SE 390: Series 300 HP-UX Internals

Introduction

Greeting & Introductions

Your Expectations

My Expectations

- This class must be SE-driven.
- This class must be practical.
- This class must keep evolving.
- We are *very* interested in constructive criticism and suggestions. As much as possible, put your comments in writing on the module evaluations or the end-of-class evaluation.
- 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

Overview of the Kernel - What's the Big Picture?

- What is it there for?
- What are the chief components?

SE 390: Series 300 HP-UX Internals

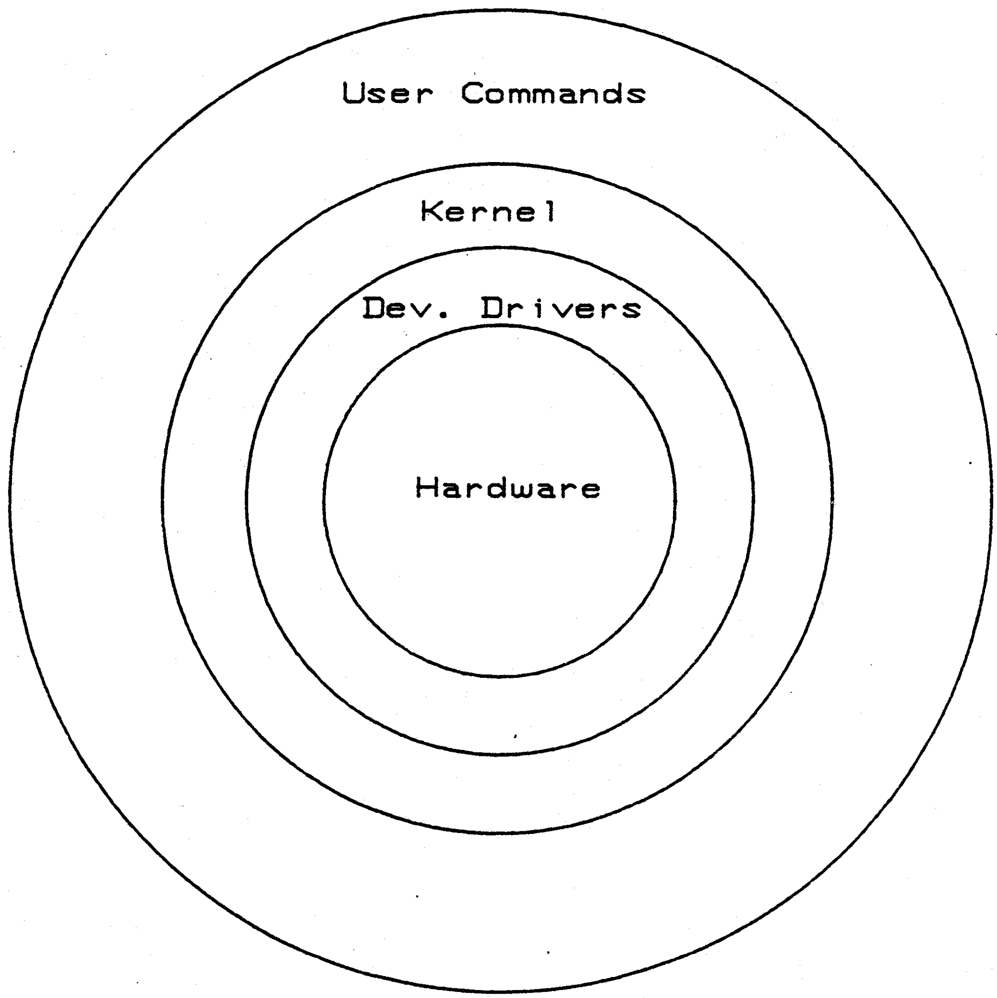
Introduction

The "Big Picture" of the Kernel

- What is it there for?
    - manage resources
    - make life easier for the programmer

- What are the major components?
    - process management
    - memory management
    - file system
    - I/O system
    - diskless nodes
    - access to the system
    - fundamental kernel data structures and library routines

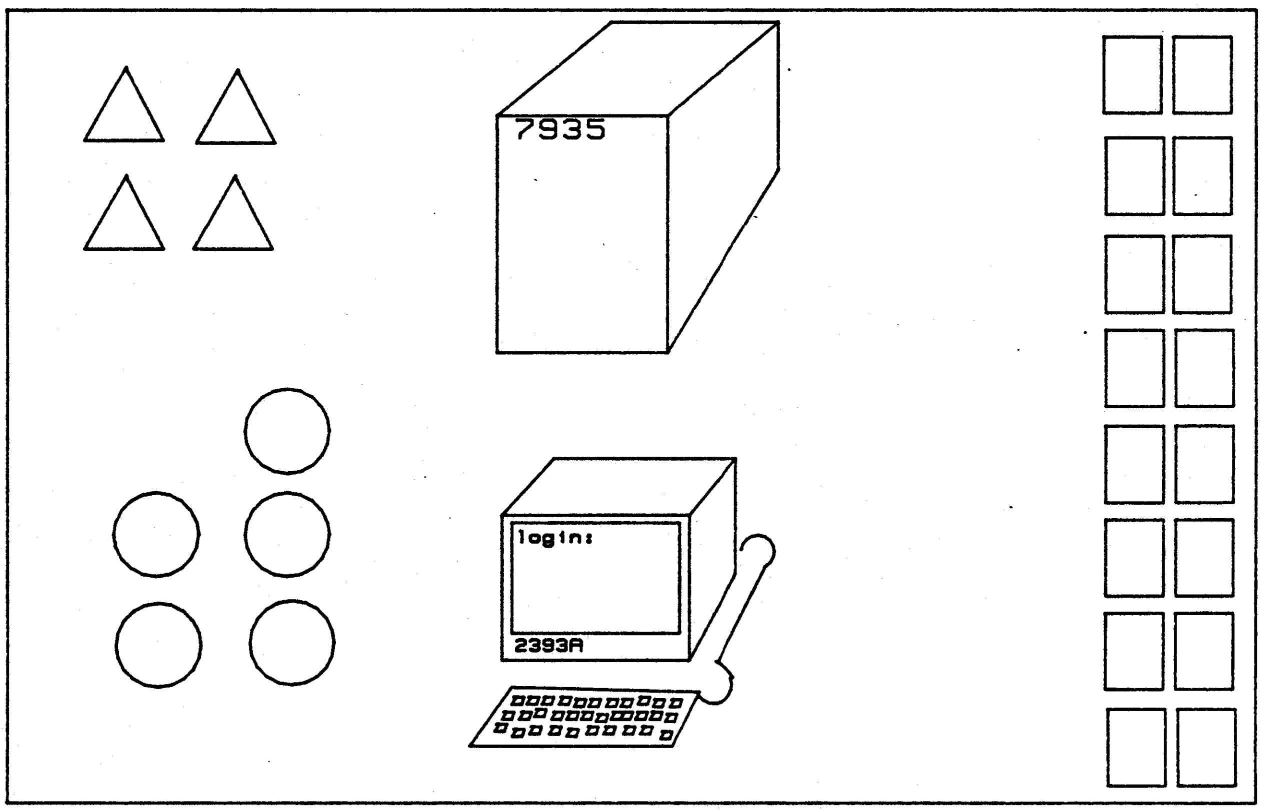
The $350* Onion



*Single-user AXE

HP-UX - The Big Picture

□ - memory   △ - system process   ○ - user process



startup - starting △'s, enabling ○'s, mapping □
shutdown - erasing △'s and ○'s; flushing □ to disk
I/O - sending stuff between □ and 7935, 2393
Memory Management - how we allocate and map □
Process Management - how we control ○'s
IPC - how ○'s talk to each other
filesystem - how we use and map the 7935

SE 390: Series 300 HP-UX Internals

Introduction

Process Management

- Creation of processes.
- Deletion of processes.
- Inter-process communication.
- CPU Scheduling.

Introduction

Memory Management

- Allocating memory.
- Freeing memory, voluntarily or otherwise.
    - voluntarily
    - pager
    - swapper
- Physical vs. virtual memory.
- Sharing of memory among many competing processes.

SE 390: Series 300 HP-UX Internals

Introduction

File System

- The Vnode layer.
- Caching.
- The HFS/Berkeley/McKusick filesystem.
- Examples:
    - open(2)
    - write(2)

Introduction

I/O System

- Block devices.
- Character devices.
- Buffering.
- Outline of driver structure.
- Flow of control in the I/O system.

Introduction

Diskless Nodes

- "Look Ma, no disk!"
- Diskless protocol.
- Vnode layer in the filesystem.
- Sharing: pids, file locking, swap space.

SE 390: Series 300 HP-UX Internals

Introduction

Access to the Kernel

- System calls.
    - front ends in libc
    - change modes with TRAP
    - trap handler calls syscall()
    - actual system call code is called indirectly
- The assembly-level debugger, adb(1).
- Calls to nlist(3).
    - YOU ARE ON YOUR OWN
    - call nlist(3) to get address of kernel symbol
    - open /dev/kmem and seek to address
    - read information
    - YOU ARE ON YOUR OWN - KERNEL DATA STRUCTURES CHANGE FROM RELEASE TO RELEASE!

SE 390: Series 300 HP-UX Internals

Introduction

Fundamental Kernal Data Structures And Library Routines
[Note that this is not meant to be complete.]

- Core map ("cmap") - has an entry for each "normal" page of physical memory. It is used by the pageout daemon.

- Resource maps - each of these is a list of {address,size} pairs that keeps track of some kernel resource.

    - swapmap - used to keep track of free swap space.

    - kernelmap - keeps track of space for page tables

- proc table - there's an entry in this for each process.

- text table - has an entry for each shared-text program.

- file table - an entry for each open file.

- inode table - used for inode caching.

- mount table - has an entry for each mounted volume.

- swap device table - has an entry for each device that is supposed to have swap space on it.

- sleep()/wakeup() - used to wait for something to become available. If a process needs some resource (like a driver or chunk of memory), it will request it and then sleep on the address of that resource. When whoever is using that resource is done, it will do a wakeup() on the address of the resource, causing all processes that were waiting to wake up. One of them will get the resource, and the others will go back to sleep.

- rminit()/rmfree()/rmalloc() - used to allocate things from the resource maps mentioned above.

- spl?() - used to change processor level. If the kernel needs to fool with a sensitive data structure, it will do an spl6() to block out interrupts, fool with it, and then set the priority back down with spl0() or splx(old_priority).

- copyin()/copyout() - used to move things between user space and kernel space.

- fuword()/suword()/fubyte()/subyte() - used to fool with a single byte or word in user space.

Introduction

Glossary Of Terms

pte - page table entry

zombie - process that has exited, but hasn't been wait(2)ed for yet

kluster - group of adjacent pages that are put out to swap space together

cluster - group of hardware pages that are grouped together for efficiency. On the 300 this isn't done since the hardware page size is 4K.

click - on the 300, a page

"push a page" - kick it out to swap space

poip - "page out in progress"

SE 390: Series 300 HP-UX Internals

Module Evaluation

INTRODUCTION

On a scale of 1-10, 1 being bad, 5 being OK/don't care/irrelevant, 10 being good, please rate the following. If you have particular comments, please write them in. Thank you!

1. Clarity of presentation:

2. Depth/complexity (1 - material was too easy, 10 - it was too hard):

3. Usefulness/applicability/relevance of material presented:

4. Speed of presentation (1 - too slow, 10 - too fast):

5. How good was the material (slides, notes, etc)?

6. How good was the instructor?

Ways this could be improved (please be specific):

General Comments:

SE 390: Series 300 HP-UX Internals

System Startup

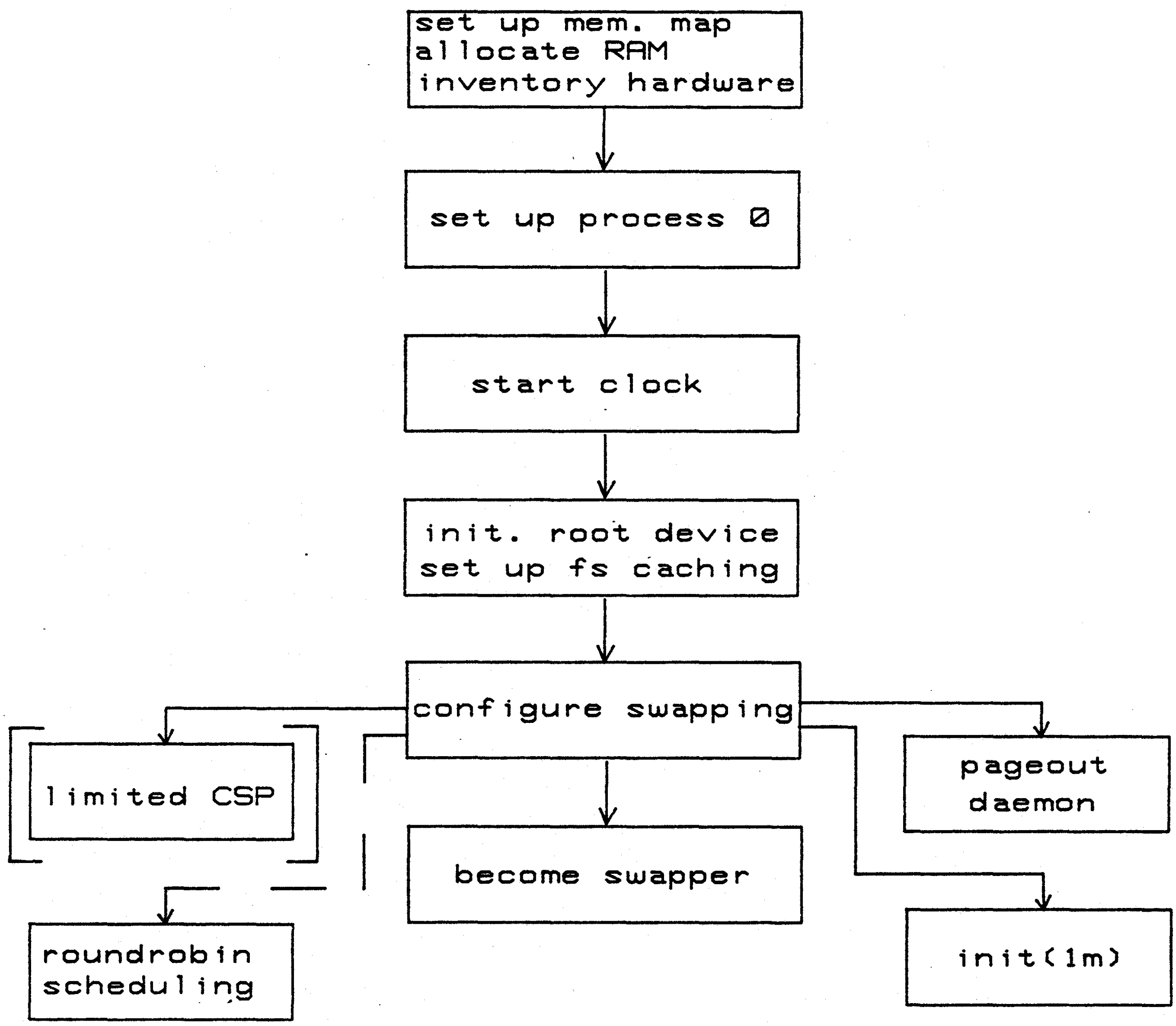
The Big Picture

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

The Little Pictures

- What is the correspondence between things being accomplished and things being printed on the console's screen?
- Configuring the virtual-memory subsystem.
- Allocating and initializing kernel data structures.
- Preparing for I/O.
- Kicking off the first three processes.

System Startup



System Startup

Internal Actions vs. External Signs

- "booting /hp-ux"

  set up kernel page table [s200/locore.s: start()]
  get information from bootrom: processor type, amount of RAM,...
  initialize the run queues [sys/kern_synch.c: rqinit()]
  allocate memory for DOS coprocessor [s200/machdep.c: startup()]
  allocate memory for the buffer cache, core map, inode table,
  file table, callout table, and other structures
  clear out memory and decide if we have enough to continue
  initialize 68881
  call device driver link routines (array of pointers to them
  is set up in /etc/conf/conf.c)
  look for ttys, init. console [s200io/kernel.c: tty_init()]

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

  enable parity detection [s200/machdep.c: parity_init()]
  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 process 0 [sys/init_main.c]
  start clock [s200/clocks.s startrtclock()]
  initialize root device [s200/machdep.c: rootinit()]
  initialize diskless

- "Local link is xxxxxxxxx"
  "Server link is yyyyyyyy"
  "Swap site is nn"
  "Root device major is xx, minor is yyyy [root site is xx]"

  set up for inode hashing [sys/ufs_inode.c: ihinit()]
  set up for block hashing [sys/init_main.c: bhinit()]
  initialize buffer cache [sys/init_main.c: binit()]

- "Swap device table: (start and size...)" \ these are present
  ".... (line for each entry) ...." / only if local swap

  configure swap devices [conf/swapconf.c: swapconf()]
  mount root filesystem
  start up CPU roundrobin scheduling
  start up paging subsystem
  start up limited CSP

- "avail mem = xxxxxxxx"
  "lockable mem = xxxxxxx"
  <copyright & restricted rights legend>

  fork init
  become the swapper

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

Starting Up The Virtual Memory System

- Set up the kernel page table such that the kernel can fool with it.
- Reserve memory for the DOS coprocessor if dos_mem_byte was specified.
- Initialize kernel memory map.
- See what swap devices are available.
- Fork process 2 to be the pageout daemon.
- Enter swap scheduling loop.

Allocating And Initializing Kernel Data Structures

- Figure out how much RAM we have, what model we are, etc.

- Allocate space for the buffer cache, proc table, inode table, file table, callout table, and various other things.

- Initialize queues and tables to be empty.

- Construct process 0 by hand - we are running as if this was a single-tasking machine, so we just put values into data structures to reflect what we're doing. "It is often easier to obtain forgiveness than permission."

- Mount the root file system and get root inode.

System Startup

Preparing For I/O.

- Call device driver initialization routines.
- See what cards are installed.
- Look for a console.

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.

- 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).

- Start CSP if we are in a diskless cluster.

SE 390: Series 300 HP-UX Internals

Module Evaluation

SYSTEM STARTUP

On a scale of 1-10, 1 being bad, 5 being OK/don't care/irrelevant, 10 being good, please rate the following. If you have particular comments, please write them in. Thank you!

1. Clarity of presentation:

2. Depth/complexity (1 - material was too easy, 10 - it was too hard):

3. Usefulness/applicability/relevance of material presented:

4. Speed of presentation (1 - too slow, 10 - too fast):

5. How good was the material (slides, notes, etc)?

6. How good was the instructor?

Ways this could be improved (please be specific):

General Comments:

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)

- Process creation/deletion.
- Fork - duplicate current process.
- Exec - replace current program with another.
- Work done on behalf of processes (system calls, interrupt handling).
- Context switching.
- Important data structures.
- Signals & IPC.
- Process states.
- Tunable parameters.

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
- Deleted by exit(2) (voluntary), or most signals (involuntary).
- 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

Process Management

What Happens When Fork(2) Is Called

- If not vfork, get *swap* space.
- Get a PID.
- Be sure there is a proc table entry and we can have it.
- If vfork and process is plocked, get lockable memory for u area and page tables.
- Copy proc table entry, changing fields where appropriate.
- Get page tables for the child.
- Copy u area, changing where appropriate.
- Clear interval timers in the child.
- If vfork, give virtual memory to child.
- Attach to text segment.
- If fork, copy virtual memory.
- Put child on run queue.
- If vfork, wait for child to exit(2) or exec(2).

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.
- Be sure it's a normal S300 object file or an old S200 one.
- Copy arguments to swap space (this is what the kernel parameter argdevnblks is all about).
- Be sure the file is big enough to have text, data, etc.
- If it's the old object format, pad it to 1/2 MB boundaries and move it up to start at 8K.
- Be sure text isn't busy: ptrace(2), open for write, etc.
- Be sure it's not too big - we can't exec a 16 GB file.
- 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.
- Get proper sysent table (there's a "compatibility" set of system calls for running old S200 executable files).
- Copy arguments from swap space to 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.

SE 390: Series 300 HP-UX Internals

Process Management

Work Done by the System For the Processes

- System calls - similar to library functions, but differ in important ways:
    - run on the kernel stack
    - have access to system data structures
    - provide protected way of getting at shared resources
- Interrupt handling
    - transparent to user
    - run in supervisor mode
- Signal sending and receiving
    - crude form of IPC
    - can be controlled somewhat with sig*(2)
    - makes use of fields in proc table entry

Context Switching - Priorities

- 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 is computed every four clock ticks for the current process

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

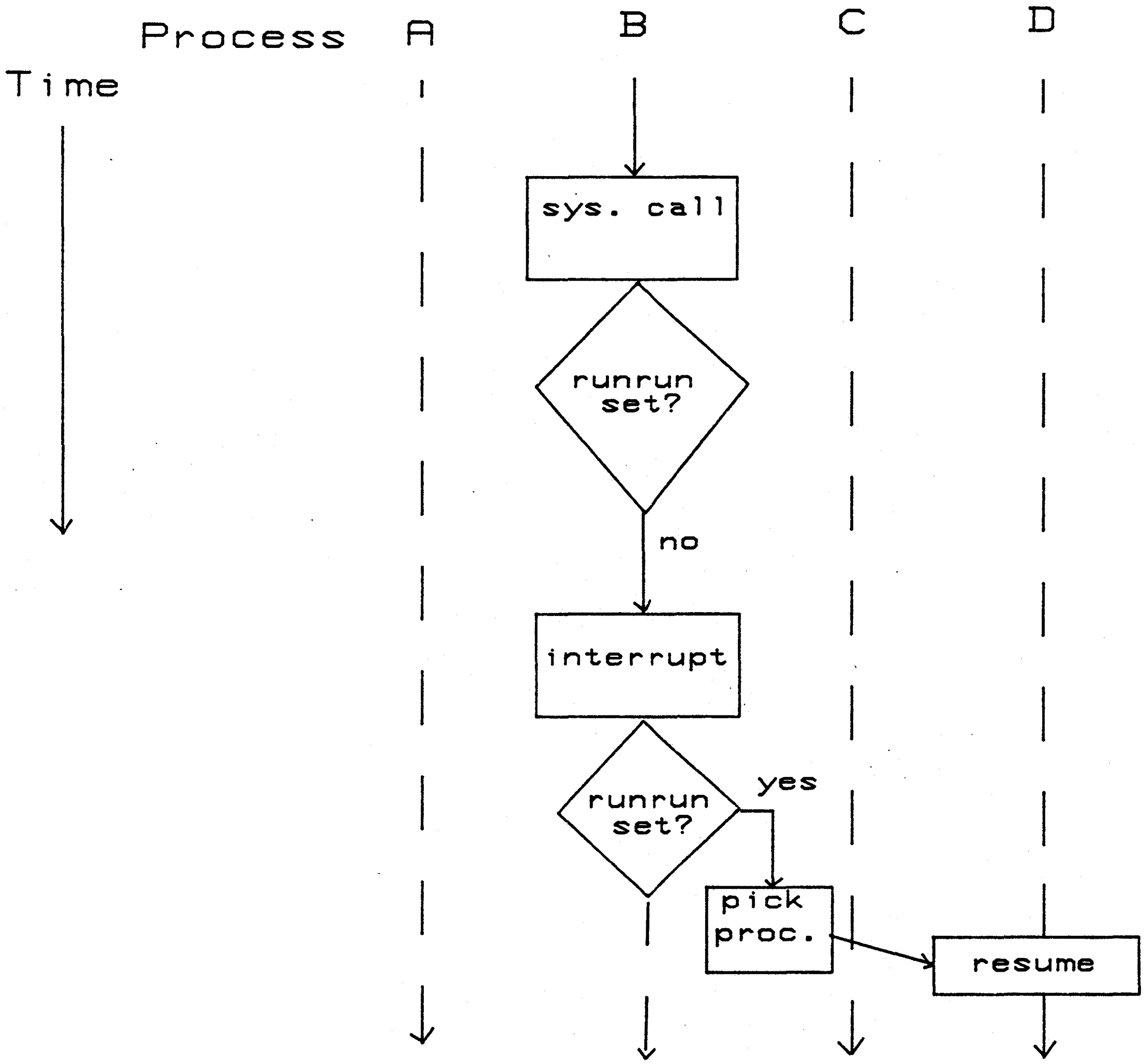
- If process has been rtprio()'ed, forget the 2nd part....

