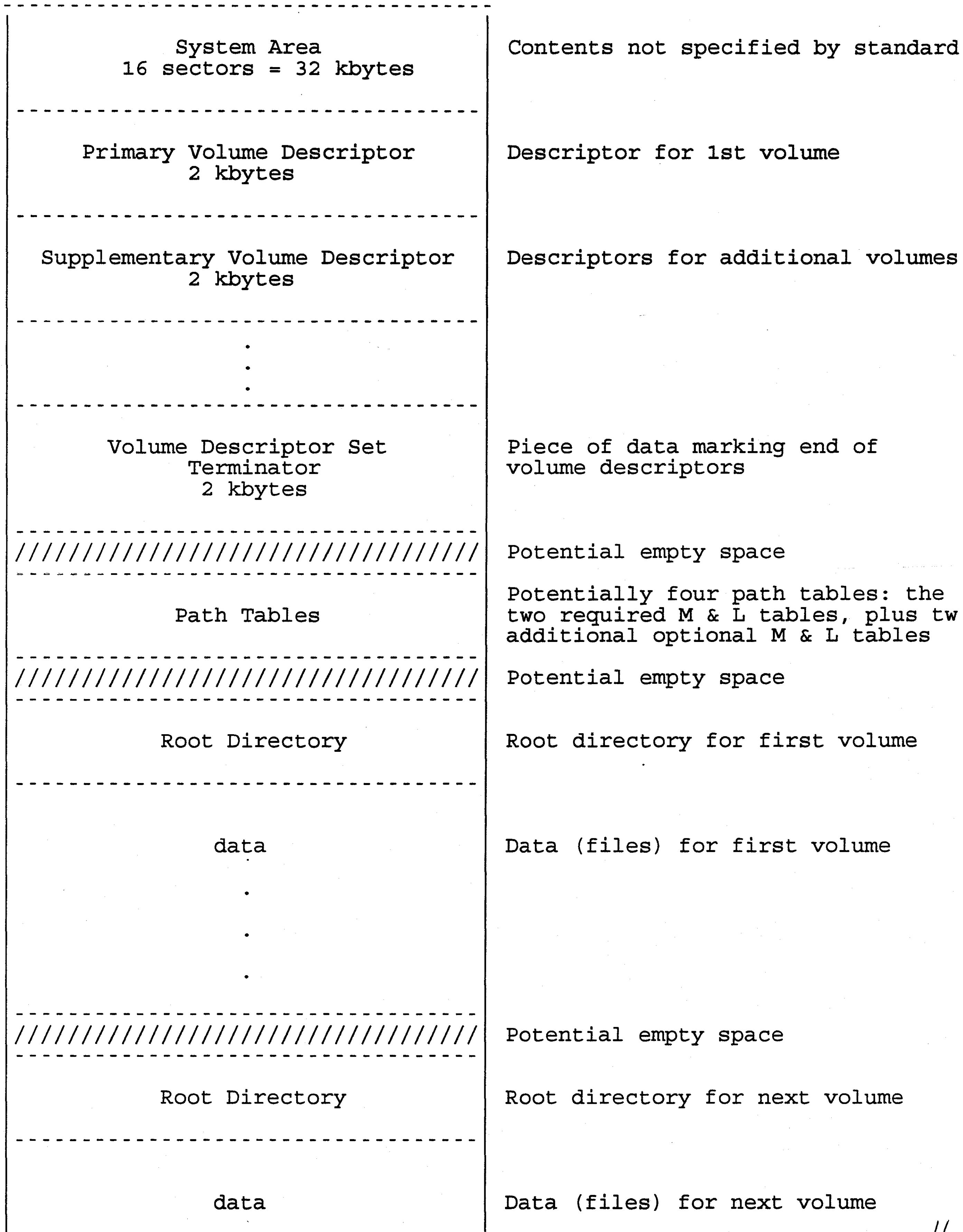
- an in-core inode looks something like this:

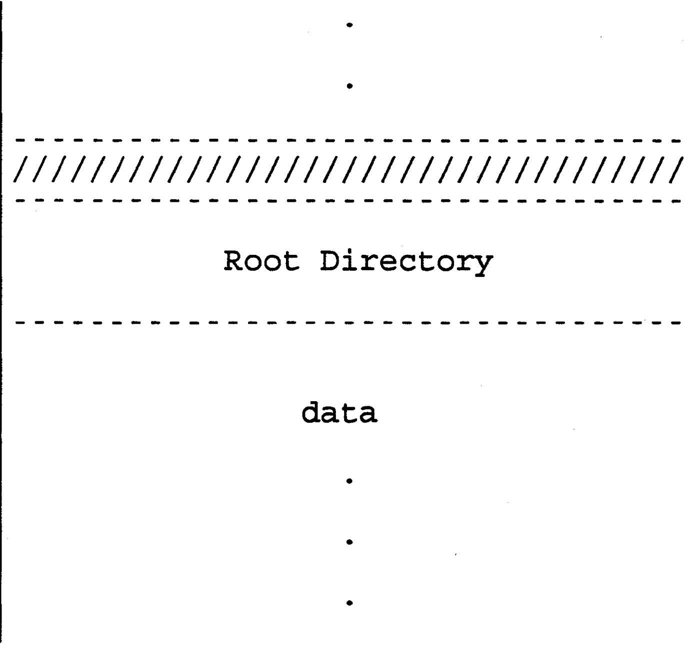


For thurs.

CD-ROM Layout

Our CD-ROM support conforms to the High Sierra & ISO-9660 standards. Here's a rough sketch of how a CD-ROM is organized:





Potential empty space

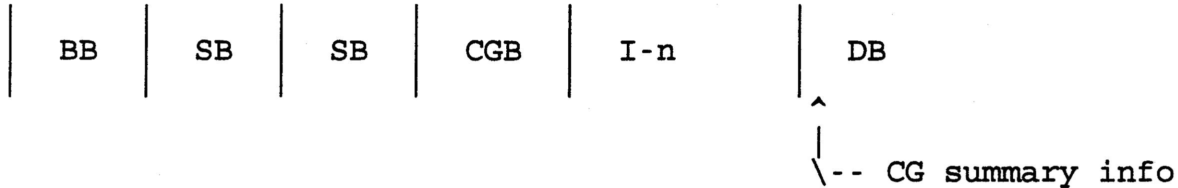
Etc., till end of CDROM or data

The Berkeley/McKusick file system

- often referred to as "HFS" or "ufs"
- retains advantages of the original Bell design
- includes remedies for most problem areas
  * throughput: larger block size (4/8 Kbytes)
  * locality: introduction of "cylinder groups" (each resembles a Bell file system)
  * robustness: superblock is replicated in each group
  * extensible: can access files of 4+ Gbytes (theoretical maximum ~ 4 Tbytes)
- see fs(4) for an explanation of many of the fields in the superblock
- minfree is a space-for-time tradeoff; the filesystem wastes some space in order to make block allocation stable and fast; note that it is a *percentage*, not a fixed amount (yes, this is still true on 1.3GB disks....)
- a cylinder group contains a backup copy of the superblock, a cylinder group block, some inodes, and some data
  - the information that changes in the superblock is the kind of thing fsck(1m) can fix, so once the filesystem is built the redundant superblocks are not normally updated (convertfs(1m) is the most common exception)
  - there is a fixed number of inodes per cylinder group
  - the information about which blocks/inodes are free is in bit maps in the cylinder group blocks, cg_free[] & cg_iused[]
  - the last time CGB was written is stored in cg_time, which is helpful to know when trying to "un-rm"

Picture of a Berkeley file system

cylinder group 0:



cylinder group 1:

cylinder group 2:

Note that the groups are "walking" to the right - this is because the system tries to stagger the backup superblocks *all over* the disk. Given this staggering of the CG beginnings, it would be hard to find the inodes or CGB or backup SB for any particular CG, except that there are macros that will do it for us.

cgsblock(&sb, 5)	will return the fragment address of the beginning of the superblock stored in CG 5
cgimin(&sb, 21)	will return the fragment address of the first inode in CG 21

File System

The space on a disk really comes from sectors that are organized into tracks that are organized into surfaces/platters.... However, it is easier to think about it in terms of a flat logical address space (which is the interface modern disks present):

```
          +----------------------+
0         |      boot block      |  both this and the primary SB are
          |______________________|  CG 0 "data"; they just don't belong
8K        | primary superblock   |  to any particular file....
          |______________________|
16K       |   CG 0 superblock    | <--- cgsblock(&super, 0)*super.fs_fsize
          |______________________|  or SBLOCK*DEV_BSIZE
24K       |    CG 0 cgblock      |
          |______________________|
32K       |    CG 0 inodes       | <--- cgimin(&super, 0)*super.fs_fsize;
          |______________________|
40K       |    CG 0 inodes       |
          |______________________|
48K       |    CG 0 inodes       |
          |______________________|
56K       |     CG 0 data        |
          |__________|___________|
64K       |          |           |
          |__________|___________|
72K       |          V           |
.         |______________________|          (rest of CG 0 is data)
.                    .
.                    .

           ______________________
2048K     |     CG 1 data        | <--- cgbase(&super, 1)*super.fs_fsize
          |______________________|  Notice this data in front of CG 1's
2056K     |     CG 1 data        |  superblock - CG 2 would have even more
          |______________________|  of it - this is to scatter superblocks
2064K     |     CG 1 data        |  all over the disk.
          |______________________|
2072K     |   CG 1 superblock    | <--- cgsblock(&super, 1)*super.fs_fsize
          |______________________|
2080K     |    cg 1 cgblock      |
          |______________________|
2088K     |    CG 1 inodes       | <--- cgimin(&super, 1)*super.fs_fsize
          |______________________|
2096K     |    CG 1 inodes       |
          |______________________|
2104K     |    CG 1 inodes       |
          |______________________|
2112K     |     CG 1 data        |
          |__________|___________|
2120K     |          |           |
          |__________|___________|
2128K     |          V           |
          |______________________|
2136K     |                      |
.         |______________________|
.                    .
.                    .
```

How UFS Files Are Accessed

(The following notes assume no non-UFS elements in the path)

- Directories contain i-number, record length, name length, and filename (the record length is in there so that deletions can be handled simply - we just add the record length of the entry being zapped to the previous one.
- Root directory is called "/" and its i-number is always 2, which is why we need both a device and an i-number to uniquely identify a file.
- The inode has things like modification/access time stamps, modes, uid/gid, etc, as well as pointers to the actual data blocks. The structure of an inode is defined in /usr/include/sys/ino*h.
- To find a file, the kernel must start from the current directory or the root (depending on whether the name starts with "/") and go through a directory and an inode per element of the path.
- The directory is the *only* place on the disk where the filename is stored; the inode has everything else about the file.
- Normally, directories should be read with opendir(3)/readdir(3); when you are reading them straight from the disk, though, be sure to use the structure defined in /usr/include/ndir.h.

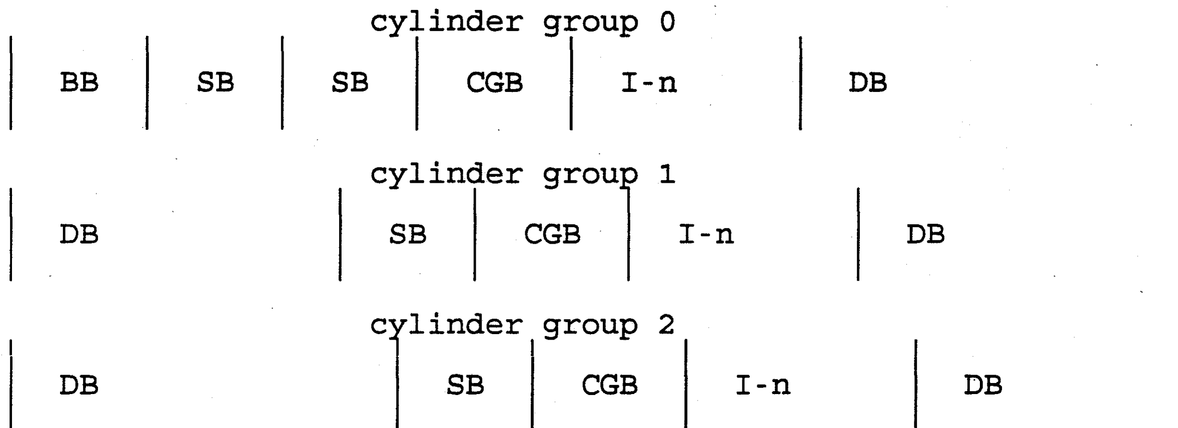
inum	rlen	nlen	name
2	12	1	.
2	12	2	..
3	20	10	lost+found
9	12	3	etc

Summary

Pathname lookup

- To use a path like "/users/se/smith", the kernel must translate it to an i-number (or cd-number, etc.) To do this, it chops the path up into individual names and lets the appropriate filesystem code handle looking for the next name in that one (assuming it's a directory; if it's not, we must be done or else the user goofed).

The McKusick filesystem staggers backup superblocks around the disk, and tries to put a file's data, directory entry, and inode close together:



On-disk data structures

- The superblock has fundamental information about the whole filesystem: the block/fragment size, the number of cylinder groups, the magic number, etc.

- All of the interesting information about a file (except its name) is in its inode.

- The actual block pointers for the file's data are expressed as fragment addresses and are found in the inode. There are 12 direct-block pointers and 3 indirect block pointers. The 1st indirect-block pointer points at a block of pointers to real disk blocks; the 2nd points to a block of pointers to blocks of pointers to real data; the 3rd is presently unused :-)

```
 1  /*
 2   * dnlc.h: $Revision: 1.3.61.2 $ $Date: 91/06/19 13:45:42 $
 3   * $Locker:  $
 4   */
 5
 6  #ifndef _SYS_DNLC_INCLUDED
 7  #define _SYS_DNLC_INCLUDED
 8
 9  /*
10   * Copyright (c) 1984 Sun Microsystems Inc.
11   */
12
13  /*
14   * This structure describes the elements in the cache of recent
15   * names looked up.
16   */
17
18  #define NC_NAMLEN       15      /* maximum name segment length we bother wi
19
20  struct  ncache {
21          struct  ncache  *hash_next, *hash_prev; /* hash chain, MUST BE FIRS
22          struct  ncache  *lru_next, *lru_prev;   /* LRU chain */
23          struct  vnode   *vp;                    /* vnode the name refers to
24          struct  vnode   *dp;                    /* vno of parent of name */
25          char            namlen;                 /* length of name */
26          char            name[NC_NAMLEN];        /* segment name */
27          struct  ucred   *cred;                  /* credentials */
28  };
29
30  #define ANYCRED ((struct ucred *) -1)
31  #define NOCRED  ((struct ucred *) 0)
32  /*
33  int     ncsize;
34  struct  ncache *ncache;
35  */
36
37
38  #define NC_HASH_SIZE            256     /* size of hash table */
39
40  /*
41   * Stats on usefulness of name cache.
42   */
43  struct  ncstats {
44          int     hits;           /* hits that we can really use */
45          int     misses;         /* cache misses */
46          int     long_enter;     /* long names tried to enter */
47          int     long_look;      /* long names tried to look up */
48          int     lru_empty;      /* LRU list empty */
49          int     purges;         /* number of purges of cache */
50  };
51
52  /*
53   * Hash list of name cache entries for fast lookup.
54   */
55  struct  nc_hash {
56          struct  ncache  *hash_next, *hash_prev;
57  };
```

```
 1  /*
 2   * @(#)fs.h: $Revision: 1.17.61.2 $ $Date: 91/06/19 15:45:29 $
 3   * $Locker:  $
 4   *
 5   */
 6
 7  /* @(#) $Revision: 1.17.61.2 $ */
 8  #ifndef _SYS_FS_INCLUDED /* allows multiple inclusion */
 9  #define _SYS_FS_INCLUDED
10  /*
11   * Each disk drive contains some number of file systems.
12   * A file system consists of a number of cylinder groups.
13   * Each cylinder group has inodes and data.
14   *
15   * A file system is described by its super-block, which in turn
16   * describes the cylinder groups.  The super-block is critical
17   * data and is replicated in each cylinder group to protect against
18   * catastrophic loss.  This is done at mkfs time and the critical
19   * super-block data does not change, so the copies need not be
20   * referenced further unless disaster strikes.
21   *
22   * For file system fs, the offsets of the various blocks of interest
23   * are given in the super block as:
24   *      [fs->fs_sblkno]         Super-block
25   *      [fs->fs_cblkno]         Cylinder group block
26   *      [fs->fs_iblkno]         Inode blocks
27   *      [fs->fs_dblkno]         Data blocks
28   * The beginning of cylinder group cg in fs, is given by
29   * the ``cgbase(fs, cg)'' macro.
30   *
31   * The first boot and super blocks are given in absolute disk addresses.
32   */
33  #define BBSIZE          8192
34  #define SBSIZE          8192
35  #define BBLOCK          ((daddr_t)(0))
36  #define SBLOCK          ((daddr_t)(BBLOCK + BBSIZE / DEV_BSIZE))
37
38  /*
39   * Addresses stored in inodes are capable of addressing fragments
40   * of `blocks'. File system blocks of at most size MAXBSIZE can
41   * be optionally broken into 2, 4, or 8 pieces, each of which is
42   * addressible; these pieces may be DEV_BSIZE, or some multiple of
43   * a DEV_BSIZE unit.
44   *
45   * Large files consist of exclusively large data blocks.  To avoid
46   * undue wasted disk space, the last data block of a small file may be
47   * allocated as only as many fragments of a large block as are
48   * necessary.  The file system format retains only a single pointer
49   * to such a fragment, which is a piece of a single large block that
50   * has been divided.  The size of such a fragment is determinable from
51   * information in the inode, using the ``blksize(fs, ip, lbn)'' macro.
52   *
53   * The file system records space availability at the fragment level;
54   * to determine block availability, aligned fragments are examined.
55   *
56   */
```

```
 57
 58  /*
 59   * Cylinder group related limits.
 60   *
 61   * For each cylinder we keep track of the availability of blocks at differe
 62   * rotational positions, so that we can lay out the data to be picked
 63   * up with minimum rotational latency.  NRPOS is the number of rotational
 64   * positions which we distinguish.  With NRPOS 8 the resolution of our
 65   * summary information is 2ms for a typical 3600 rpm drive.
 66   */
 67  #define NRPOS           8       /* number distinct rotational positions */
 68
 69  /*
 70   * MAXIPG bounds the number of inodes per cylinder group, and
 71   * is needed only to keep the structure simpler by having the
 72   * only a single variable size element (the free bit map).
 73   *
 74   * N.B.: MAXIPG must be a multiple of INOPB(fs).
 75   */
 76  #define MAXIPG          2048    /* max number inodes/cyl group */
 77
 78  /*
 79   * MINBSIZE is the smallest allowable block size.
 80   * In order to insure that it is possible to create files of size
 81   * 2^32 with only two levels of indirection, MINBSIZE is set to 4096.
 82   * MINBSIZE must be big enough to hold a cylinder group block,
 83   * thus changes to (struct cg) must keep its size within MINBSIZE.
 84   * MAXCPG is limited only to dimension an array in (struct cg);
 85   * it can be made larger as long as that structures size remains
 86   * within the bounds dictated by MINBSIZE.
 87   * Note that super blocks are always of size MAXBSIZE,
 88   * and that MAXBSIZE must be >= MINBSIZE.
 89   */
 90  #define MINBSIZE        4096
 91  #define MAXCPG          32      /* maximum fs_cpg */
 92
 93  /* MAXFRAG is the maximum number of fragments per block */
 94  #define MAXFRAG         8
 95
 96  #ifndef NBBY
 97  #define NBBY            8       /* number of bits in a byte     */
 98                                  /* NOTE: this is also defined   */
 99                                  /* in param.h.  So if NBBY gets */
100                                  /* changed, change it in        */
101                                  /* param.h also                 */
102  #endif
103
104  /*
105   * The path name on which the file system is mounted is maintained
106   * in fs_fsmnt. MAXMNTLEN defines the amount of space allocated in
107   * the super block for this name.
108   * The limit on the amount of summary information per file system
109   * is defined by MAXCSBUFS. It is currently parameterized for a
110   * maximum of two million cylinders.
111   */
112  #define MAXMNTLEN 512
```

```
113  #define MAXCSBUFS 32
114
115  /*
116   * Per cylinder group information; summarized in blocks allocated
117   * from first cylinder group data blocks.  These blocks have to be
118   * read in from fs_csaddr (size fs_cssize) in addition to the
119   * super block.
120   *
121   * N.B. sizeof(struct csum) must be a power of two in order for
122   * the ``fs_cs'' macro to work (see below).
123   */
124  struct csum {
125          long    cs_ndir;        /* number of directories */
126          long    cs_nbfree;      /* number of free blocks */
127          long    cs_nifree;      /* number of free inodes */
128          long    cs_nffree;      /* number of free frags */
129  };
130
131  /*
132   * Super block for a file system.
133   */
134  #define FS_MAGIC        0x011954
135
136  /*
137   * Magic number for file system allowing long file names.
138   */
139  #define FS_MAGIC_LFN    0x095014
140
141  /*
142   * Magic number for file systems which have their fs_featurebits field
143   * set up.
144   */
145  #define FD_FSMAGIC      0x195612
146
147  /*
148   * Flags for fs_featurebits field.
149   */
150  #define FSF_LFN         0x1     /* long file names */
151  #define FSF_KNOWN       (FSF_LFN)
152  #define FSF_UNKNOWN(bits) ((bits) & ~(FSF_KNOWN))
153
154  /*
155   * Quick check to see if inode is in a file system allowing
156   * long file names.
157   */
158  #define IS_LFN_FS(ip) \
159      (((ip)->i_fs->fs_magic == FS_MAGIC_LFN) || \
160      ((ip)->i_fs->fs_featurebits & FSF_LFN))
161
162  #define FS_CLEAN        0x17
163  #define FS_OK           0x53
164  #define FS_NOTOK        0x31
165
166  /* fs_flags fields */
167  #define FS_INSTALL      0x80
168  #define FS_QCLEAN       0x01
```

```
169  #define FS_QOK          0x02
170  #define FS_QNOTOK       0x03
171  #define FS_QMASK        0x03
172  #define FS_QFLAG(p)     ((p)->fs_flags & FS_QMASK)
173  #define FS_QSET(p,val)  ((p)->fs_flags &= ~FS_QMASK, (p)->fs_flags |= (val)
174
175  /* Mirstate describes the mirror states of the root and primary swap */
176  /* devices.  This information is only recorded in the super block of */
177  /* the root file system.  If root and swap devices are mirrored, the */
178  /* bootup code will configure their states based on mirstate.        */
179
180  struct mirinfo {
181          struct mirstate {               /* mirror states for root and swap
182                  u_int   root:4,         /* root mirror states */
183                          rflag:1,        /* root clean/unconf flag */
184                          swap:4,         /* swap mirror states */
185                          sflag:1,        /* swap clean/unconf flag */
186                          spare:22;       /* spare bits */
187          } state;
188          long mirtime;                   /* mirror time stamp */
189  };
190
191  struct  fs
192  {
193          struct  fs *fs_link;            /* linked list of file systems */
194          struct  fs *fs_rlink;           /*     used for incore super blocks
195          daddr_t fs_sblkno;              /* addr of super-block in filesys *
196          daddr_t fs_cblkno;              /* offset of cyl-block in filesys *
197          daddr_t fs_iblkno;              /* offset of inode-blocks in filesy
198          daddr_t fs_dblkno;              /* offset of first data after cg */
199          long    fs_cgoffset;            /* cylinder group offset in cylinde
200          long    fs_cgmask;              /* used to calc mod fs_ntrak */
201          time_t  fs_time;                /* last time written */
202          long    fs_size;                /* number of blocks in fs */
203          long    fs_dsize;               /* number of data blocks in fs */
204          long    fs_ncg;                 /* number of cylinder groups */
205          long    fs_bsize;               /* size of basic blocks in fs */
206          long    fs_fsize;               /* size of frag blocks in fs */
207          long    fs_frag;                /* number of frags in a block in fs
208  /* these are configuration parameters */
209          long    fs_minfree;             /* minimum percentage of free block
210          long    fs_rotdelay;            /* num of ms for optimal next block
211          long    fs_rps;                 /* disk revolutions per second */
212  /* these fields can be computed from the others */
213          long    fs_bmask;               /* ``blkoff'' calc of blk offsets *
214          long    fs_fmask;               /* ``fragoff'' calc of frag offsets
215          long    fs_bshift;              /* ``lblkno'' calc of logical blkno
216          long    fs_fshift;              /* ``numfrags'' calc number of frag
217  /* these are configuration parameters */
218          long    fs_maxcontig;           /* max number of contiguous blks */
219          long    fs_maxbpg;              /* max number of blks per cyl group
220  /* these fields can be computed from the others */
221          long    fs_fragshift;           /* block to frag shift */
222          long    fs_fsbtodb;             /* fsbtodb and dbtofsb shift consta
223          long    fs_sbsize;              /* actual size of super block */
224          long    fs_csmask;              /* csum block offset */
```

```
225         long    fs_csshift;             /* csum block number */
226         long    fs_nindir;              /* value of NINDIR */
227         long    fs_inopb;               /* value of INOPB */
228         long    fs_nspf;                /* value of NSPF */
229         long    fs_id[2];               /* file system id */
230         struct  mirinfo fs_mirror;      /* mirror states of root/swap */
231         long    fs_featurebits;         /* feature bit flags */
232         long    fs_optim;               /* optimization preference - see be
233 /* sizes determined by number of cylinder groups and their sizes */
234         daddr_t fs_csaddr;              /* blk addr of cyl grp summary area
235         long    fs_cssize;              /* size of cyl grp summary area */
236         long    fs_cgsize;              /* cylinder group size */
237 /* these fields should be derived from the hardware */
238         long    fs_ntrak;               /* tracks per cylinder */
239         long    fs_nsect;               /* sectors per track */
240         long    fs_spc;                 /* sectors per cylinder */
241 /* this comes from the disk driver partitioning */
242         long    fs_ncyl;                /* cylinders in file system */
243 /* these fields can be computed from the others */
244         long    fs_cpg;                 /* cylinders per group */
245         long    fs_ipg;    for lab 2.   /* inodes per group */
246         long    fs_fpg;                 /* blocks per group * fs_frag */
247 /* this data must be re-computed after crashes */
248         struct  csum fs_cstotal;        /* cylinder summary information */
249 /* these fields are cleared at mount time */
250         char    fs_fmod;                /* super block modified flag */
251         char    fs_clean;               /* file system is clean flag */
252         char    fs_ronly;               /* mounted read-only flag */
253         char    fs_flags;               /* currently unused flag */
254         char    fs_fsmnt[MAXMNTLEN];    /* name mounted on */
255 /* these fields retain the current block allocation info */
256         long    fs_cgrotor;             /* last cg searched */
257         struct  csum *fs_csp[MAXCSBUFS];/* list of fs_cs info buffers */
258         long    fs_cpc;                 /* cyl per cycle in postbl */
259         short   fs_postbl[MAXCPG][NRPOS];/* head of blocks for each rotatio
260         long    fs_magic;               /* magic number */
261         char    fs_fname[6];            /* file system name */
262         char    fs_fpack[6];            /* file system pack name */
263         u_char  fs_rotbl[1];            /* list of blocks for each rotation
264 /* actually longer */
265 };
266 /*
267  * Preference for optimization.
268  */
269 #define FS_OPTTIME      0       /* minimize allocation time */
270 #define FS_OPTSPACE     1       /* minimize disk fragmentation */
271
272 /*
273  * Convert cylinder group to base address of its global summary info.
274  *
275  * N.B. This macro assumes that sizeof(struct csum) is a power of two.
276  */
277 #define fs_cs(fs, indx) \
278         fs_csp[(indx) >> (fs)->fs_csshift][(indx) & ~(fs)->fs_csmask]
279
280 /*
```

```
281    * MAXBPC bounds the size of the rotational layout tables and
282    * is limited by the fact that the super block is of size SBSIZE.
283    * The size of these tables is INVERSELY proportional to the block
284    * size of the file system. It is aggravated by sector sizes that
285    * are not powers of two, as this increases the number of cylinders
286    * included before the rotational pattern repeats (fs_cpc).
287    * Its size is derived from the number of bytes remaining in (struct fs)
288    */
289   #define MAXBPC  (SBSIZE - sizeof (struct fs))
290
291   /*
292    * Cylinder group block for a file system.
293    */
294   #define CG_MAGIC                0x090255
295   struct  cg {
296           struct  cg *cg_link;            /* linked list of cyl groups */
297           struct  cg *cg_rlink;           /*     used for incore cyl groups *
298           time_t  cg_time;                /* time last written */
299           long    cg_cgx;                 /* we are the cgx'th cylinder group
300           short   cg_ncyl;                /* number of cyl's this cg */
301           short   cg_niblk;               /* number of inode blocks this cg *
302           long    cg_ndblk;               /* number of data blocks this cg */
303           struct  csum cg_cs;             /* cylinder summary information */
304           long    cg_rotor;               /* position of last used block */
305           long    cg_frotor;              /* position of last used frag */
306           long    cg_irotor;              /* position of last used inode */
307           long    cg_frsum[MAXFRAG];      /* counts of available frags */
308           long    cg_btot[MAXCPG];        /* block totals per cylinder */
309           short   cg_b[MAXCPG][NRPOS];    /* positions of free blocks */
310           char    cg_iused[MAXIPG/NBBY];  /* used inode map */
311           long    cg_magic;               /* magic number */
312           u_char  cg_free[1];             /* free block map */
313   /* actually longer */
314   };
315
316   /*
317    * MAXBPG bounds the number of blocks of data per cylinder group,
318    * and is limited by the fact that cylinder groups are at most one block.
319    * Its size is derived from the size of blocks and the (struct cg) size,
320    * by the number of remaining bits.
321    */
322   #define MAXBPG(fs) \
323           (fragstoblks((fs), (NBBY * ((fs)->fs_bsize - (sizeof (struct cg))))
324
325   /*
326    * Turn file system block numbers into disk block addresses.
327    * This maps file system blocks to device size blocks.
328    */
329   #define fsbtodb(fs, b)  ((b) << (fs)->fs_fsbtodb)
330   #define dbtofsb(fs, b)  ((b) >> (fs)->fs_fsbtodb)
331
332   /*
333    * Cylinder group macros to locate things in cylinder groups.
334    * They calc file system addresses of cylinder group data structures.
335    */
336   #define cgbase(fs, c)   ((daddr_t)((fs)->fs_fpg * (c)))
```

useful for lab?

```
337  #define cgstart(fs, c) \
338          (cgbase(fs, c) + (fs)->fs_cgoffset * ((c) & ~((fs)->fs_cgmask)))
339  #define cgsblock(fs, c) (cgstart(fs, c) + (fs)->fs_sblkno)      /* super bl
340  #define cgtod(fs, c)    (cgstart(fs, c) + (fs)->fs_cblkno)      /* cg block
341  #define cgimin(fs, c)   (cgstart(fs, c) + (fs)->fs_iblkno)      /* inode bl
342  #define cgdmin(fs, c)   (cgstart(fs, c) + (fs)->fs_dblkno)      /* 1st data
343
344  /*
345   * Give cylinder group number for a file system block.
346   * Give cylinder group block number for a file system block.
347   */
348  #define dtog(fs, d)     ((d) / (fs)->fs_fpg)
349  #define dtogd(fs, d)    ((d) % (fs)->fs_fpg)
350
351  /*
352   * Extract the bits for a block from a map.
353   * Compute the cylinder and rotational position of a cyl block addr.
354   */
355  #define blkmap(fs, map, loc) \
356      (((map)[(loc) / NBBY] >> ((loc) & (NBBY-1))) & (0xff >> (NBBY - (fs)->f
357  #define cbtocylno(fs, bno) \
358          ((bno) * NSPF(fs) / (fs)->fs_spc)
359  #define cbtorpos(fs, bno) \
360          ((bno) * NSPF(fs) % (fs)->fs_nsect * NRPOS / (fs)->fs_nsect)
361
362  /*
363   * The following macros optimize certain frequently calculated
364   * quantities by using shifts and masks in place of divisions
365   * modulos and multiplications.
366   */
367  #define blkoff(fs, loc)         /* calculates (loc % fs->fs_bsize) */ \
368          ((loc) & ~(fs)->fs_bmask)
369  #define fragoff(fs, loc)        /* calculates (loc % fs->fs_fsize) */ \
370          ((loc) & ~(fs)->fs_fmask)
371  #define lblkno(fs, loc)         /* calculates (loc / fs->fs_bsize) */ \
372          ((loc) >> (fs)->fs_bshift)
373  #define numfrags(fs, loc)       /* calculates (loc / fs->fs_fsize) */ \
374          ((loc) >> (fs)->fs_fshift)
375  #define blkroundup(fs, size)    /* calculates roundup(size, fs->fs_bsize) *
376          (((size) + (fs)->fs_bsize - 1) & (fs)->fs_bmask)
377  #define fragroundup(fs, size)   /* calculates roundup(size, fs->fs_fsize) *
378          (((size) + (fs)->fs_fsize - 1) & (fs)->fs_fmask)
379  #define fragstoblks(fs, frags)  /* calculates (frags / fs->fs_frag) */ \
380          ((frags) >> (fs)->fs_fragshift)
381  #define blkstofrags(fs, blks)   /* calculates (blks * fs->fs_frag) */ \
382          ((blks) << (fs)->fs_fragshift)
383  #define fragnum(fs, fsb)        /* calculates (fsb % fs->fs_frag) */ \
384          ((fsb) & ((fs)->fs_frag - 1))
385  #define blknum(fs, fsb)         /* calculates rounddown(fsb, fs->fs_frag) *
386          ((fsb) &~ ((fs)->fs_frag - 1))
387
388  /*
389   * Determine the number of available frags given a
390   * percentage to hold in reserve
391   */
392  #define freespace(fs, percentreserved) \
```

```
393             (blkstofrags((fs), (fs)->fs_cstotal.cs_nbfree) + \
394             (fs)->fs_cstotal.cs_nffree - ((fs)->fs_dsize * (percentreserved) /
395
396     /*
397      * Determining the size of a file block in the file system.
398      */
399     #define blksize(fs, ip, lbn) \
400             (((lbn) >= NDADDR || (ip)->i_size >= ((lbn) + 1) << (fs)->fs_bshift
401                 ? (fs)->fs_bsize \
402                 : (fragroundup(fs, blkoff(fs, (ip)->i_size))))
403     #define dblksize(fs, dip, lbn) \
404             (((lbn) >= NDADDR || (dip)->di_size >= ((lbn) + 1) << (fs)->fs_bshi
405                 ? (fs)->fs_bsize \
406                 : (fragroundup(fs, blkoff(fs, (dip)->di_size))))
407
408     /*
409      * Number of disk sectors per block; assumes DEV_BSIZE byte sector size.
410      */
411     #define NSPB(fs)        ((fs)->fs_nspf << (fs)->fs_fragshift)
412     #define NSPF(fs)        ((fs)->fs_nspf)
413
414     #endif /* not _SYS_FS_INCLUDED */
```

 on disk inodes

```
 1  /* @(#) $Revision: 1.14.61.3 $ */
 2  /* $Source: /ws_src/sys.UDL_MERGE_800/ufs/RCS/ino.h,v $
 3   * $Revision: 1.14.61.3 $       $Author: rsh $
 4   * $State: Exp $         $Locker:  $
 5   * $Date: 91/11/19 11:21:14 $
 6   */
 7  #ifndef _SYS_INO_INCLUDED /* allows multiple inclusion */
 8  #define _SYS_INO_INCLUDED
 9
10  struct dinode {
11          union {
12                  struct  icommon di_icom;
13                  char    di_size[128];
14          } di_un;
15  };
16
17  struct cinode {
18          union {
19                  struct  icont   ci_icont;
20                  char    ci_size[128];
21          } ci_un;
22  };
23
24  #define di_ic           di_un.di_icom
25  #define di_mode         di_ic.ic_mode
26  #define di_nlink        di_ic.ic_nlink
27  #define di_uid          di_ic.ic_uid
28  #define di_gid          di_ic.ic_gid
29  #define di_size         di_ic.ic_size.val[1]
30  #define di_db           di_ic.ic_un2.ic_reg.ic_db
31  #define di_ib           di_ic.ic_un2.ic_reg.ic_un.ic_ib
32  #define di_atime        di_ic.ic_atime
33  #define di_mtime        di_ic.ic_mtime
34  #define di_ctime        di_ic.ic_ctime
35  #define di_symlink      di_ic.ic_un2.ic_symlink
36  #define di_flags        di_ic.ic_flags
37  #define di_rdev         di_ic.ic_un2.ic_reg.ic_db[0]
38  #define di_pseudo       di_ic.ic_un2.ic_reg.ic_db[1]
39  #define di_rsite        di_ic.ic_un2.ic_reg.ic_db[2]
40  #define di_blocks       di_ic.ic_blocks
41  #define di_gen          di_ic.ic_gen
42  #define di_fversion     di_ic.ic_fversion
43  #define di_frptr        di_ic.ic_un2.ic_reg.ic_un.ic_fifo.if_frptr
44  #define di_fwptr        di_ic.ic_un2.ic_reg.ic_un.ic_fifo.if_fwptr
45  #define di_frcnt        di_ic.ic_un2.ic_reg.ic_un.ic_fifo.if_frcnt
46  #define di_fwcnt        di_ic.ic_un2.ic_reg.ic_un.ic_fifo.if_fwcnt
47  #define di_fflag        di_ic.ic_un2.ic_reg.ic_un.ic_fifo.if_fflag
48  #define di_fifosize     di_ic.ic_un2.ic_reg.ic_un.ic_fifo.if_fifosize
49  #define di_contin       di_ic.ic_contin
50
51  #define ci_ic           ci_un.ci_icont
52  #define ci_mode         ci_ic.icc_mode
53  #define ci_nlink        ci_ic.icc_nlink
54  #define ci_acl          ci_un.ci_icont.icc_acl
55
56  #endif /* _SYS_INO_INCLUDED */
```