- 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.

ontext 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. This case only happens when someone else becomes more important to run and the system has noticed.
- Save current context into u area, which is mapped into the top of the process' address space.
- Restore other process' context from its u area.
- Resume execution.

Context Switching



SE 390: Series 300 HP-UX Internals

Process Management

The Context of a Process

- 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, kernel stack, etc.
- The proc table entry - defined in /usr/include/sys/proc.h
    - stuff that needs to always be available - priority, PID, signal masks, etc.

Signal Handling

- Signal sending
    - crude form of IPC
    - accomplished with kill(2)
    - SIGUSR[12] are available for cooperating processes

- Signal receiving or "catching"
    - 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

- 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....

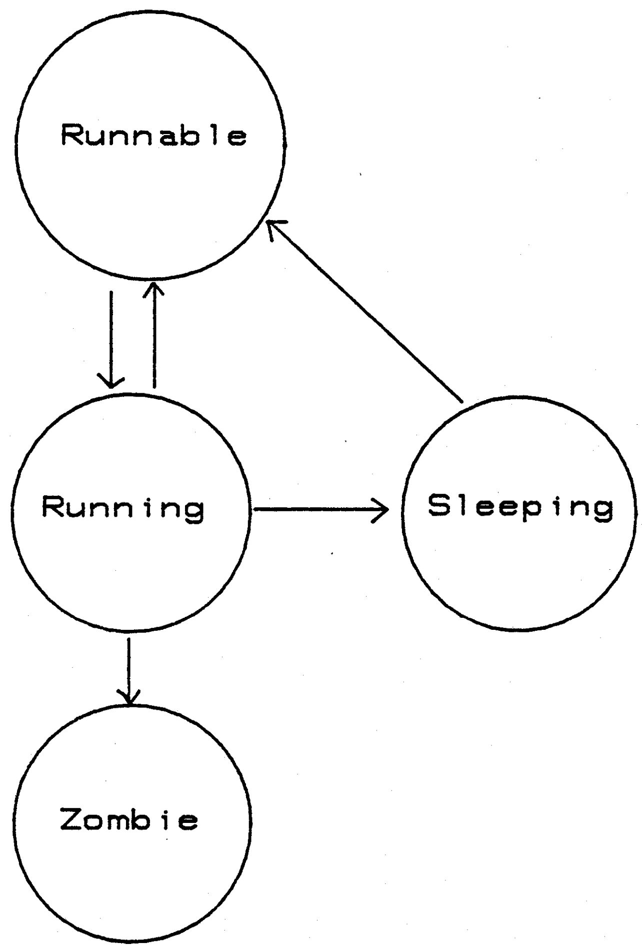
    - if we were in the middle of a system call, we may restart it or we may return an error - depends on what user asked for.

Process Management

Process States

- 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
- Sleeping - we are waiting for a resource.
    - in a sleep queue
- Zombie - we've exited, but parent hasn't done a wait(2) on us yet. All that's left is the proc table entry.

Process States



SE 390: Series 300 HP-UX Internals

Process Management

Tunable Parameters

- argdevnblk - limits number of exec()s that can happen concurrently
  - each exec takes about 12K of swap space for arguments
- 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 time
  - this is used to size a static array, the proc table
- timeslice - length of timeslice for round-robin CPU scheduling
  - normally 5 clock ticks, which is 100ms
  - 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

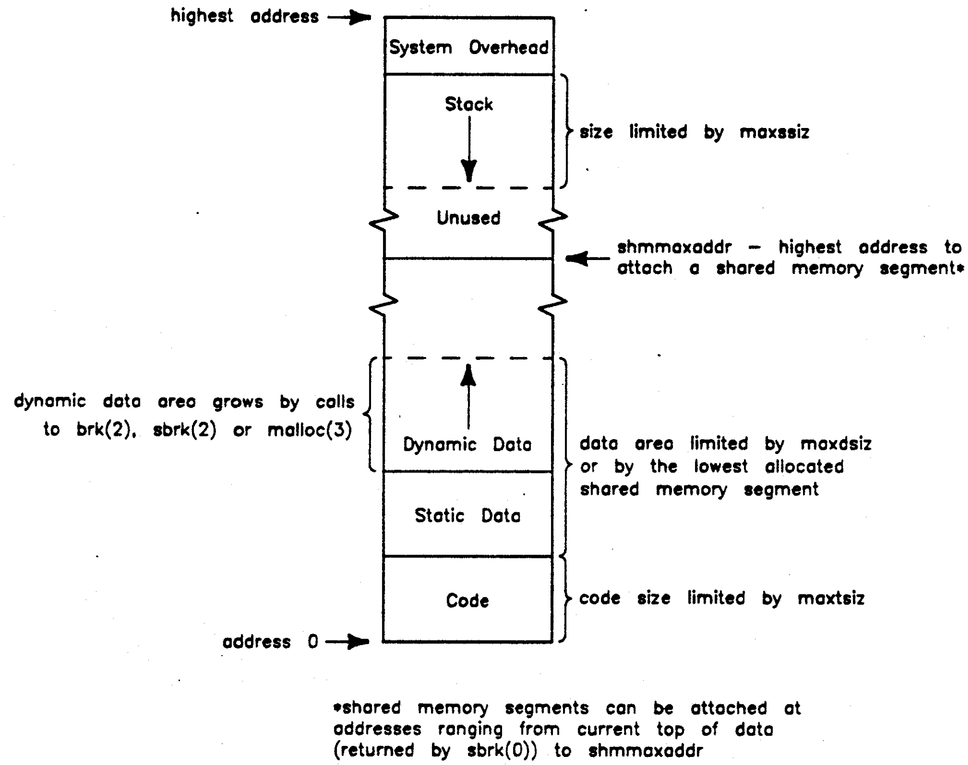


Figure 2-10. User Process Logical Address Space

Physical Memory Utilization

The maximum amount of physical memory you can install on your Series 300 computer is 7½ Megabytes for Models 310 and 320, 4 Mbytes for the Model 318, 16 Mbytes for the Model 319, 8 Mbytes for the Model 330, and 32 Mbytes for Model 350. The minimum amount of RAM for a non-networked single-user Series 300 HP-UX system is 2 Mbytes. The minimum amount of RAM for a Series 300 acting as the root server for an HP-UX cluster is 3 Mbytes. As more users are added on a multi-user system, more memory may be required for adequate performance. The computer's performance will also depend on the applications you run and on the peripheral devices attached to the system.

SE 390: Series 300 HP-UX Internals

Module Evaluation

PROCESS MANAGEMENT

On a scale of 1-10, 1 being bad, 5 being OK/don't care/irrelevant, 10 being good, please rate the following. If you have particular comments, please write them in. Thank you!

1. Clarity of presentation:

2. Depth/complexity (1 - material was too easy, 10 - it was too hard):

3. Usefulness/applicability/relevance of material presented:

4. Speed of presentation (1 - too slow, 10 - too fast):

5. How good was the material (slides, notes, etc)?

6. How good was the instructor?

Ways this could be improved (please be specific):

General Comments:

SE 390: Series 300 HP-UX Internals

Memory Management

The Big Picture

- How does HP-UX distribute and control physical and virtual memory?

The Little Picture(s)

- Physical <---> Virtual memory.
- Paging
- Swapping
- Important data structures.
- Tunable parameters.

SE 390: Series 300 HP-UX Internals

Memory Management

Virtual Memory

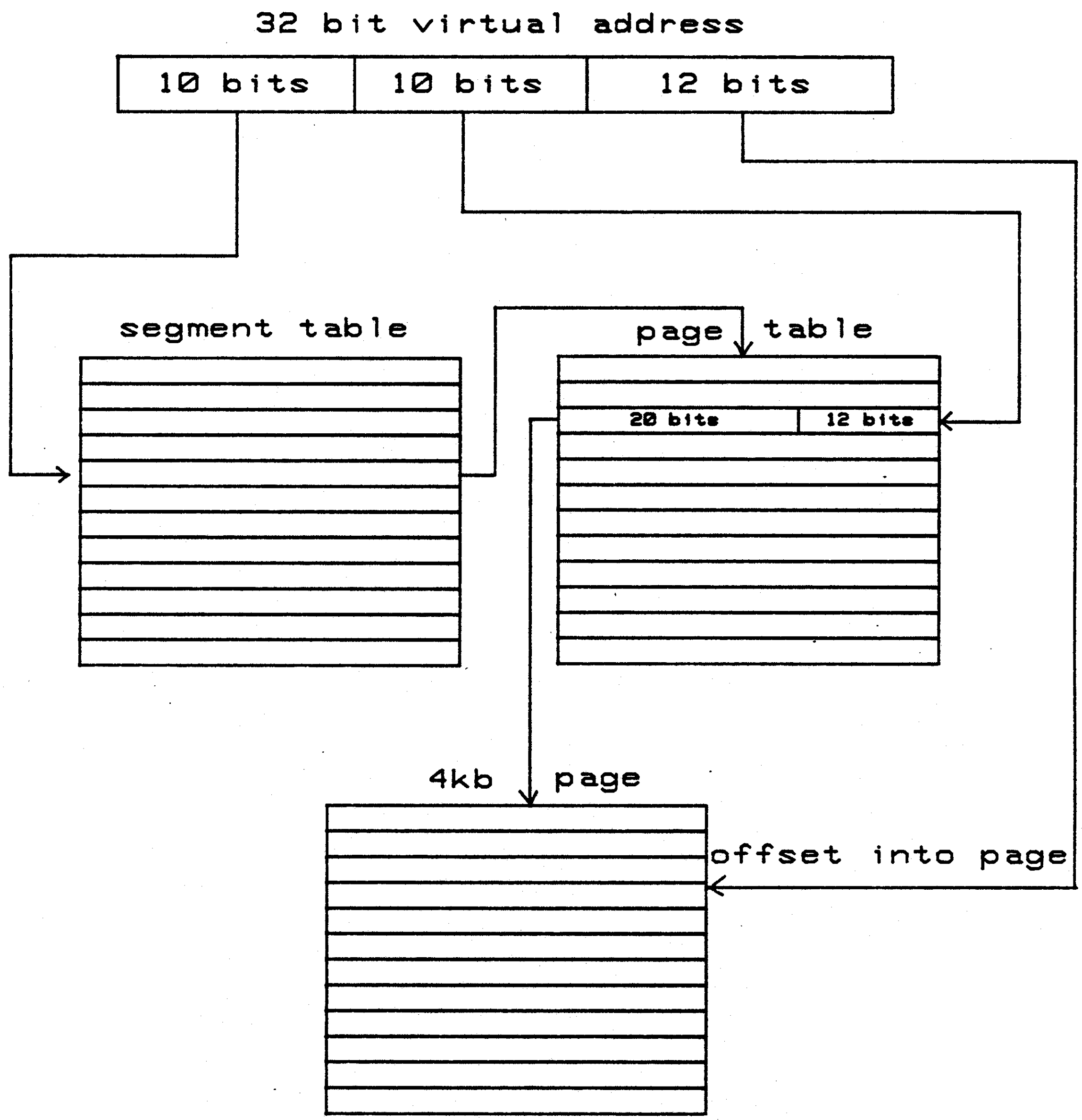
Why?

- allow all programs to think they are running by themselves
- allow for (fairly) efficient stretching of memory

How?

- Virtual address translation
    - 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.
- Pageout daemon kicks out pages if we're running short and they aren't being referenced often enough.
- A process always has enough swap space to hold whatever it is doing; it may or may not have enough physical pages for everything.

Virtual Address Translation



Memory Management

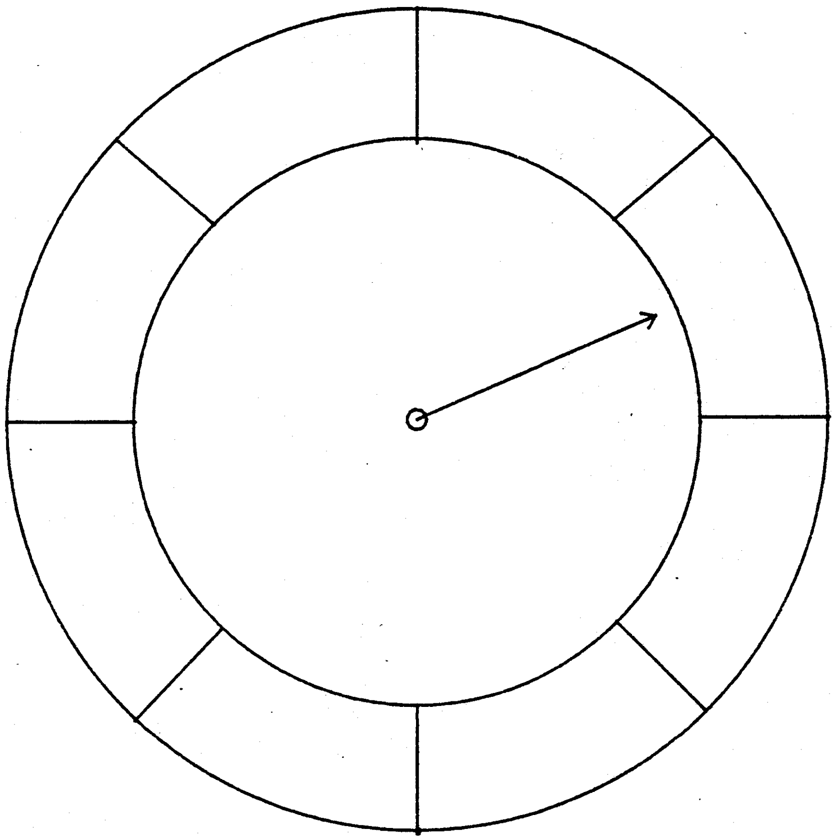
The Paging Game

- A (somewhat) graceful way of stretching the amount of available memory.
- Implemented with a clock algorithm:
  - "hand" goes around at a calculated rate, marking pages
  - if a marked page is referenced, a "soft" page fault occurs and the mark is erased
- Speed of hand 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.

Process 2: The Pageout Daemon

```
loop:
        free pages that have been written out

        sleep until somebody needs us and wakes us up


        while (we haven't scanned too many pages) and (free_mem < lotsfree) {
top:
                grab coremap entry for page

                if it's free, locked, or a system page
                        goto skip

                if reference bit is set
                        clear it
                        take page if process has too many
                else {
                        if process isn't using many pages
                                goto skip

                        if page is dirty {

                                if we're pushing pages too fast
                                        goto skip

                                if process is exiting or being swapped
                                        goto skip

                                free pages that have been written out

                                if we're out of swap headers
                                        goto top

                                lock page

                                adjust ptes, poip counts, etc.

                                "kluster" adjacent pages together

                                write page(s) out

                                goto skip
                        }

                        decrement count of pages process has in memory
                        free the page
                }

skip:           check to make sure wheels aren't spinning; if they are,
                wait until next clock tick
        }
        goto loop
```

Page Replacement



When To Do What

```
                      Available Memory

          +----------------------------------------+
          |                                        |
          |                                        |
          |                                        |
          |                                        |
          |                                        |
          |                                        |
          |                                        |
          |                                        |
          |                                        |
          |                                        |
          |                                        |
          |  min(256K, 25% of user memory)         |
lotsfree  +----------------------------------------+
          |  pageout daemon runs below here        |
          |                                        |
          |                                        |
          |                                        |
          |                                        |
          |  min(200K, 12.5% of user memory)       |
desfree   +----------------------------------------+
          |  swapper will run below here           |
          |                                        |
          |  min(64K, desfree/2)                   |
minfree   +----------------------------------------+
          |  swapper will force active processes   |
          |  out below here                        |
          |                                        |
          +----------------------------------------+
```

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.
- Space is allocated in chunks of at least dmmin, but <= dmmax.
- Only happens when we are really worried about the amount of memory available.

Process 0: The Swapper

```
loop:
        if (want kernelmap) or ((>= 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 so far
                                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

        if it's not critical to bring someone in
                wait awhile
                goto loop

 rdswap:
        walk through proc table {

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

                if process is currently being swapped out or has shm locked
                        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 or asleep)
                        see how it is
                        if it's one of the biggest we've seen
                                remember it
        }

        if we didn't find a long sleeper
                pick "oldest" job (based on nice value and time since swapin)

        if (found a sleeper) or (desperate and found *someone* to swap out) 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
        goto loop
```

SE 390: Series 300 HP-UX Internals

Memory Management

Important Data Structures

- Core map - used for paging. There's an entry in it for each page of non-kernel memory.

- Swap map - used for mapping the swap space. Allocated in chunks of dmmin <= size <= dmmax.

- Segment table - one for each process. Each table has 1024 entries, each of which points at a page table.

- Page table - 1024 entries, each of which points to a 4kb page.

Memory Management

Tunable Parameters

- dos_mem_byte - allocates memory for the DOS Coprocessor's use
- maxdsiz - maximum size of the data segment for an executing process
- maxssiz - maximum size of the stack segment for an executing process
- maxtsiz - maximum size of the text segment for an executing process
- minswapchunks - minimum amount of swap for a diskless node. It is always allocated to the node.
- maxswapchunks - maximum amount of swap space a node is allowed to allocate.
- unlockable_mem - amount of RAM that can not be locked
- dmmin - minimum size of chunk that can be allocated from swap area
- dmmax - maximum size of chunk that can be allocated from swap area
- dmtext - maximum amount of swap space that can be allocated for text (code) in a single request
- dmshm - maximum amount of swap space that can be allocated for System V shared memory usage in a single request

SE 390: Series 300 HP-UX Internals

Module Evaluation

MEMORY MANAGEMENT

On a scale of 1-10, 1 being bad, 5 being OK/don't care/irrelevant, 10 being good, please rate the following. If you have particular comments, please write them in. Thank you!

1. Clarity of presentation:

2. Depth/complexity (1 - material was too easy, 10 - it was too hard):

3. Usefulness/applicability/relevance of material presented:

4. Speed of presentation (1 - too slow, 10 - too fast):

5. How good was the material (slides, notes, etc)?

6. How good was the instructor?

Ways this could be improved (please be specific):

General Comments:

SE 390: Series 300 HP-UX Internals

System Shutdown

The Big Picture

- How do we (gracefully) go from a running system to a halted one?

The Little Pictures

- What is the correspondence between things being accomplished and things being printed on the console's screen?
- Updating the file system.
- Halting, gracefully or ungracefully.
- Interpreting panic dumps.

System Shutdown

Internal Actions vs. External Signs

- "Shutdown at <time>"

  mask signals (SIGINT, SIGQUIT, SIGHUP)

- "System going down ..."
  "System shutdown time has arrived."

  idle init process by sending it a signal
  update file system

- "Syncing disks..."

  close file systems

- "done"

  mask interrupts
  [dump stack and uts info]
  [symbolic traceback if >= 6.0]

- "halted"

System Shutdown

Updating the File System

- Write back modified superblock and cylinder group summary information.
- Write out inodes.
- Write out delayed-write blocks in the buffer cache.

System Shutdown

Halting the System

- Unmount all file systems after updating them.
- Print kernel stack to console if we're panicking.
- If we're panicking and running 6.0 or later, do symbolic traceback.
- If we're rebooting, copy boot code and jump to it.
- Loop on a "stop" instruction, waiting for something to happen.

Interpreting Panic Dumps

- First column consists of stack addresses.
- Numbers in the other columns that are in the first one or sandwiched by numbers in the first one are probably frame pointers.
- Find first appropriate address (frame pointer).
- 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.

System Shutdown

(Hopefully un)Common Kinds of Panics

- Parity error - sometimes this can be helped (concealed :-)) by changing the kernel parameter parity_option.

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

- {file,callout,text,...}: table is full - some kernel table is full. These can often be fixed by adjusting a kernel parameter.

- 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.