 incore version of inode

```
 1  /* @(#) $Revision: 1.37.61.13 $ */
 2  /* $Source: /ws_src/sys.UDL_MERGE_800/ufs/RCS/inode.h,v $
 3   * $Revision: 1.37.61.13 $        $Author: smp $
 4   * $State: Exp $          $Locker:  $
 5   * $Date: 92/05/04 09:28:13 $
 6   */
 7  #ifndef _SYS_INODE_INCLUDED /* allows multiple inclusion */
 8  #define _SYS_INODE_INCLUDED
 9
10  #ifndef _SYS_STDSYMS_INCLUDED
11  #    include <sys/stdsyms.h>
12  #endif   /* _SYS_STDSYMS_INCLUDED  */
13
14  /*
15   * The I node is the focus of all file activity in UNIX.
16   * There is a unique inode allocated for each active file,
17   * each current directory, each mounted-on file, text file, and the root.
18   * An inode is 'named' by its dev/inumber pair. (iget/iget.c)
19   * Data in icommon is read in from permanent inode on volume.
20   */
21
22  #include <sys/sem_beta.h>
23
24  #ifndef SITEARRAYSIZE
25  #include <sys/sitemap.h>
26  #endif /* SITEARRAYSIZE */
27
28  #include <sys/vnode.h>
29
30  #include <sys/acl.h>
31
32  #define NDADDR  12               /* direct addresses in inode */
33  #define NIADDR  3                /* indirect addresses in inode */
34                                   /* fifo's depends on this value */
35                                   /* if this value changes, look */
36                                   /* at icommon.ic_un2.ic_reg.ic_un */
37
38  /*
39   * Fast symlinks --
40   *    symbolic links with paths short than MAX_FASTLINK_SIZE
41   *    are stored in the inode where the direct and indirect
42   *    block pointers are normally stored.  The flag IC_FASTLINK
43   *    (in i_flags) indicates that the symbolic link is of the
44   *    "fast" variety.
45   *
46   * This implementation cannot change, or the filesystem will
47   * not be compatible with the OSF/1 "ufs" filesystem.
48   */
49  #define MAX_FASTLINK_SIZE ((NDADDR + NIADDR) * sizeof (daddr_t))
50  #define IC_FASTLINK     0x00000001
51
52  struct inode {
53          struct  inode *i_chain[2];      /* must be first */
54          struct  vnode i_vnode;  /* vnode associated with this inode */
55          struct  vnode *i_devvp; /* vnode for block i/o */
56          u_int   i_flag;
```

```
 57         dev_t   i_dev;          /* device where inode resides */
 58         ino_t   i_number;       /* i number, 1-to-1 with device address */
 59         int     i_diroff;       /* offset in dir, where we found last entry
 60         struct  inode *i_contip; /* pointer to the continuation inode */
 61         struct  fs *i_fs;       /* file sys associated with this inode */
 62         struct  duxfs *i_dfs;
 63         struct  dquot *i_dquot; /* quota structure controlling this file */
 64
 65 /*  Put the i_rdev here so the remote device stuff can change it
 66  *  and still have the real device number around
 67  */
 68         dev_t   i_rdev;         /* if special, the device number */
 69
 70         union {
 71                 daddr_t if_lastr;       /* last read (read-ahead) */
 72                 struct  socket *is_socket;
 73         } i_un;
 74         struct  {
 75                 struct inode  *if_freef;        /* free list forward */
 76                 struct inode **if_freeb;        /* free list back */
 77         } i_fr;
 78         struct i_select {
 79                 struct proc *i_selp;
 80                 short i_selflag;
 81         } i_fselr, i_fselw;
 82         struct locklist *i_locklist;    /* locked region list */
 83         struct sitemap i_opensites;     /* map of sites with file open */
 84         struct sitemap i_writesites;    /* map of sites writing to file */
 85         site_t i_ilocksite;             /* site holding ilock */
 86         short   i_pid;          /* pid of last process to lock this inode *
 87         union
 88         {
 89         struct sitemap is_execsites;    /* map of sites executing the file
 90         struct sitemap is_fifordsites;  /* map of sites reading fifo */
 91         } i_siteu;
 92 #define i_execsites i_siteu.is_execsites
 93 #define i_fifordsites i_siteu.is_fifordsites
 94         struct dcount i_execdcount;     /* # of local process exec the file
 95         struct dcount i_refcount;       /* real and virtual reference count
 96         struct sitemap i_refsites;      /* all other references */
 97         struct mount *i_mount;          /* mount table entry
 98                                          * note this can be calculated as:
 99                                          * (struct mount *)
100                                          *      (ITOV(ip)->v_vfsp->v_data)
101                                          * but since this is a relatively
102                                          * frequent operation in DUX, we
103                                          * save it here to make it more
104                                          * efficient.
105                                          */
106         union
107         {
108             struct icommon
109             {
110                 u_short ic_mode;        /*  0: mode and type of file */
111                 short   ic_nlink;       /*  2: number of links to file */
112                 ushort  ic_uid;         /*  4: owner's user id */
```

```
113                 ushort  ic_gid;         /*  6: owner's group id */
114                 quad    ic_size;        /*  8: number of bytes in file */
115  #ifdef _KERNEL
116                 struct timeval ic_atime;/* 16: time last accessed */
117                 struct timeval ic_mtime;/* 24: time last modified */
118                 struct timeval ic_ctime;/* 32: last time inode changed */
119  #else
120                 time_t  ic_atime;       /* 16: time last accessed */
121                 long    ic_atspare;
122                 time_t  ic_mtime;       /* 24: time last modified */
123                 long    ic_mtspare;
124                 time_t  ic_ctime;       /* 32: last time inode changed */
125                 long    ic_ctspare;
126  #endif /* _KERNEL */
127                 union {
128                     struct {
129                         daddr_t ic_db[NDADDR];  /* 40: disk block addresses
130                         union {
131                             daddr_t ic_ib[NIADDR]; /* 88: indirect blocks *
132                             struct ic_fifo
133                             {
134                                 short if_frptr;
135                                 short if_fwptr;
136                                 short if_frcnt;
137                                 short if_fwcnt;
138                                 short if_fflag;
139                                 short if_fifosize;
140                             } ic_fifo;
141                         } ic_un;
142                     } ic_reg;
143                     char ic_symlink[MAX_FASTLINK_SIZE]; /* 40: short symlin
144                 } ic_un2;
145
146                 long    ic_flags;       /* 100: status */
147                 long    ic_blocks;      /* 104: blocks actually held */
148                 long    ic_gen;         /* 108: generation number */
149                 long    ic_fversion;    /* 112: file version number */
150                 long    ic_spare[2];    /* 116: reserved, currently unused
151                 ino_t   ic_contin;      /* 124: continuation inode number *
152             } i_ic;
153             struct icont
154             {
155                 ushort  icc_mode;
156                 short   icc_nlink;      /*  2: number of links to file */
157                                         /* 4: The optional entries of the
158                                          * access control list
159                                          */
160  #ifdef _KERNEL
161                 struct  acl_tuple icc_acl[NOPTTUPLES];
162  #else /* not _KERNEL */
163                 struct  acl_entry_internal icc_acl[NOPTENTRIES];
164  #endif /* else not _KERNEL */
165                 char    icc_spare[46];  /* 82: currently unused */
166             } i_icc;
167         } i_icun;
168  #ifdef HPNSE
```

```
169             struct stdata *i_sptr;  /* HP-UX NSE, associated stream */
170     #endif
171             unsigned char i_ord_flags;      /* copied to buf for ordered writes
172     };
173
174     #define L_REMOTE 0x1    /*  The process holding the lock is remote      */
175
176     /* NOTE: Watch out for IWANT = 0x10, which is also used as a lock flag */
177     #define NFS_WANTS_LOCK 0x2      /* NFS lock manager is waiting for lock */
178
179     struct  locklist
180     {
181          /* NOTE link must be first in struct */
182          struct  locklist *ll_link;     /* link to next lock region */
183          short   ll_count;              /* reference count */
184          short   ll_flags;              /* current flags: L_REMOTE, IWANT, ILB
185          union
186           { struct  proc *llu_proc;         /* process which owns region */
187             struct
188               { site_t llur_psite;          /*  Site where process lives    */
189                 short  llur_pid;            /*  PID of process              */
190               } llu_remote;
191           } ll_u;
192
193     #define ll_proc  ll_u.llu_proc
194     #define ll_psite ll_u.llu_remote.llur_psite
195     #define ll_pid   ll_u.llu_remote.llur_pid
196          off_t   ll_start;              /* starting offset */
197          off_t   ll_end;                /* ending offset, zero is eof */
198          short   ll_type;               /* type of lock (for fnctl) */
199          struct inode *ll_ip;           /* Inode owning this locklist */
200     };
201     enum lockf_type {L_LOCKF, L_READ, L_WRITE, L_COPEN, L_FCNTL};
202
203     #define i_mode          i_icun.i_ic.ic_mode
204     #define i_nlink         i_icun.i_ic.ic_nlink
205     #define i_uid           i_icun.i_ic.ic_uid
206     #define i_gid           i_icun.i_ic.ic_gid
207     #define i_size          i_icun.i_ic.ic_size.val[1]
208     #define i_db            i_icun.i_ic.ic_un2.ic_reg.ic_db
209     #define i_ib            i_icun.i_ic.ic_un2.ic_reg.ic_un.ic_ib
210     #define i_atime         i_icun.i_ic.ic_atime
211     #define i_mtime         i_icun.i_ic.ic_mtime
212     #define i_ctime         i_icun.i_ic.ic_ctime
213     #define i_symlink       i_icun.i_ic.ic_un2.ic_symlink
214     #define i_flags         i_icun.i_ic.ic_flags
215     #define i_blocks        i_icun.i_ic.ic_blocks
216     /*  Define 1) new name for real device number 2) name for device site # */
217     #define i_device        i_icun.i_ic.ic_un2.ic_reg.ic_db[0]
218     #define i_rsite         i_icun.i_ic.ic_un2.ic_reg.ic_db[2]
219     #define i_gen           i_icun.i_ic.ic_gen
220     #define i_lastr         i_un.if_lastr
221     #define i_socket        i_un.is_socket
222     #define i_forw          i_chain[0]
223     #define i_back          i_chain[1]
224     #define i_freef         i_fr.if_freef
```

```
225 #define i_freeb        i_fr.if_freeb
226 #define i_frptr        i_icun.i_ic.ic_un2.ic_reg.ic_un.ic_fifo.if_frptr
227 #define i_fwptr        i_icun.i_ic.ic_un2.ic_reg.ic_un.ic_fifo.if_fwptr
228 #define i_frcnt        i_icun.i_ic.ic_un2.ic_reg.ic_un.ic_fifo.if_frcnt
229 #define i_fwcnt        i_icun.i_ic.ic_un2.ic_reg.ic_un.ic_fifo.if_fwcnt
230 #define i_fflag        i_icun.i_ic.ic_un2.ic_reg.ic_un.ic_fifo.if_fflag
231 #define i_fifosize     i_icun.i_ic.ic_un2.ic_reg.ic_un.ic_fifo.if_fifosize
232 #define i_fifo         i_icun.i_ic.ic_un2.ic_reg.ic_un.ic_fifo
233 #define i_fversion     i_icun.i_ic.ic_fversion
234
235 #define i_contin       i_icun.i_ic.ic_contin
236 #define i_acl          i_icun.i_icc.icc_acl
237
238
239 /*
240  * Only include ino.h if we are defining _KERNEL.  No need otherwise.
241  */
242 #ifdef _KERNEL
243 #include <sys/ino.h>
244 #endif /* _KERNEL */
245
246 #ifdef _KERNEL
247 #ifdef  __hp9000s800
248 extern  struct inode *inode;            /* the inode table itself */
249 extern  struct inode *inodeNINODE;      /* the end of the inode table */
250 extern  int     ninode;                 /* number of slots in the table */
251
252 extern struct vnodeops ufs_vnodeops;    /* vnode operations for ufs */
253 extern struct vnodeops dux_vnodeops;    /* vnode operations for dux */
254
255 extern  struct  vnode *rootdir;         /* pointer to inode of root directo
256 extern struct locklist locklist[]; /* The lock table itself */
257 #endif /* __hp9000s800 */
258
259 #ifdef __hp9000s300
260 struct inode *inode;            /* the inode table itself */
261 struct inode *inodeNINODE;      /* the end of the inode table */
262 int     ninode;                 /* number of slots in the table */
263 extern struct vnodeops ufs_vnodeops;    /* vnode operations for ufs */
264 extern struct vnodeops dux_vnodeops;    /* vnode operations for dux */
265
266 struct  vnode *rootdir;                 /* pointer to inode of root directo
267 struct locklist locklist[]; /* The lock table itself */
268 #endif /* __hp9000s300 */
269
270 struct  inode *ialloc();
271 struct  inode *iget();
272 struct  inode *ifind();
273 struct  inode *owner();
274 struct  inode *maknode();
275 struct  inode *namei();
276
277 ino_t   dirpref();
278 #endif /* _KERNEL */
279
280 /* flags */
```

```
281  #define ILOCKED         0x1             /* inode is locked */
282  #define IUPD            0x2             /* file has been modified */
283  #define IACC            0x4             /* inode access time to be updated
284  #ifdef  notdef
285  #define IMOUNT          0x8             /* inode is mounted on */
286  #endif
287  #define IWANT           0x10            /* some process waiting on lock */
288  #define ITEXT           0x20            /* inode is pure text prototype */
289  #define ICHG            0x40            /* inode has been changed */
290  #ifdef  notdef
291  #define ISHLOCK         0x80            /* file has shared lock */
292  #define IEXLOCK         0x100           /* file has exclusive lock */
293  #endif
294  #define ILWAIT          0x200           /* someone waiting on file lock */
295  #define IREF            0x400           /* inode is being referenced */
296                                          /* change is use DUX !!! */
297  #define ILBUSY          0x800           /* lock is not available */
298  #define IRENAME         0x1000          /* this inode is the source of a
299                                             rename operation */
300  #define IACLEXISTS      0x2000          /* An acl exists for this inode */
301
302
303  #define ISYNCLOCKED     0x10000         /* inode locked for synchronization
304  #define ISYNC           0x20000         /* synchronous I/O required */
305  #define IDUXMNT         0x40000         /* inode mounted remotely */
306  #define ISYNCWANT       0x80000         /* a process waiting on ISYNCLOCKED
307  #define IDUXMRT         0x100000        /* root inode of remotely mounted d
308  #define IBUFVALID       0x200000        /* incore buffers presumed valid */
309  #define IPAGEVALID      0x400000        /* incore exec pages presumed valid
310  #define IOPEN           0x800000        /* inode is currently being opened
311
312  #define IFRAG           0x01000000      /* fragment was allocated, must ref
313
314  #define IHARD           0x2000000       /* hardened inode */
315  #define INOFLUSH        0x4000000       /* for iflush */
316
317  #if defined(__hp9000s800) && !defined(_WSIO)
318  #define IF_MI_DEV       0x08000000      /* dev_t has mgr_index already */
319  #else /* __hp9000s800 */
320  #define IF_MI_DEV       0x00000000      /* s200 doesn't have mgr_index */
321  #endif /* __hp9000s800 */
322  #define IFRAGSYNC       0x10000000      /* need synch. frag_fit() */
323
324  /* modes */
325  #define IFMT            0170000         /* type of file */
326  #define IFIFO           0010000         /* fifo */
327  #define IFCHR           0020000         /* character special */
328  #define IFDIR           0040000         /* directory */
329  #define IFBLK           0060000         /* block special */
330  #define IFCONT          0070000         /* continuation inode */
331  #define IFREG           0100000         /* regular */
332  #define IFNWK           0110000         /* network special */
333  #define IFLNK           0120000         /* symbolic link */
334  #define IFSOCK          0140000         /* socket */
335
336  #define ISUID           04000           /* set user id on execution */
```

```
337  #define ISGID           02000           /* set group id on execution */
338  #define IENFMT          02000           /* enforced file locking */
339  #define ISVTX           01000           /* save swapped text even after use
340  #define IREAD           0400            /* read, write, execute permissions
341  #define IWRITE          0200
342  #define IEXEC           0100
343
344  #define IFIR            01              /* fifo read waiting for write flag
345  #define IFIW            02              /* fifo write waiting for read flag
346  #define PIPSIZ          8192            /* fifo buffer size */
347  #define FSEL_COLL       01              /* select collision flag */
348
349  /* for ILOCK and related macros - PA */
350
351  #define DUX_ILOCK(ip)   (ip)->i_ilocksite = u.u_site
352
353  #define NFS_ILOCK(ip)   (ip)->i_pid = u.u_procp->p_pid
354
355  #ifdef QFS
356  #define QFS_ILOCK(ip)   record_lock((int) ip)
357  #define QFS_IUNLOCK(ip) remove_lock((int) ip)
358  #else   /* not QFS */
359  #define QFS_ILOCK(ip)
360  #define QFS_IUNLOCK(ip)
361  #endif  /* not QFS */
362
363  #define ILOCK(ip) { \
364          QFS_ILOCK(ip);  \
365          while ((ip)->i_flag & ILOCKED) { \
366                  (ip)->i_flag |= IWANT; \
367                  sleep((caddr_t)(ip), PINOD); \
368          } \
369          (ip)->i_flag |= ILOCKED; \
370          DUX_ILOCK(ip);  \
371          NFS_ILOCK(ip);  \
372  }
373
374  #define IUNLOCK(ip) { \
375          (ip)->i_flag &= ~ILOCKED; \
376          QFS_IUNLOCK(ip); \
377          if ((ip)->i_flag&IWANT) { \
378                  (ip)->i_flag &= ~IWANT; \
379                  wakeup((caddr_t)(ip)); \
380          } \
381  }
382
383  #ifdef _KERNEL
384  /*
385   * Convert between inode pointers and vnode pointers
386   */
387  #define VTOI(VP)        ((struct inode *)(VP)->v_data)
388  #define ITOV(IP)        ((struct vnode *)&(IP)->i_vnode)
389
390  /*
391   * Convert between vnode types and inode formats
392   */
```

```
393  extern enum vtype       iftovt_tab[];
394  extern int              vttoif_tab[];
395  #define IFTOVT(M)       ((((M)&IFMT) == IFNWK)?VFNWK:((((M)&IFMT) == IFIFO)
396  #define VTTOIF(T)       (vttoif_tab[(int)(T)])
397
398  #define MAKEIMODE(T, M) (VTTOIF(T) | (M))
399
400  #define ESAME (-1)               /* trying to rename linked files (special)
401  #ifdef __hp9000s300
402  #define EREMOVE (-2)             /* "source" file of link removed in the
403                                      middle of operation (happens only
404                                      originate from client)*/
405  #endif /* __hp9000s300 */
406  #ifdef __hp9000s800
407  #define EREMOVE (-2)             /* "source" file of link removed in the
408                                      middle of operation (happens only
409                                      originate from client)*/
410  #endif /* __hp9000s800 */
411  #define ERENAME (-3)             /* the inode being rename'd is in the path
412                                     of another rename operation*/
413  #define EPATHCONF_NONAME (-4)    /* The posix standard says that if a user
414                                     requests an unknown name, it should not
415                                     change errorno but should return an erro
416                                     This indicates that is the case. */
417
418  /*
419   * Check that file is owned by current user or user is su.
420   */
421  /* We can't do a straight comparision of (CR)->cr_uid against (IP)->i_uid.
422   * We also need to check the case where we are NFS, and network root (-2)
423   * and the inode is owned by "nobody" because i_uid is an ushort and -2 is
424   * stored as 65534.
425   */
426  /* name conflict with DIL */
427  #define OWNER_CR(CR, IP) \
428      (((CR)->cr_uid == (IP)->i_uid)? 0: \
429          ((((CR)->cr_uid == -2) && ((IP)->i_uid == (ushort)-2))? 0: \
430              (suser()? 0: u.u_error)))
431
432  /*
433   * enums
434   */
435  enum de_op      { DE_CREATE, DE_LINK, DE_RENAME };      /* direnter ops */
436
437  #endif /* _KERNEL */
438  /*
439   * This overlays the fid structure (see vfs.h).  Used mainly in support
440   * of NFS 3.2 file handles, the fid structure should contain the minimum
441   * information necessary to uniquely identify a file, GIVEN a pointer to
442   * the file system.
443   */
444  struct ufid {
445          u_short ufid_len;
446          ino_t   ufid_ino;
447          long    ufid_gen;
448  };
```

```
 1
 2  /*
 3   * The vnode is the focus of all file activity in UNIX.
 4   * There is a unique vnode allocated for each active file,
 5   * each current directory, each mounted-on file, text file, and the root.
 6   */
 7
 8  /*
 9   * vnode types. VNON means no type.
10   */
11  enum vtype      { VNON, VREG, VDIR, VBLK, VCHR, VLNK, VSOCK, VBAD, VFIFO, VFNW
12  enum vfstype    { VDUMMY, VNFS, VUFS, VDUX, VDUX_PV, VDEV_VN, VNFS_SPEC,
13                          VNFS_BDEV, VNFS_FIFO, VCDFS, VDUX_CDFS, VDUX_CDFS_PV }
14
15  struct vnode {
16          u_short         v_flag;                 /* vnode flags (see below)*/
17          u_short         v_shlockc;              /* count of shared locks */
18          u_short         v_exlockc;              /* count of exclusive locks */
19          u_short         v_tcount;               /* private data for fs */
20          int             v_count;                /* reference count */
21          struct vfs      *v_vfsmountedhere;      /* ptr to vfs mounted here */
22          struct vnodeops *v_op;                  /* vnode operations */
23          struct socket   *v_socket;              /* unix ipc */
24          struct vfs      *v_vfsp;                /* ptr to vfs we are in */
25          enum vtype      v_type;                 /* vnode type */
26          dev_t           v_rdev;                 /* device (VCHR, VBLK) */
27          caddr_t         v_data;                 /* private data for fs */
28          enum vfstype    v_fstype;               /* file system type*/
29          struct vas      *v_vas;                 /* vm data structures */
30          vm_sema_t       v_lock;                 /* vnode lock */
31          struct buf      *v_ord_lastdatalink;    /* for ordered writes */
32          struct buf      *v_ord_lastmetalink;    /* for ordered writes */
33          struct buf      *v_cleanblkhd;          /* clean buffer head */
34          struct buf      *v_dirtyblkhd;          /* dirty buffer head */
35  };
36
37  /*
38   * vnode flags.
39   */
40  #define VROOT           0x01    /* root of its file system */
41  #define VTEXT           0x02    /* vnode is a pure text prototype */
42  #define VEXLOCK         0x10    /* exclusive lock */
43  #define VSHLOCK         0x20    /* shared lock */
44  #define VLWAIT          0x40    /* proc is waiting on shared or excl. lock */
45  #define VMMF            0x100   /* Vnode memory mapped */
46
47
48  /*
49   * Operations on vnodes.
50   */
51
52        struct vnodeops {
53          int     (*vn_open)(__farg);
54          int     (*vn_close)(__farg);
55          int     (*vn_rdwr)(__farg);
56          int     (*vn_ioctl)(__farg);
```

```
 57          int     (*vn_select)(__farg);
 58          int     (*vn_getattr)(__farg);
 59          int     (*vn_setattr)(__farg);
 60          int     (*vn_access)(__farg);
 61          int     (*vn_lookup)(__farg);
 62          int     (*vn_create)(__farg);
 63          int     (*vn_remove)(__farg);
 64          int     (*vn_link)(__farg);
 65          int     (*vn_rename)(__farg);
 66          int     (*vn_mkdir)(__farg);
 67          int     (*vn_rmdir)(__farg);
 68          int     (*vn_readdir)(__farg);
 69          int     (*vn_symlink)(__farg);
 70          int     (*vn_readlink)(__farg);
 71          int     (*vn_fsync)(__farg);
 72          int     (*vn_inactive)(__farg);
 73          int     (*vn_bmap)(__farg);
 74          int     (*vn_strategy)(__farg);
 75          int     (*vn_bread)(__farg);
 76          int     (*vn_brelse)(__farg);
 77          int     (*vn_pathsend)(__farg);
 78          int     (*vn_setacl)(__farg);
 79          int     (*vn_getacl)(__farg);
 80          int     (*vn_pathconf)(__farg);
 81          int     (*vn_fpathconf)(__farg);
 82          /*
 83           * Add VOPs for support NFS 3.2 file locking.  See below for more info
 84           */
 85          int     (*vn_lockctl)(__farg);
 86          int     (*vn_lockf)(__farg);
 87          int     (*vn_fid)(__farg);
 88          int     (*vn_fsctl)(__farg);
 89          int     (*vn_prefill)(__farg);
 90          int     (*vn_pagein)(__farg);
 91          int     (*vn_pageout)(__farg);
 92          int     (*vn_dbddup)(__farg);
 93          int     (*vn_dbddealloc)(__farg);
 94      };
 95
 96
 97  #ifdef _KERNEL
 98
 99  #define VOP_OPEN(VPP,F,C)                (*(*(VPP))->v_op->vn_open)(VPP, F, C)
100  #define VOP_CLOSE(VP,F,C)                (*(VP)->v_op->vn_close)(VP,F,C)
101  #define VOP_RDWR(VP,UIOP,RW,F,C)         (*(VP)->v_op->vn_rdwr)(VP,UIOP,RW,F,C)
102  #define VOP_IOCTL(VP,C,D,F,CR)           (*(VP)->v_op->vn_ioctl)(VP,C,D,F,CR)
103  #define VOP_SELECT(VP,W,C)               (*(VP)->v_op->vn_select)(VP,W,C)
104  /*An additional parameter specifying synchronization has been added to getattr
105  #define VOP_GETATTR(VP,VA,C,S)           (*(VP)->v_op->vn_getattr)(VP,VA,C,S)
106  #define VOP_SETATTR(VP,VA,C,N)           (*(VP)->v_op->vn_setattr)(VP,VA,C,N)
107  #define VOP_ACCESS(VP,M,C)               (*(VP)->v_op->vn_access)(VP,M,C)
108  #define VOP_LOOKUP(VP,NM,VPP,C,MVP)              (*(VP)->v_op->vn_lookup)(VP,NM
109  #define VOP_CREATE(VP,NM,VA,E,M,VPP,C)   (*(VP)->v_op->vn_create) \
110                                                   (VP,NM,VA,E,M,VPP,C)
111  #define VOP_REMOVE(VP,NM,C)              (*(VP)->v_op->vn_remove)(VP,NM,C)
112  #define VOP_LINK(VP,TDVP,TNM,C)          (*(VP)->v_op->vn_link)(VP,TDVP,TNM,C)
```

```
113   #define VOP_RENAME(VP,NM,TDVP,TNM,C)      (*(VP)->v_op->vn_rename) \
114                                                     (VP,NM,TDVP,TNM,C)
115   #define VOP_MKDIR(VP,NM,VA,VPP,C)         (*(VP)->v_op->vn_mkdir)(VP,NM,VA,VPP,C
116   #define VOP_RMDIR(VP,NM,C)                (*(VP)->v_op->vn_rmdir)(VP,NM,C)
117   #define VOP_READDIR(VP,UIOP,C)            (*(VP)->v_op->vn_readdir)(VP,UIOP,C)
118   #define VOP_SYMLINK(VP,LNM,VA,TNM,C)      (*(VP)->v_op->vn_symlink) \
119                                                     (VP,LNM,VA,TNM,C)
120   #define VOP_READLINK(VP,UIOP,C)           (*(VP)->v_op->vn_readlink)(VP,UIOP,C)
121   #define VOP_FSYNC(VP,C, S)                        (*(VP)->v_op->vn_fsync)(VP,C,
122   #define VOP_INACTIVE(VP,C)                (*(VP)->v_op->vn_inactive)(VP,C)
123   #define VOP_BMAP(VP,BN,VPP,BNP)           (*(VP)->v_op->vn_bmap)(VP,BN,VPP,BNP)
124   #define VOP_STRATEGY(BP)                  (*(BP)->b_vp->v_op->vn_strategy)(BP)
125   #define VOP_BREAD(VP,BN,BPP)              (*(VP)->v_op->vn_bread)(VP,BN,BPP)
126   #define VOP_BRELSE(VP,BP)                 (*(VP)->v_op->vn_brelse)(VP,BP)
127   #define VOP_PATHSEND(VPP,PNP,FOLLOW,NLINKP,DIRVPP,COMPVPP,OPCODE,DEPENDENT) \
128           ((*(*(VPP))->v_op->vn_pathsend) ? \
129           (*(*(VPP))->v_op->vn_pathsend) \
130                   (VPP,PNP,FOLLOW,NLINKP,DIRVPP,COMPVPP,OPCODE,DEPENDENT) : \
131                   (panic("VOP_PATHSEND"),EINVAL))
132   #define VOP_SETACL(VP,NT,BP)              (*(VP)->v_op->vn_setacl)(VP,NT,BP)
133   #define VOP_GETACL(VP,NT,BP)              (*(VP)->v_op->vn_getacl)(VP,NT,BP)
134   #define VOP_PATHCONF(VP,NT,BP,CR)         (*(VP)->v_op->vn_pathconf)(VP,NT,BP,CR
135   #define VOP_FPATHCONF(VP,NT,BP,CR)        (*(VP)->v_op->vn_fpathconf)(VP,NT,BP,C
136
137   /*
138    * VOPs for NFS 3.2 file locking.  Ours are different because we support
139    * local file locking already in the kernel.  VOP_LOCKCTL() is called from
140    * fcntl() to process a lock request.  We have an extra parameters because
141    * the lower level routines will need the file structure for the file
142    * being locked.  The Lower Bound and Upper Bound are passed in because the
143    * higher level routine already computed them for error checking.  This means
144    * that ALL functions calling these routines MUST include reasonable values
145    * for LB and UB.   Also, Sun does not have a VOP_LOCKF() because they
146    * emulate lockf() as a library on top of fcntl(), instead of two separate
147    * system calls like ours.
148    */
149   #define VOP_LOCKCTL(VP,LD,CMD,C,FP,LB,UB)   (*(VP)->v_op->vn_lockctl) \
150                                                 (VP,LD,CMD,C,FP,LB,UB)
151   #define VOP_LOCKF(VP,CMD,SIZE,C,FP,LB,UB)       (*(VP)->v_op->vn_lockf) \
152                                                 (VP,CMD,SIZE,C,FP,LB,UB)
153   /*
154    * Support for NFS 3.2 file handles.  Given a vnode pointer, generate
155    * a "file id" which can be used to recreate the vnode later on.
156    */
157   #define VOP_FID(VP, FIDPP)                (*(VP)->v_op->vn_fid)(VP, FIDPP)
158   #define VOP_FSCTL(VP, COMMAND, UIOP, CRED)  (*(VP)->v_op->vn_fsctl) \
159                                                     (VP, COMMAND, UIOP, CRED)
160   #define VOP_PREFILL(VP,PRP)       (*(VP)->v_op->vn_prefill)(PRP)
161   #define VOP_DBDDUP(VP,DBD)        (*(VP)->v_op->vn_dbddup)(VP, DBD)
162   #define VOP_DBDDEALLOC(VP,DBD) \
163           (((VP)->v_op->vn_dbddealloc)?(*(VP)->v_op->vn_dbddealloc)(VP,DBD):1)
164   #define VOP_PAGEOUT(VP,PRP,START,END,FLAGS) \
165           (*(VP)->v_op->vn_pageout)(PRP,START,END,FLAGS)
166
167   #define VOP_PAGEIN(VP,PRP,WRT,SPACE,VADDR,START) \
168           (*(VP)->v_op->vn_pagein)(PRP,WRT,SPACE,VADDR,START)
```

```
169
170  /*
171   * flags for above
172   */
173  #define IO_UNIT         0x01            /* do io as atomic unit for VOP_RDWR *
174  #define IO_APPEND       0x02            /* append write for VOP_RDWR */
175  #define IO_SYNC         0x04            /* sync io for VOP_RDWR */
176
177  #endif /* _KERNEL */
178
179  /*
180   * Vnode attributes.  A field value of -1
181   * represents a field whose value is unavailable
182   * (getattr) or which is not to be changed (setattr).
183   */
184  /*DUX MESSAGE STRUCTURE*/
185  struct vattr {
186          enum vtype      va_type;        /* vnode type (for create) */
187          u_short         va_mode;        /* files access mode and type */
188          u_short         va_uid;         /* owner user id */
189          u_short         va_gid;         /* owner group id */
190          /*moved va_nlink for alignment*/
191          short           va_nlink;       /* number of references to file */
192          long            va_fsid;        /* file system id (dev for now) */
193          long            va_nodeid;      /* node id */
194          u_long          va_size;        /* file size in bytes (quad?) */
195          long            va_blocksize;   /* blocksize preferred for i/o */
196          struct timeval  va_atime;       /* time of last access */
197          struct timeval  va_mtime;       /* time of last modification */
198          struct timeval  va_ctime;       /* time file ``created */
199          dev_t           va_rdev;        /* device the file represents */
200          long            va_blocks;      /* kbytes of disk space held by file *
201          site_t          va_rsite;       /* site the device file represents */
202          site_t          va_fssite;      /* file system site (dev site ) */
203          dev_t           va_realdev;     /* The real devcie number of device
204                                             containing the inode for this file
205          u_short         va_basemode;    /* the base mode bits unaltered */
206          u_short         va_acl:1,       /* set if optional ACL entries */
207                          va_fstype:3,
208                          :12;
209  };
210
211  /*
212   *  Modes. Some values same as Ixxx entries from inode.h for now
213   */
214  #define VSUID   04000           /* set user id on execution */
215  #define VSGID   02000           /* set group id on execution */
216  #define VSVTX   01000           /* save swapped text even after use */
217  #define VREAD   0400            /* read, write, execute permissions */
218  #define VWRITE  0200
219  #define VEXEC   0100
```