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

07ca?i
_biowait+0x22:          addq.w  &0x8,%a7

107af?10i
_biowait+0x7:           bgt.w   _bmap+0x523
                        eor.b   %d4,%d0
                        ori.b   &0xFFFEC2D,%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

0062?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

_6cc8?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

 e4?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)

 odc?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()
```

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

Module Evaluation

SYSTEM SHUTDOWN

On a scale of 1-10, 1 being bad, 5 being OK/don't care/irrelevant, 10 being good, please rate the following. If you have particular comments, please write them in. Thank you!

1. Clarity of presentation:

2. Depth/complexity (1 - material was too easy, 10 - it was too hard):

3. Usefulness/applicability/relevance of material presented:

4. Speed of presentation (1 - too slow, 10 - too fast):

5. How good was the material (slides, notes, etc)?

6. How good was the instructor?

Ways this could be improved (please be specific):

General Comments:

SE 390: Series 300 HP-UX Internals

Obscure Commands

he Big Picture

- What are some of the obscure but important commands and how do they work?

The Little Pictures

- Init(1m).
- Config(1m).
- Cluster(1m).
- Mkfs(1m).
- Getty(1m).
- Login(1).

Obscure Commands

init(1m)

- PID 1; started by kernel after we're up and running.
- Parent of all user processes.
- Reads table to find out what it is supposed to do.
- Is a state machine - when it is in state n, it decides what needs to be happening by looking in /etc/inittab for entries with n listed as the state. It starts up those commands, and if they are marked "respawn" it keeps starting them up whenever they die.
- When you launch a 2nd init(1m) or do a telinit(1m), it looks to see what PID it is; if it is not 1, it will take the parameter from the command line and send that as a signal to PID 1. Thus,

  ```
  $ telinit 3
  ```

  sends signal #3 to PID 1, which is the real init(1m).
- Init(1m) acts as a cleaner-upper - it arranges to catch SIGCHLD whenever it fork/execs a new process. It also inherits children when parents die without wait(2)ing for their children.

Obscure Commands

config(1m)

```
deal with arguments

open files (default: conf.c, /etc/master, config.mk, mkdev)

read master file to get
        1) list of devices, handlers, major & minor numbers, etc
        2) aliases for the above (such as 7914 --> cs80)
        3) tunable parameters and default values

while there's another line in the dfile {

        case line of

                "root" : record major and minor number

                "swap" : add entry to swdevt[]

                "swapsize" : set sizes for all swap entries so far

                tunable parameter : record value

                otherwise :
                        look it up in master list
                        be sure user specified minor# if not a card
                        be sure it's a legal address
                        add to list of devices on the system
}

set defaults for tunable parameters that weren't set

be sure all required devices are in list

write out the config file

write out the makefile
```

```
* HPUX_ID:  @(#)dfile.full.lan  15.1            86/12/09
* This is the configuration file for a full system, with LAN
* drivers
cs80
amigo
tape
printer
stape
srm
ptymas
ptyslv
ieee802
ethernet
hpib
gpio
ciper
rje
* cards
98624
98625
98626
98628
98642
* Tunable parameters
num_lan_cards 2
```

```
* HPUX_ID: @(#)master   49.10     87/11/18
*
* The following devices are those that can be specified in the system
* description file.  The name specified must agree with the name shown,
* or with an alias.
*
* name              handle            type    mask    block    char
*
cs80                cs80              3       3FB     0        4
scsi                scsi              3       3FB     7        47
flex                mf                3       1FA     1        6
amigo               amigo             3       3FB     2        11
rdu                 rdu               B       3C      6        45
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
lla                 lla               9       1FD     -1       18
lan01               lla               9       0FD     -1       19
hpib                hpib              1       FB      -1       21
gpio                hpib              1       1FB      -1      22
ciper               ciper             1       DA      -1       26
nsdiag0             nsdiag0           9       EA      -1       46
snalink             snalink           1       1C0     -1       36
dos                 dos               1       F9      -1       27
vme                 vme               1       1F8     -1       32
vme2                stealth           1       1C8     -1       44
dskless             dskless           18      100     -1       -1
rfa                 rfai              10      100     -1       -1
nfs                 nfsc              10      100     -1       -1
ramdisc             ram               3       FB      4        20
*
* The following cards are those that can be specified in the system
* description file.  The name specified must agree with the name shown,
* or with an alias.
*
* name   handle  type   mask   block   char
*
98624    ti9914  10     100    -1      -1
98625    simon   10     100    -1      -1
98626    sio626  10     100    -1      -1
98628    sio628  10     100    -1      -1
98642    sio642  10     100    -1      -1
*
* The following devices must not be specified in the system description
* file.  They are here to supply information to the config program.
*
* name              handle            type    mask    block    char
swapdev             swap              E       0       3        -1
swapdev1            swap1             E       0       5        -1
console             cons              D       FD      -1       0
ttyXX               tty               D       FD      -1       1
tty                 sy                D       FD      -1       2
mem                 mm                D       32      -1       3
swap                swap              D       30      -1       8
iomap               iomap             D       F9      -1       10
graphics            graphics          5       1F9     -1       12
r8042               r8042             D       C8      -1       23
hil                 hil               D       EC      -1       24
nimitz              nimitz            D       E4      -1       25
ite                 ite200            1C      100     -1       -1
$$$
```

```
*
* The following entries form the alias table.
* field 1: product #    field 2: driver name
*
[bunch of stuff deleted here and following]
7935     cs80
ct       cs80
7906     amigo
7925     amigo
9133V    amigo
9895     amigo
int      flex
fd       flex
7971     tape
mt       tape
7974     stape
7978     stape
lp       printer
2225     printer
2227     printer
2934     printer
*
* Several printers listed below can also be
* supported on hpib and RS-232
*
2563     ciper
98629    srm
98641    rje
98643    lla
*
* Plotters can also be supported on RS-232
*
plt      hpib
7550     hpib
inthpib  98624
ti9914   98624
simon    98625
98644    98626
sio626   98626
sio628   98628
sio642   98642
mux      98642
98577    vme2
stealth  vme2
ieee802  lla
ethernet lan01
$$$
*
* The following entries form the tunable parameter table.
*
maxusers        MAXUSERS        8                        0
timezone        TIMEZONE        420                      0
dst             DST             1                        0
nproc           NPROC           (20+8*MAXUSERS+(NGCSP))          6
num_cnodes      NUM_CNODES      0                        0
dskless_node    DSKLESS_NODE    0                        0
server_node     SERVER_NODE     0                        0
ninode  NINODE  ((NPROC+16+MAXUSERS)+32+(2*NPTY)+SERVER_NODE*18*NUM_CNODES)
nfile           NFILE           (16*(NPROC+16+MAXUSERS)/10+32+(2*NPTY)) 14
argdevnblk      ARGDEVNBLK      0                        0
nbuf            NBUF            0                        0
dos_mem_byte    DOS_MEM_BYTE    0                        0
ncallout        NCALLOUT   (16+NPROC+USING_ARRAY_SIZE+SERVING_ARRAY_SIZE) 6
ntext           NTEXT           (40+MAXUSERS)            10
unlockable_mem  UNLOCKABLE_MEM  102400                   0
nflocks         NFLOCKS         200                      2
```

```
npty            NPTY            82                     1
maxuprc         MAXUPRC         25                     3
dmmin           DMMIN           16                     16
dmmax           DMMAX           512                    256
dmtext          DMTEXT          512                    256
dmshm           DMSHM           512                    256
maxdsiz         MAXDSIZ         0x01000000             0x00040000
maxssiz         MAXSSIZ         0x00200000             0x00040000
maxtsiz         MAXTSIZ         0x01000000             0x00040000
shmmaxaddr      SHMMAXADDR      0x01000000             0x00040000
parity_option   PARITY_OPTION   2                      0
timeslice       TIMESLICE       0                      -1
acctsuspend     ACCTSUSPEND     2                      -100
acctresume      ACCTRESUME      4                      -100
ndilbuffers     NDILBUFFERS     30                     1
filesizelimit   FILESIZELIMIT   0x1fffffff             0x00000010
dskless_mbufs DSKLESS_MBUFS (((SERVING_ARRAY_SIZE+(2*USING_ARRAY_SIZE))/32)+1
dskless_cbufs   DSKLESS_CBUFS   (DSKLESS_MBUFS*2)      6
using_array_size        USING_ARRAY_SIZE    (NPROC)  1
serving_array_size SERVING_ARRAY_SIZE (SERVER_NODE*NUM_CNODES*MAXUSERS+2*MAXU
dskless_fsbufs          DSKLESS_FSBUFS    (SERVING_ARRAY_SIZE)       0
selftest_period SELFTEST_PERIOD 120     0
*
* The next two parameters, check_alive_period and retry_alive_period, should
* never be changed by a customer.  Only a qualified Hewlett-Packard service
* engineer should change these parameters.  Diskless node crashes could occur
* if either of these parameters is changed improperly!
*
check_alive_period      CHECK_ALIVE_PERIOD      4       4
retry_alive_period      RETRY_ALIVE_PERIOD      21      21
maxswapchunks   MAXSWAPCHUNKS   512     1
minswapchunks   MINSWAPCHUNKS   4       1
num_lan_cards   NUM_LAN_CARDS   2                      0
netmemmax       NETMEMMAX       250000                 75000
netmemthresh    NETMEMTHRESH    100000                 -1
ngcsp           NGCSP           (8*NUM_CNODES)         0
scroll_lines    SCROLL_LINES    100                    100
*
* Messages, Semaphores, and Shared Memory Constants
mesg     MESG     1               0
msgmap   MSGMAP   (MSGTQL+2)      3
msgmax   MSGMAX   8192            0
msgmnb   MSGMNB   16384           0
msgmni   MSGMNI   50              1
msgssz   MSGSSZ   1               1
msgtql   MSGTQL   40              1
msgseg   MSGSEG   16384           1
sema     SEMA     1               0
semmap   SEMMAP   (SEMMNI+2)      4
semmni   SEMMNI   64              2
semmns   SEMMNS   128             2
semmnu   SEMMNU   30              1
semume   SEMUME   10              1
semvmx   SEMVMX   32767           1
semaem   SEMAEM   16384           0
shmem    SHMEM    1               0
shmmax   SHMMAX   0x00600000      0x00200000
shmmin   SHMMIN   1               1
shmmni   SHMMNI   30              1
shmseg   SHMSEG   10              1
shmbrk   SHMBRK   16              0
shmall   SHMALL   2048            2048
fpa      FPA      1               0
```

SE 390: Series 300 HP-UX Internals

Module Evaluation

COMMANDS

On a scale of 1-10, 1 being bad, 5 being OK/don't care/irrelevant, 10 being good, please rate the following. If you have particular comments, please write them in. Thank you!

1. Clarity of presentation:

2. Depth/complexity (1 - material was too easy, 10 - it was too hard):

3. Usefulness/applicability/relevance of material presented:

4. Speed of presentation (1 - too slow, 10 - too fast):

5. How good was the material (slides, notes, etc)?

6. How good was the instructor?

Ways this could be improved (please be specific):

General Comments:

SE 390: Series 300 HP-UX Internals

File System

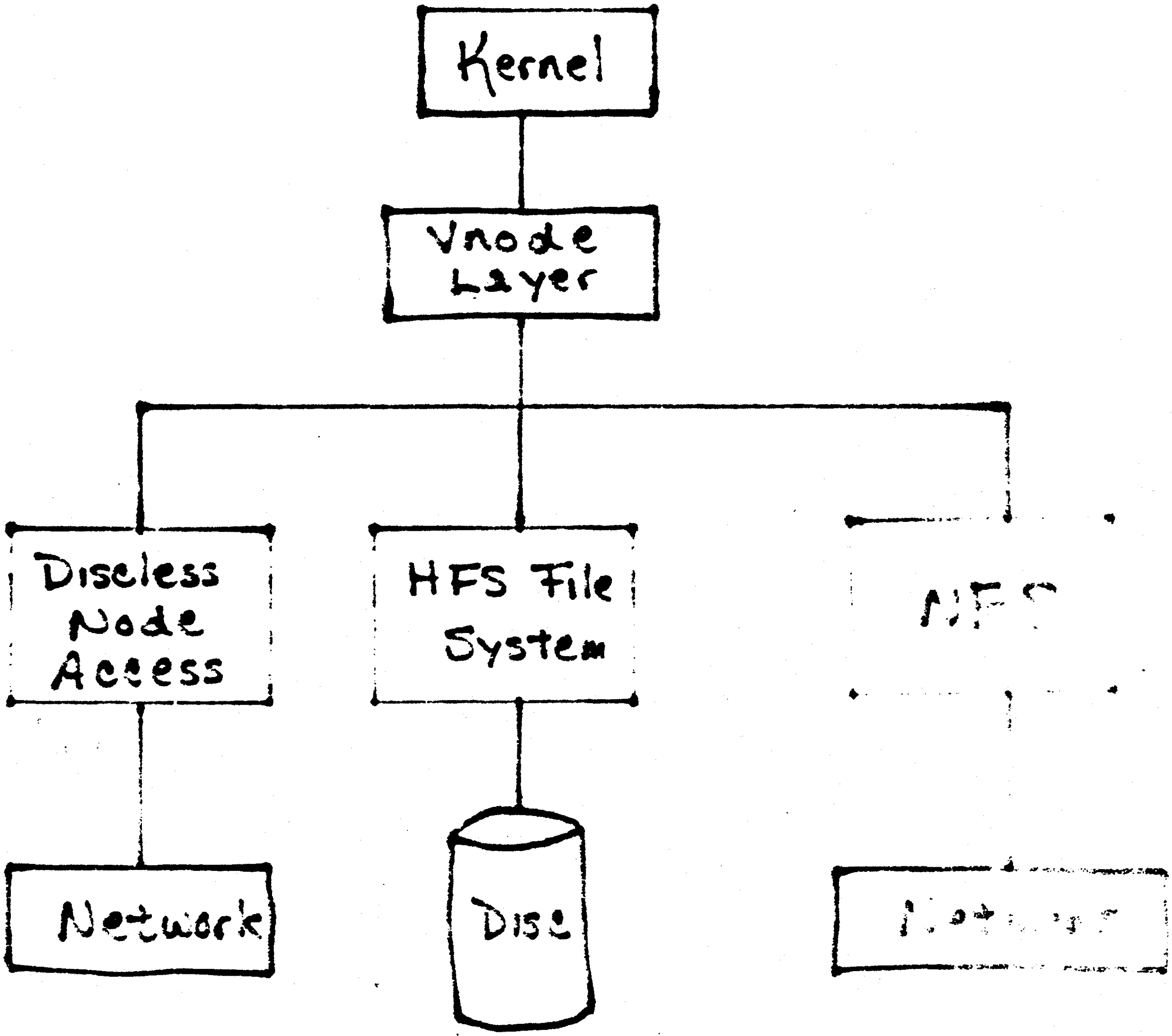
The Big Picture

How does HP-UX organize disks and access files?

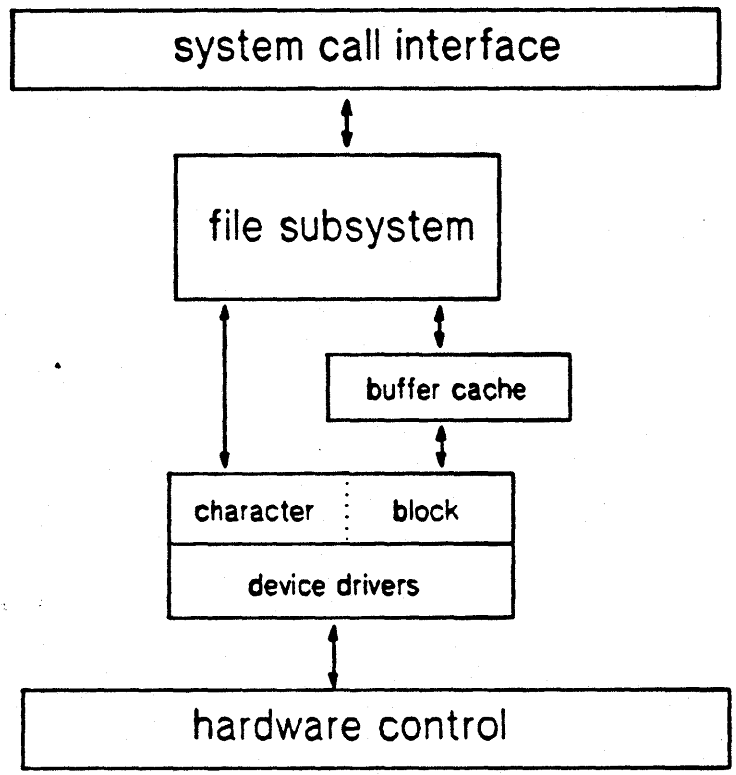
The Little Pictures

- The Vnode layer.
- Caching: buffers, inodes, and directory names.
- The HFS/Berkeley/McKusick filesystem.
    - History and layout.
    - Allocation policies.
    - Locking.
    - Recovering from messes.
- Examples:
    - open(2)
    - write(2)

File System


Kernel
Vnode Layer
Discless Node Access
HFS File System
Network
Disc

The Big Picture

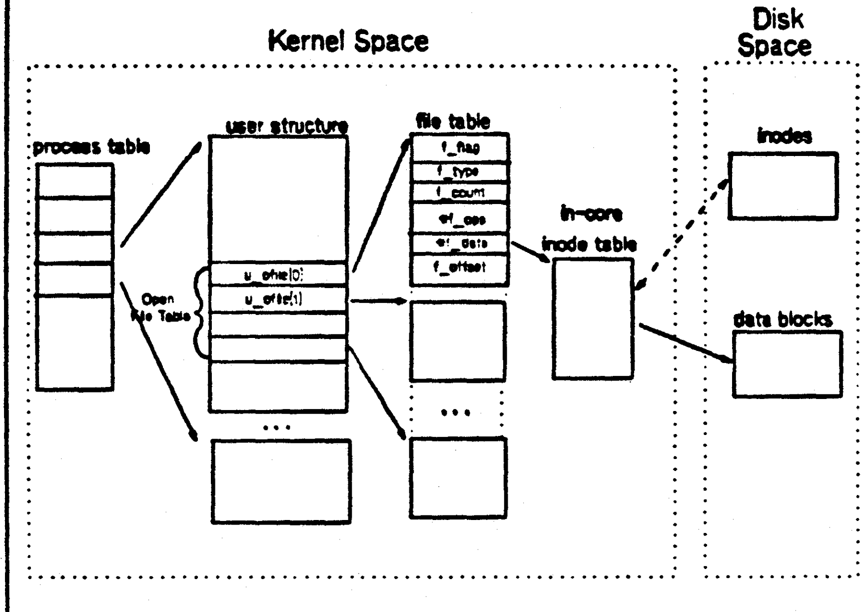


ufs10002

☐ HP-UX File System Overview

☐ Notes

Kernel Structure Overview



The Vnode Layer

- Why?

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

    - 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.

    - The function namei() has been replaced by lookupname(). It returns a pointer to a locked vnode. This function is called whenever the system needs to translate a pathname like "/usr/mail/fred" to something that will let it get at the stuff in the file. In the case of 5.5/namei(), this was a pointer to an inode. Now it is a vnode pointer.

File System

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.

- 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.

- Directory name cache - used to keep us from having to always translate pathnames. If we just accessed a particular path, we'll keep the name around since there is a fair chance we'll want it again.

SE 390: Series 300 HP-UX Internals

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
  * no alignment constraints on data transfers
  * easy to implement
- Disadvantages:
  * limited file I/O throughput
  * lack of locality on disk
  * lack of robustness
  * designed for "small" systems/disks

File System

Picture of a Bell file system

```
|        |        |                 |                     |
| Boot   | Super  | I-nodes         | Data                |
| Block  | Block  |                 | Blocks              |
|        |        |                 |                     |

  (BB)     (SB)     (I-n)             (DB
```

File System

The Berkeley/McKusick file system (aka "HFS")

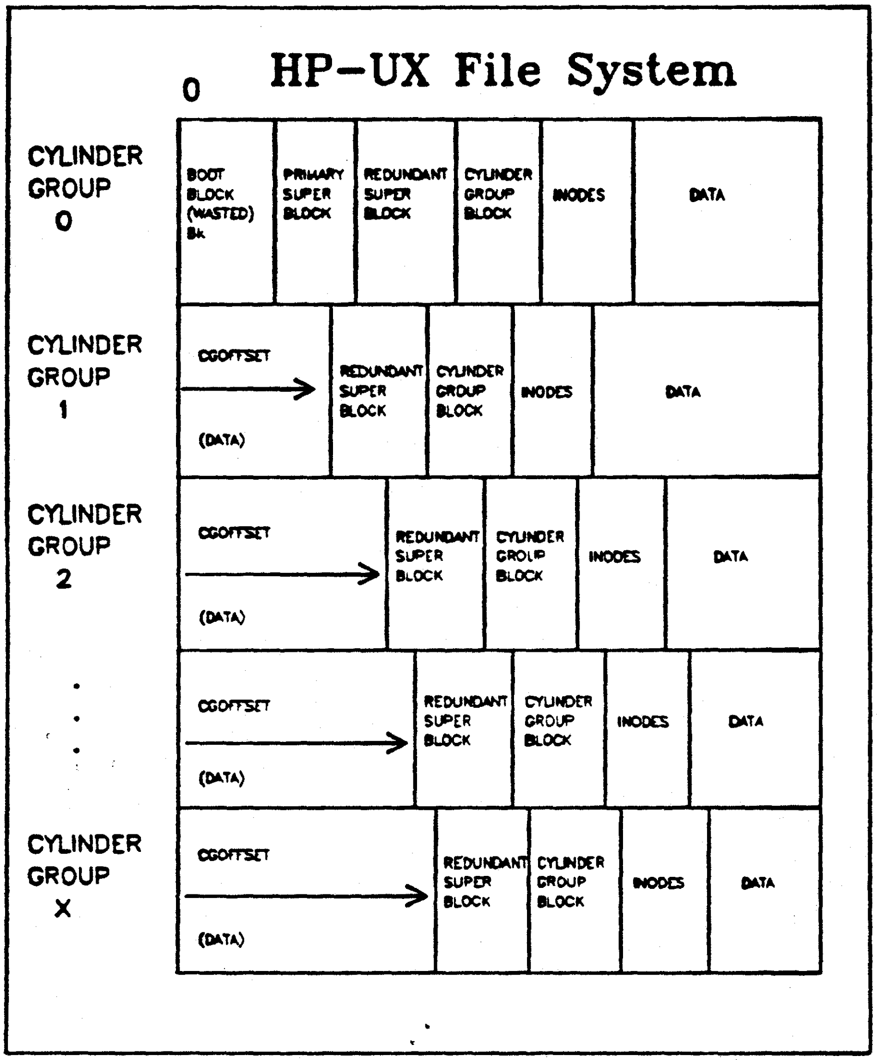
- retains advantages of the original Bell design
- includes remedies for most problem areas
  * throughput: larger block size (4/8 Kbytes)
  * locality: introduction of "cylinder groups" (each resembles a Bell file system)
  * robustness: superblock is replicated in each group

    staged modifications to file system
  * extensible: can access files of 4+ Gbytes (theoretical maximum ~ 4 Tbytes)

    parameterizes disk features
- HP-UX extensions:
  * fs_clean flag
- what s300 HP-UX does not include (as of now)
  * partitions (aka "disk sections")
  * long file names (coming soon to a filesystem near you :-))
  * disk quotas

HP-UX File System Overview

Notes



ufs10005

SE 390: Series 300 HP-UX Internals

File System

Picture of a Berkeley file system

cylinder group 0:

```
|       |       |       |        |              |                        |
|  BB   |  SB   |  SB   |  CGB   |  I-n         |  DB                    |
|       |       |       |        |              |                        |
```

cylinder group 1:

```
|                   |       |        |              |                    |
|  DB               |  SB   |  CGB   |  I-n         |  DB                |
|                   |       |        |              |                    |
```

cylinder group 2:

```
|                      |       |        |              |                 |
|  DB                  |  SB   |  CGB   |  I-n         |  DB             |
|                      |       |        |              |                 |
```

etc.

File Locking

- Byte oriented - process can lock any part of a file.
- For enforcement mode locking to work, the setgid bit MUST be on and the group execute bit MUST be off.
- Enforcement locking is provided in 6.0 - this is possible with a stateful system like our Diskless system, but difficult/impossible with a stateless system like NFS.
- Does not work at all for device files. It would not be good to lock /dev/dsk/* :-)
- Implemented with a locklist that is kept in each inode. The list has an entry for each lock, and is sorted by PID and starting offset in the file. When a user wants to lock a chunk of a file, the system will walk through the list and make sure that no other process has a lock on a section that overlaps or includes the one being requested.
- There is code to check for deadlock. Suppose that process A is waiting on something that process B has locked, and process B is doing the same with process C. We do not want to let C wait on A!

Recovering From Messes

- fsck(1m) - this will fix most problems, but not all.

- fsdb(1m) - this is capable of doing most anything in the hands of a skilled operator, but they are rare :-)

- disked(1m) - roughly equivalent to fsdb(1m) in power, but has a MUCH nicer user interface. Unfortunately, it's not supported.

File System

An Example: `fd = open("/usr/mail/fred", O_RDONLY);`

- put "open" code on stack and trap to get into the kernel's syscall()
- allocate slots in system and user open file tables
    - if there's not a user entry, user has exceeded limit of 60
    - if there's not a system entry, will get a "tablefull" message on the console
- look up the name and get a vnode pointer - this will involve interaction with the remote server, local HFS filesystem, etc to do the actual looking through directories
- check file permissions and accessability of filesystem
    - if filesystem is mounted readonly, we can't let user write

File System

Example 2: n = write(fd, buf, buflen);

- Put "write" code on stack and trap to get to kernel's syscall().
- Generic write() will package up the parameters, do checking, etc with the help of some other routines.
- The file structure "knows" what functions should be called to do particular things to it, so the system jumps to the appropriate one.
- Lock the inode.
- Call the driver (if it's a character device) or go through the buffer cache or whatever is appropriate.
- Since we are writing, the system must check to see if there is enough space in the block(s) that is/are presently allocated to the file; if not, allocate more according to the rules mentioned previously.
- Update and unlock the inode.