J. Wakerly
26 February 1982

The following description has appeared in a number of informal publications of computer users, and has been variously attributed to Jeff Berryman, Bruce VanAtta, and probably others as well. I'm not sure who the original author is, but read, understand, and enjoy.

The Paging Game -- Rules

1. Each player gets several million things.
2. Things are kept in crates that hold 4096 things each. Things in the same crate are called crate-mates.
3. Crates are stored either in the workshop or the warehouse. The workshop is almost always too small to hold all the crates.
4. There is only one workshop but there may be several warehouses. Everyone shares them.
5. Each thing has its own thing number.
6. What you do with a thing is to zark it. Everyone takes turns zarking.
7. You can only zark your things, not anyone else's.
8. Things can only be zarked when they are in the workshop.
9. Only the Thing King knows whether a thing is in the workshop or in a warehouse.
10. The longer a thing goes without being zarked, the grubbier it is said to become.
11. The way you get things is to ask the Thing King. He only gives out things in multiples of eight. This is to keep the royal overhead down.
12. The way you zark a thing is to give its thing number. If you give the number of a thing that happens to be in a workshop it gets zarked right away. If it is in a warehouse, the Thing King packs the crate containing your thing back into the workshop. If there is no room in the workshop, he first finds the grubbiest crate in the workshop, whether it be yours or somebody else's, and packs it off with all its crate-mates to a warehouse. In its place he puts the crate containing your thing. Your thing then gets zarked and you never know that it wasn't in the workshop all along.
13. Each player's stock of things have the same numbers as everybody else's. The Thing King always knows who owns what thing and whose turn it is, so you can't ever accidentally zark somebody else's thing even if it has the same thing number as one of yours.

Notes

1. Traditionally, the Thing King sits at a large, segmented table and is attended to by pages (the so-called "table pages") whose job it is to help the king remember where all the things are and who they belong to.

2. Rules 9 and 12 free players to concentrate on zarking their things, letting the King do the worrying about where the things are located.

3. One consequence of Rule 13 is that everybody's thing numbers will be similar from game to game, regardless of the number of players.

4. The Thing King has a few things of his own, some of which move back and forth between workshop and warehouse just like anybody else's, but some of which are just too heavy to move out of the workshop.

5. With the given set of rules, oft-zarked things tend to get kept mostly in the workshop, while little-zarked things stay mostly in a warehouse. This is efficient stock control.

6. Sometimes even warehouses get full. The Thing King then has to start piling things on the dump out back. This makes the game slower because it takes a long time to get things off of the dump when they are needed in the workshop. A forthcoming change in the rules will allow the Thing King to select the grubbiest things in the warehouses and send them to the dump in his spare time, thus keeping the warehouses from getting too full. This means that the most infrequently-zarked things will end up in the dump so the Thing King won't have to get things from the dump so often. This should speed up the game when there are a lot of players and the warehouses are getting full.

7. Every player is a winner in the paging game despite the apparent autocratic nature of the King.

LONG LIVE THE THING KING!

SE 390: Series 300 HP-UX Internals

Memory Management

Virtual Memory

Why?

- allow for (fairly) efficient stretching of memory
- allow all programs to think they are running by themselves by providing virtual address space for each process

How?

- There will always be swap space reserved for a process' memory; it may or may not have enough physical RAM for all it is doing.
- Pageout daemon kicks out pages if we're running short and they aren't being referenced often enough; swapper kicks out whole processes if we're *really* getting short.
- Virtual address translation
  - 68K
    - 32 bit address
    - 10 bits tell which segment table entry
    - 10 more tell which page table entry (pte)
    - 12 bits for offset into 4k page
    - pte has 20 bit physical address (of 4k page) and has 12 bits left over for protection information, flags, etc.
    - 68040 requires 3-level tables, but the idea is the same.
  - PA
    - system shares *large* virtual address space; each process gets 4 1GB chunks of it;
    - when there is a TLB miss, the system will use the PDIR (reverse page table) to resolve the address

Memory Management

Foundation Principles

- Lots of things will be shared; the VM system should encourage this by making it efficient:
    - copy-on-write - allows for efficient fork(), etc
    - shared libraries; allow sharing of text at granularity of library rather than a.out
- A process address space is nothing more than a bunch of collections of pages (abstracted as pregions/regions).
- Machine independence:
    - the bulk of the VM system is shared between 300/400 and 700/800 - the Hardware-Independent Layer ("HIL").
    - the parts specific to one or the other are well compartmentalized and there are clean interfaces to this code - the Hardware-Dependent Layer ("HDL").
- The bulk of the system should deal in pages, but shouldn't know much about them - all the HIL knows is that pages are NBPG bytes in size and it can get at them via pfdat[].

SE 390: Series 300 HP-UX Internals

Memory Management

Regions

- Regions are the building blocks for the whole VM system.

- A region is a logically contiguous set of pages that are used for *one* thing such as stack, text, shared lib, etc.

- Regions contain (among other things)

    - the type of this region (unused, private, or shared)

    - the number of pages in this region

    - the number of physical pages in this region

    - "disk block descriptors" - tell where the data can be paged/swapped to; one for each page in the region

    - a vnode * that tells which device/filesystem the data in this region came/comes from

    - " " goes to

    (The vnodes tell *which* device/filesystem; the DBDs tell *where* on that device/filesystem.)

Pregions (keep track of which parts of the virtual addr space is in use.

- A pregion can be thought of as a connection between a region and a process.

- Note that in the region data structure there is no place for things like the virtual address at which the region is mapped; this is because regions are system-wide structures, and that sort of information is per-process. To connect regions to processes, we use structures called pregions. Some of the more important fields in a pregion:

    - pointers to the pregions on either side

    - a pointer back to the vas

    - the type of this pregion (text, data, stack, mapped file, I/O, shared memory, etc)

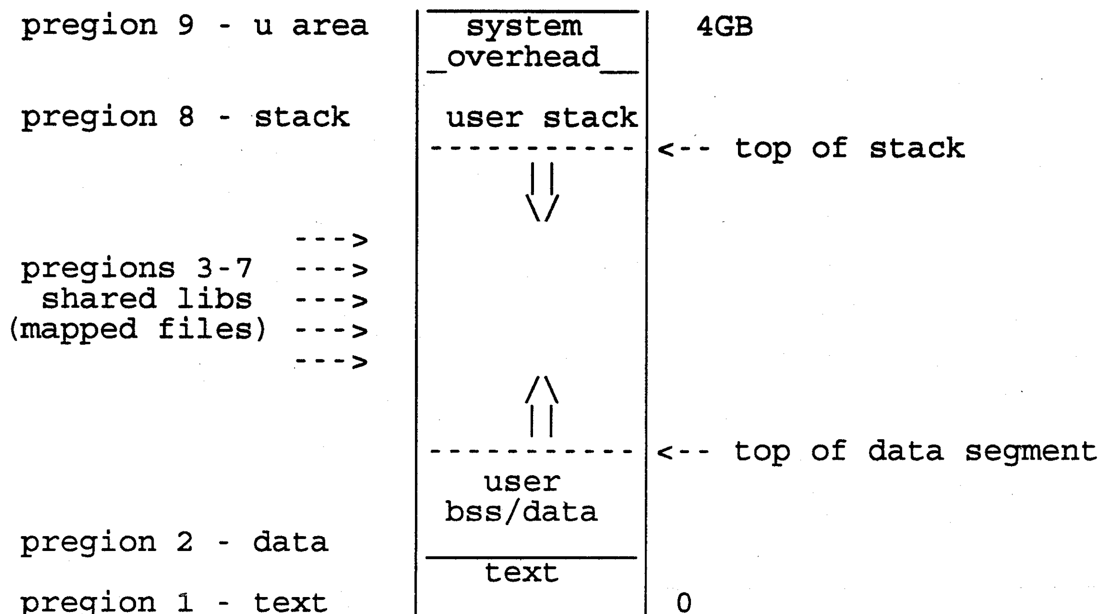
    - the virtual address (in the process' address space) this pregion is mapped to

    - a count of the number of pages this pregion is mapping

    - a pointer to the region

Per-process VM Structures

A process' memory map is represented by something called a "vas" (virtual address space), which is little more than a doubly-linked list of pregions. A typical process will have 4 "normal" pregions as well as some extra ones....



We said above that a process' address space was represented by a "vas". For any process, there is a pointer in its "proc structure" that points to its vas. The vas has several things in it, most notably

- a pair of pointers to a doubly linked list of pregions, sorted by where they are in the process' address space

- a pointer to hardware-dependent structures (such as the segment/page tables for 680x0)

Note the hierarchy: each process has its own vas, which gets us to the pregions, which point at (system-wide) region structures. All of this is for the kernel; the MMU still uses segment and page tables to do (virtual ---> physical) translation.

When To Do What

Available Memory

```
           +---------------------------------------+
           |                                       |
           |                                       |
           |   If the amount of free physical      |
           |   memory stays up here, life is       |
           |   wonderful.  If it falls             |
           |                   down                |
           |                   here,               |
           |                   though,             |
           |                   we're in            |
           |                   trouble...          |
           |                                       |
           |  min(512K, 25% of user memory)        |
lotsfree   +---------------------------------------+
           |  pageout daemon runs below here       |
           |       scans pages and may page        |
           |       a few out                       |
           |                                       |
           |  min(200K, 12.5% of user memory)      |
desfree    +---------------------------------------+
           |  swapper will run below here, and     |
           |       vhand will try harder           |
           |                                       |
           |  min(64K, desfree/2)                  |
minfree    +---------------------------------------+
           |  swapper will force active processes  |
           |  out below here                       |
           |                                       |
           +---------------------------------------+
```

Note: these numbers may change from release to release; the general idea is likely to be around for a while.

The Paging Game

- A (somewhat) graceful way of stretching the amount of available memory.
- Implemented with a clock algorithm:
    - "age hand" goes around at a calculated rate, marking pages by clearing their reference bits
    - if the process accesses the page, the reference bit will be set again
    - if the "steal hand" comes around and the reference bit is still clear, the page is likely to get kicked out
    - if the process accesses a page that has been "kicked out" but hasn't been given to someone else yet, a "soft" page fault occurs and the page can be reclaimed
    - the "hands" only look at active pregions; this way no time is wasted looking at physical pages that can't be paged out (i.e. a driver grabs some memory; that memory can't be paged, so there's no reason for the kernel to look at it)
    - in 8.0 there is a severe problem with this scheme, because if we kick out 20 pages in a row, they probably all came from 1-2 pregions, and those were probably from 1-2 processes :-(*** this is fixed in 9.0 ***
- Speed of hands is calculated to keep overhead <= 10% of CPU time.
- Pageout daemon is process 2; doesn't run at all if more than "lotsfree" memory available.

The pager views memory as if it was around the face of a clock. For our purposes, we'll unroll the clock and look at it as a straight line (numbers above each pregion indicate its size):

```
        50          140          30           40            95
   |  X text  |  X data  |  X stack  |  mwm text  |  mwm data...
```

In 9.0 the pager will look through 1/16 of each pregion's pages at a time, so it will go around the whole "clock" 16 times to visit all of the eligible memory.