```
/* @(#) $Revision: 56.1 $ */
/*
 * Each disk drive contains some number of file systems.
 * A file system consists of a number of cylinder groups.
 * Each cylinder group has inodes and data.
 *
 * A file system is described by its super-block, which in turn
 * describes the cylinder groups.  The super-block is critical
 * data and is replicated in each cylinder group to protect against
 * catastrophic loss.  This is done at mkfs time and the critical
 * super-block data does not change, so the copies need not be
 * referenced further unless disaster strikes.
 *
 * For file system fs, the offsets of the various blocks of interest
 * are given in the super block as:
 *      [fs->fs_sblkno]         Super-block
 *      [fs->fs_cblkno]         Cylinder group block
 *      [fs->fs_iblkno]         Inode blocks
 *      [fs->fs_dblkno]         Data blocks
 * The beginning of cylinder group cg in fs, is given by
 * the ''cgbase(fs, cg)'' macro.
 *
 * The first boot and super blocks are given in absolute disk addresses.
 */
#define BBSIZE          8192
#define SBSIZE          8192
#define BBLOCK          ((daddr_t)(0))
#define SBLOCK          ((daddr_t)(BBLOCK + BBSIZE / DEV_BSIZE))

/*
 * Addresses stored in inodes are capable of addressing fragments
 * of 'blocks'. File system blocks of at most size MAXBSIZE can
 * be optionally broken into 2, 4, or 8 pieces, each of which is
 * addressible; these pieces may be DEV_BSIZE, or some multiple of
 * a DEV_BSIZE unit.
 *
 * Large files consist of exclusively large data blocks.  To avoid
 * undue wasted disk space, the last data block of a small file may be
 * allocated as only as many fragments of a large block as are
 * necessary.  The file system format retains only a single pointer
 * to such a fragment, which is a piece of a single large block that
 * has been divided.  The size of such a fragment is determinable from
 * information in the inode, using the ''blksize(fs, ip, lbn)'' macro.
 *
 * The file system records space availability at the fragment level;
 * to determine block availability, aligned fragments are examined.
 *
 */

/*
 * Cylinder group related limits.
 *
 * For each cylinder we keep track of the availability of blocks at different
 * rotational positions, so that we can lay out the data to be picked
 * up with minimum rotational latency.  NRPOS is the number of rotational
 * positions which we distinguish.  With NRPOS 8 the resolution of our
 * summary information is 2ms for a typical 3600 rpm drive.
 */
#define NRPOS           8       /* number distinct rotational positions */

/*
 * MAXIPG bounds the number of inodes per cylinder group, and
 * is needed only to keep the structure simpler by having the
 * only a single variable size element (the free bit map).
 *
 * N.B.: MAXIPG must be a multiple of INOPB(fs).
```

```
 */
#define MAXIPG          2048    /* max number inodes/cyl group */

/*
 * MINBSIZE is the smallest allowable block size.
 * In order to insure that it is possible to create files of size
 * 2^32 with only two levels of indirection, MINBSIZE is set to 4096.
 * MINBSIZE must be big enough to hold a cylinder group block,
 * thus changes to (struct cg) must keep its size within MINBSIZE.
 * MAXCPG is limited only to dimension an array in (struct cg);
 * it can be made larger as long as that structures size remains
 * within the bounds dictated by MINBSIZE.
 * Note that super blocks are always of size MAXBSIZE,
 * and that MAXBSIZE must be >= MINBSIZE.
 */
#define MINBSIZE        4096
#define MAXCPG          32      /* maximum fs_cpg */

/* MAXFRAG is the maximum number of fragments per block */
#define MAXFRAG         8

#ifndef NBBY
#define NBBY            8       /* number of bits in a byte     */
                                /* NOTE: this is also defined   */
                                /* in param.h.  So if NBBY gets */
                                /* changed, change it in        */
                                /* param.h also                 */
#endif

/*
 * The path name on which the file system is mounted is maintained
 * in fs_fsmnt. MAXMNTLEN defines the amount of space allocated in
 * the super block for this name.
 * The limit on the amount of summary information per file system
 * is defined by MAXCSBUFS. It is currently parameterized for a
 * maximum of two million cylinders.
 */
#define MAXMNTLEN 512
#define MAXCSBUFS 32

/*
 * Per cylinder group information; summarized in blocks allocated
 * from first cylinder group data blocks.  These blocks have to be
 * read in from fs_csaddr (size fs_cssize) in addition to the
 * super block.
 *
 * N.B. sizeof(struct csum) must be a power of two in order for
 * the ''fs_cs'' macro to work (see below).
 */
struct csum {
        long    cs_ndir;        /* number of directories */
        long    cs_nbfree;      /* number of free blocks */
        long    cs_nifree;      /* number of free inodes */
        long    cs_nffree;      /* number of free frags */
};

/*
 * Super block for a file system.
 */
#define FS_MAGIC        0x011954

#define FS_CLEAN        0x17
#define FS_OK           0x53
#define FS_NOTOK        0x31

struct  fs
```

```
{
        struct  fs *fs_link;            /* linked list of file systems */
        struct  fs *fs_rlink;           /*     used for incore super blocks */
        daddr_t fs_sblkno;              /* addr of super-block in filesys */
        daddr_t fs_cblkno;              /* offset of cyl-block in filesys */
        daddr_t fs_iblkno;              /* offset of inode-blocks in filesys */
        daddr_t fs_dblkno;              /* offset of first data after cg */
        long    fs_cgoffset;            /* cylinder group offset in cylinder */
        long    fs_cgmask;              /* used to calc mod fs_ntrak */
        time_t  fs_time;                /* last time written */
        long    fs_size;                /* number of blocks in fs */
        long    fs_dsize;               /* number of data blocks in fs */
        long    fs_ncg;                 /* number of cylinder groups */
        long    fs_bsize;               /* size of basic blocks in fs */
        long    fs_fsize;               /* size of frag blocks in fs */
        long    fs_frag;                /* number of frags in a block in fs */
/* these are configuration parameters */
        long    fs_minfree;             /* minimum percentage of free blocks */
        long    fs_rotdelay;            /* num of ms for optimal next block */
        long    fs_rps;                 /* disk revolutions per second */
/* these fields can be computed from the others */
        long    fs_bmask;               /* ''blkoff'' calc of blk offsets */
        long    fs_fmask;               /* ''fragoff'' calc of frag offsets */
        long    fs_bshift;              /* ''lblkno'' calc of logical blkno */
        long    fs_fshift;              /* ''numfrags'' calc number of frags */
/* these are configuration parameters */
        long    fs_maxcontig;           /* max number of contiguous blks */
        long    fs_maxbpg;              /* max number of blks per cyl group */
/* these fields can be computed from the others */
        long    fs_fragshift;           /* block to frag shift */
        long    fs_fsbtodb;             /* fsbtodb and dbtofsb shift constant */
        long    fs_sbsize;              /* actual size of super block */
        long    fs_csmask;              /* csum block offset */
        long    fs_csshift;             /* csum block number */
        long    fs_nindir;              /* value of NINDIR */
        long    fs_inopb;               /* value of INOPB */
        long    fs_nspf;                /* value of NSPF */
        long    fs_id[2];               /* file system id */
        long    fs_sparecon[4];         /* reserved for future constants */
/* sizes determined by number of cylinder groups and their sizes */
        daddr_t fs_csaddr;              /* blk addr of cyl grp summary area */
        long    fs_cssize;              /* size of cyl grp summary area */
        long    fs_cgsize;              /* cylinder group size */
/* these fields should be derived from the hardware */
        long    fs_ntrak;               /* tracks per cylinder */
        long    fs_nsect;               /* sectors per track */
        long    fs_spc;                 /* sectors per cylinder */
/* this comes from the disk driver partitioning */
        long    fs_ncyl;                /* cylinders in file system */
/* these fields can be computed from the others */
        long    fs_cpg;                 /* cylinders per group */
        long    fs_ipg;                 /* inodes per group */
        long    fs_fpg;                 /* blocks per group * fs_frag */
/* this data must be re-computed after crashes */
        struct  csum fs_cstotal;        /* cylinder summary information */
/* these fields are cleared at mount time */
        char    fs_fmod;                /* super block modified flag */
        char    fs_clean;               /* file system is clean flag */
        char    fs_ronly;               /* mounted read-only flag */
        char    fs_flags;               /* currently unused flag */
        char    fs_fsmnt[MAXMNTLEN];    /* name mounted on */
/* these fields retain the current block allocation info */
        long    fs_cgrotor;             /* last cg searched */
        struct  csum *fs_csp[MAXCSBUFS];/* list of fs_cs info buffers */
        long    fs_cpc;                 /* cyl per cycle in postbl */
        short   fs_postbl[MAXCPG][NRPOS];/* head of blocks for each rotation */
```

```
        long    fs_magic;                       /* magic number */
        char    fs_fname[6];                    /* file system name */
        char    fs_fpack[6];                    /* file system pack name */
        u_char  fs_rotbl[1];                    /* list of blocks for each rotation */
/* actually longer */
};

/*
 * Convert cylinder group to base address of its global summary info.
 *
 * N.B. This macro assumes that sizeof(struct csum) is a power of two.
 */
#define fs_cs(fs, indx) \
        fs_csp[(indx) >> (fs)->fs_csshift][(indx) & ~(fs)->fs_csmask]

/*
 * MAXBPC bounds the size of the rotational layout tables and
 * is limited by the fact that the super block is of size SBSIZE.
 * The size of these tables is INVERSELY proportional to the block
 * size of the file system. It is aggravated by sector sizes that
 * are not powers of two, as this increases the number of cylinders
 * included before the rotational pattern repeats (fs_cpc).
 * Its size is derived from the number of bytes remaining in (struct fs)
 */
#define MAXBPC  (SBSIZE - sizeof (struct fs))

/*
 * Cylinder group block for a file system.
 */
#define CG_MAGIC        0x090255
struct  cg {
        struct  cg *cg_link;                    /* linked list of cyl groups */
        struct  cg *cg_rlink;                   /*     used for incore cyl groups */
        time_t  cg_time;                        /* time last written */
        long    cg_cgx;                         /* we are the cgx'th cylinder group */
        short   cg_ncyl;                        /* number of cyl's this cg */
        short   cg_niblk;                       /* number of inode blocks this cg */
        long    cg_ndblk;                       /* number of data blocks this cg */
        struct  csum cg_cs;                     /* cylinder summary information */
        long    cg_rotor;                       /* position of last used block */
        long    cg_frotor;                      /* position of last used frag */
        long    cg_irotor;                      /* position of last used inode */
        long    cg_frsum[MAXFRAG];              /* counts of available frags */
        long    cg_btot[MAXCPG];                /* block totals per cylinder */
        short   cg_b[MAXCPG][NRPOS];            /* positions of free blocks */
        char    cg_iused[MAXIPG/NBBY];          /* used inode map */
        long    cg_magic;                       /* magic number */
        u_char  cg_free[1];                     /* free block map */
/* actually longer */
};

/*
 * MAXBPG bounds the number of blocks of data per cylinder group,
 * and is limited by the fact that cylinder groups are at most one block.
 * Its size is derived from the size of blocks and the (struct cg) size,
 * by the number of remaining bits.
 */
#define MAXBPG(fs) \
        (fragstoblks((fs), (NBBY * ((fs)->fs_bsize - (sizeof (struct cg))))))

/*
 * Turn file system block numbers into disk block addresses.
 * This maps file system blocks to device size blocks.
 */
#define fsbtodb(fs, b)  ((b) << (fs)->fs_fsbtodb)
#define dbtofsb(fs, b)  ((b) >> (fs)->fs_fsbtodb)
```

```
/*
 * Cylinder group macros to locate things in cylinder groups.
 * They calc file system addresses of cylinder group data structures.
 */
#define cgbase(fs, c)   ((daddr_t)((fs)->fs_fpg * (c)))
#define cgstart(fs, c) \
        (cgbase(fs, c) + (fs)->fs_cgoffset * ((c) & ~((fs)->fs_cgmask)))
#define cgsblock(fs, c) (cgstart(fs, c) + (fs)->fs_sblkno)      /* super blk */
#define cgtod(fs, c)    (cgstart(fs, c) + (fs)->fs_cblkno)      /* cg block */
#define cgimin(fs, c)   (cgstart(fs, c) + (fs)->fs_iblkno)      /* inode blk */
#define cgdmin(fs, c)   (cgstart(fs, c) + (fs)->fs_dblkno)      /* 1st data */

/*
 * Give cylinder group number for a file system block.
 * Give cylinder group block number for a file system block.
 */
#define dtog(fs, d)     ((d) / (fs)->fs_fpg)
#define dtogd(fs, d)    ((d) % (fs)->fs_fpg)

/*
 * Extract the bits for a block from a map.
 * Compute the cylinder and rotational position of a cyl block addr.
 */
#define blkmap(fs, map, loc) \
    (((map)[loc / NBBY] >> (loc % NBBY)) & (0xff >> (NBBY - (fs)->fs_frag)))
#define cbtocylno(fs, bno) \
        ((bno) * NSPF(fs) / (fs)->fs_spc)
#define cbtorpos(fs, bno) \
        ((bno) * NSPF(fs) % (fs)->fs_nsect * NRPOS / (fs)->fs_nsect)

/*
 * The following macros optimize certain frequently calculated
 * quantities by using shifts and masks in place of divisions
 * modulos and multiplications.
 */
#define blkoff(fs, loc)         /* calculates (loc % fs->fs_bsize) */ \
        ((loc) & ~(fs)->fs_bmask)
#define fragoff(fs, loc)        /* calculates (loc % fs->fs_fsize) */ \
        ((loc) & ~(fs)->fs_fmask)
#define lblkno(fs, loc)         /* calculates (loc / fs->fs_bsize) */ \
        ((loc) >> (fs)->fs_bshift)
#define numfrags(fs, loc)       /* calculates (loc / fs->fs_fsize) */ \
        ((loc) >> (fs)->fs_fshift)
#define blkroundup(fs, size)    /* calculates roundup(size, fs->fs_bsize) */ \
        (((size) + (fs)->fs_bsize - 1) & (fs)->fs_bmask)
#define fragroundup(fs, size)   /* calculates roundup(size, fs->fs_fsize) */ \
        (((size) + (fs)->fs_fsize - 1) & (fs)->fs_fmask)
#define fragstoblks(fs, frags)  /* calculates (frags / fs->fs_frag) */ \
        ((frags) >> (fs)->fs_fragshift)
#define blkstofrags(fs, blks)   /* calculates (blks * fs->fs_frag) */ \
        ((blks) << (fs)->fs_fragshift)
#define fragnum(fs, fsb)        /* calculates (fsb % fs->fs_frag) */ \
        ((fsb) & ((fs)->fs_frag - 1))
#define blknum(fs, fsb)         /* calculates rounddown(fsb, fs->fs_frag) */ \
        ((fsb) &~ ((fs)->fs_frag - 1))

/*
 * Determine the number of available frags given a
 * percentage to hold in reserve
 */
#define freespace(fs, percentreserved) \
        (blkstofrags((fs), (fs)->fs_cstotal.cs_nbfree) + \
        (fs)->fs_cstotal.cs_nffree - ((fs)->fs_dsize * (percentreserved) / 100))

/*
```

```
 * Determining the size of a file block in the file system.
 */
#define blksize(fs, ip, lbn) \
        (((lbn) >= NDADDR || (ip)->i_size >= ((lbn) + 1) << (fs)->fs_bshift) \
            ? (fs)->fs_bsize \
            : (fragroundup(fs, blkoff(fs, (ip)->i_size))))
#define dblksize(fs, dip, lbn) \
        (((lbn) >= NDADDR || (dip)->di_size >= ((lbn) + 1) << (fs)->fs_bshift) \
            ? (fs)->fs_bsize \
            : (fragroundup(fs, blkoff(fs, (dip)->di_size))))