The Pageout Daemon ("vhand"; process 2)

```
loop:
        pages_to_scan = maxmem/scanrate/tune.t_vhandr            #1

        if we have plenty of RAM                                 #2
                pages_to_free = 0
        else
                pages_to_free = desfree - freemem

        if pages_to_free > 0
                do
                        look for pageable pregion; if found              #3
                                get _pageout routine from appropriate
                                  filesystem to steal the pages; normally
                                  this will be the "devswap" filesystem

                while we haven't yet stolen pages_to_free pages


        while pages_to_scan > 0

                find an "ageable" pregion (one that's not locked right now)

                clear ref bits for its pages, starting where we left off
                  last time and dropping pages_to_scan appropriately

        goto loop
```

Notes

1. "maxmem" is basically the number of pages the kernel didn't take at boot time; "scanrate" is the number of seconds it should take to go around the clock, assuming that vhand shouldn't take too much of the system's time and that it should run faster/slower depending on how much memory is currently free; "tune.t_vhandr" tells how many times per second to run vhand - it is part of a larger structure that controls the pager's operations

2. The fact that the pager is running means the system is short of memory; how short it is will govern whether we actually steal pages or not

3. The pager doesn't want to know about devices, so it hides behind the vnode layer; when it wants to page out some of a pregion's pages, it calls the filesystem associated with the region; this would normally be the pseudo-filesystem "devswap" (which only has pagein/pageout routines)

Swapping

- A cumbersome way of stretching the amount of available memory.
- Can consume lots of the system's resources.
- Kick out whole process at a time, not just part of it.
- Only happens when we are really worried about the amount of memory available.
- If the swapper runs much at all, the system is underconfigured.
- The basic plan is to kick out junk; if that fixes the problem, we're OK. Only as a last resort will an active process get swapped out.
- Deactivation (new in 9.0)
    - move the process to a priority that the scheduler will ignore (keep it from running, period)
    - let the pager steal its pages
    - swap out the u area & kernel stack, since the pager is not allowed to touch those
    - motivation is to keep from overloading the system with swap traffic (pager is much nicer to system than swapper)

Process 0: The Swapper

```
loop:
        if ((>= 2 runnable procs) and (very short of RAM))
                goto hardswap

        walk through proc table, switching on p_stat {

                case runnable but swapped out:
                        if this guy is the highest priority we've seen
                                remember him

                case sleeping or stopped:
                        if this guy is dead in the water
                                kick him out
        }

        if nobody wants in
                sleep until we're needed
                goto loop

        if it's not critical to bring someone in
                wait awhile
                goto loop
        else
                try to swap most important process in (usually works)
                if it worked, goto loop

hardswap:
        walk through proc table {

                if process isn't swappable or is a zombie
                        skip it

                if (proc. is stopped) or (has slept awhile at int'ible pri)
                        if it has slept longer than anyone we've seen
                                remember it
                else if (don't have sleeper yet) and (it's runnable|asleep)
                        see how big it is
                        if it's one of the biggest we've seen
                                remember it
        }

        if we didn't find a long sleeper
                pick "oldest" big job (based on nice value and time in-core)

        if (found a sleeper) or (desperate and found *someone* to swap) or
           (someone needs in and someone else has been in for awhile) {
                if we're desperate
                        fake like we're still short on memory
                try to swap this guy out (will usually succeed)
                goto loop
        }

        wait awhile and then goto loop
```

Swap Space Allocation/Management

- Space allocation - per region

  W/ c. kalle

  - A page of swap is reserved for each page of the region (assuming it is a data/stack sort of region).

  - The number of pages of swap available to reserve is in a kernel global variable called "swapspc_cnt"; the maximum is in "swapspc_max".

  - Space won't be allocated until we need it; at that point, an address (really indices into the swaptab[]/swapmap[] below) will be put into the DBD for the page.

- Space allocation - shared objects

  - Shared text can be released if no processes are using it; note that it is not swapped; we just arrange to fault it in when it is referenced again.

  - Shared memory can be swapped out if no processes are using it (implying that doing constant shmat(2)/shmdt(2)s is a bad idea).

- Space allocation - system-wide

  - "swaptab" is an array with MAXSWAPCHUNKS entries, each corresponding to 2 MB (default - parameter is named "swchunk" and it defaults to 2048 (1k units)) of swap space.

  - The major component of a swaptab[] entry is an array called "swapmap" - it has an entry in it for each page of space in this chunk.

  - "swdevt" is an array, one element per disk that has swap space on it. It is in /etc/conf/conf.c.

  - If the swap space is spread over >1 disk, the space is taken from equal-priority disks in a round-robin fashion. Device swap is regarded as a higher priority than filesystem swap, for a given priority (e.g. device swap at priority 5 will get used before fs swap at priority 5 which will get used before device swap at 6)

  - Filesystem swap is normally allocated from the filesystem when it is needed and returned when not; exception if system manager specifies a minimum amount to take (and keep).

  - Note that we never guarantee contiguous chunks, but will certainly accept them :-)

Swap Space Allocation/Management

```
        swaptab                                                        swdevt

   +----------------+\                                        +----------------+
   |      2 MB      | \       +------------+         -----|                |
   ------------------   ==>   | 7959; 4 MB | <--/           +----------------+
   |      2 MB      | /       +------------+        /----|                |
   ------------------'                               |      +----------------+
   |       .        |\                               |  /-|                |
   ------------------ \       +------------+         |  |   +----------------+
   |       .        |   ==>   | 7945; 6 MB | <--/       |
   ------------------  /      +------------+            |
   |       .        | /                                 |
   ------------------'        +------------+            |
   |       .        |   ==>   | 7937; 2 MB | <-----/          fswdevt
   +----------------+         +------------+                +----------------+
   |                |\                              /---|                |
   +----------------+ \       +------------+       |      +----------------+
   |                |   ==>   | 7958; fs   | <---/
   +----------------+  /      +------------+
   |                | /
   +----------------+/


       swdev_pri                                                   fswdev_pri

   +----------------+       +----------------+
   | 7937; pri 0    | ---> |   7959; pri 0   |
   +----------------+       +----------------+
                                                          +----------------+
                                                          |  7958; pri 0   |
                                                          +----------------+
   +----------------+
   | 7945; pri 1    |
   +----------------+


       swaptab               each swaptab entry points at a swapmap[],
                                 which is an array of structures,
   +----------------+        1 structure for each page worth of swap
   |      2 MB      | ---> {} {} {} {} {} {} {} {} {} {} {} {} {} {}
   +----------------+
                             {} = use count and
                                  ptr to next free entry
```

In a region, each page will have a VFD and a DBD. When a page has been pushed out to the swap area, the DBD will have an index into the swaptab[] and an index into that entry's swapmap[].

SE 390: Series 300 HP-UX Internals

Memory Management

Important Data Structures

- pfdat - used to keep track of physical memory. There's an entry in it for each page of non-kernel memory. The structure is defined in sys/pfdat.h.

- 68K:
  - Segment table - one for each process. Each table has 1024 entries, each of which can point at a page table (or block table, if 3-level tables are being used). The structure for these tables is in machine/pte.h.

  - Page table - 1024 entries, each of which can point to a 4K page of RAM.

- swdevt[] - an array of structures, one element per disk that has swap on it; the structure contains things like where the swap starts, how many blocks are there, etc. There is a similar structure called "fswdevt" for filesystem swap.

- swaptab[] - an array of structures, one for each 2MB (default) of swap space. It is sized by the kernel parameter MAXSWAPCHUNKS, and each entry points at a swapmap[]...

- swapmap[] - (not related to pre-8.0 swapmap!) - an array that hangs off of a swaptab[] entry; there is an entry in a swapmap[] for each page of swap space in the (by default) 2MB chunk. The entries consist of a use count and a pointer to the next free entry in the swapmap.

- swdev_pri[] - an array of prioritized pointers to swap disks; each disk that is at a particular swap priority has an entry in swdev_pri[that_priority]

- vmmeter and vmtotal - see the respective header files for these structures; they have important summary information that things like top and monitor display

Memory Management

Tunable Parameters

- maxdsiz, maxssiz, maxtsiz - maximum sizes of the respective parts of a process. There is no built-in "cost" for raising these parameters - they are here as sanity checks.
- minswapchunks - minimum amount of swap for a diskless node. It is always allocated to the node (this applies to other systems as well, but is primarily an issue for diskless systems that get their swap from a server).
- maxswapchunks - maximum amount of swap space a system is allowed to allocate; note that this is enforced on the node itself, not by the diskless server ==> each system has its own value
- nswapdev - no. of entries in swdevt[]; if this number is more than the number of "swap..." lines in the dfile, there will be room for dynamic swapon(1m) commands after boot time.
- swchunk - size of chunk in swaptab[] - defaults to 2048 Kbyte which means 2MB
- unlockable_mem - amount of RAM that can not be locked

Note that other parameters (such as nbuf) can have an effect on the VM system (what if nbuf was 1024 on an 8MB system?)

Kernel Variables Of Interest

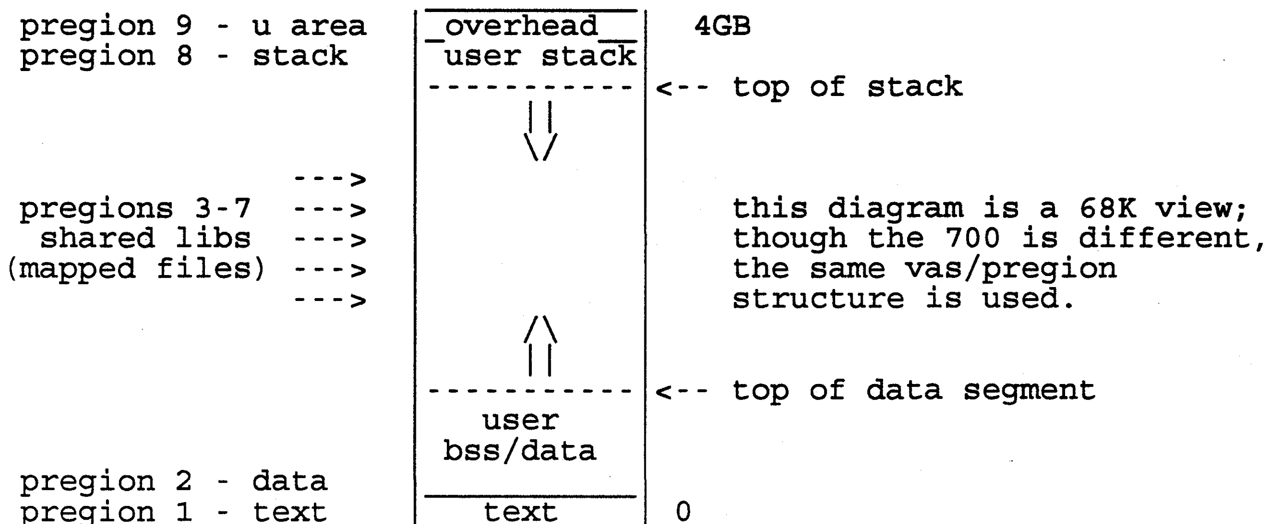
- _max?siz from above; all integers
- segment and page tables - see pte.h
- _lotsfree, _desfree, _minfree - integers used by pageout daemon
- _freemem - integer used by pager to keep track of free memory
- _swdevt - array defined in conf.c

SE 390: Series 300 HP-UX Internals

Memory Management

Summary

A process' memory map is represented by something called a "vas" (virtual address space), which is little more than a doubly-linked list of pregions. A typical process will have 4 "normal" pregions as well as some extra ones....



- Virtual-to-physical mapping is handled by the MMU, with the aid of per-process segment and page tables. The first part of the address indexes into the segment table, the next indexes into the page table that the STE pointed at, and the last piece is a 12-bit index into the page.

- "Regions" are groups of pages that are all of the same type (e.g. text, stack, etc), and the system is set up to allow easy sharing of them. If a process is using a particular region, it will have a "pregion" that points to the region and tells where in the process address space that region is mapped.

- "Paging" refers to kicking out individual pages (loosely based on frequency of use) and then faulting them back in if needed; "swapping" refers to kicking out and bringing in whole processes. Paging is a much gentler way to stretch the amount of memory.