/*
 * Number of disk sectors per block; assumes DEV_BSIZE byte sector size.
 */
#define NSPB(fs)        ((fs)->fs_nspf << (fs)->fs_fragshift)
#define NSPF(fs)        ((fs)->fs_nspf)
```

SWAPPING

SWAPPING

Before 6.0, the swap space was managed on the local system using the swap map.

Swapmap is an Array of Address, Size pairs where each entry defines an available Piece of swap area.

swapmap

Addr 1	Size 1
Addr 2	Size 2
Addr 3	Size 3

As swap space is required by a process, it is taken from the swapmap. The address, size data is updated.

SWAPPING

swap map entry

1	1000

Process A requests 100 bytes swap

Updated swapmap

101	900

The location of the swap space in use by process A is kept in the per process region table of the process's U area.

When the process terminates, the swap space will be returned to the swapmap. An attempt will be made to coalesce the released space with an existing swapmap entry.

SWAPPING

swap map entry

101	900

Process B requests 200 bytes swap

Updated swapmap

301	700

Process A terminates

Updated swapmap

1	100
301	700

It was not possible to coalesce the freed space. A new array entry was created. The swap area has become fragmented. If these areas still free when process B terminates, the space will be merged back into a single swapmap entry.

After 6.0 release, systems which have local swap devices build a chunk map at bootup. This includes standalone systems, root servers and diskless nodes with local swap.

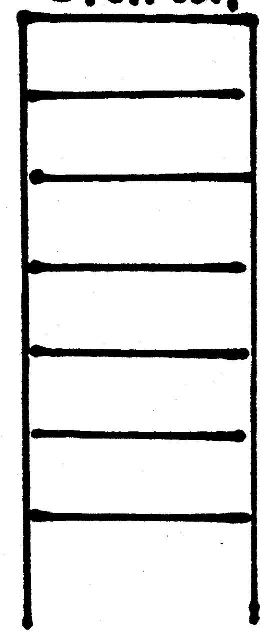
Each chunk map entry corresponds to

DMMAX * 1024 bytes

of swap space.

(Default DMMAX = 512)

chmap



A chunk map is not built on a diskless node with no local swap disks.

SWAPPING

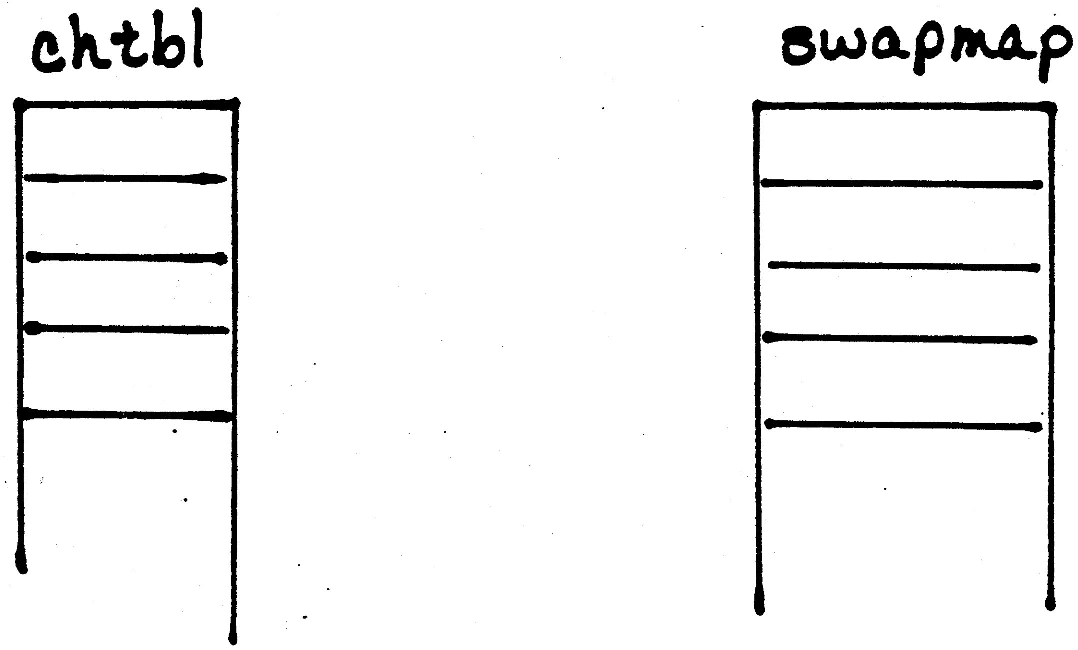
If a system has multiple swap devices configured in the kernel, chunk map entries for each of the devices is interleaved in a circular fashion.

The last entry for each swap device may be less than DMMAX * 1024.

Entries for dises other than the default swap device only become valid after swapon has been executed.

SWAPPING

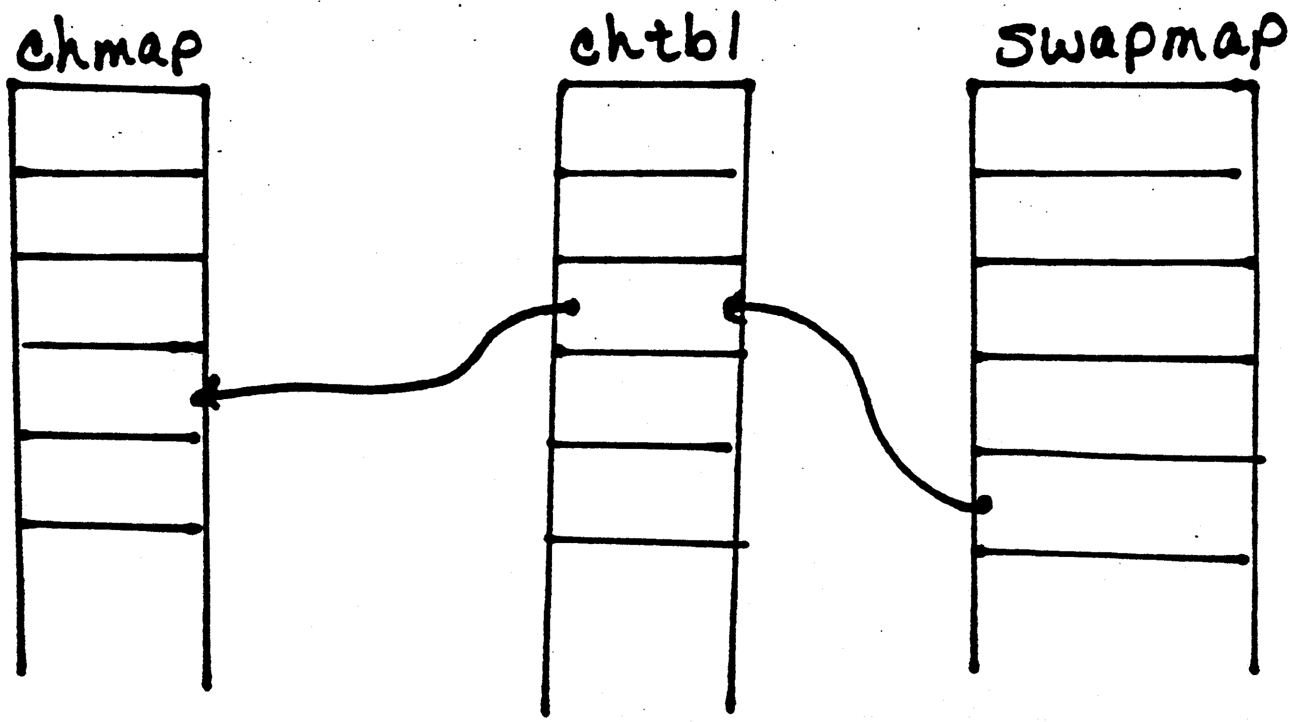
All systems, those that swap remotely and those that swap on local discs, create a chunk table and a swapmap.



chtbl is dimensioned for maxcnodeswchunks entries. It is used to keep track of swap chunks allocated from the chunk map.

The swapmap is used as in 5.X systems.

SWAPPING

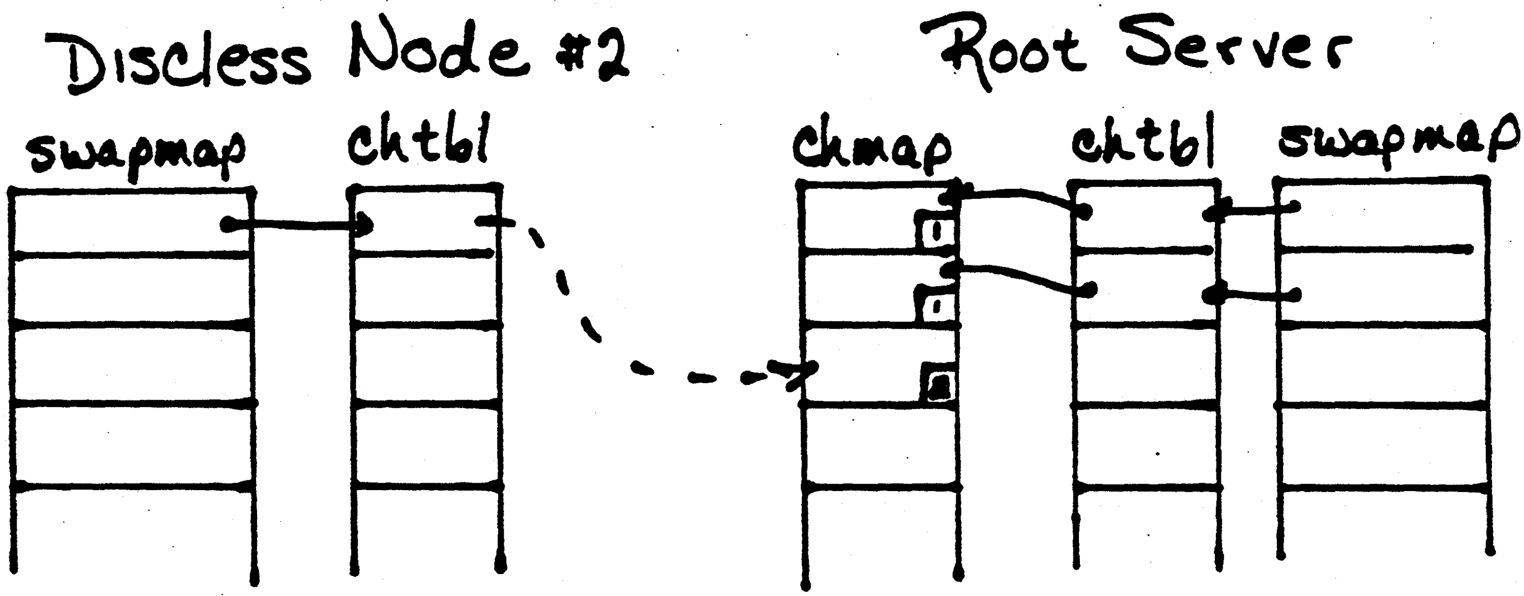


As swap space is required, a request is ade for additional swap space.

An entry in the chunk map is allocated. An entry in the chunk table is made to keep track of the chunk of swap space allocated.

New entries in the swap map are created to show the new available swap space.

SWAPPING

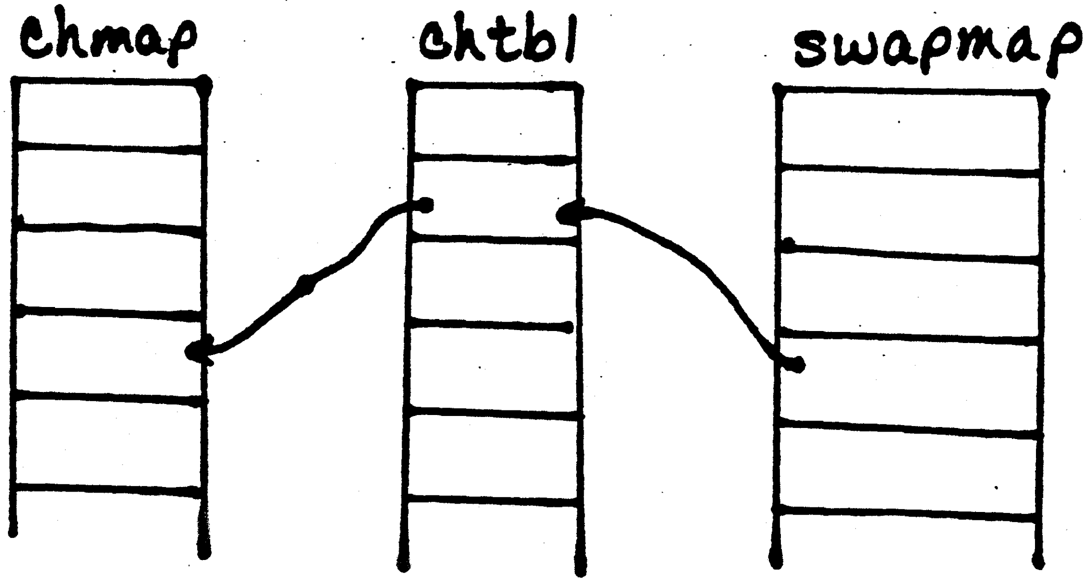


The chunk map is used to manage the global shared swap space on the Root Server.

There is a cnode ID field in the chunk map entry to record what cnode is using each chunk of swap space.

When a cnode fails, the chunk map is scanned for chunks allocated to the failed cnode. These chunks are freed and returned to pool of available chunks.

SWAPPING



A kernel daemon runs every 30 seconds on every cnode to check for chunks of swap space that may be returned to the global ool.

If a chunk of swap is not being used, the entire chunk will be in the swapmap.

The swap daemon will set a reference bit in the chunk table entry for any chunk not being used. If this chunk is still unused the next time the daemon runs, the swapmap entries will be removed and the chunk will be freed.

SWAPPING

DMMAX

Must be same for all nodes swapping on
Root Server

Swap space partitioned in chunks of
DMMAX * 1024

DMMAX on Root Server used by all nodes
swapping on Server

DMSHM and DMTEXT
Not forced to be same as Root Server
Not allowed to exceed DMMAX

SWAPPING

maxswapchunks

Configurable kernel parameter

Number of DMMAX * 1024 chunks that
a node may allocate

Defaults to 512

May be used to restrict swap space
allowed to a node

SWAPPING

minswapchunks

Configurable kernel parameter

Number of DMMAX * 1024 chunks
allocated to node at bootup

Node always retains this minimum
amount of swap

Defaults to 4

Discless Network Protocol

Discless Network Protocol

- Better Performance than Existing Protocols
- Used only by Discless Kernel
- Communication on a Single LAN only
- Does Not support Gateways
- Co-Exists with other Networking Services

****

The S300 discless workstation implementation does not use any of the standard networking protocols such as TCP/IP, LLA or UDP. The Discless protocol is a special HP proprietary protocol that has been designed for optimum performance in a discless environment.

This protocol is only used by the HP-UX kernel in support of discless workstations. At the 6.0 release, it supports communication between a root server and its discless nodes over a single LAN only. Because this is not an IP protocol, it is not supported across an IP gateway.

The discless protocol co-exists with the other HP supported networking services. Both the server and discless nodes may use NS and ARPA/Berkeley services to communicate with systems outside the cluster as well as inside the cluster. NFS may be used by both the server and discless nodes to access files systems outside the cluster.

Allows Use of a Simple Protocol

- Messages rarely Lost in a LAN
- No need to copy Data between Buffers
- Certain Requests are Idempotent
- Preallocates space for Reply Message
- Not for General Purpose Communication

****

Because the discless protocol is limited to a single LAN, certain assumptions ay be made which allow for a simple protocol. These include:

Messages are rarely lost in a single LAN

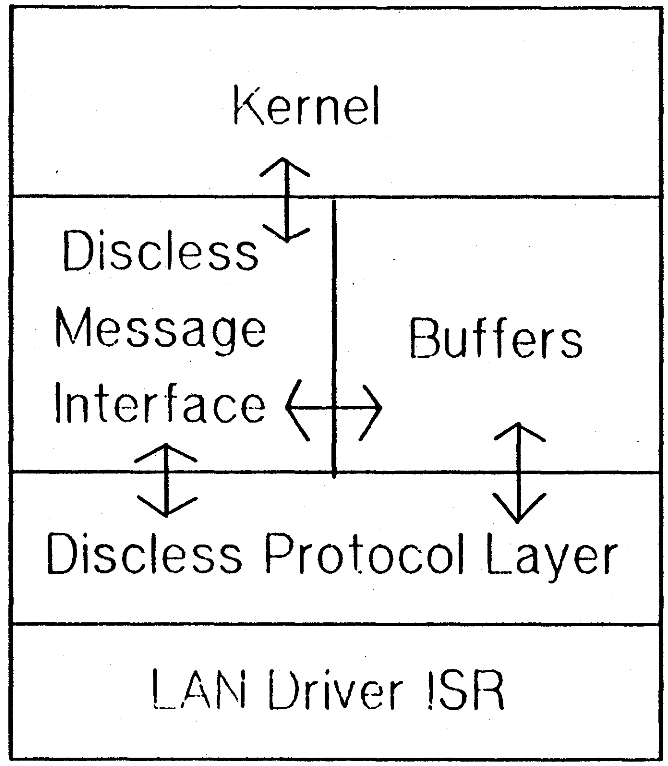
Significant time is lost in copying data between network buffers and user buffers. Therefore, the discless protocol copies directly from the LAN card into the discless netbufs.

Certain requests are idempotent - this means that these requests may be re-executed with no change in effect. An example of such a request would be a sync. Idempotent requests do not require acknowledgement.

Performance can be improved by preallocating space for the reply message before the request is sent out.

Since this protocol is not designed for general purpose communication, it may be tailored to the specific requirements of the cluster.

Discless Network Protocol



****

This diagram shows how the discless message interface provides for communication between the kernel and the Discless Protocol Layer. The Discless Protocol Layer interfaces directly with the LAN hardware. The protocol and uffer management are hidden from the kernel.

Inbound messages are passed to the Discless Protocol Layer by the LAN Software ISR while Outbound messages are passed directly to the LAN Hardware ISR.

Discless Network Protocol

The Discless protocol is a Request/Reply protocol.

Non-Idempotent Request

Non-Idempotent requests must be executed only once so these requests must be acknowledged. An example of such a request is a file open request.

```
Requestor                                    Receiver
                                                  (Server)

                  -----request------------>

                  <----------reply---------

                  ----acknowledge--------->
```

The serving node will continue to transmit the reply until an acknowledge is received.

If the requestor does not receive the reply within a timeout period, the request will be sent again. The server will ignore duplicate requests.

Discless Network Protocol

Idempotent Request

Requestor **Receiver**

-------request-------->

<--------reply---------

Idempotent requests are requests which produce the same effect when executed multiple times - such as a sync request. Such requests do not need to be acknowledged.

If no reply is received by the requestor within a timeout period, the request will be sent again. Even if the request has already been executed, it will be executed again.

Discless Network Protocol

"Slow Requests"

Some requests may take an indefinitely long time to complete - such as a request to write to a file which is locked.

Requestor   Server

-------request----------------->

(timeout expires)

----duplicate request----------->

<--"this is a slow request"------

(request is completed)

<---------reply-----------------

-----acknowledge--------------->

When a duplicate request is received by the server and the server has not been able to service the request, the server will send an acknowledgement to the requestor indicating this is a "slow request". The requestor will then stop repeating requests and wait for the reply.

Datagram Messages

- For Accessing Network with Minimal Overhead
- Datagrams not Queued, Network Driver called directly
- Single Packet Request or Reply
- No Acknowledgement or Reply Required

Datagram Messages

- Used for these Types of Messages

  Clocksync

  NAK's

  I'm Alive

  Broadcast Failure

- These messages use Statically allocated mbufs to Increase Probability of being Delivered

mbufs

- For Discless Message Headers
- 128 Byte Buffers
- Allocated at Cluster Time
- Required for each Message
- Configured by dskless_mbufs
- dskless_mbufs = Number of pages of mbufs

****

mbufs are a linked list of 128 byte buffers allocated at cluster time. Pages of memory are allocated and "chopped" into 128 byte pieces which are linked together.

The kernel parameter maxdiscless_mbufs is a number of memory pages to be used for mbufs - 32 mbufs per page of memory.

If a system has insufficient mbufs allocated, look for "Cannot allocate message buffer" messages. The mbuf utilization may be checked using the M screen of the 6.0 version of monitor.

cbufs

- For Discless Message Data
- 1024 Byte Buffers
- Allocated at Cluster Time
- Required if entire Message does not fit in an mbuf
- Configured by dskless_cbufs
- dskless_cbufs = Number of pages of cbufs
- If entire Message does not fit in a cbuf, file system Buffer is allocated

****

cbuf are a linked list of 1024 byte buffers allocated at cluster time. Pages of memory are allocated and "chopped" into 1024 byte pieces which are linked together.

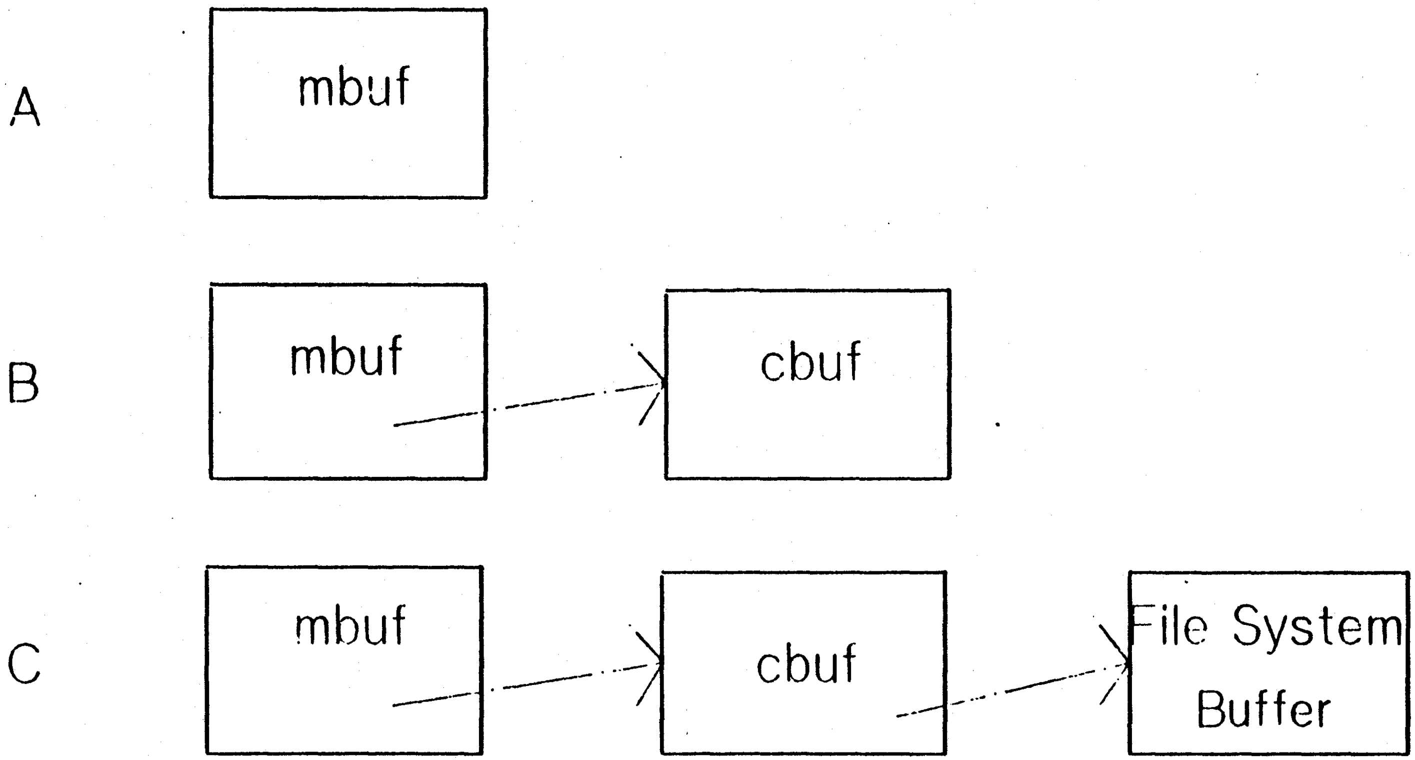
The kernel parameter maxdiscless_clusters determines the number of memory pages to be allocated for cbufs - cbufs per page of memory. The default value of maxdiscless_clusters is ????

If the system runs out of the preallocated pool of cbufs, more pages of memory will be taken from the page pool and used for cbufs. Once these buffers have been freed, the pages will be returned to the page pool. The cbuf utilization may be checked using the M screen of the 6.0 version of monitor.

Performance considerations:

If the root server system is going to be used as a server only, it is a good idea to allocate more than sufficient memory to the discless protocol. If the server is not being used as a workstation it does not make sense to be dropping messages due to lack of protocol buffer resou

Discless Outgoing Messages



****

Outgoing messages may be of varying sizes. A message may require only an mbuf which means that the message is just a header with the information encoded in fields of the header. Examples of this type of message are `end Alive requests, "I'm Alive" messages, etc.

For longer messages, an cbuf is also used. The mbuf is still required build the header and the additional message data is stored in the cbuf. The mbuf contains a pointer which points to the cbuf buffer.

Very long messages, such as file system buffer write requests, will require more space than the mbuf and cbuf combined. For these requests, file system buffers are allocated. The file system buffers may be either 4K bytes or 8K bytes. The following combinations are possible for these long messages

mbuf + 4K File System Buffer = 128 + 4096

mbuf + 8K File System Buffer = 128 + 8192

mbuf + cbuf + 4K File System Buffer = 128 + 1024 + 4096

mbuf + cbuf + 8K File system Buffer = 128 + 1024 + 8192

Network Buffers

- Preallocated Pool of File System Buffers
- Available to Kernel under Interrupt
- Used to Receive Incoming Request Message
- Prevents having to Copy Data to a File System Buffer

****

These is also a preallocated pool of Network Buffers called Netbufs.

Netbufs are a pool of file system buffers allocated at cluster time which e available to the kernel to be used under interrupt when receiving a request. is is done so that when requests are received under interrupt, the kernel oes not have to wait for a buffer to be allocated.

When a request is received, the message is copied directly from the hardware buffer into a netbuf for processing. The use of netbufs prevents having to copy the data into file system buffers. Since netbufs are preallocated file system buffers they can be used directly by the file system by exchanging pointers.

If a system is out of available netbufs at the time a request is received, the request must be dropped. A NAK will be sent to the requesting node. You may look at netbuf utilization by checking the M screen of the 6.0 version of monitor.

FSBUFS - This is the number of netbuf headers allocated

FSPAGES - This is the number of memory pages used for netbuf data buffers allocated.

FSPAGES will probably be greated than FSBUFS because some messages will require more than 4K for data space.

Discless Network Protocol

When Sending a Discless Request

- mbuf, cbuf or file system buffer is used to build the request message

- Buffer space for the reply message is preallocated from the mbufs and file system buffers

- The message is placed in the Discless message queue. In the interest of performance, the Discless protocol message takes precedence over other protocols.

- When the message is sent out, data is copied directly from the Discless buffers into the hardware buffer.

Discless Network Protocol

```
   Requestor                           Server

       |                                 /\
       |                                  |
      \/                                  |

Get Buffers for Request          Request handled under
Preallocate buffers for          interrupt or given to
  reply                           gcsp/ucsp/lcsp

       |                                 /\
       |                                  |
      \/                                  |

Give Message to Discless         Request handled by Protocol
Message Layer                            Layer

       |                                 /\
       |                                  |
      \/                                  |

Pass message off to              Request is received by
Protocol Layer by                Discless Receive Routines
placing on queue                     (Serving_Array)
(Using_Array)

       |                                 /\
       |                                  |
      \/                                  |

Message is put on                 LAN H/W ISR
Network

       |                                 /\
       |                                  |
       |__________________________________|
```

Discless Network Protocol

All cluster nodes maintain a using_array and a serving_array.

using_array Keeps track of outstanding/active requests made by the local node.

serving_array Keeps track of received requests which are being serviced.

Root Server	Needs small using_array Needs large serving_array
Discless Node	Needs large using_array Needs small serving_array

The sizes are determined by

using_array_size

serving_array_size

KERNEL PARAMETERS

New Configurable Kernel Parameters

For new libraries:

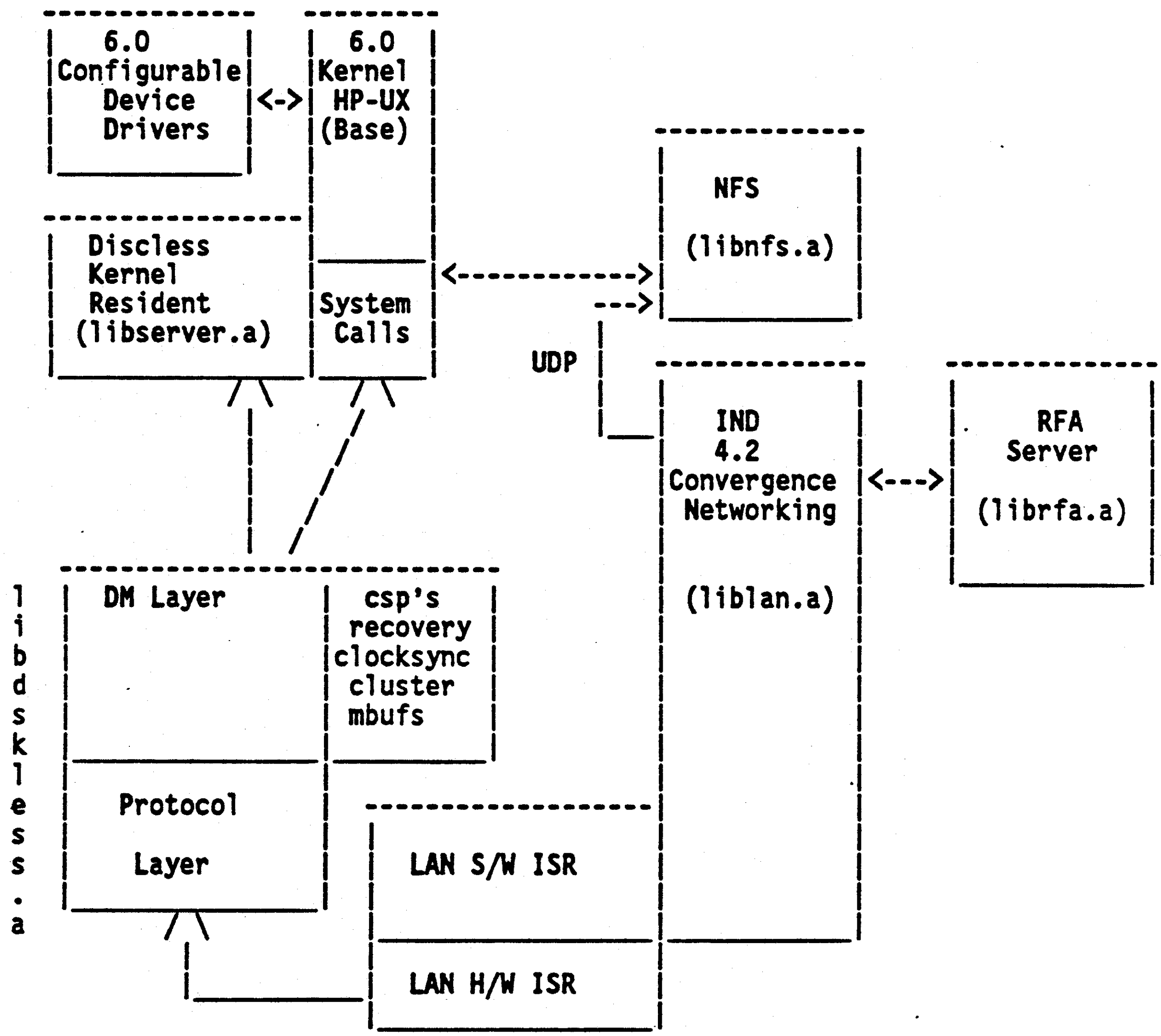
dskless - brings in libdskless.a
routines required to run discless
Discless protocol, Clock Sync,
Crash Recovery, CSP's
Also lan, rdu, nsdiag

rfa - brings in librfa.a
NS and RFA routines

lla and/or lan01
brings in liblan.a
LAN drivers and 4.2
convergence networking code

nfs - brings in libnfs.a
NFS routines

The Configurable Kernel Pieces



KERNEL PARAMETERS

Drivers:

nsdiag	- LAN diagnostics
rdu	- Driver used in remote swapping
fpa	- Dragon floating point accellerator support
vme	- vme support
stealth	- vme backplane support
dos	- DOS card support
scsi	- Small computer system interface driver

Kernel Parameters

LAN:

num_lan_cards	- Number of LAN cards to be supported on system default = 2
netmemmax netmemthresh	- Since npowerup command has been replaced by ifconfig, these parameters are used to allocate sufficient memory buffers for networking.

KERNEL PARAMETERS

Discless:

num_cnodes

Limiter for discless resource allocation Similar to maxusers. Does not limit the number of cnodes supported by the server.

Used in sizing NINODE, NGCSP, serving_array_size, num_retry_reply, num_retry_request

dskless_node

Value should be set to 1 for discless node and set to 0 for the root server. Used in sizing the using_array.

server_node

Value should be set to 1 for root server and set to 0 for a discless node. Used in sizing the serving_array.

KERNEL PARAMETERS

Discless:

using_array_size

Determines size of the using_array.

Default: NPROC

serving_array_size

Determines size of the serving_array

Default:

(server_node * num_cnodes * maxusers)+(2 * maxusers)

dskless_fsbufs

Determines the size of the pool of netbufs

Default = serving_array_size

KERNEL PARAMETERS

Discless:

dskless_mbufs

Numbers of mbufs to allocate at cluster time.

Default =

(((serving_array_size + (2 * using_array_size))/32) + 1)

dskless_cbufs

Number of cbufs to allocate at cluster time.

Default = dskless_mbufs * 2

ngcsp

Determines number of general CSP's
that may run on a system

Default = 4 * num_cnodes

KERNEL PARAMETERS

Discless:

maxswapchunks

Determines size of the Swap Space Chunk Table. Default is 512.

minswapchunks

Size of Swap Area always allocated to a node. Default is Default = 4 chunks.

Kernel Parameters

Discless:

selftest_period

Period in seconds between executions of kernel selftest routine.

Default - 140 sec

Maximum - 300 sec

If set to 0, turns off selftest.

check_alive_period

Period in seconds between executions of Check Alive routine.

Default - 10 sec

Minimum - 10 sec

KERNEL PARAMETERS

retry_alive_period

Number of times to retry Send Alive messages to a site before executing Cable Break detection routine.

Default - 20 sec

Minimum - 10 sec

retryselftest_period

Selftest retry period if selftest detects a failure.

Default - 4 sec

Maximum - 1/2 selftest_period

SE 390: Series 300 HP-UX Internals

Module Evaluation

DISKLESS

On a scale of 1-10, 1 being bad, 5 being OK/don't care/irrelevant, 10 being good, please rate the following. If you have particular comments, please write them in. Thank you!

1. Clarity of presentation:

2. Depth/complexity (1 - material was too easy, 10 - it was too hard):

3. Usefulness/applicability/relevance of material presented:

4. Speed of presentation (1 - too slow, 10 - too fast):

5. How good was the material (slides, notes, etc)?

6. How good was the instructor?

Ways this could be improved (please be specific):

General Comments:

I/O Section - Intro

I. Overview
   A. What is a driver?
   B. Types of drivers?
   C. How is the driver accessed?
   D. How a driver is configured into the kernel

II. Review a simple driver (RAM Disc driver)
   A. What the Kernel does for you
   B. What the driver does for you
      1. Block device routines
      2. Character device routines

III. Review a "real" driver (gpio card)
   A. Walk thru an open call
   B. Walk thru a read system call
      1. not using DMA
      2. using DMA

IV. RS-232 drivers
   A. Use of buffers
   B. What is "canonical processing?"

Types of Drivers

Block Mode - uses a buffer cache that is maintained by the File System
- usually associated with the File System, and deals with blocks of data of the same size
- used with devices that have random access.
- ideal for using DMA type transfers

Character Mode - usually sequential devices (e.g. printers, terminals, 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

Character Mode Drivers fall into three main types:

1) Very similar to the Block Mode driver. For example, the CS80 driver uses much of the same code for its block & character mode access. The driver uses a buffer header like the block mode driver, and may actually "borrow" one from the buffer cache. The buffer space is (usually) the user's buffer, which is mapped into the kernel space. This method does not require copying data from users space to a kernel buffer. Used with drivers that perform large transfers and DMA capable.

2) Serial drivers use internal buffers (Clists/Cblocks) for holding the data for transfer. They (can) perform processing of the data using canonical processing (e.g. ERASE, KILL, etc.). Data is transfered between these buffers and the user's buffers. They usually deal with small/slow transfers.

3) The third type contains internal buffers (like serial drivers) and transfers the data between the user's space and kernel space for the I/O transfers. It will use the CPU (via interrupts) to transfer the data. An example of this type of driver is the rje driver. This type does not use DMA for transfers.

DIL added another type of driver, which is CPU intensive. It uses the IOMAP facility to map the I/O card into the users address space and then copies data directly between the user's buffer and the card.

How Drivers are Accessed

- I/O to/from devices are accessed using the same semantics as normal files in the file system. By using this method, a program does not have to treat access to a file or a device any differently.

- All I/O starts with accessing the File System (during the open). The "open" system call accesses the file system and puts the device file info into the file descriptor table. It also will perform any necessary device dependent operations.

- I/O Reads/Writes follow the same path as used to read and write files in the upper levels of the kernel. This is also true for Pipes (FIFOS), Directories, Networked Special Files, Symbolic Links. At this point in the kernel, we diverge to the different areas in the kernel (drivers for I/O).

What is a Driver?

Provides the window to interface to the outside world

Provides the hardware specific routines

Provides a common interface to the kernel

How A Driver Is Configured?

- /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. 98642). Some drivers may combine both, as in the gpio driver.

- The driver information in /etc/master tells "config" what entries to make in the conf.c file created. The following gives examples of these entries.

```
*
* name          handle  type    mask    block   char
*
cs80            cs80    3       3FB     0       4
tape            tp      1       FA      -1      5
ramdisc         ram     3       FB      4       20
*
* name          handle  type    mask    block   char
*
98624           ti9914  10      100     -1      -1
98625           simon   10      100     -1      -1
98626           sio626  10      100     -1      -1
98628           sio628  10      100     -1      -1
98642           sio642  10      100     -1      -1
*
* name          handle  type    mask    block   char
*
tty             sy      D       FD      -1      2
```

- A description of the fields are:

```
name - the name used in the "dfile" signifying the requested driver
handle - the "handle" actually used for the subroutine calls in the
      kernel (e.g. for tty driver, the open routine would be 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 (always TRUE for select)
      | | | | | | | \----- 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" & "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 apporpriate driver. By modifying /etc/master's driver numbers, you can change the "major" numbers for your drivers.

```
struct bdevsw bdevsw[] = {
/* 0*/ cs80_open, cs80_close, cs80_strategy, cs80_size, C_ALLCLOSES,
/* 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 routines. These routines check to see if the card is theirs. If so, it may perform some initializations and it reports finding the card. If not, the make_entry() routine will call the next make_entry() routine. 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,
        srm629_link,
        rje_link,
        ptys_link,
        lla_link,
        hpib_link,
        vme_link,
        stealth_link,
        rfai_link,
```

```
                ti9914_link,
                simon_link,
                sio626_link,
                sio628_link,
                sio642_link,
                ite200_link,
                (int (*)())0
        };
```

```
* UX_ID:  @(#)dfile.full.lan  49.3              87/09/28
* dfile.full.lan
*
* This is the configuration file for a full system, with LAN
*
* DEVICE DRIVERS
* disc drivers
cs80
scsi
amigo
* tape drivers
tape
stape
* printer drivers
printer
ciper
* shared resource management driver
srm
* pseudo terminal drivers (needed for windows)
ptymas
ptyslv
* dil hpib driver (includes plotters)
hpib
* dil gpio driver
gpio
*   note job entry
r
*    98286 DOS Coprocessor driver (see dos_mem_byte parameter)
dos
* HP 98646 VME driver
vme
* HP 98577 VME expander
vme2
* If you want to run NFS, uncomment the following line.
*nfs
* lan drivers (formerly: ieee802 & ethernet drivers)
lla
lan01
nsdiag0
* RFA server code
rfa
* CARDS
* HP-IB interface
98624
* high_speed HP-IB interface
98625
* RS-232 serial interface
98626
* RS-232 datacomm interface
98628
* RS-232 multiplexer
98642
```

```
/*
 *  Configuration information
 */


#define MAXUSERS        8
#define TIMEZONE        420
#define DST     1
#define NPROC   (20+8*MAXUSERS+(NGCSP))
#define NUM_CNODES      0
#define DSKLESS_NODE    0
#define SERVER_NODE     0
#define NINODE  ((NPROC+16+MAXUSERS)+32+(2*NPTY)+SERVER_NODE*18*NUM_CNODES)
#define NFILE   (16*(NPROC+16+MAXUSERS)/10+32+(2*NPTY))
#define ARGDEVNBLK      0
#define NBUF    0
#define DOS_MEM_BYTE    0
#define NCALLOUT        (16+NPROC+USING_ARRAY_SIZE+SERVING_ARRAY_SIZE)
#define NTEXT   (40+MAXUSERS)
#define UNLOCKABLE_MEM  102400
#define NFLOCKS 200
#define NPTY    82
#define MAXUPRC 25
#define DMMIN   16
#define DMMAX   512
#define DMTEXT  512
#define DMSHM   512
#define MAXDSIZ 0x01000000
#define MAXSSIZ 0x00200000
#define MAXTSIZ 0x01000000
#define SHMMAXADDR      0x01000000
#define PARITY_OPTION   2
#define TIMESLICE       0
#define ACCTSUSPEND     2
#define ACCTRESUME      4
#define NDILBUFFERS     30
#define FILESIZELIMIT   0x1ffffff
#define DSKLESS_MBUFS   (((SERVING_ARRAY_SIZE+(2*USING_ARRAY_SIZE))/32)+1)
#define DSKLESS_CBUFS   (DSKLESS_MBUFS*2)
#define USING_ARRAY_SIZE        (NPROC)
#define SERVING_ARRAY_SIZE      (SERVER_NODE*NUM_CNODES*MAXUSERS+2*MAXUSERS)
#define DSKLESS_FSBUFS  (SERVING_ARRAY_SIZE)
#define SELFTEST_PERIOD 120
#define CHECK_ALIVE_PERIOD      4
#define RETRY_ALIVE_PERIOD      21
#define MAXSWAPCHUNKS   512
#define MINSWAPCHUNKS   4
#define NUM_LAN_CARDS   2
#define NETMEMMAX       250000
#define NETMEMTHRESH    100000
#define NGCSP   (8*NUM_CNODES)
#define SCROLL_LINES    100
#define MESG    1
#define MSGMAP  (MSGTQL+2)
#define MSGMAX  8192
```

```
#define MSGMNB  16384
#define MSGMNI  50
#define MSGSSZ  1
#define MSGTQL  40
#define MSGSEG  16384
#define SEMA    1
#define SEMMAP  (SEMMNI+2)
#define SEMMNI  64
#define SEMMNS  128
#define SEMMNU  30
#define SEMUME  10
#define SEMVMX  32767
#define SEMAEM  16384
#define SHMEM   1
#define SHMMAX  0x00600000
#define SHMMIN  1
#define SHMMNI  30
#define SHMSEG  10
#define SHMBRK  16
#define SHMALL  2048
#define FPA     1

#include        "/etc/conf/h/param.h"
#include        "/etc/conf/h/systm.h"
#include        "/etc/conf/h/tty.h"
#include        "/etc/conf/h/space.h"
#include        "/etc/conf/h/opt.h"
#include        "/etc/conf/h/conf.h"

#define ieee802_open    lan_open
#define ieee802_close   lan_close
#define ieee802_read    lan_read
#define ieee802_write   lan_write
#define ieee802_link    lan_link
#define ieee802_select  lan_select
#define ethernet_open   lan_open
#define ethernet_close  lan_close
#define ethernet_read   lan_read
#define ethernet_write  lan_write
#define ethernet_link   lan_link
#define ethernet_select lan_select
#define hpib_link       gpio_link
#define lla_link        lan_link
#define lan01_link      lan_link

extern nodev(), nulldev();
extern seltrue();

extern cs80_open(), cs80_close(), cs80_read(), cs80_write(), cs80_ioctl(), cs80_
extern amigo_open(), amigo_close(), amigo_read(), amigo_write(), amigo_ioctl(),
extern swap_strategy();
extern swap1_strategy();
extern scsi_open(), scsi_close(), scsi_read(), scsi_write(), scsi_ioctl(), scsi_
extern cons_open(), cons_close(), cons_read(), cons_write(), cons_ioctl(), cons_
extern tty_open(), tty_close(), tty_read(), tty_write(), tty_ioctl(), tty_select
extern sy_open(), sy_close(), sy_read(), sy_write(), sy_ioctl(), sy_select();
```

```
extern mm_read(), mm_write();
extern tp_open(), tp_close(), tp_read(), tp_write(), tp_ioctl();
extern lp_open(), lp_close(), lp_write(), lp_ioctl();
extern swap_read(), swap_write();
extern stp_open(), stp_close(), stp_read(), stp_write(), stp_ioctl();
extern iomap_open(), iomap_close(), iomap_read(), iomap_write(), iomap_ioctl();
extern graphics_open(), graphics_close(), graphics_read(), graphics_write(), gra
extern srm629_open(), srm629_close(), srm629_read(), srm629_write(), srm629_link
extern rje_open(), rje_close(), rje_read(), rje_write(), rje_ioctl(), rje_link()
extern ptym_open(), ptym_close(), ptym_read(), ptym_write(), ptym_ioctl(), ptym_
extern ptys_open(), ptys_close(), ptys_read(), ptys_write(), ptys_ioctl(), ptys_
extern lla_open(), lla_close(), lla_read(), lla_write(), lla_ioctl(), lla_select
extern lla_open(), lla_close(), lla_read(), lla_write(), lla_ioctl(), lla_select
extern hpib_open(), hpib_close(), hpib_read(), hpib_write(), hpib_ioctl();
extern hpib_open(), hpib_close(), hpib_read(), hpib_write(), hpib_ioctl(), hpib_
extern r8042_open(), r8042_close(), r8042_ioctl();
extern hil_open(), hil_close(), hil_read(), hil_ioctl(), hil_select();
extern nimitz_open(), nimitz_close(), nimitz_read(), nimitz_select();
extern ciper_open(), ciper_close(), ciper_write(), ciper_ioctl();
extern dos_open(), dos_close(), dos_read(), dos_write(), dos_ioctl();
extern vme_open(), vme_close(), vme_read(), vme_write(), vme_ioctl(), vme_link()
extern stealth_open(), stealth_close(), stealth_ioctl(), stealth_link();
extern nsdiag0_open(), nsdiag0_close(), nsdiag0_read(), nsdiag0_ioctl();

extern rfai_link();
extern ti9914_link();
extern simon_link();
extern sio626_link();
extern sio628_link();
extern sio642_link();
extern ite200_link();

struct bdevsw bdevsw[] = {
/* 0*/  cs80_open, cs80_close, cs80_strategy, cs80_size, C_ALLCLOSES,
/* 1*/  nodev, nodev, nodev, nodev, 0,
/* 2*/  amigo_open, amigo_close, amigo_strategy, amigo_size, C_ALLCLOSES,
/* 3*/  nodev, nodev, swap_strategy, 0, 0,
/* 4*/  nodev, nodev, nodev, nodev, 0,
/* 5*/  nodev, nodev, swap1_strategy, 0, 0,
/* 6*/  nodev, nodev, nodev, nodev, 0,
/* 7*/  scsi_open, scsi_close, scsi_strategy, scsi_size, C_ALLCLOSES,
};

struct cdevsw cdevsw[] = {
/* 0*/  cons_open, cons_close, cons_read, cons_write, cons_ioctl, cons_select, C
/* 1*/  tty_open, tty_close, tty_read, tty_write, tty_ioctl, tty_select, C_ALLCL
/* 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_ALL
/* 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,
/* 8*/  nulldev,  nulldev, swap_read, swap_write, nodev, nodev, 0,
/* 9*/  stp_open, stp_close, stp_read, stp_write, stp_ioctl, seltrue, 0,
/*10*/  iomap_open, iomap_close, iomap_read, iomap_write, iomap_ioctl, nodev, C_
/*11*/  amigo_open, amigo_close, amigo_read, amigo_write, amigo_ioctl, seltrue,
```

```
/*12*/   graphics_open, graphics_close, graphics_read, graphics_write, graphics_i
/*13*/   srm629_open, srm629_close, srm629_read, srm629_write, nodev, seltrue, 0,
/*14*/   nodev, nodev, nodev, nodev, nodev, nodev, 0,
/*15*/   rje_open, rje_close, rje_read, rje_write, rje_ioctl, seltrue, 0,
/*16*/   ptym_open, ptym_close, ptym_read, ptym_write, ptym_ioctl, ptym_select, 0
/*17*/   ptys_open, ptys_close, ptys_read, ptys_write, ptys_ioctl, ptys_select, C
/*18*/   lla_open, lla_close, lla_read, lla_write, lla_ioctl, lla_select, C_ALLCL
/*19*/   lla_open, lla_close, lla_read, lla_write, lla_ioctl, lla_select, C_ALLCL
/*20*/   nodev, nodev, nodev, nodev, nodev, nodev, 0,
/*21*/   hpib_open, hpib_close, hpib_read, hpib_write, hpib_ioctl, seltrue, C_ALL
/*22*/   hpib_open, hpib_close, hpib_read, hpib_write, hpib_ioctl, seltrue, C_ALL
/*23*/   r8042_open, r8042_close, nodev, nodev, r8042_ioctl, nodev, 0,
/*24*/   hil_open, hil_close, hil_read, nodev, hil_ioctl, hil_select, 0,
/*25*/   nimitz_open, nimitz_close, nimitz_read, nodev, nodev, nimitz_select, 0,
/*26*/   ciper_open, ciper_close, nodev, ciper_write, ciper_ioctl, seltrue, 0,
/*27*/   dos_open, dos_close, dos_read, dos_write, dos_ioctl, nodev, C_ALLCLOSES,
/*28*/   nodev, nodev, nodev, nodev, nodev, nodev, 0,
/*29*/   nodev, nodev, nodev, nodev, nodev, nodev, 0,
/*30*/   nodev, nodev, nodev, nodev, nodev, nodev, 0,
/*31*/   nodev, nodev, nodev, nodev, nodev, nodev, 0,
/*32*/   vme_open, vme_close, vme_read, vme_write, vme_ioctl, nodev, 0,
/*33*/   nodev, nodev, nodev, nodev, nodev, nodev, 0,
/*34*/   nodev, nodev, nodev, nodev, nodev, nodev, 0,
/*35*/   nodev, nodev, nodev, nodev, nodev, nodev, 0,
/*36*/   nodev, nodev, nodev, nodev, nodev, nodev, 0,
/*37*/   nodev, nodev, nodev, nodev, nodev, nodev, 0,
/*38*/   nodev, nodev, nodev, nodev, nodev, nodev, 0,
/*39*/   nodev, nodev, nodev, nodev, nodev, nodev, 0,
/*40*/   nodev, nodev, nodev, nodev, nodev, nodev, 0,
/*41*/   nodev, nodev, nodev, nodev, nodev, nodev, 0,
/*42*/   nodev, nodev, nodev, nodev, nodev, nodev, 0,
/*43*/   nodev, nodev, nodev, nodev, nodev, nodev, 0,
/*44*/   stealth_open, stealth_close, nodev, nodev, stealth_ioctl, nodev, 0,
/*45*/   nodev, nodev, nodev, nodev, nodev, nodev, 0,
/*46*/   nsdiag0_open, nsdiag0_close, nsdiag0_read, nodev, nsdiag0_ioctl, seltrue
/*47*/   scsi_open, scsi_close, scsi_read, scsi_write, scsi_ioctl, seltrue, C_ALL
};

int      nblkdev = sizeof (bdevsw) / sizeof (bdevsw[0]);
int      nchrdev = sizeof (cdevsw) / sizeof (cdevsw[0]);

dev_t    rootdev = makedev(-1,0xFFFFFF);

/* The following three variables are dependent upon bdevsw and cdevsw. If
   either changes then these variables must be checked for correctness */

dev_t    swapdev1 = makedev(5, 0x000000);
int      brmtdev = 6;
int      crmtdev = 45;

struct swdevt swdevt[] = {
         { makedev(-1,0xFFFFFF), 0, -1, 0 },
         { 0, 0, 0, 0 }
}
```

```
int     (*driver_link[])() =
{
        cs80_link,
        amigo_link,
        scsi_link,
        graphics_link,
        srm629_link,
        rje_link,
        ptys_link,
        lla_link,
        hpib_link,
        vme_link,
        stealth_link,
        rfai_link,
        ti9914_link,
        simon_link,
        sio626_link,
        sio628_link,
        sio642_link,
        ite200_link,
        (int (*)())0
};
```

```
##
##  HP-UX System Makefile
##

# .SILENT
IDENT = -Dhp9000s200 -DKERNEL -Dhpux -DLOCKF -DHFS
REALTIME = -DRTPRIO -DPROCESSLOCK

CC = /bin/cc +X
AS = /bin/as
LD = /bin/ld
SHELL = /bin/sh

LIBSERVER = `if [ -f /etc/conf/libserver.a ]; then echo /etc/conf/libserver.a; f

LIBDSKLESS = `if [ -f /etc/conf/libdskless.a ]; then echo /etc/conf/libdskless.a

LIBLAN = `if [ -f /etc/conf/liblan.a ]; then echo /etc/conf/liblan.a; fi`

LIBS = -lc
LIBS1 = /etc/conf/libkreq.a\
        /etc/conf/libdreq.a\
        /etc/conf/libsysV.a\
        /etc/conf/libmin.a\
        /etc/conf/libdevelop.a\
        /etc/conf/libdil_srm.a\
        $(LIBNFS)\
        $(LIBSERVER)\
        $(LIBDSKLESS)\
        $(LIBLAN)

CFLAGS = -O -Wc,-Nd3500,-Ns3500
COPTS = $(IDENT) $(REALTIME) -DWOPR

all:    hp-ux

hp-ux:  conf.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 \
                $(LIBS1) $(LIBS)
        rm -f locore.o vers.o name.o funcentry.o
        chmod 755 hp-ux

conf.o:
        @echo 'Compiling conf.c ...'
        $(CC) $(CFLAGS) $(COPTS) -c conf.c
```