  Swap space is reserved whenever a process starts (via fork/exec or grows (via malloc (==> sbrk/brk)); it is actually allocated to a particular page in a region when that page is about to get swapped/paged. It is mapped via DBDs in the region; these index into the swaptab[]/swapmap[] structure for the system.

```
1
2   From 9.0 /etc/conf/h/pregion.h:
3
4     /*      Each process has a number of pregions which describe the
5      *      regions which are attached to the process.
6      */
7     struct p_lle {
8             struct pregion *lle_next;       /* First pregion in list */
9             struct pregion *lle_prev;       /* Last pregion in list */
10    };
11
12    typedef struct pregion {
13            struct p_lle p_ll;      /* Linked list of pregions in vas */
14    #define p_next p_ll.lle_next
15    #define p_prev p_ll.lle_prev
16            short   p_flags;
17            short   p_type;
18            reg_t   *p_reg;         /* Pointer to the region. */
19            space_t p_space;        /* virtual space for region */
20            caddr_t p_vaddr;        /* virtual offset for region */
21            size_t  p_off;          /* offset in region */
22            size_t  p_count;        /* number of pages mapped by pregion */
23            short   p_prot;         /* protection ID of region */
24            ushort  p_ageremain;    /* remaining number of pages to age */
25            size_t  p_agescan;      /* index of next scan for vhand's age hand *
26            size_t  p_stealscan;    /* index of next scan for vhand's steal hand
27            struct vas *p_vas;      /* Pointer to vas we're under */
28            struct pregion *p_forw; /* Active chain of pregions */
29            struct pregion *p_back;
30            struct pregion *p_prpnext; /* list of pregions off region */
31            struct pregion *p_prpprev; /* list of pregions off region */
32            size_t p_lastfault;     /* last page faulted by this pregion */
33            size_t p_lastpagein;    /* last page-in scheduled for this pregion *
34            short p_trend_diff;     /* difference between last two page faults *
35            ushort p_trend_strength;/* number of times p_trend_diff was the same
36            struct hdlpregion p_hdl;/* HDL specific info for pregion */
37    } preg_t;
38
39    /*      Pregion flags.
40     */
41
42    #define PF_ALLOC        0x0001          /* Pregion allocated            */
43    #define PF_MLOCK        0x0002          /* region is memory locked      */
44    #define PF_EXACT        0x0004          /* map pregion exactly          */
45    #define PF_ACTIVE       0x0008          /* Pregion on active chain      */
46    #define PF_NOPAGE       0x0010          /* Pregion locked against paging */
47                                            /* either another pregion is    */
48                                            /* responsible for paging this  */
49                                            /* region or we don't want it   */
50                                            /* paged (UAREA and NULLDREF)   */
51    #define PF_NOMAP        0x0020          /* Translations should not be   */
52                                            /* resolved through this preg   */
53                                            /* by HIL code (for priveleged  */
54                                            /* shared libraries).           */
55    #define PF_PUBLIC       0x0040          /* May be public (for shared    */
56                                            /* libraries)                   */
57    #define PF_DAEMON       0x0080          /* pregion is for kernel daemon */
58    #define PF_WRITABLE     0x0100          /* May grant write access to    */
59                                            /* pages.                       */
60    #define PF_INHERIT      0x0200          /* Inherit across exec()        */
61    #define PF_VTEXT        0x0400          /* vnode was marked as VTEXT    */
62    #define PF_MMFATTACH    0x0800          /* MMF pregion is being attached*/
63
64    #define PREGMLOCKED(PRP)        (PRP->p_flags & PF_MLOCK)
65
```

```
66    /*      Pregion types.
67     */
68
69    #define PT_UNUSED       0              /* Unused pregion.              */
70    #define PT_UAREA        1              /* U area                       */
71    #define PT_TEXT         2              /* Text region.                 */
72    #define PT_DATA         3              /* Data region.                 */
73    #define PT_STACK        4              /* Stack region.                */
74    #define PT_SHMEM        5              /* Shared memory region.        */
75    #define PT_NULLDREF     6              /* Null pointer dereference page */
76    #define PT_LIBTXT       7              /* shared library text region   */
77    #define PT_LIBDAT       8              /* shared library data region   */
78    #define PT_SIGSTACK     9              /* signal stack                 */
79    #define PT_IO           10             /* I/O region                   */
80    #define PT_MMAP         11             /* Memory mapped file           */
81    #define PT_GRAFLOCKPG   12             /* Framebuffer lock page        */
82    #define PT_NTYPES       13             /* Total # pregion types defined */
83
84
85  From 9.0 /etc/conf/h/vas.h
86
87    #define      VA_CACHE_SIZE   1
88
89    struct vas {
90          struct p_lle va_ll;      /* Doubly linked list of pregions */
91    #define va_next va_ll.lle_next
92    #define va_prev va_ll.lle_prev
93          preg_t *va_cache[VA_CACHE_SIZE];
94          int     va_refcnt;       /* Number of pointers to this vas */
95          vm_sema_t va_lock;       /* Lock structure */
96          u_int va_rss;            /* Cached approx. of shared res. set size */
97          u_int va_prss;           /* Cached approx. of private RSS (in mem) */
98          u_int va_swprss;         /* Cached approx. of private RSS (on swap) */
99          u_long va_flags;         /* various flags */
100         struct file *va_fp;      /* file table entry for MMFs psuedo-vas */
101         u_long va_wcount;        /* count of writable MMFs sharing psuedo-vas *
102         struct proc *va_proc;    /* pointer to process, if there is one */
103         struct hdlvas va_hdl;    /* HW Dependent info for vas */
104   };
105
106   typedef struct vas vas_t;  /* this needs to be visible to compile proc.h */
107
108   /*
109    * Values for va_flags
110    */
111   #define VA_HOLES      0x00000001        /* vas may have holes within pregions
112   #define VA_IOMAP      0x00000002        /* there may be an iomap pregion in th
113   #define VA_NOTEXT     0x00000004        /* No text region in vas (EXEC_MAGIC a
114
115
116
```

```
117  From 9.0 /etc/conf/h/region.h:
118
119    /*
120     * Per region descriptor.  One is allocated for
121     * every active region in the system.  Beware if you add
122     * data elements here:  Dupreg may need to copy them.
123     */
124
125    typedef struct  region  {
126            ushort  r_flags;
127            ushort  r_type;          /* type of region */
128            size_t  r_pgsz;          /* size in pages */
129            size_t  r_nvalid;        /* number of valid pages in region */
130            size_t  r_swnvalid;      /* resident set size of swapped region */
131                                     /* (r_nvalid value when region swapped) */
132            size_t  r_swalloc;       /* for RF_SWLAZY, # pgs actually allocated *
133            ushort  r_refcnt;        /* number of users pointing at region */
134            size_t  r_off;           /* offset into vnode (page aligned) */
135            ushort  r_incore;        /* number of users pointing at region */
136            short   r_mlockcnt;      /* number of processes that locked this */
137                                     /* region in memory. */
138            int     r_dbd;           /* dbd for vfd's when swapped */
139            struct vnode *r_fstore;  /* pointer to vnode where blocks come from *
140            struct vnode *r_bstore;  /* pointer to vnode where blocks go */
141            struct region  *r_forw;  /* links for list of all regions */
142            struct region  *r_back;
143            short   r_zomb;          /* set by xinval to indicate text bad */
144            struct region            /* hash for region */
145                    *r_hchain;
146            union {
147                struct old_aout {
148                    u_int r_ubyte;    /* byte off in fstore (for old a.out) */
149                    u_int r_ubytelen; /* byte len in fstore (for old a.out) */
150                } r_byt;
151                struct mmf {
152                    struct ucred *r_ummfcred; /* credentials for MMF */
153                    u_long r_filler1;         /* unused */
154                } r_mmf;
155            } r_un;
156            vm_sema_t r_lock;        /* region lock */
157            vm_sema_t r_mlock;       /* wait for region to be locked in memory */
158            int  r_poip;             /* number of page I/Os in progress
159                                      *
160                                      *  NOTE: must hold the region lock and the
161                                      *  sleep_lock to increment the r_poip
162                                      *  field (start an I/O). Must hold
163                                      *  the sleep_lock to decrement.
164                                      */
165            struct broot             /* Root of btree of vfd/dbd's */
166                    *r_root;
167            unsigned long r_key;     /* Each region contains chunk and one key */
168            chunk_t *r_chunk;
169            struct region *r_next;   /* links for regions sharing pages */
170            struct region *r_prev;
171            struct pregion *r_pregs;/* list of pregions pointing to this region
172            struct hdlregion         /* HDL fields in region */
173                    r_hdl;
174    } reg_t;
175
176    #define r_byte          r_un.r_byt.r_ubyte
177    #define r_bytelen       r_un.r_byt.r_ubytelen
178    #define r_mmfcred       r_un.r_mmf.r_ummfcred;
179
```

```
180     /*
181      * Region flags
182      */
183     #define RF_NOFREE       0x0001  /* Don't free region on last detach */
184     #define RF_ALLOC        0x0004  /* region is not on free list */
185     #define RF_MLOCKING     0x0008  /* set when locking region in memory */
186                                     /* wake up processes waiting on r_mlock */
187                                     /* when resetting this flag. */
188     #define RF_ZOMB         0x0010  /* set in xinval when a text turns bad */
189     #define RF_UNALIGNED    0x0020  /* Region is an unaligned view of vnode */
190                                     /* (support old a.out) */
191     #define RF_SWLAZY       0x0040  /* Don't allocate all swap space up front */
192     #define RF_WANTLOCK     0x0080  /* someone else wants to lock this reg, */
193                                     /* so wakeup(rp) them. CHANGE FOR MP*/
194     #define RF_HASHED       0x0100  /* region is hashed (fstore, byte) */
195     #define RF_EVERSWP      0x0200  /* set if region has ever been swapped */
196     #define RF_NOWSWP       0x0400  /* set if region is now swapped */
197     #define RF_DAEMON       0x0800  /* set if region is for a kernel daemon */
198     #define RF_UNMAP        0x1000  /* MMF region is being unmapped */
199     #define RF_IOMAP        0x2000  /* region is an iomap(7) region */
200
201     /*
202      * Logical index from region offset to vnode offset in bytes.
203      */
204     #define vnodindx(RP, PGINDX) (ptob(PGINDX + (RP)->r_off))
205
206     /*
207      * Region types
208      */
209     #define RT_UNUSED       0       /* Region not being used.       */
210     #define RT_PRIVATE      1       /* Private (non-shared) region. */
211     #define RT_SHARED       2       /* Shared region */
212
213
214
215
```

```
216  From 9.0 /etc/conf/h/conf.h:
217
218    /*
219     * Swap device information
220     */
221    typedef struct swdevt
222    {
223            dev_t   sw_dev;         /* swap device          */
224            int     sw_enable;      /* enabled              */
225            int     sw_start;       /* offset for 300/700   */
226            int     sw_nblks;       /* number of blocks     */
227            int     sw_nfpgs;       /* # of free pages      */
228            int     sw_priority;    /* priority of device   */
229            int     sw_head;        /* first swaptab[] entry*/
230            int     sw_tail;        /* last swaptab[] entry */
231            struct swdevt *sw_next; /* next swap device     */
232    } swdev_t;
233
234
235  From 9.0 /etc/conf/h/swap.h:
236
237    int fs_swap_debug;
238
239    /*      The following structure contains the data describing a
240     *      swap file.
241     */
242
243    typedef struct swapmap {
244            ushort  sm_ucnt;        /* number of users on this page */
245            short   sm_next;        /* index of free swapmap[]      */
246    } swpm_t;
247
248    typedef struct swaptab {
249            short   st_free;          /* index of 1st free swapmap[]*/
250            short   st_next;          /* index of next chunk for    */
251                                      /* same dev or fs             */
252            int     st_flags;         /* flags defined below.       */
253            struct swdevt *st_dev;    /* swap device.               */
254            struct fswdevt *st_fsp;   /* swap file system.          */
255            struct vnode *st_vnode;   /* dev or fs vnode            */
256                                      /* system chunk               */
257            int     st_nfpgs;         /* nbr of free pages on device*/
258            struct swapmap *st_swpmp; /* ptr to swapmap[] array.    */
259            int     st_site;          /* site number (DUX)          */
260            union {
261            int     st_start;         /* starting addr on S300      */
262            int     st_swptab;        /* server swaptab[] index     */
263            } st_union;
264    } swpt_t;
265
266    typedef struct fswdevt{
267            struct fswdevt *fsw_next;       /* next fs w/ same pri  */
268            int fsw_enable;                 /* enabled              */
269            int fsw_nfpgs;                  /* # free pages         */
270            int fsw_allocated;              /* # of blocks allocated*/
271            uint fsw_min;                   /* min # preallocated   */
272            uint fsw_limit;                 /* max # to allocate    */
273            uint fsw_reserve;               /* # to reserve         */
274            int fsw_priority;               /* priority             */
275            struct vnode *fsw_vnode;        /* file system vnode    */
276            short   fsw_head;               /* 1st swaptab[] entry  */
277            short   fsw_tail;               /* last swaptab[] entry */
278            char fsw_mntpoint[256];         /* file system mount pt.*/
279    } fswdev_t;
280
```

```
281  typedef struct devpri{
282          struct swdevt *first;   /* first fs for a priority      */
283          struct swdevt *curr;    /* allocate from this fs first  */
284  } devpri_t;
285
286  typedef struct fspri{
287          struct fswdevt *first;  /* first fs for a priority      */
288          struct fswdevt *curr;   /* allocate from this fs first  */
289  } fspri_t;
290
291
292  /*
293   * This is an overlay structure for a regular dbd.
294   * It MUST be the same size as a dbd.
295   */
296  typedef struct swpdbd {
297          uint dbd_type:4,
298               dbd_swptb:14,
299               dbd_swpmp:14;
300  } swpdbd_t;
301
302
303  extern nswapfs;
304  extern nswapdev;
305  extern swchunk;
306  extern maxswapchunks;
307  extern swapmem_cnt;
308  extern swapspc_cnt;
309  extern maxfs_pri;
310  extern maxdev_pri;
311  extern struct vnode *swapdev_vp;
312  extern struct swaptab *swapMAXSWAPTAB;
313  extern vm_sema_t swap_lock;       /* Lock for all swap entries   */
314  extern vm_lock_t rswap_lock;       /* Lock for reserveing swap */
315  extern int       swapwant;        /* Set non-zero if someone is  */
316                                    /* waiting for swap space.     */
317
318  #define SWTYPE_DEV      0x1     /* raw disk swap dev */
319  #define SWTYPE_FS       0x2     /* file system swap device */
320  #define SWTYPE_LAN      0x4     /* diskless (lan) swap device */
321
322
323
324
```

```
325  From 9.0 /etc/conf/h/vmmeter.h:
326
327     /*
328      * Virtual memory related instrumentation
329      */
330     struct vmmeter
331     {
332     #define v_first v_swtch
333             unsigned v_swtch;        /* context switches */
334             unsigned v_trap;         /* calls to trap */
335             unsigned v_syscall;      /* calls to syscall() */
336             unsigned v_intr;         /* device interrupts */
337             unsigned v_pdma;         /* pseudo-dma interrupts */
338             unsigned v_pswpin;       /* pages swapped in */
339             unsigned v_pswpout;      /* pages swapped out */
340             unsigned v_pgin;         /* pageins */
341             unsigned v_pgout;        /* pageouts */
342             unsigned v_pgpgin;       /* pages paged in */
343             unsigned v_pgpgout;      /* pages paged out */
344             unsigned v_intrans;      /* intransit blocking page faults */
345             unsigned v_pgrec;        /* total page reclaims */
346             unsigned v_xsfrec;       /* found in free list rather than on swapdev
347             unsigned v_xifrec;       /* found in free list rather than in filsys
348             unsigned v_exfod;        /* pages filled on demand from executables *
349             unsigned v_zfod;         /* pages zero filled on demand */
350             unsigned v_vrfod;        /* fills of pages mapped by vread() */
351             unsigned v_nexfod;       /* number of exfod's created */
352             unsigned v_nzfod;        /* number of zfod's created */
353             unsigned v_nvrfod;       /* number of vrfod's created */
354             unsigned v_pgfrec;       /* page reclaims from free list */
355             unsigned v_faults;       /* total faults taken */
356             unsigned v_scan;         /* scans in page out daemon */
357             unsigned v_rev;          /* revolutions of the hand */
358             unsigned v_seqfree;      /* pages taken from sequential programs */
359             unsigned v_dfree;        /* pages freed by daemon */
360             unsigned v_cwfault;      /* Copy on write faults */
361             unsigned f_bread;        /* total bread requests */
362             unsigned f_breadcache;   /* total bread cache hits */
363             unsigned f_breadsize;    /* total bread bytes */
364             unsigned f_breada;       /* total read aheads */
365             unsigned f_breadacache;  /* total read ahead cache hits */
366             unsigned f_breadasize;   /* total read ahead bytes */
367             unsigned f_bwrite;       /* total bwrite requests */
368             unsigned f_bwritesize;   /* total bwrite bytes */
369             unsigned f_bdwrite;      /* total bdwrite requests */
370             unsigned f_bdwritesize;  /* total bdwrite bytes */
371     #ifdef __hp9000s800
372             unsigned v_pgtlb;        /* tlb flushes */
373             unsigned v_swpwrt;       /* swap writes */
374     #endif /* __hp9000s800 */
375             unsigned v_fastpgrec;    /* fast reclaims in locore */
376             unsigned f_clnbkfl;      /* clean block found immediatly on free list
377             unsigned f_flsempty;     /* free list empty */
378             unsigned f_bufbusy;      /* buffer busy */
379             unsigned f_delwrite;     /* delayed write buffer written */
380     #define v_last   f_delwrite
381             unsigned v_free;         /* free memory pages */
382             unsigned v_swpin;        /* swapins */
383             unsigned v_swpout;       /* swapouts */
384             unsigned v_runq;         /* current length of run queue */
385     };
386
387
388     #ifdef  _KERNEL
389     extern  struct  vmmeter cnt, rate, sum;
390     #endif
```

```
391
392    /* systemwide totals computed every five seconds */
393    struct vmtotal
394    {
395            unsigned int t_rq;      /* length of the run queue */
396            unsigned int t_dw;      /* jobs in ``disk wait'' (neg priority) */
397            unsigned int t_pw;      /* jobs in page wait */
398            unsigned int t_sl;      /* jobs sleeping in core */
399            unsigned int t_sw;      /* swapped out runnable/short block jobs */
400            int     t_vm;           /* total virtual memory */
401            int     t_avm;          /* active virtual memory */
402            unsigned int t_rm;      /* total real memory in use */
403            unsigned int t_arm;     /* active real memory */
404            int     t_vmtxt;        /* virtual memory used by text */
405            int     t_avmtxt;       /* active virtual memory used by text */
406            unsigned int t_rmtxt;   /* real memory used by text */
407            unsigned int t_armtxt;  /* active real memory used by text */
408            unsigned int t_free;    /* free memory pages */
409    };
410    #ifdef  _KERNEL
411    extern struct   vmtotal total;
412    #endif
413
414
415
416
417  From 9.0 /etc/conf/h/vmsystm.h:
418
419    /*
420     * Miscellaneous virtual memory subsystem variables and structures.
421     */
422
423    #ifdef _KERNEL
424    extern  int     freemem;        /* remaining blocks of free memory */
425    extern  int     freemem_cnt;    /* number of processes waiting on freemem */
426    extern  int     avefree;        /* moving average of remaining free blocks *
427    extern  int     avefree30;      /* 30 sec (avefree is 5 sec) moving average
428    extern  int     deficit;        /* estimate of needs of new swapped in procs
429    extern  int     nscan;          /* number of scans in last second */
430    extern  int     multprog;       /* current multiprogramming degree */
431    extern  int     desscan;        /* desired pages scanned per second */
432
433    /* writable copies of tunables */
434    extern  int     maxslp;         /* max sleep time before very swappable */
435    extern  int     lotsfree;       /* max free before clock freezes */
436    extern  int     minfree;        /* minimum free pages before swapping begins
437    extern  int     desfree;        /* no of pages to try to keep free via daemo
438    extern  int     saferss;        /* no pages not to steal; decays with slptim
439
440    /* AGEFRACTION of n means we want to age 1/n of a region before going on */
441    /* AGEFRACTION of 16 is the smallest possible since p_ageremain is a short *
442    #define LOGAGEFRACTION 4
443    #define AGEFRACTION (1 << LOGAGEFRACTION)
444    #define AGEFRACTIONMASK (AGEFRACTION - 1)
445    #endif
446
```

SE 390: Series 300 HP-UX Internals

Diskless

The Big Picture

- How does HP-UX do without a disk?

The Little Picture(s)

- What a cnode can and can't do
- Context
- Crash Detection
- The server's view
- References

Diskless

What a Cnode Can And Can't Do

- It can...

  ...run programs & deal with I/O, context switching, etc.

  ...handle its own swapping if a local swap disk is present

  ...be a fully functional networking node/gateway

- It can't...

  ...access its own filesystem - there's no disk!

  - 8.0 allows "locally mounted filesystems"; really "locally attached filesystem disks", since they are part of the cluster's filesystem
    ^^^^^^^^^

  ...allocate its own PIDs independently

  ...swap (assuming no local disk)

  - local swap has always been allowed

  - in 8.0 one cnode can act as the "swap server" for other cnodes iff

    a) it has a local swap disk and
    b) its cnode id is shown as their swap site in /etc/clusterconf

  ...automagically keep its clock in synch

  ...access devices on the server or other nodes (what is a device file? what would remote device support imply?)

SE 390: Series 300 HP-UX Internals

Diskless

Context

- Set at boot time.
- Provides a general mechanism for matching files with machines and/or capabilities.
    - If a machine has a floating point accelerator in it, that implies that it needs to "see" a different math library than a normal machine would need.
    - In theory, this sort of thing could be used to allow for having both UCB and AT&T command sets available, or providing for a S300 and S800 to get their respective executables off of the same disk. This is in fact what is done in 7.0/8.0 when we have an S800 serving S300 clients; /bin and many other things become CDFs.
- The key place it is used in the kernel is in pathname lookup. When the search for "/etc/reboot" finds its way to the actual disk, the system will notice if the file is a CDF. If it is, it will drop down into the directory and start looking for files that match a context string.
- What are the implications of having "system" files be CDFs?

Fun With CDFs

- they're tricky!

- be sure to use "-hidden" with find(1) if you care about CDFs

- "ll -H" is your friend :-)

- if something isn't a CDF when you first install the system, it probably shouldn't be, e.g. making /etc a CDF so that the passwd, group, ... files can be customized on each client may seem clever at first, but will seem decidedly un-clever next time you want to boot :-(

- be conscious of different "priorities" of context elements, i.e. having a CDF element for the server (by its name) and one for "localroot" too is a bad plan

- cnode-specific device files are often confused with CDFs, but are something different - basically a cnode-specific device file is one that can only be used on a particular cnode. By default, a device file can only be used on the machine it is created on; specifying additional options to mknod(1m) can yield a device file that is 1) specific to another cnode; 2) global (usable by the whole cluster)

Crash Detection

- When in the course of human events a diskless node goes out to lunch, it takes cluster resources with it. It is important that this be detected quickly, since other nodes may be waiting on files or memory or whatever.

- Whenever a node receives a packet from another node, it keeps track of this. If it notices that it hasn't received a packet from a node very recently, it will send a message to that node asking it to respond. If it does, fine; if not, it is declared dead and its resources are reclaimed.

- The kernel parameters check_alive_period and retry_alive_period deal with this. If for some reason it is OK/expected that nodes will be unable to respond quickly, they may need to be raised, but in general they should be left alone.

Recovery-on
Recovery-off
dup.test
dup.test
Recovery on
Recovery-off.

SE 390: Series 300 HP-UX Internals

Diskless

The Server's View

- The server is an ordinary system except that it has a few extra processes running.

  120-150 pri

- When a server cluster(1m)'s, it starts up a "Limited CSP". This CSP is only willing to do certain things; if it is asked to do something that might take a while, it will put the request on a queue and let a "General CSP" handle it.
- CSPs run at "important" priorities, i.e. better than normal user processes, but not real-time.
- When a request comes in from a cnode, it is put on a queue. When a CSP becomes available, it will grab the request and start working on it.
- If a request takes too long, the CSP will commit suicide when it finishes - it will already have been replaced.
- The server is responsible for keeping the clocks synchronized (otherwise make wouldn't work right), allocating chunks of PIDs to cnodes (lots of things use PIDs to generate filenames), and doing the swap and filesystem serving.
- The server must find out quickly if a node fails, so that resources can be reclaimed.
- If the server needs to reboot, it must shut down all the clients first, which is why /etc/reboot ~~is a CDF~~.

  must have 2 paths. One for Remote root, one for local boot.

SE 390: Series 300 HP-UX Internals

System Startup

The Big Picture

- How do we get from a doing-nothing system to a system running HP-UX?

The Little Pictures

- The boot ROM and secondary loader.
- Configuring the virtual-memory subsystem.
- Preparing for I/O.
- Kicking off the first processes.
- What is the correspondence between things being accomplished and things being printed on the console's screen?

System Startup

Boot Rom and Secondary Loader (S300/400)

- The first 8K block of the disk is a boot block, which contains a LIF directory and the secondary loader. A copy of this block can be found in /etc/boot.

```
        first 8K                        (not to scale!)
+-------------------------------------------------------+
|SYSHPUX  : secon- |                    |              |
|SYSBCKUP :  dary  |     filesystem     |     swap     |
|SYSDEBUG : loader |                    |              |
|         :        |                    |              |
+-------------------------------------------------------+
```

- The boot rom reads the LIF directory for each disk present and allows the user to choose one of the entries (assuming attended boot).

- Once an entry has been chosen, the bootrom loads the secondary loader and starts it running. The secondary loader "knows" where the bootrom keeps some of its variables, and it goes and looks to see which of the possible filenames was picked.

  (The bootrom uses the top page of physical RAM to store variables. The kernel also has the top page mapped, and the name of the kernel we booted is accessible via the kernel variable "sysname"; the disk is designated by "msus".)

- The bootrom provides some very simple I/O routines, and the secondary loader uses these to print out the message, "booting /hp-ux" (assuming the default case) and to read in the kernel.

  The secondary loader has a bare-bones knowledge of the file system, and is smart enough to go look in /etc/clusterconf and pick an appropriate kernel out of /hp-ux+ based on that.

- Once the kernel has been read in, the loader jumps to it, passing it the processor type, the address at which it was loaded, etc.

- The kernel is now on its own....

System Startup

System Boot (S700)

- The first 8K of the disk is a boot block, which contains a LIF directory. Doing a "lifls -l" of some bootable disk will show that there are quite a few entries in the directory: filesystem, swap, HP-UX, some stuff for debuggers, etc. Most of this stuff is in the "boot area", which is at the end of a 700's disk.

  A typical system disk might be laid out something like this:

```
+------------------------------------------------------------+
|LIF dir| filesystem                  | swap     | ISL, etc |
+------------------------------------------------------------+
```

  Note that this looks much like a 300/400 disk. The major difference is that the "secondary loader" for a PA machine is too big to fit into the 1st 8K block of the disk like the 300 would do, so it has been moved to a 2MB area at the end.

- The bootrom will search for possible boot devices and consoles if it hasn't been told in advance where to boot from. To interact with it, press and hold ESC shortly after powering on the machine; this will cause it to enter a menu-driven mode in which lots of things can be set/changed (things like boot paths, console/keyboard paths, the LANIC address, etc.

  NOTE: typing "secure on" at this point will keep you from ever being able to change bootpaths, console paths, etc.*

- Once a device has been chosen to boot from, find something else to do; it will be quite a while before anything happens on the console. Once the kernel is loaded and initialized, though, the 700 will make up for its initial sluggishness. It will ID cards (and really look quite a bit like a 300) as it boots and observers will be hard-pressed to keep up with what is being displayed.

- Once the kernel is running, the system will go through all of the normal user-space things like /etc/rc, /etc/netlinkrc, etc.

Starting Up The Virtual Memory System

- Set up the kernel page table ("Sysmap") and turn on the MMU.
- Initialize kernel memory mapping. The kernel *must* know about all physical memory: some is allocated to the kernel itself, some is allocated to user processes, and *all* of it must be kept track of.
- See what swap devices are available. The table is specified in conf.c, and is called swdevt[]. At boot time it is scanned, and the disks are checked to make sure the space is really there, etc. This is when the system prints

```
Swap device table: start and size given in 512-byte blocks...
    entry 0: autoconfigured on root device; start=X, size=Y
```

- Enable the first swap device in swdevt[].
- Fork process 2 to be the pageout daemon.
- Start looking for jobs to swap in/out.

Preparing For I/O.

- Call device driver link routines. Note the *_link routines in /etc/conf/conf.c after you have run config(1m). At bootup time, the system will walk that whole list, calling each routine in it. The routine will add an entry for its driver to a list that will be used when we actually find cards.

- See what cards are installed. When a card is found, walk the list mentioned above. When a driver claims the card as its own, it will allocate data structures and do any other startup initialization (e.g. adding an entry to rupttable on the 68K).

- Look for a console. See the Facilities (Concepts & Tutorials) manual for the order in which things will be chosen.

- Mount the root filesystem. This is done by asking each disk driver whether it knows about the disk the bootrom says we booted from (this information is put in the top page of RAM by the bootrom along with the name ("SYSHPUX", "SYSBCKUP", etc)). When we find a driver that claims the disk, we can call its "open" routine and mount the disk.

System Startup

Starting The First Processes

- Build process 0 by hand; it will become the swapper.

- Start roundrobin scheduling. This isn't really a process, but sort of acts like one. What we actually do is arrange for a routine to be called every <timeslice> cpu ticks.