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 32bit value, with 8 bits for major number and 24bits 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   /* (256 bytes) "sector" size (log2) of the ram discs */

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];
```

```
/
**  Open the ram device.
*/
ram_open(dev, flag)
dev_t dev;
int flag;
{
        register unsigned long size;
        register struct ram_descriptor *ram_des_ptr;

        /* check if this is status open */
        if (RAM_MINOR(dev) == 0)
                return(0);

        /* check if this device is greater than max number of volumes */
        if ((size = RAM_DISC(dev)) > RAM_MAXVOLS)
                return(EINVAL);

        ram_des_ptr = &ram_device[size];

        /* check the size of the ram disc less than 16 sectors */
        if ((size = RAM_SIZE(dev)) < 16)
                return(EINVAL);

        /* check if already allocated */
        if (ram_des_ptr->addr != NULL) {

                /* then check if size changed; must be the same size */
                if (ram_des_ptr->size != size)
                        return(EINVAL);

                /* bump open count */
                ram_des_ptr->opencount++;
        } else {
                /* allocate the memory for the ram disc */
                if ((ram_des_ptr->addr =
                       (char *)sys_memall(size<<LOG2SECSIZE)) == NULL) {
                       return(ENOMEM);
                }
                /* save size in 256 byte "sectors" */
                ram_des_ptr->size = size;

                /* open count should be zero */
                if (ram_des_ptr->opencount++) {
                       panic("ram_open count wrong\n");
                }
        }
        return(0);
}
```

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, then 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() routine (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);
}
```

RAMdisk Strategy

This routine will actually perform the "I/O" to the RAM disc. The buf structure passed to the strategy routine contains the necessary information for the transfer. This info is filled in by kernel routines. In the case of a character device, physio() performs this task; for block devices, the file system takes care of filling in the data.

```
ram_strategy(bp)
register struct buf *bp;
{
        register block_d7;
        register char *addr;
        register struct ram_descriptor *ram_des_ptr;

        /* check if this is a status request, return the ram_device structure */
        if (RAM_MINOR(bp->b_dev) == 0) {
                if ((bp->b_flags & B_PHYS) && /* must be char (raw) device */
                        (bp->b_flags & B_READ) &&
                        (bp->b_bcount == sizeof(ram_device))) {
                        bp->b_resid = bp->b_bcount; /*normally done by bpcheck*/

                        /* return the "ram_device" structure to the caller */
                        bcopy(&ram_device[0], bp->b_un.b_addr,
                                sizeof(ram_device));
                } else {
                        bp->b_error = EIO;
                        bp->b_flags = B_ERROR;
                }
                goto done;
        }
        /* do the normal reads and writes to ram disc */
        ram_des_ptr = &ram_device[RAM_DISC(bp->b_dev)];

        /* sanity check if we got the memory */
        if ((addr = ram_des_ptr->addr) == NULL) {
                panic("no memory in ram_strategy\n");
        }
        /* make sure the request is within the domain of the "disc" */
        if (bpcheck(bp, ram_des_ptr->size, LOG2SECSIZE, 0))
                return;

        /* calculate address to do the transfer */
        addr += bp->b_un2.b_sectno<<LOG2SECSIZE;

        /* for debugging file system only */
        block_d7 = bp->b_un2.b_sectno>>2;
```

```
        if (bp->b_flags & B_READ) {
                pbcopy(addr, bp->b_un.b_addr, bp->b_bcount);
                switch (bp->b_bcount/1024) {
                case 1: ram_des_ptr->rd1k++;
                        break;
                case 2: ram_des_ptr->rd2k++;
                        break;
                case 3: ram_des_ptr->rd3k++;
                        break;
                case 4: ram_des_ptr->rd4k++;
                        break;
                case 5: ram_des_ptr->rd5k++;
                        break;
                case 6: ram_des_ptr->rd6k++;
                        break;
                case 7: ram_des_ptr->rd7k++;
                        break;
                case 8: ram_des_ptr->rd8k++;
                        break;
                default: ram_des_ptr->rdother++;
                }
        } else { /* WRITE */
                pbcopy(bp->b_un.b_addr, addr, bp->b_bcount);
                switch (bp->b_bcount/1024) {
                case 1: ram_des_ptr->wt1k++;
                        break;
                case 2: ram_des_ptr->wt2k++;
                        break;
                case 3: ram_des_ptr->wt3k++;
                        break;
                case 4: ram_des_ptr->wt4k++;
                        break;
                case 5: ram_des_ptr->wt5k++;
                        break;
                case 6: ram_des_ptr->wt6k++;
                        break;
                case 7: ram_des_ptr->wt7k++;
                        break;
                case 8: ram_des_ptr->wt8k++;
                        break;
                default: ram_des_ptr->wtother++;
                }
        }
done:
        bp->b_resid -= bp->b_bcount;
        biodone(bp);
}
```

```
/* this routine is put in here because I want it to be in the profiles */
/* bcopy could just as well be used if profiling is not used */

asm("   global  _pbcopy                 # physio enforces word alignmemt! ");
asm("_pbcopy:                           # 0 thru 256 Kbytes!!!            ");
asm("   movm.l  4(%sp),%d0/%a0-%a1      # d0 = src; a0 = dst; a1 = cnt    ");
asm("   exg     %d0,%a1                 # d0 = cnt; a1 = src              ");
asm("   subq.l  &1,%d0                  # make a counter                  ");
asm("   blt     Llpcopy4                # less or = zero?                 ");
asm("   ror.l   &2,%d0                                                    ");
asm("   bra     Llpcopy2                # move 4 bytes at a time          ");
asm("Llpcopy1:                                                            ");
asm("   mov.l   (%a1)+,(%a0)+           # move large block                ");
asm("Llpcopy2:                                                            ");
asm("   dbra    %d0,Llpcopy1                                              ");
asm("   swap    %d0                     # get remaining bytes             ");
asm("   rol.w   &2,%d0                  # position to low bits            ");
asm("Llpcopy3:                                                            ");
asm("   mov.b   (%a1)+,(%a0)+           # 1 to 4 bytes last bytes         ");
asm("   dbra    %d0,Llpcopy3                                              ");
asm("Llpcopy4:                                                            ");
asm("   rts                                                               ");
```

RAMdisk Ioctl

```
The ioctl routine:
        executed via ioctl(2);
        purpose:
           handles commands passed to it via ioctl
        implement the various ioctls by including statements of the
        following form:
           #define CMD task(t, n, arg)
        where:
           CMD  command name
           t    arbitrary letter
           n    sequential number (unique for each ioctl define for a
                given ioctl routine)
           arg  optional arg for command
        "task" is one of the following (task is a macro defined in sys/ioctl.h
           _IO    no arg
           _IOR   user reads info from the driver into arg
           _IOW   user writes info to driver from data in (or pointed to by)
                  arg
           _IOWR  both _IOR and _IOW

There are two ioctl's defined for the RAM disc driver.  They are as follows:


/* ioctl to deallocate ram volume */
#define RAM_DEALLOCATE   _IOW(R, 1, int)

/* ioctl to reset the access counter to ram volume */
#define RAM_RESETCOUNTS _IOW(R, 2, int)
```

```
ram_ioctl(dev, cmd, addr, flag)
dev_t dev;
int cmd;
caddr_t addr;
int flag;
{
        register struct ram_descriptor *ram_des_ptr;
        register volume;

        /* check if dev is the status dev */
        if (RAM_MINOR(dev) != 0)
                return(EIO);

        /* check if 0 - 15 disc volume */
        volume = *(int *)addr;
        if ((volume % RAM_MAXVOLS) != volume)
                return(EIO);

        /* calculate which ram volume it is */
        ram_des_ptr = &ram_device[volume];

        /* if not allocated, then return error */
        if (ram_des_ptr->addr == NULL) {
                return(ENOMEM);
        }
        switch(cmd) {

        /* mark for memory release on last close */
        case RAM_DEALLOCATE:
                ram_des_ptr->flag = RAM_RETURN;
                break;

        /* clear out access counts */
        case RAM_RESETCOUNTS:
                ram_des_ptr->rd8k =     0;
                ram_des_ptr->rd7k =     0;
                ram_des_ptr->rd6k =     0;
                ram_des_ptr->rd5k =     0;
                ram_des_ptr->rd4k =     0;
                ram_des_ptr->rd3k =     0;
                ram_des_ptr->rd2k =     0;
                ram_des_ptr->rd1k =     0;
                ram_des_ptr->rdother =  0;
                ram_des_ptr->wt8k =     0;
                ram_des_ptr->wt7k =     0;
                ram_des_ptr->wt6k =     0;
                ram_des_ptr->wt5k =     0;
                ram_des_ptr->wt4k =     0;
                ram_des_ptr->wt3k =     0;
                ram_des_ptr->wt2k =     0;
                ram_des_ptr->wt1k =     0;
                ram_des_ptr->wtother =  0;
                break;
        default:
                return(EIO);
```

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

RAMdisk Close

The close routine may typically perform some driver specific operations. It may flush buffers if the device supports asyncronous I/O (e.g. tty driver). It will usually decrement an "open" counter and may release I/O buffers, etc. on close.

The RAM disk driver just decrements an open count and will release memory on last close if the RAM_RETURN flag has previously been set by an ioctl call.

```
#define RAM_RETURN 1

struct ram_descriptor {
        char    *addr;
        int     size;
        short   opencount;
        short   flag;
        int     rd1k;
                :
                :
} ram_device[RAM_MAXVOLS];

ram_close(dev)
dev_t dev;
{
        register struct ram_descriptor *ram_des_ptr;
        register i;

        /* check if this is status close */
        if (RAM_MINOR(dev) != 0) {
                ram_des_ptr = &ram_device[RAM_DISC(dev)];

                if (--ram_des_ptr->opencount < 0)
                        panic("ram_close count less than zero\n");
        }

        /* free all ram volumes with flag set and open count = 0 */
        /* RAM_RETURN flag is set by an ioctl call                */

        ram_des_ptr = &ram_device[0];
        for (i = 0; i < RAM_MAXVOLS; i++, ram_des_ptr++) {
                if ((ram_des_ptr->flag & RAM_RETURN) == 0)
                        continue;
                if (ram_des_ptr->opencount != 0)
                        continue;
                /* release the system memory */
                sys_memfree(ram_des_ptr->addr, ram_des_ptr->size<<LOG2SECSIZE);

                /* zero the whole entry */
                bzero((char *)ram_des_ptr, sizeof(struct ram_descriptor));
        }
}
```

```
/* HPUX_ID: @(#)ram.h   49.1     87/08/21  */

#include <sys/ioctl.h>

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

/* ioctl to deallocate ram volume */
#define RAM_DEALLOCATE  _IOW(R, 1, int)

/* ioctl to reset the access counter to ram volume */
#define RAM_RESETCOUNTS _IOW(R, 2, int)

/* 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   /* (256 bytes) "sector" size (log2) of the ram discs */

#define RAM_RETURN 1

struct ram_descriptor {
        char    *addr;
        int     size;
        short   opencount;
        short   flag;
        int     rd1k;
        int     rd2k;
        int     rd3k;
        int     rd4k;
        int     rd5k;
        int     rd6k;
        int     rd7k;
        int     rd8k;
        int     rdother;
        int     wt1k;
        int     wt2k;
        int     wt3k;
        int     wt4k;
        int     wt5k;
        int     wt6k;
        int     wt7k;
        int     wt8k;
        int     wtother;
} ram_device[RAM_MAXVOLS];
```

```
/* HPUX_ID: @(#)ram_disc.c      49.1     87/08/21  */

/****************************************************************************/
/* This driver allows you to create up to 16 "ram_disc" volumes, doing a     */
/* "mkfs" on them and then "mount"ing them as a file system.  Be careful to  */
/* not use up too much ram on the "disc".  You still must have some left for */
/* running normal processes.                                                 */
/*                           System Software Operation                       */
/*                             Fort Collins, Co 80526                        */
/*                                  Oct 14, 1986                             */
/* Note:                                                                     */
/*    There is a bug in 5.2 and earlier systems.  The "special" dev is left  */
/*    open if there is an error during a "mount" command.  This will make it */
/*    impossible to deallocate a disc volume if a "mount" error occurs.  So  */
/*    be carefull to do a "mkfs" on the disc volume before trying to mount it.*/
/*                                                                           */
/* Revision History:                                                         */
/*      11-21-86 added the status request                                    */
/*      12-09-86 changed the ramfree request                                 */
/*                                                                           */
/****************************************************************************/
/*
******   STEPS TO ADD THE RAM_DISC DRIVER TO YOUR KERNEL
*

STEP 1) Login as "root"
        # cd /etc/conf

STEP 2)  make a mod to the "/etc/master" file as follows:

* HPUX_ID: @(#)master   10.3     85/11/14
*
* The following devices are those that can be specified in the system
* description file.  The name specified must agree with the name shown,
* or with an alias.
*
* name           handle          type    mask    block   char
*
cs80           cs80            3       3FB     0       4
.
.
.
ramdisc        ram             3       FB      4       20
.
.
Note: Major number 4 for block device and 20 for char (raw) device may
        need to be different on your system.  Reflect these different
        numbers in the "mknod" command below.

STEP 3)  modify the "/etc/conf/dfile...your_favorite" with the addition of
        "ramdisc"

STEP 4) # ar -rv libmin.a ram_disc.o

STEP 5) # config dfile...your_favorite
```

```
STEP 6) # make -f config.mk

STEP 7) # mv /hp-ux /SYSBCKUP
        # mv ./hp-ux /hp-ux

STEP 8) # reboot
        and login as "root"

STEP 9) # /etc/mknod /dev/ram  b  4 0xVSSSSS            (block device)
        # /etc/mknod /dev/rram c 20 0xVSSSSS            (char device)
                Where V = volume number 0 - F           (0 - 15)
                Where SSSSS = number of 256 byte sectors in volume (in hex).
        I.E.
        # /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/ram2M     b  4 0x202000       (block 2Mb ram volume)
        # /etc/mknod /dev/rram2M    c 20 0x202000       (char  2Mb ram volume)

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

        # /etc/mknod /dev/ramAM     b  4 0xA0A000       (block 10Mb ram volume)
        # /etc/mknod /dev/rramAM    c 20 0xA0A000       (char  10Mb ram volume)
        (Note: I don't know if this works yet - don't have this much mem)

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/ram2M   2048          (make file system for 2Mb volume)
        # mkfs /dev/ram4M   4096          (make file system for 4Mb volume)

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

        # mkdir /ram1M
        # mount /dev/ram1M /ram1M                       (mount 1Mb ram volume)

        # mkdir /ram2M
        # mount /dev/ram2M /ram2M                       (mount 2Mb ram volume)

        # mkdir /ram4M
        # mount /dev/ram4M /ram4M                       (mount 4Mb ram volume)

STEP 12) To unmount volume
        # umount /dev/ram1M

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

```
                -or-  if you use the above /dev/ram convention.
        # ramstat -d 1

 To get a status of all  memory volumes
        # ramstat /dev/ram
                -or-
        # ramstat

 To reset the access counters of a memory volume # 1.
        # ramstat -r 1 /dev/ram
                -or-
        # ramstat -r1
*****************************************************************************/

#ifdef KERNEL
#include "../h/param.h"
#include "../h/errno.h"
#include "../h/buf.h"
#include "../s200io/ram.h"
#else
#include <sys/param.h>
#include <sys/errno.h>
unsigned minphys();     /* XXX needed only with user version of buf.h */
#include <sys/buf.h>
#include "ram.h"
#endif

/*
** Open the ram device.
*/
ram_open(dev, flag)
dev_t dev;
int flag;
{
        register unsigned long size;
        register struct ram_descriptor *ram_des_ptr;

        /* check if status open */
        if (RAM_MINOR(dev) == 0)
                return(0);

        /* check if greater than max number of volumes */
        if ((size = RAM_DISC(dev)) > RAM_MAXVOLS)
                return(EINVAL);

        ram_des_ptr = &ram_device[size];

        /* check the size of the ram disc less than 16 sectors */
        if ((size = RAM_SIZE(dev)) < 16)
                return(EINVAL);

        /* check if already allocated */
        if (ram_des_ptr->addr != NULL) {

                /* then check if size changed */
                if (ram_des_ptr->size != size)
```

```
                                return(EINVAL);

                        /* bump open count */
                        ram_des_ptr->opencount++;
                } else {
                        /* allocate the memory for the ram disc */
                        if ((ram_des_ptr->addr =
                                (char *)sys_memall(size<<LOG2SECSIZE)) == NULL) {
                                return(ENOMEM);
                        }
                        /* save size in 256 byte "sectors" */
                        ram_des_ptr->size = size;

                        /* open count should be zero */
                        if (ram_des_ptr->opencount++) {
                                panic("ram_open count wrong\n");
                        }
                }
                return(0);
}

ram_close(dev)
dev_t dev;
{
        register struct ram_descriptor *ram_des_ptr;
        register i;

        /* check if status open */
        if (RAM_MINOR(dev) != 0) {
                ram_des_ptr = &ram_device[RAM_DISC(dev)];

                if (--ram_des_ptr->opencount < 0)
                        panic("ram_close count less than zero\n");
        }
/*
   NOTE: 5.2 9000/300 and earlier systems may have a bug that the memory
   cannot be released if there was ever a "mount" error because the open
   count will never reach zero -- so be carefull to do a "mkfs" before
   a "mount".
*/
        /* free all ram volumes with flag set and open count = 0 */
        ram_des_ptr = &ram_device[0];
        for (i = 0; i < RAM_MAXVOLS; i++, ram_des_ptr++) {
                if ((ram_des_ptr->flag & RAM_RETURN) == 0)
                        continue;
                if (ram_des_ptr->opencount != 0)
                        continue;
                /* release the system memory */
                sys_memfree(ram_des_ptr->addr, ram_des_ptr->size<<LOG2SECSIZE);

                /* zero the whole entry */
                bzero((char *)ram_des_ptr, sizeof(struct ram_descriptor));
        }
}

ram_strategy(bp)
```

```
register struct buf *bp;
{
        register block_d7;
        register char *addr;
        register struct ram_descriptor *ram_des_ptr;

        /* check if status request, return the ram_device structure */
        if (RAM_MINOR(bp->b_dev) == 0) {
                if ((bp->b_flags & B_PHYS) && /* must be char (raw) device */
                        (bp->b_flags & B_READ) &&
                        (bp->b_bcount == sizeof(ram_device))) {
                        bp->b_resid = bp->b_bcount; /*normally done by bpcheck*/

                        /* return the "ram_device" structure to the caller */
                        bcopy(&ram_device[0], bp->b_un.b_addr,
                                sizeof(ram_device));
                } else {
                        bp->b_error = EIO;
                        bp->b_flags = B_ERROR;
                }
                goto done;
        }
        /* do the normal reads and writes to ram disc */
        ram_des_ptr = &ram_device[RAM_DISC(bp->b_dev)];

        /* sanity check if we got the memory */
        if ((addr = ram_des_ptr->addr) == NULL) {
                panic("no memory in ram_strategy\n");
        }
        /* make sure the request is within the domain of the "disc" */
        if (bpcheck(bp, ram_des_ptr->size, LOG2SECSIZE, 0))
                return;

        /* calculate address to do the transfer */
        addr += bp->b_un2.b_sectno<<LOG2SECSIZE;

        /* for debugging file system only */
        block_d7 = bp->b_un2.b_sectno>>2;

        if (bp->b_flags & B_READ) {
                pbcopy(addr, bp->b_un.b_addr, bp->b_bcount);
                switch (bp->b_bcount/1024) {
                case 1: ram_des_ptr->rd1k++;
                        break;
                case 2: ram_des_ptr->rd2k++;
                        break;
                case 3: ram_des_ptr->rd3k++;
                        break;
                case 4: ram_des_ptr->rd4k++;
                        break;
                case 5: ram_des_ptr->rd5k++;
                        break;
                case 6: ram_des_ptr->rd6k++;
                        break;
                case 7: ram_des_ptr->rd7k++;
                        break;
```

```
                case 8: ram_des_ptr->rd8k++;
                        break;
                default: ram_des_ptr->rdother++;
                }
        } else { /* WRITE */
                pbcopy(bp->b_un.b_addr, addr, bp->b_bcount);
                switch (bp->b_bcount/1024) {
                case 1: ram_des_ptr->wt1k++;
                        break;
                case 2: ram_des_ptr->wt2k++;
                        break;
                case 3: ram_des_ptr->wt3k++;
                        break;
                case 4: ram_des_ptr->wt4k++;
                        break;
                case 5: ram_des_ptr->wt5k++;
                        break;
                case 6: ram_des_ptr->wt6k++;
                        break;
                case 7: ram_des_ptr->wt7k++;
                        break;
                case 8: ram_des_ptr->wt8k++;
                        break;
                default: ram_des_ptr->wtother++;
                }
        }
done:
        bp->b_resid -= bp->b_bcount;
        biodone(bp);
}