- Fork process 2 to become the pageout daemon.

- Start CSP if this is a diskless node.

- Fork process 1 to become init. We actually do some stuff to set this up as a user process so that when /etc/init is exec(2)ed, it is a normal user process. It is somewhat special, however, because the kernel sort of looks out for it in a few areas (such as not letting someone send SIGKILL to it, panic()ing if it exit(2)s, etc).

  In 8.0, the kernel runs /etc/pre_init_rc before starting /etc/init so that the root filesystem can be checked without any interference from user processes. Note that pre_init_rc checks /dev/rroot, which is a character-special file that represents the root disk (major & minor are both -1). If /dev/rroot gets destroyed or isn't there for some reason,

  ```
  # mknod /dev/rroot c -1 -1
  ```
  will fix it.

SE 390: Series 300 HP-UX Internals

System Startup

Internal Actions vs. External Signs (on a 68K system; 700 is similar)

```
- "booting /hp-ux"

        set up kernel page table
        get info. from bootrom: processor type, amount of RAM,...
        allocate RAM for buffer cache, cmap, inodes, etc.
        clear out memory and decide if we have enough to continue
        call device driver link routines
        look for ttys, init. console

- "Console is ITE"
  "ITE + 0 ports"
  "680x0 processor"
  "MC68881 coprocessor"

        look for I/O cards

- "xxxxx at select code yy" - for each card found
  "real mem = xxxxxxxx"
  "mem reserved for dos = xxxxxxx"
  "using xxx buffers containing yyyyyy bytes of memory"

        twiddle data structures to reflect proc. 0
        start clock
        initialize root device
        initialize diskless stuff

- "Local link is xxxxxxxxx"      \
  "Server link is yyyyyyyy"       >   diskless systems only...
  "Swap site is nn"              /
  "Root device major is xx, minor is yyyy [root site is xx]"

        initialize buffer cache

- "Swap device table: (start and size...)"  \  these are present
  ".... (line for each entry) ...."          /  only if local swap
  "Savecore image of xx pages will be saved at block yy in swap area

        configure swap devices
        mount root filesystem
        start up CPU roundrobin scheduling
        start up paging subsystem
        start up limited CSP

8.0: check root filesystem via /etc/pre_init_rc

- "avail mem = xxxxxxxx"
  "lockable mem = xxxxxxx"

        fork init
        become the swapper

<any further (normal) messages will be from init or its children>
```

Good CS Reference Books

The Design of the UNIX Operating System - Maurice Bach

Advanced Programming in the UNIX Environment - W. Richard Stevens

Modern Operating Systems - Andrew Tanenbaum

Operating Systems: Design and Implementation - Andrew Tanenbaum

Operating Systems Design: The XINU Approach - Douglas Comer

The Design and Implementation of the 4.3BSD UNIX Operating System
Leffler, McKusick, Karels, and Quarterman

Algorithms + Data Structures = Programs - Wirth

Algorithms - Sedgewick

Computer Networks - Tanenbaum

UNIX Network Programming - W. Richard Stevens

Fundamentals of Interactive Computer Graphics - Foley & Van Dam

Internetworking With TCP/IP - Comer

Practical UNIX Security - Garfinkel & Spafford

Software Tools In Pascal - Kernighan & Plauger

The Elements of Programming Style - Kernighan & Plauger

The UNIX Programming Environment - Kernighan & Pike

UNIX System Administration Handbook - Nemeth, Snyder, & Seebass

A good place to get the above if you can't find them locally...

Computer Literacy Bookshop
408-730-9955
520 Lawrence Expressway, Sunnyvale, CA 94086

2 blocks south of US 101, next to TOGO's
Open 7 days/week; mail orders, phone orders welcome
America's largest computer bookstore
10,000 professional and PC titles

Kernel Debugging Hints

1. Dealing with "hung" processes.

   When a process needs something that it can't have (inside the kernel), it will call a kernel routine named sleep(). One of the arguments it is called with is a priority; if this is less than PZERO (see param.h), this means that the sleep is *not* interruptible. If this is the case, the sleep() had better be pretty short; if it turns out not to be, we will wind up with a non-killable hung process. This is not A Good Thing.

   How to deal with it? There are several ways. The first is to run monitor and see what its "single process info" screen will tell you about the process. The second is to use "ps -l" to get the sleep channel and priority. If the priority is < PZERO, chances are this is a driver bug. If we want to keep on investigating, we can feed this address to adb(1) to find out what's being waited for:

   ```
   adb /hp-ux /dev/kmem
   ```

   This will usually work, but there's a catch. Suppose the sleep channel is 0x12345678. By default, adb(1) is only willing to look at addresses less than 0x1000000 (16 MB). If the sleep address is above this, it will be necessary to change adb(1)'s mapping, like this:

   ```
   /m 0 0x1fffffff 0
   ```

   This tells adb(1) to use a big piece of the address space, instead of just a tiny one.

   Once the mapping is straightened out, use a command like this:

   ```
   0x<sleep_channel>/i
   ```

   If adb(1) can find a symbol near that address, it will print out something like this:

   ```
   _Bufferaddr+0x94:
   ```

   This tells us that we may be waiting on a buffer. Sometimes this is helpful, sometimes not; it is worth remembering.

2. Figuring out what went wrong in a system call or library routine.

   This shouldn't be in here, but in the interest of fending off questions, it is :-)

   Let's suppose someone writes a new version of cat(1), like this:

```
#include <stdio.h>
#include <fcntl.h>

main(argc, argv)
int argc;
char *argv[];
{
	int fd, n;
	char buf[8192];

	fd = open(argv[1], O_RDONLY);

	while ((n = read(fd, buf, 8192)) > 0)
		write(1, buf, n);

	close(fd);
}
```

Suppose that this is invoked on some file, and nothing comes out. Is it necessarily because there isn't anything in the file? What if...the mode of the file didn't allow access?

3. Miscellaneous.

If you are getting absolutely *bizarre* behavior from your system, consider the possibility that you have a mismatch between different parts (kernel vs. commands, part of kernel vs. another part, etc). I once had an SE call in with a *strange* set of symptoms that I simply couldn't explain. It turned out that he had mixed 5.5 and 6.0 kernel library archives!

CDFs can cause pretty bizarre behavior if you aren't watching out for them.

If a device driver (or some other configurable part of the kernel) is not configured in, the error one gets back isn't necessarily clear.... For instance, if diskless is not configured into the server's kernel the cluster(1m) command will fail with "no such device or address". How enlightening :-)

Driver Writing Information & Hints

Introduction

This document is taken from the prestudy for SE327, the now-defunct driver-writing class. If you are looking for a basic introduction to the concepts, this is worth reading. If you want more detailed information, order the HP-UX Driver Development Guide (98577-90013 as of August 1991).

What is a Driver?

Just what is a driver, anyway?

A. A "driver" is one of four distinct personality types, the other three being "amiable", "expressive", and "analytic".

B. A "driver", along with the "iron", the "wedge", and the "putter", comprise the equipment needed for a game of golf. A driver is designed to deliver maximum force to the ball, and to sink fastest when thrown into water hazards in disgust. It also can be used to create larger divots when irons are insufficient for the task.

C. A "driver" is the person sitting behind the steering apparatus of a locomotion vehicle. The only known exception to this rule is the "mother-in-law", which can be seated anywhere within the vehicle and still drive effectively.

D. A "driver" is a piece of code which enables communication between the user and a particular piece of hardware.

The correct answer, of course, is D. The driver bridges the gap between the user and the target hardware.

User-Land Versus Kernel Drivers

A driver can run as a user process (in "user-land") or as a kernel process. A driver executing as a user-land process runs at normal user priorities, and is subject to the same scheduling rules as any other process. The advantages of a user-land driver are:

1. There is no kernel re-build or reboot necessary.

2. The driver writer can use adb/cdb for debugging.

3. The driver writer can use familiar user libraries in his/her code.

4. The driver writer has no need of kernel knowledge.

An example of a product which requires user-land drivers is the old VME expander (98646A). Drivers for VME cards installed in that product had to run in user-land.

Some of the disadvantages of user-land drivers are:

1. They're slow!

2. Interrupts aren't available.

3. DMA isn't available.

The driver writer needs to evaluate his/her application and weigh the trade-offs between user-land and kernel drivers before deciding which is right for the task. Often, a simple user-land program will do the job in situations which don't require great speed, interrupts, or DMA. Some tools available for writing user-land drivers are:

1. Pseudo-terminals (ptys) - for RS-232/serial devices;

2. Device I/O Library (DIL) - for HP-IB or GPIO devices;

3. Iomap - useful with just about any interface card for which the driver writer has a register map. Maps a particular chunk of physical memory into user space.

Since the purpose of the SE327 driver writing class is to fully describe kernel drivers, only kernel drivers will be discussed from this point on.

Types of Drivers

There are two types of kernel drivers: interface drivers and device drivers.

The interface driver communicates with a particular type of interface card and doesn't concern itself with the devices connected to that card, if any. For example, there are interface drivers for the MUX card and the HP-IB card.

A device driver communicates with a particular class of device and doesn't care about the interface it's connected to. For example, a device driver would talk to a CS/80 disc, a ciper printer, or a serial device.

These two types of kernel drivers can be combined into one driver if only one class of device can be connected to a particular interface card. Some of the more complex interfaces, like HP-IB, have three interface drivers (for the 98624, the 98625, and the internal HP-IB interfaces) and a multitude of device drivers (for HP-IB printers, ciper printers, CS/80 discs, other discs, 9-track mag tape, etc.).

Types of Driver Access

There are two types of kernel driver access: block access and character (raw) access.

When block access is used, data transferred between a user process and a device is buffered. Data transfer occurs in units called blocks.

When character access is used, there is no particular buffering scheme used, although the driver writer can use a buffering scheme if he/she so desires. Data is transferred in units of one or more bytes.

The type of access used depends heavily on the device to which the driver talks. Devices having the following characteristics are good candidates for block access:

1. The device supports random access of blocks.

2. The data in each block is stable.

3. The data is not available until it is requested.

Typical block devices are discs and tapes.

Devices having the following characteristics are good candidates for character access:

1. The data cannot be accessed randomly.

2. The data is not stable.

3. The data can be available before any process requests it.

Typical character devices are terminals and printers.

Note that most devices can be accessed both ways. However, one type of access is usually optimal for a particular type of device.

Driver Entry Points

The HP-UX kernel expects all drivers to consist of one or more routines whose names are consistent across all drivers. These routine names are called "entry points". A driver may or may not have a particular entry point, but if it does, that entry point will always have the same name (how an entry point for one driver is distinguished from the same entry point for another driver is discussed later in this document).

There are a different set of entry points for character drivers and block drivers. The character driver entry points are:

Entry Point	Function
open	Called from open(2)
close	Called from close(2)
read	Called from read(2)
write	Called from write(2)
ioctl	Called from ioctl(2)
select	Called from select(2)

For block drivers, the entry points are:

Entry Point	Function
open	Called from open(2)
close	Called from close(2)
strategy	Called from read(2) or write(2)
size	Not user-callable; returns size of swap area on device, if any

These two sets of entry points simply mean that these are the routines the kernel knows how to call, given a particular type of driver access. Nothing stops the driver writer from writing a strategy routine for a character driver (in fact this is often done). The kernel won't know how to call it, but the driver code itself can explicitly call it.

In addition to these entry points, there are three more entry points for interface routines (used in DIO drivers only). The purpose of these routines will be discussed in class. These interface routines are:

* link
* make_entry
* init

Finally, there are three "pseudo-driver" entry points. They are:

Entry Point	Function
nulldev	Does nothing; kernel returns successfully to user.
nodev	Does nothing; kernel returns an error to user.
seltrue	Does nothing; kernel returns successfully to user.

                    Used in place of a select routine when device is
                    always ready for I/O.
==

These pseudo-driver entry points will be discussed in more detail later in this document, and in class. Note that "seltrue" has identical functionality to "nulldev". It exists at all simply because it is part of AT&T's standard UNIX release.

The Cdevsw and Bdevsw Tables

How does the kernel keep track of the routines in each driver?

There are two data structures, called the cdevsw table and the bdevsw table, which maintain pointers to the routines in each driver. The cdevsw table is used for character drivers, and the bdevsw table is used for block drivers.

Each table is an array of structures. The array is indexed by the major number of the driver. Thus, at bdevsw[0] one would expect to find pointers to entry points in the block CS/80 driver (major number 0), and in cdevsw[4] one would expect to find pointers to entry points in the character CS/80 driver (major number 4).

Each cdevsw table entry looks like this:

```
struct cdevsw {
    int (*d_open)();
    int (*d_close)();
    int (*d_read)();
    int (*d_write)();
    int (*d_ioctl)();
    int (*d_select)();
    int d_flags;
};
```

Each cdevsw table entry contains pointers for the six character driver entry points, and a parameter "d_flags" to contain flags. The available flags are:

C_ALLCLOSES   specifies that the close entry point shall be called on all closes of the device, instead of only the last close.

C_NODELAY   specifies that the kernel shall not wait for I/O to complete, but shall return immediately to the user process.

Each bdevsw table entry is similar:

```
struct bdevsw {
    int (*d_open)();
    int (*d_close)();
    int (*d_strategy)();
    int (*d_psize)();
};
```

Each bdevsw table entry contains pointers for the four block driver entry points, and the same flags parameter "d_flags".

Installing a Driver

The procedure for installing a driver into a Series 300 HP-UX kernel is really quite simple. The overall procedure is given here, with more detail given in later sections. The procedure is:

1. Compile driver.

2. Modify /etc/master.

3. Add driver name to dfile.

4. Execute "config".

5. Modify config.mk.

6. Execute "make".

This creates a new kernel which must be moved to /hp-ux. Once the system is rebooted, the new kernel is active.

Compile the Driver

Once the driver writer has written all his/her code, it must be compiled to create a ".o" file.

Modify /etc/master

This is probably the most time-consuming step. A line of information regarding the new driver must be added to /etc/master. The "config" routine uses this information in setting up the cdevsw and bdevsw tables and other data structures in conf.c.

Each line in the first section of /etc/master gives information for one driver. Each line is of the form:

```
name  prefix  type  mask  bmajor  cmajor
```

"Name" is the driver name for use in config's dfile. Use any descriptive name not already in use.

"Prefix" can be the same as "name", or some other descriptive string. It is this string that the kernel uses to differentiate your kernel driver's entry points from other drivers' entry points. For example, if you specify a "prefix" of "mycode", the kernel expects to find entry points named "mycode_open", "mycode_close", etc. The driver writer presumably knows this and codes his/her routine names accordingly.

"Type" is a five-bit attribute flag. It has the following form:

```
---------------------
| 4 | 3 | 2 | 1 | 0 |
---------------------
```

The meanings of the bits are:

bit 0 - Set this bit if the driver should have an entry in the cdevsw table (which it should if it is a character driver).

bit 1 - Set this bit if the driver should have an entry in the bdevsw table (which it should if it is a block driver).

bit 2 - Set this bit if the driver is a required driver. "Config" will include the driver in the new kernel whether its name appears in dfile or not.

bit 3 - Set this bit if the driver name may only be specified once in dfile. If the driver's name appears in dfile more than once, an error is generated. Normally this is not an error.

bit 4 - Set this bit if this driver is an interface driver. This implies the presence of link, make_entry, and init routines.

"Mask" is a 10-bit driver routine flag. It has the following form:

```
-----------------------------------------
| 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
-----------------------------------------
```

The meanings of the bits are:

bit 0 - Set this bit if the C_ALLCLOSES flag is desired. Otherwise, this flag is left unset.

bit 1 - Set this bit if the "seltrue" pseudo entry point is desired instead of an actual "select" entry point.

bit 2 - Set this bit if the driver has a select routine.

bit 3 - Set this bit if the driver has an ioctl routine.

bit 4 - Set this bit if the driver has a write routine.

bit 5 - Set this bit if the driver has a read routine.

bit 6 - Set this bit if the driver has a close routine.

bit 7 - Set this bit if the driver has an open routine.

bit 8 - Set this bit if the driver has a link routine.

bit 9 - Set this bit if the driver has a size routine.

(Note that there is no bit specifying whether or not a block driver has a strategy routine. It turns out that config expects to find a strategy routine in all block drivers. An undefined external results if a block driver having no strategy routine is installed.)

"Bmajor" is the block major number of the driver, if any. Specify -1 otherwise.

"Cmajor" is the character major number of the driver, if any. Specify -1 otherwise.

Determine values for all fields of the /etc/master line, and enter that line in /etc/master. Here are some sample entries:

```
* name       prefix      type      mask      block      char
*
cs80         cs80         3         3FB        0          4
flex         mf           3         1FA        1          6
amigo        amigo        3         3FB        2         11
tape         tp           1          FA       -1          5
printer      lp           1          DA       -1          7
stape        stp          1          FA       -1          9
srm          srm629       1         1F2       -1         13
plot.old     pt           1          F2       -1         14
rje          rje          1         1FA       -1         15
ptymas       ptym         9          FC       -1         16
ptyslv       ptys         9         1FD       -1         17
ieee802      ieee802      1         1FD       -1         18
ethernet     ethernet     1         1FD       -1         19
hpib         hpib         1          FB       -1         21
gpio         hpib         1         1FB       -1         22
ciper        ciper        1          DA       -1         26
snalink      snalink      1         1C0       -1         36
dos          dos          1          F9       -1         27
```

For example, in the "cs80" line above, the CS/80 driver should have both a

cdevsw and bdevsw table entry (according to "type"), and contains routines for all entry points except a true select routine (seltrue is used instead). The block major number is 0, and the character major number is 4.

Add Driver Name to Dfile

Edit an existing dfile, or create your own, and add the name of your driver to it (the name to enter is the same as "name" in the /etc/master entry you created). This causes "config" to include it in the new kernel.

Execute Config

Execute the "config" routine as follows:

```
    config dfile
```

Config uses the information in dfile and /etc/master to create a conf.c file and a makefile called config.mk. The conf.c file contains all kernel configuration information modified per the instructions in /etc/master and dfile. For example, conf.c contains the new bdevsw and cdevsw tables, new kernel parameter settings, if any, etc. The config.mk makefile contains the instructions needed by "make" to compile and link a new kernel.

For each driver name mentioned in dfile, config finds a line in /etc/master whose first field matches that name, and uses the information on that line to complete configuration for that driver. It builds the cdevsw and bdevsw tables by looking at "type" (to determine if entries should be built at all) and "mask" (to determine which entry points the driver contains). Config fills in the cdevsw/bdevsw tables with pointers to the actual routine names by adding the "prefix" and an underscore to the beginning of each entry point defined by that driver, and installing the resulting string into the table. It also adds an "external" declaration for the resulting routine name to conf.c.

A portion of a cdevsw table from conf.c is shown below:

```
struct cdevsw cdevsw[] = {
/* 0*/  cons_open,cons_close,cons_read,cons_write,cons_ioctl,cons_select,
        C_ALLCLOSES,
/* 1*/  tty_open, tty_close, tty_read, tty_write, tty_ioctl, tty_select,
        C_ALLCLOSES,
/* 2*/  sy_open, sy_close, sy_read, sy_write, sy_ioctl, sy_select,
        C_ALLCLOSES,
/* 3*/  nulldev,  nulldev, mm_read, mm_write, nodev, seltrue, 0,
/* 4*/  cs80_open, cs80_close, cs80_read, cs80_write, cs80_ioctl, seltrue,
        C_ALLCLOSES,
/* 5*/  tp_open, tp_close, tp_read, tp_write, tp_ioctl, seltrue, 0,
/* 6*/  nodev, nodev, nodev, nodev, nodev, nodev, 0,
/* 7*/  lp_open, lp_close, nodev, lp_write, lp_ioctl, seltrue, 0,
    .
    .
    .
};
```

The commented numbers help identify which character major number each line is associated with.

Note that missing entry points are automatically filled in by "config" with "nodev". (Whether or not an entry point is missing is specified by "mask".) This means that the kernel will do nothing and return an error if a user process calls a system call corresponding to the entry point in whose slot the "nodev" exists. For example, using the above table fragment, if a user issues a read(2) system call on a device file using the lp driver (major number 7), the kernel will do nothing and return an error.

"Nodev" is appropriate anytime a driver does not have a particular entry point routine, and when calling that routine is considered an error. "Nulldev" can be used instead if calling a missing routine is not really erroneous. If you want to specify "nulldev" instead of "nodev" for particular entries in the cdevsw or bdevsw tables, you must edit conf.c by hand after "config" has finished executing.

The C_ALLCLOSES flag can be specified via "mask". If it is not specified, a zero appears in that slot. If the C_NODELAY flag is desired, it must be manually added after "config" is finished executing.

Modify Config.mk

The name of the new driver's object file must be added to the makefile created by "config". The object file name must be added to the HP-UX dependencies line and to the line containing the linker command string. The exact placement is shown below (the new driver's object file is represented by "MYDRIVER.o"):

```
    .
    .
    .
    hp-ux:   conf.o MYDRIVER.o
             rm -f hp-ux
             ar x /etc/conf/libkreq.a locore.o vers.o name.o funcentry.o
             @echo 'Loading hp-ux...'
             $(LD) -m -n -o hp-ux -e _start -x \
                    locore.o vers.o conf.o name.o funcentry.o MYDRIVER.o \
                    $(LIBS1) $(LIBS)
             rm -f locore.o vers.o name.o funcentry.o
             chmod 755 hp-ux
    .
    .
    .
```

Execute Make

Now execute "make" with

```
    make -f config.mk
```

This will compile the new conf.c file and link it with the various kernel libraries to produce a new HP-UX kernel. The new kernel is called "hp-ux", and is created in your current directory (usually /etc/conf).

Install the new kernel with:

```
    mv /hp-ux /SYSBCKUP
    mv hp-ux /hp-ux
```

and then reboot the system.

NAME

disked - interactive disk editor for HFS

SYNOPSIS

disked [-w] [-b <#>] <special-file>

DESCRIPTION

Disked is an interactive disk editor that examines and modifies an HFS file system. It operates on either a character or block device associated with a file system. The file system should be unmounted while disked is being run on the file system.

Disked reads commands from standard input and writes to either standard output or standard error. Although it was designed to be run interactively it can be used in batch mode by redirecting standard input. Most of the commands read data from disk into a buffer maintained by disked. Each command which reads from disk will overwrite this buffer.

Disked normally opens special-file read-only. If the w option is specified then special-file is opened for reading and writing. Only by setting the w option is it possible for the user to damage the file system.

If b option is specified, disked will use the specified alternate superblock instead of the primary superblock to interpret the file system.

Disked maintains two buffers called the browser and edit buffers. At any point in time only one of these two buffers is considered the current buffer. The x command can be used to switch the current buffer from the browser buffer to the edit buffer and vice-versa. The only significant difference between these two buffers is that it is possible to modify the disk when using the edit buffer. Disked initially sets the current buffer to the browser buffer. For more information see the section on Buffer Commands.

The output of most of the commands can be redirected using the disked operators ">", ">>", and "|". The ">" symbol is used to redirect the output of an individual command to a file. The ">>" symbol provides the same functionality except that the output is appended. The "|" symbol is used to pipe the output of an individual command to any Unix command. For example, if the user wanted to redirect the output of the s command (display primary super-block) to a file called "foo". The following command would work:

```
s > foo
```

The following is a detailed list of commands:

General Commands:

command	short description
b <n> <c>	display <c> bytes starting from byte <n>
f <n> <c>	display <c> bytes starting from fragment <n>
r <n> <c>	display <c> bytes starting from sector <n>

These commands are used to display data. The user is given the option of specifying a byte address (b command), a fragment number (f command), or a raw disk sector (r command - note: a raw disk sector is in terms of DEV_BSIZE units.). The count argument <c> is optional for the f and r commands, and defaults to the fragment size or to DEV_BSIZE bytes, repsectively. Each command displays data in the same format. The format is a byte address counter followed by a sequence of numbers and the character representation of those bytes. With the default settings, each number represents 4 bytes and is displayed in hex. The counter is initially displayed in decimal. The default values are changed by setting the variables wordsize, displayin and countin (see User settable variables below). All of these commands will allow the user to display from 1 to MAXBSIZE worth of data.

command	short description
i <n>	display inode
p <path>	display inode
d <n> <c>	display <c> bytes of directory entries starting from fragment <n>

These commands allow the user to traverse the directory tree. The i command can be used to display the contents of the specified inode. The root inode of an HFS file system is inode 2. The p command can be used to display the contents of the inode represented by <path>. If <path> is a relative pathname (does not begin with a '/'), it will be interpreted as though the file system were mounted as the root file system and the current working directory were the root directory. An absolute pathname will be interpreted first as if the file system were mounted at the current or last mount point of a larger file hierarchy (using the fs_mnt field of the superblock); failing this, <path> will be interpreted as though the file system were mounted as the root file system.

The d command is useful for displaying the data blocks of a directory inode as directory entries. Because data block addresses in the inode are really fragment numbers, this command (like the f command) takes an optional count argument <c>. If <c> is not specified, it defaults to the size of a fragment.

command	short description
q	exit the program

Allow normal termination of the program. If the edit buffer has been modified the q command will not allow the user to exit disked (see Q command).

Buffer Commands:

command	short description
x	switch current buffer
X	switch meaning of browser and edit buffers

The x command is used to switch which buffer is the current buffer. When disked is first invoked the current buffer is the browser buffer. To edit the disk the user must change the current buffer to be the edit buffer. Then the user can read the data into the edit buffer and modify it. It is then possible to leave the changes in the edit buffer and switch buffers to the browser buffer. The user can then search through the disk without losing the changes. When the user wants to write the changes out, the user can switch back to the edit buffer, and use the W command to write the data to disk.

The X command is similar to the x command except that X swaps the meaning of the browser and edit buffers such that the current browser buffer, along with its contents, becomes the edit buffer and vice versa. This makes it convenient to modify data already in the browser buffer without having to switch buffers and read in the same data to the edit buffer.

Modification Commands (edit buffer only):

command	short description
m <off>[:<rep>] <arglist>	modify buffer
m <start>[-<stop>] <arglist>	modify buffer

W write modified buffer

The m command allows the user to modify the current buffer (which must be the edit buffer) at buffer offset <off> to be <arglist>. <arglist> is a list of numbers or characters separated by one or more blanks. If a rep is specified then the arglist will be repeated that many times. Off may be specified as either a number or as an offset into a known structure (for a list of known offsets type h offsets). Alternatively, the user may specify a range within the buffer to be modified. Each term in the arglist is put into a different word. Each word represents 1, 2 or 4 bytes depending on the value of wordsize. The only legal values for wordsize are 1, 2 or 4. The terms in the arglist will be padded so that each term completely fills one wordsize unit.

The W command is used to write the modified buffer to disk.

Note: Two ways exist to undo changes made to the current buffer. The first is to read data into the current buffer. This can be done with almost any of the commands. The second is to abort the program using the Q command.

command	short description
Q	abort program

Abort the program even if the edit buffer has been modified. All changes are ignored and the program is terminated.

Internal Data Structure Commands:

command	short description
s [s][r]	display primary super-block
s <n> [r]	display redundant super-block <n>

These commands are used to display either the primary super-block or the redundant super-block associated with each cylinder group. Included in each super-block is a rotational table. The r option is used to to display this table. In addition, the first n blocks of data space contain summary information. The s option can be used to display this while displaying the primary super-block.

command short description

c <n>   display cylinder group <n>

This command is used to display the contents of any cylinder group.

Use of expressions:

Many disked commands expect one or more numbers as arguments. If a command expects a number then the number can always be replaced with an expression. An expression is either an integer or a parenthesized expression containing one or more of the following arithmetic operators: |, &, *, /, +, -. Further, an expression can contain any number of macros. Disked maintains a list of macros which can be invoked (type - h macros). As an example suppose the user wanted to display the cylinder group associated with a particular inode. One mechanism would be to use knowledge of how a disk is laid out and calculate the number by hand. The preferable method is to use the c command passing as an argument itog(<inode number>).

Free List Manipulation

command	short description
w > <file>	write current buffer to <file>
w >> <file>	append current buffer to <file>

With these two commands it is possible to walk through the free lists and recover lost data.

example:

In the following manner it is possible to read the free data blocks of one **unmounted** file system and write the data blocks to a file on a **mounted** file system. The c command can be used to obtain a list of free fragments in each cylinder group. With this information the f command can be used to read the free fragment into the current buffer. The following formula will convert a cylinder group relative fragment number to a file system relative fragment number (<fragment-number> + cgbase(<cylinder-group-number>)). Once the data has been read into the current buffer, it can be written to any file on a mounted file system with the w

command.

Extended commands

command	short description
copyi <inode number>	display data for inode
map	display a map of this disk
tell <fragment>	describe fragment
bgrep "string" <b> <e>	

Copyi takes as input an inode number and displays the data blocks associated with it. It is very important that the user ensure that the specified inode is valid. The size and blocks fields in the inode must be correct or disked might not be able to display the data blocks. In addition, it is very important that checking not be turned off when this command is executed (see User settable variables).

Map is used to display a fragment map of all fragments on the disk.

Tell takes as input a fragment number and provides information about the specified fragment.

Bgrep searches for the specified string starting from fragment <b> and until fragment <e> and displays the fragment number of any fragment that contains this string. If <b> and <e> are not specified, then the search defaults to the whole disc. The string must be enclosed in double quotes and may contain C style escape characters and **grep(1)** style regular expressions.

User settable variables:

command	short description
set <variable> <value>	assign <value> to <variable>

This command is used to set any one of a number of different global variables. What follows is a list of variables and their possible values and then a description of what each variable does:

variable	possible values (default values are in bold)

check	(**on**, off)
countin	(octal, hex, **decimal**)
displayin	(octal, **hex**, decimal)
display	(**on**, off)
init	(**on**, off)
wordsize	(1, 2, or **4**)

check
This variable controls whether or not certain error checks are performed by _disked_. _Disked_ goes to great lengths to prevent the user from damaging the file system. Turning this variable off will prevent _disked_ from performing these checks. This should obviously be done only with great care if _disked_ is being used with the _w_ option.

countin
This variable determines the radix in which the counter is displayed for the _b_, _f_, and _r_ commands.

displayin
This variable determines the radix in which data is displayed for output (with the _b_, _f_, and _r_ commands).

display
This variable controls whether or not the _b_, _f_ or _r_ commands will display the data when it is read in. It is useful to unset this variable when copying a known set of free blocks from the device to a file on another disk.

init This variable controls whether or not the _edit_ and _browser_ buffers are re-initialized when a new disk is opened (see _n_ command). By unsetting this variable it is possible to copy at most MAXBSIZE worth of data from one disk to another.

wordsize
This variable controls the primary wordsize (number of bytes in a word) for the program. On output, it affects the amount of data to be displayed at any point in time. On input, it will control the amount of data overwritten for each _argument_ in the _arglist_ of the _m_ command.

Miscellaneous commands:

command _short description_

```
h <topic>                            provide on-line help
? <topic>                            provide on-line help
h help                               list topics available for help
B                                    display current buffer as data
C                                    display current buffer as cylinder
D                                    display current buffer as director
F                                    display current buffer as data
I                                    display current buffer as inodes
R                                    display current buffer as data
S                                    display current buffer as super-bl
! <command>                          execute monitor command
n [-w] [-b <#> ] <special-file>      restart program using <special-fil
                                     and specified options
```

command	short description
= <n>	display number

This command takes as input an expression and displays the value of that expression in hex, octal and decimal.

command	short description
$<a-z> = expr	assign a value to a local variable

This command assigns the expression to a local variable. There are 26 local variables $a - $z. Once a local variable has a value it can be used in any expression. To display the value of a local variable use the = command.

In addition to the 26 local variables, disked supports two local variables called $size and $address. These variables are the size and address of the current buffer. They may be used in any expression where a local variable is used. This enables the user to reference the size and address of the current buffer, without typing in the actual numbers. Further, if the current buffer is the edit buffer then the user can change the values of $size and $address. This has the effect of changing where disked believes the data resides. By changing $address and then writing the edit buffer out, the user can move data from one place to another on the disk.

example:

The following example display the contents of the n-th cylinder group; where n is (0x314 + 12) / 013.

```
$a = (0x314 + 12) / 013
c $a
```


21

SE 390: Series 300 HP-UX Internals

Monday Afternoon Labs

0. If you have NOT used "monitor" much, run it and take a look at each of the screens of information. Use the online help facility. What things does monitor(1m) tell you that you can't (yet) make use of?

1. Using the template provided (ppt.c), print out the values of at least 5 kernel parameters. Verify 2-3 of them with monitor(1m). If you want ideas on what to print, look at space.h or monitor's C screen.

2. Look through the "pm" and "misc" directories in the examples archive I gave you. Are there useful functions (or whole programs)?

3. Start work on your version of monitor, focusing on process stuff. Consider printing (among other things)

- the process table (like ps does)
- the proc table entry and u area for a given process
- relevant kernel parameters

SE 390: Series 300 HP-UX Internals

Tuesday Afternoon Labs

1. Change the major number of some driver in /etc/master and rebuild your kernel. Then make a corresponding device file and reboot. Change something that 1) you can verify and 2) won't kill your machine if you mess up. A good candidate would be character-mode SCSI/CS80 (whichever one your disk is).

2. Install the ramdisk driver on the system and add code to print out the the size and 1k block address whenever a block is read or written (there is a printf() in the kernel just like there is in libc for user programs). You will probably need to replace the one that is already there (use "ar t" to figure out which library it is in).

3. Reconfigure your kernel and look at the conf.c that gets generated. Which parts of it come from dfile? Which come from /etc/master?

4. Force your system to panic and interpret the resulting stack trace. (misc/th_init is helpful here... :-))

5. Take a look at the supplied pseudo-driver called "pdisk". How does it compare to the pty drivers (the things that enable telnet/X11/script to work)?

SE 390: Series 300 HP-UX Internals

Wednesday Afternoon Labs

*** Be sure to look at the examples in the "fs" directory before doing these labs; also, note that many of them are easier on a ramdisk ***

0. Write a program to hunt for superblocks on a disk.

1. Translate a pathname to an i-number using adb(1), fsdb(1m), disked(1m), or a C program you write.

2. Modify "myls.c" to be something along the lines of "myll.c"; in other words, get the inode for each file and print things like the size, owner UID, etc.

3. Use the ramdisk driver (or pdisk driver/server) to learn about the filesystem's layout and "habits". How is the filesystem affected by fs_async?

4. Mess up the disk using disked(1m) or some other command (You needn't get too violent - how about dd(1)ing over the 1st 16K?) Then fix it using fsck(1m), disked(1m), or whatever you want (dd(1)ing from another disk is strictly an option of last resort :-))

***** OR *****

Write a version of cat(1) that uses only a disk device file.

Diskless

1. Cluster your system with another, and look closely at what monitor(1m) will tell you about both machines.

2. Locally mount a ramdisk, and make it so that noone else in the class can access the stuff down under the mount point. This is not tricky/hard/etc :-)

SE390: Series 300 HP-UX Internals

VM System Labs

0. Look through the "vm" directory in the examples archive I gave you. Are there useful functions (or whole programs)?

1. See what monitor, iostat(1), and vmstat(1) will tell you about the state of the VM system. How does their output change if you run a program that chews up lots of RAM (try memory/paging.c)?

2. Write a program that will summarize swap space usage by looking at swaptab[], swapspc_max, and swapspc_cnt. It should produce output something like this:

```
        there is a total of XXX MB on the system
        YYY MB is free
        ZZZ MB is allocated
        AAA MB is reserved but not yet allocated
```

You might want to enhance it to summarize diskless client usage as well, i.e.

```
        BBB MB has been allocated to <name of client 1>
        CCC MB has been allocated to <name of client 2>...
```

Note that you do not need to walk through each swaptab[]'s swapmap array.

3. What had to change in "top" for it to work in 8.0? Change it so that it sorts by size instead of CPU usage (i.e. have it print the 10 (or whatever) *biggest* programs, rather than the 10 that are using the most CPU time).

4. Add some VM-related stuff to the "monitor" you started on Monday.

SE 390: Series 300 HP-UX Internals

Friday Labs

0. Shut down the system and reboot it, watching carefully to see what gets printed out. What is the last line printed by the kernel? What is the first line printed by init(1m)?

1. Finish/clean up your labs, and see if there are things in monitor that you recognize now that didn't make sense earlier.

2. Give your instructor a copy of your monitor and your filesystem programs. Please put them in a {shell,cpio,tar} archive. Thanks!

```
 1  #include <stdio.h>
 2  #include <sys/param.h>
 3  #include <fcntl.h>
 4  #include <sys/user.h>
 5  #include <sys/proc.h>
 6
 7  /*
 8   *  Example of reading /dev/kmem to get at kernel data
 9   *  structures.  Note that this is NON-PORTABLE and
10   *  UNSUPPORTED - it may break with future releases of
11   *  HP-UX.  It's fun, though :-)
12   *
13   *
14   *  first we declare a data structure that will be passed to nlist(3);
15   *  note that we are only filling in the first member of each structure
16   *  in the array, and that we end with a null member
17   */
18
19  struct nlist nl[] = {               /*  setup for calls to nlist(3)  */
20  #ifdef hp9000s800
21          { "nproc" },                /*  # entries in process table  */
22          { "proc" },                 /*  pointer to process table  */
23  #else
24          { "_nproc" },               /*  # entries in process table  */
25          { "_proc" },                /*  pointer to process table  */
26  #endif
27          { "" }
28  };
29
30  #define C_NPROC 0                   /*  indices into the above array  */
31  #define C_PROC  1
32
33  int kmem;                           /*  file descriptor for kernel mem  */
34
35
36  main()
37  {
38          startup();
39          walk_table();
40          exit(0);
41  }
42
43
44  startup()                   /*  read symbol table & open kernel memory  */
45  {
46          if (nlist("/hp-ux", nl) < 0) {
47                  perror("nlist(3)");     /*  can't get symbol table  */
48                  exit(1);
49          }
50
51          if ((kmem = open("/dev/kmem", O_RDONLY)) < 0) {
52                  perror("open(2)");      /*  can't open kernel mem  */
53                  exit(1);
54          }
55  }
56
```

```
 57
 58
 59
 60
 61  walk_table()            /* step through the process table  */
 62  {
 63          int i, nproc;
 64          long pt_addr;
 65          struct proc *proc_table, *p;
 66
 67
 68          /*
 69           *      first go get the value of nproc from /dev/kmem, using
 70           *      the address nlist(3) returned to us
 71           */
 72          lseek(kmem, nl[C_NPROC].n_value, 0);
 73          read(kmem, &nproc, sizeof nproc);
 74          proc_table = (struct proc *) calloc(nproc, sizeof(struct proc));
 75
 76          /*
 77           *      now get the *address* of the proc table, seek there,
 78           *      and get the real thing; this is because proc is a
 79           *      pointer rather than a simple variable
 80           */
 81          lseek(kmem, nl[C_PROC].n_value, 0);
 82          read(kmem, &pt_addr, sizeof pt_addr);
 83          lseek(kmem, pt_addr, 0);
 84          if ((i = read(kmem, proc_table, sizeof(struct proc)*nproc)) < 0) {
 85                  perror("read proc_table");
 86                  close(kmem);
 87                  exit(1);
 88          }
 89
 90          /*
 91           *      we have the proc table; get in a loop and step through
 92           *      the whole thing, printing a line for each slot that
 93           *      is being used
 94           */
 95
 96          p = proc_table;
 97
 98          for (i = 0; i < nproc; i++) {
 99                  if (p->p_stat)                  /*  if entry in use  */
100                          printf("pid, pgrp, uid, ruid are %d %d %d %d\n",
101                                  p->p_pid, p->p_pgrp, p->p_uid, p->p_suid);
102                  p++;
103          }
104
105          close(kmem);
106  }
107
108
109
```

A Quick Introduction to adb(1)

When in the course of human events it becomes necessary to patch a kernel or examine it, there are very few commands that will do the job. One possibility is adb(1), a general-purpose debugger that is capable of doing most anything. It is hard to use, but sometimes it's the only thing available....

If you need to use adb(1), here are some annotated examples. Note that adb(1) really only knows about executable files and core files; since /hp-ux is an executable and /dev/kmem is kernel memory (which has basically the same format as a core file), we can use it to work on the kernel. The "# " in each example was printed by the shell; everything else left of the arrows below was typed in by the intrepid hacker :-)

```
# adb /hp-ux
dfile_data?s            <--- print variable "dfile_data" as a string
                             from /hp-ux (note the "?")

19232?10i               <--- disassemble; print 10 instructions
                             starting at address 19232

# adb -w /hp-ux /dev/kmem
fs_async/D              <--- print variable "fs_async" as an integer
                             from /dev/kmem (note the "/")
/W 0                    <--- set it to 0 (turn it off) in /dev/kmem
```

Note that using "/" will cause adb(1) to work with the "core" file (/dev/kmem) and that this will either take effect immediately (for a simple variable) or not work at all (for something like nproc which sizes a data structure).

Using "?" will direct adb(1) to the "a.out" (/hp-ux), which won't take effect until you reboot (which may be what you want, and which is your only choice if you are changing the size of a table in the kernel).

One last thing: adb(1) is, uh, somewhat lacking in its user interface :-) It is *very* picky about syntax, case, etc; in the string "fs_async/D" above, it really does have to be a capital "D". To get out of the program, use either "$q" or the old standby, "<ctrl-d>".

SE390: Series 300 HP-UX Internals

Memory Management

A Thousand Words Worth :-)