/* this routine is put in here because I want it to be in the profiles */
/* bcopy could just as well be used if profiling is not used */

asm("   global   _pbcopy                # physio enforces word alignmemt! ");
asm("_pbcopy:                           # 0 thru 256 Kbytes!!!           ");
asm("   movm.l   4(%sp),%d0/%a0-%a1     # d0 = src; a0 = dst; a1 = cnt   ");
asm("   exg      %d0,%a1                # d0 = cnt; a1 = src             ");
asm("   subq.l   &1,%d0                 # make a counter                 ");
asm("   blt      Llpcopy4               # less or = zero?                ");
asm("   ror.l    &2,%d0                                                  ");
asm("   bra      Llpcopy2               # move 4 bytes at a time         ");
asm("Llpcopy1:                                                           ");
asm("   mov.l    (%a1)+,(%a0)+          # move large block               ");
asm("Llpcopy2:                                                           ");
asm("   dbra     %d0,Llpcopy1                                            ");
asm("   swap     %d0                    # get remaining bytes            ");
asm("   rol.w    &2,%d0                 # position to low bits           ");
asm("Llpcopy3:                                                           ");
asm("   mov.b    (%a1)+,(%a0)+          # 1 to 4 bytes last bytes        ");
asm("   dbra     %d0,Llpcopy3                                            ");
asm("Llpcopy4:                                                           ");
asm("   rts                                                              ");

ramread(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);
}

ram_ioctl(dev, cmd, addr, flag)
dev_t dev;
int cmd;
caddr_t addr;
int flag;
{
        register struct ram_descriptor *ram_des_ptr;
        register volume;

        /* check if dev is the status dev */
        if (RAM_MINOR(dev) != 0)
                return(EIO);

        /* check if 0 - 15 disc volume */
        volume = *(int *)addr;
        if ((volume % RAM_MAXVOLS) != volume)
                return(EIO);

        /* calculate which ram volume it is */
        ram_des_ptr = &ram_device[volume];

        /* if not allocated, then return error */
        if (ram_des_ptr->addr == NULL) {
                return(ENOMEM);
        }
        switch(cmd) {

        /* mark for memory release on last close */
        case RAM_DEALLOCATE:
                ram_des_ptr->flag = RAM_RETURN;
                break;

        /* clear out access counts */
        case RAM_RESETCOUNTS:
                ram_des_ptr->rd8k =     0;
                ram_des_ptr->rd7k =     0;
                ram_des_ptr->rd6k =     0;
                ram_des_ptr->rd5k =     0;
                ram_des_ptr->rd4k =     0;
                ram_des_ptr->rd3k =     0;
                ram_des_ptr->rd2k =     0;
                ram_des_ptr->rd1k =     0;
                ram_des_ptr->rdother =  0;
                ram_des_ptr->wt8k =     0;
```

```
                ram_des_ptr->wt7k =     0;
                ram_des_ptr->wt6k =     0;
                ram_des_ptr->wt5k =     0;
                ram_des_ptr->wt4k =     0;
                ram_des_ptr->wt3k =     0;
                ram_des_ptr->wt2k =     0;
                ram_des_ptr->wt1k =     0;
                ram_des_ptr->wtother =  0;
                break;
        default:
                return(EIO);
        }
        return(0);
}
```

Read System Call

The read(2) system call is a very short assembly language stub. It puts into register d0 what the system call is (3 for read) and performs a trap 0. Upon return, it checks what the status is and jumps to an error routine if a -1 is returned.

example call:

```
        read(fd,buff,10);


        # KLEENIX_ID @(#)read.s 49.1 86/12/18
        # C library -- read

        # nread = read(file, buffer, count);
        # nread ==0 means eof; nread == -1 means error

                set     READ,3
                global  _read
                global  __cerror

        _read:
                ifdef(`PROFILE',`
                mov.l   &p_read,%a0
                jsr     mcount
                ')
                movq    &READ,%d0
                trap    &0
                bcc.b   noerror
                jmp     __cerror
        noerror:
                rts

        ifdef(`PROFILE',`
                        data
        p_read: long    0
                ')
```

Xsyscall (kernel)

Xsyscall is the code executed (in the kernel) due to receiving a trap 0. It saves the registers and the pointer to the user's stack onto the kernel stack. Then we jump to the syscall (C) routine , the system call "gateway" routine. Upon return from the system call, we restore the user's stack pointer and other register values from kernel stack, and return from execption.

```
# HPUX_ID: @(#)locore.s 49.3            87/10/01
```



```
        global  _xsyscall,_syscall

_xsyscall:
        movm.l  %d0-%d7/%a0-%a7,-(%sp)  #save all 16 registers
        mov.l   %usp,%a0
        mov.l   %a0,60(%sp)             #save usr stack ptr
                                        # pass exception frame
        jsr     _syscall                #C handler for syscalls
#       fall into xreturn code

xreturn:
        mov.l   60(%sp),%a0
        mov.l   %a0,%usp                        #restore usr stack ptr
        movm.l  (%sp)+,%d0-%d7/%a0-%a6  #restore all other registers
        addq.l  &4,%sp                  # and pop off sp
        addq.l  &6,%sp                  #sp and alignment word
        bclr    &POP_STACK_BIT,_u+PCB_FLAGS
        bne.b   xreturn1
        rte
```

Syscall Kernel Routine

The syscall routine is passed the address of the execption stack, the values in the user's registers at the time of the trap. The syscall routine removes from the stack the system call number (3 for a read). This number is used in a table lookup to determine what system routine to call and how many parameters were put onto the user's stack. These parameters are then copied from the user's stack into the process's u_area. After setting up the u_area with the system call information, we call the routine pointed to by the system call number (the kernel read routine in this case).

Upon return from the read routine, syscall checks what the error value is in the u_area.

- If there was an error, the error value is put into register d0 (on the exception stack).
- If the read routine successfully completed (no interrupt), the return value in the u_area is put into register d0 (on the exception stack).
- If the call was interrupted (for any reason) and the system call is set up for RESTART, then the PC in the exception stack is backed up two instructions (back to the trap 0 statement in read(2)).

We then update the u_area, and check the "runrun" flag to see if another process has a higher priority than we have. If there is, we let the system switch to that process. If not, then we return to xsyscall, then back to "userland".

Read Kernel Routine

The read kernel routine sets up a "uap" structure that will contain the information necessary for the I/O. It contains the file descriptor, address of the user's buffer, and count (retrieved from the u_area). It takes this information and puts it into an iovec structure, containing the buffer location and count. The iovec struct is placed into a "uio" structure along with the number of iovec structures (1 for a read, greater than 1 for readv). Read then calls rwuio() with the "uio" structure and a flag indicating "read".

```
/* HPUX_ID: @(#)sys_gen.c       49.1            87/08/21 */
```



```
/* Read system call.  */
read()
{
        register struct a {
                int     fdes;
                char    *cbuf;
                unsigned count;
        } *uap = (struct a *)u.u_ap;
        struct uio auio;
        struct iovec aiov;

        aiov.iov_base = (caddr_t)uap->cbuf;
        aiov.iov_len = uap->count;
        auio.uio_iov = &aiov;
        auio.uio_iovcnt = 1;
        rwuio(&auio, UIO_READ);
```

Rwuio Kernel Routine

The rwuio routine determines from the file descripter what file we are dealing with. It ensures that we have permission to execute the request (we have "read" permission on the file). It sets up some of the uio fields (e.g. residual count = 0) and ensures that the iovectors are valid (non-negative). We determine the total number of bytes requested (total of each iovec count) and set the uio offset to the present file pointer offset. Then we will call the routine "~~ufs_rdwr~~" via a pointer to the routine in the file pointer structure (the routine is filled in by the open system call). Upon return we update the return value in the u_area (bytes transfered) and the file pointer offset.

vno_rw

```
/* HPUX_ID: @(#)sys_gen.c       49.1            87/08/21 */

/*
(c) Copyright 1983, 1984, 1985, 1986, 1987 Hewlett-Packard Company.
(c) Copyright 1979 The Regents of the University of Colorado, a body corporate
(c) Copyright 1979, 1980, 1983 The Regents of the University of California
(c) Copyright 1980, 1984 AT&T Technologies.  All Rights Reserved.
The contents of this software are proprietary and confidential to the Hewlett-
Packard Company, and are limited in distribution to those with a direct need
to know.  Individuals having access to this software are responsible for main-
taining the confidentiality of the content and for keeping the software secure
when not in use.  Transfer to any party is strictly forbidden other than as
expressly permitted in writing by Hewlett-Packard Company.  Unauthorized trans-
fer to or possession by any unauthorized party may be a criminal offense.

                    RESTRICTED RIGHTS LEGEND

          Use,  duplication,  or disclosure by the Government  is
          subject to restrictions as set forth in subdivision (b)
          (3)  (ii)  of the Rights in Technical Data and Computer
          Software clause at 52.227-7013.

                     HEWLETT-PACKARD COMPANY
                        3000 Hanover St.
                      Palo Alto, CA  94304
*/

rwuio(uio, rw)
        register struct uio *uio;
        enum uio_rw rw;
{
        struct a {
                int     fdes;
        };
        register struct file *fp;
        register struct iovec *iov;
        int i, count;
```

```
        GETF(fp, ((struct a *)u.u_ap)->fdes);
        if ((fp->f_flag&(rw==UIO_READ ? FREAD : FWRITE)) == 0) {
                u.u_error = EBADF;
                return;
        }
        uio->uio_resid = 0;
        uio->uio_segflg = 0;
        iov = uio->uio_iov;
        for (i = 0; i < uio->uio_iovcnt; i++) {
                if (iov->iov_len < 0) {
                        u.u_error = EINVAL;
                        return;
                }
                uio->uio_resid += iov->iov_len;
                if (uio->uio_resid < 0) {
                        u.u_error = EINVAL;
                        return;
                }
                iov++;
        }
        count = uio->uio_resid;
        uio->uio_offset = fp->f_offset;
        if ((u.u_procp->p_flag&SOUSIG) == 0 && setjmp(&u.u_qsave)) {
                if (uio->uio_resid == count)
                        u.u_eosys = RESTARTSYS;
        } else
                u.u_error = (*fp->f_ops->fo_rw)(fp, rw, uio);
        u.u_r.r_val1 = count - uio->uio_resid;
        fp->f_offset += u.u_r.r_val1;
        u.u_ru.ru_ioch += u.u_r.r_val1; /* for System V accounting */
}
```

Vno_rw Kernel Routine

The routine vno_rw is the vnode layer read/write routine. It sets up some values from the uio and file pointer structures and calls VOP_RDWR, a macro routine which calls the proper vnode operation routine (in this case ufs_rdwr).

```
/* HPUX_ID: @(#)vfs_io.c        49.1            87/08/21 */

/*
(c) Copyright 1983, 1984, 1985, 1986, 1987 Hewlett-Packard Company.
(c) Copyright 1979 The Regents of the University of Colorado, a body corporate
(c) Copyright 1979, 1980, 1983 The Regents of the University of California
(c) Copyright 1980, 1984 AT&T Technologies.  All Rights Reserved.
The contents of this software are proprietary and confidential to the Hewlett-
Packard Company, and are limited in distribution to those with a direct need
to know.  Individuals having access to this software are responsible for main-
taining the confidentiality of the content and for keeping the software secure
when not in use.  Transfer to any party is strictly forbidden other than as
expressly permitted in writing by Hewlett-Packard Company.  Unauthorized trans-
fer to or possession by any unauthorized party may be a criminal offense.

                    RESTRICTED RIGHTS LEGEND

          Use,  duplication,  or disclosure by the Government  is
          subject to restrictions as set forth in subdivision (b)
          (3)  (ii)  of the Rights in Technical Data and Computer
          Software clause at 52.227-7013.

                     HEWLETT-PACKARD COMPANY
                        3000 Hanover St.
                      Palo Alto, CA  94304
*/

int
vno_rw(fp, rw, uiop)
        struct file *fp;
        enum uio_rw rw;
        struct uio *uiop;
{
        register struct vnode *vp;
        register int count;
        register int error;

        vp = (struct vnode *)fp->f_data;
        /*
         * Ir write make sure filesystem is writable
         */
        if ((rw == UIO_WRITE) && (vp->v_vfsp->vfs_flag & VFS_RDONLY))
                return(EROFS);
        count = uiop->uio_resid;
        if (vp->v_type == VREG) {
                error =
```

```
                VOP_RDWR(vp, uiop, rw,
                    ((fp->f_flag & FAPPEND) != 0?
                        IO_APPEND|IO_UNIT: IO_UNIT), fp->f_cred);
```

```
        } else {
                error =
                    VOP_RDWR(vp, uiop, rw,
                        ((fp->f_flag & FAPPEND) != 0?
                            IO_APPEND: 0), fp->f_cred);
        }
        if (error)
                return(error);
        if (fp->f_flag & FAPPEND) {
                /*
                 * The actual offset used for append is set by VOP_RDWR
                 * so compute actual starting location
                 */
                fp->f_offset = uiop->uio_offset - (count - uiop->uio_resid);
        }
        return(0);
}
```

Ufs_rdwr Kernel Routine

The ufs_rdwr routine is the read/write vnode operation routine. It is a short routine that converts the vnode pointer into an inode pointer, and (for a device file), calls the rwip routine (we are getting close to the driver!)

```
/* HPUX_ID: @(#)ufs_vnops.c     49.7            87/10/16 */

/*
(c) Copyright 1983, 1984, 1985, 1986, 1987 Hewlett-Packard Company.
(c) Copyright 1979 The Regents of the University of Colorado, a body corporate
(c) Copyright 1979, 1980, 1983 The Regents of the University of California
(c) Copyright 1980, 1984 AT&T Technologies.  All Rights Reserved.
The contents of this software are proprietary and confidential to the Hewlett-
Packard Company, and are limited in distribution to those with a direct need
to know.  Individuals having access to this software are responsible for main-
taining the confidentiality of the content and for keeping the software secure
when not in use.  Transfer to any party is strictly forbidden other than as
expressly permitted in writing by Hewlett-Packard Company.  Unauthorized trans-
fer to or possession by any unauthorized party may be a criminal offense.

                    RESTRICTED RIGHTS LEGEND

          Use,  duplication,  or disclosure by the Government  is
          subject to restrictions as set forth in subdivision (b)
          (3)  (ii)  of the Rights in Technical Data and Computer
          Software clause at 52.227-7013.

                      HEWLETT-PACKARD COMPANY
                         3000 Hanover St.
                       Palo Alto, CA  94304
*/

/* read or write a vnode */
int
ufs_rdwr(vp, uiop, rw, ioflag, cred)
        struct vnode *vp;
        struct uio *uiop;
        enum uio_rw rw;
        int ioflag;
        struct ucred *cred;
{
        register struct inode *ip;
        int error;
        int type;

        ip = VTOI(vp);
        type = ip->i_mode&IFMT;
```

```
        if (type == IFREG || type == IFIFO || type == IFNWK) {
                /* don't have file pointer */
                ILOCK(ip);
                if ((ioflag & IO_APPEND) && (rw == UIO_WRITE)) {
                        /*
                         * in append mode start at end of file.
                         */
                        uiop->uio_offset = ip->i_size;
                }
                error = rwip(ip, uiop, rw, ioflag);
                IUNLOCK(ip);
        } else {
                error = rwip(ip, uiop, rw, ioflag);
        }
        return (error);
}
```

Rwip Kernel Routine

The rwip routine "distributes" the read/write request based on the type of file. We will update the access time in the inode since this is a read routine (if we perform a write, we update the "update" and "change" values in the inode). Since this is a character device (type = IFCHAR), the routine determines what the major number is (21 for gpio) from the dev number (stored in the inode structure). We then perform a jump to the "gpio.read" (actually hpib.read) through the cdev_sw[] table. Now we are off and running to the driver!!

```
/* HPUX_ID: @(#)ufs_vnops.c     49.7            87/10/16 */

/*
(c) Copyright 1983, 1984, 1985, 1986, 1987 Hewlett-Packard Company.
(c) Copyright 1979 The Regents of the University of Colorado, a body corporate
(c) Copyright 1979, 1980, 1983 The Regents of the University of California
(c) Copyright 1980, 1984 AT&T Technologies.  All Rights Reserved.
The contents of this software are proprietary and confidential to the Hewlett-
Packard Company, and are limited in distribution to those with a direct need
to know.  Individuals having access to this software are responsible for main-
taining the confidentiality of the content and for keeping the software secure
when not in use.  Transfer to any party is strictly forbidden other than as
expressly permitted in writing by Hewlett-Packard Company.  Unauthorized trans-
fer to or possession by any unauthorized party may be a criminal offense.

                      RESTRICTED RIGHTS LEGEND

          Use,  duplication,  or disclosure by the Government  is
          subject to restrictions as set forth in subdivision (b)
          (3)  (ii)  of the Rights in Technical Data and Computer
          Software clause at 52.227-7013.

                       HEWLETT-PACKARD COMPANY
                          3000 Hanover St.
                        Palo Alto, CA  94304
*/

int
rwip(ip, uio, rw, ioflag)
        register struct inode *ip;
        register struct uio *uio;
        enum uio_rw rw;
        int ioflag;
{
        dev_t dev;
        struct vnode *devvp;
        struct buf *bp;
        struct fs *fs;
        daddr_t lbn, bn;
        register int n, on, type;
        int size;
        long bsize;
        extern int mem_no;
```

```
        extern int ieee802_no;
        extern int ethernet_no;
        int error = 0;
        int total;
        int syncio_flag;
        int dirsz = 0;

        dev = (dev_t)ip->i_rdev;
        if (rw != UIO_READ && rw != UIO_WRITE)
                panic("rwip");
        if (rw == UIO_READ && uio->uio_resid == 0)
                return (0);
        type = ip->i_mode&IFMT;
        /* uio_offset can go negative for software drivers that open, read,
           or write continuously, and never close */
        if ((uio->uio_offset < 0) || (uio->uio_offset + uio->uio_resid) < 0) {
                if (type != IFCHR)
                        return(EINVAL);
                else {
                    if (major(dev) == ethernet_no || major(dev) == ieee802_no)
                                /* kludge!  how do we set f_offset to 0 ??? */
                                uio->uio_offset = 0;
                    else {
                                if (major(dev) != mem_no)
                                        return(EINVAL);
                    }
                }
        }
        /* If the inode is remote, call the appropriate routine.  Note,
         * We could completely separate out all DUX code from this
         * routine by having a separate vnode entry, but it would mean
         * duplicating all the preliminary tests.
         */
        if ((type != IFCHR && type != IFBLK) && remoteip(ip))
                return (dux_rwip(ip, uio, rw, ioflag));
        if (rw == UIO_READ)
                imark(ip, IACC);
        switch (type) {
        case IFIFO:
                if (rw == UIO_READ)
                        error = fifo_read(ip, uio);
                else
                        error = fifo_write(ip, uio);
                return(error);
                break;
        case IFCHR:
                if (rw == UIO_READ) {
                        error = (*cdevsw[major(dev)].d_read)(dev, uio);
                } else {
                        imark(ip, IUPD|ICHG);
                        error = (*cdevsw[major(dev)].d_write)(dev, uio);
                }
                return (error);
                break;

        case IFBLK:
```

```
        case IFREG:
        case IFDIR:
        case IFLNK:
        case IFNWK:
                :
                :
}
```

Hpib_read Driver Routine

This routine is much like the RAMdisk read routine. The main difference is that we use a dil buffer instead of requesting one from the file system. This buffer is located off the user's u_area, and was acquired when the user opened the device. We return to "rwip" the return status of physio().

```
/* HPUX_ID: @(#)dil_hpib.c      49.3            87/10/14 */

/*
(c) Copyright 1983, 1984, 1985, 1986, 1987 Hewlett-Packard Company.
(c) Copyright 1979 The Regents of the University of Colorado, a body corporate
(c) Copyright 1979, 1980, 1983 The Regents of the University of California
(c) Copyright 1980, 1984 AT&T Technologies.  All Rights Reserved.
The contents of this software are proprietary and confidential to the Hewlett-
Packard Company, and are limited in distribution to those with a direct need
to know.  Individuals having access to this software are responsible for main-
taining the confidentiality of the content and for keeping the software secure
when not in use.  Transfer to any party is strictly forbidden other than as
expressly permitted in writing by Hewlett-Packard Company.  Unauthorized trans-
fer to or possession by any unauthorized party may be a criminal offense.

                    RESTRICTED RIGHTS LEGEND

          Use,  duplication,  or disclosure by the Government  is
          subject to restrictions as set forth in subdivision (b)
          (3)  (ii)  of the Rights in Technical Data and Computer
          Software clause at 52.227-7013.

                     HEWLETT-PACKARD COMPANY
                        3000 Hanover St.
                      Palo Alto, CA  94304
*/

hpib_read(dev, uio)
dev_t dev;
struct uio *uio;
{
        register struct buf *bp;
        register struct dil_info *info;

        bp = (struct buf *)u.u_fp->f_buf;       /* get buffer */
        info = (struct dil_info *) bp->dil_packet;
        info->dil_procp = u.u_procp;

        return(physio(hpib_strategy, bp, dev, B_READ,minphys,uio));
}
```

Physio Kernel Routine

Back in the kernel again! The physio routine takes each vector from iovec (buffer ptr/count) and breaks the routine into "mincnt" size chunks (64k if mincnt is minphys). It converts the user's addresses into physical memory addresses, and locks down those pages. Then it will call the proper strategy routine (in this case hpib_strategy) via the physstrat routine. When physio is finished with a transfer, it releases the pages that were locked. Physio also keeps track of how many bytes have actually been transfered, updating the residual as it runs through the transfer.

While physio is using the buffer "bp", it marks the buffer "BUSY" so no one else will try to use it. When physio finishes with the buffer, it marks the buffer "unBUSY" and checks if someone wanted it (B_WANTED bit set). If so, physio calls wakeup to wake up all processes sleeping on the buffer (note: this should not happen for this read call).

```
/* HPUX_ID: @(#)vm_swp.c        49.1            87/08/21 */

/*
(c) Copyright 1983, 1984, 1985, 1986, 1987 Hewlett-Packard Company.
(c) Copyright 1979 The Regents of the University of Colorado, a body corporate
(c) Copyright 1979, 1980, 1983 The Regents of the University of California
(c) Copyright 1980, 1984 AT&T Technologies.  All Rights Reserved.
The contents of this software are proprietary and confidential to the Hewlett-
Packard Company, and are limited in distribution to those with a direct need
to know.  Individuals having access to this software are responsible for main-
taining the confidentiality of the content and for keeping the software secure
when not in use.  Transfer to any party is strictly forbidden other than as
expressly permitted in writing by Hewlett-Packard Company.  Unauthorized trans-
fer to or possession by any unauthorized party may be a criminal offense.

                    RESTRICTED RIGHTS LEGEND

          Use,  duplication,  or disclosure by the Government  is
          subject to restrictions as set forth in subdivision (b)
          (3)  (ii)  of the Rights in Technical Data and Computer
          Software clause at 52.227-7013.

                     HEWLETT-PACKARD COMPANY
                        3000 Hanover St.
                      Palo Alto, CA  94304
*/

/*
 * Raw I/O. The arguments are
 *      The strategy routine for the device
 *      A buffer, which will always be a special buffer
 *        header owned exclusively by the device for this purpose
 *      The device number
 *      Read/write flag
 * Essentially all the work is computing physical addresses and
 * validating them.
 * If the user has the proper access privilidges, the process is
 * marked 'delayed unlock' and the pages involved in the I/O are
```

```
aulted and locked. After the completion of the I/O, the above pages
are unlocked.
*/
```

```
physio(strat, bp, dev, rw, mincnt, uio)
        int (*strat)();
        register struct buf *bp;
        dev_t dev;
        int rw;
        unsigned (*mincnt)();
        struct uio *uio;
{
        register struct iovec *iov;
        register int ccount, npf;
        char *base;
        int s, error = 0;
        register long *upte, *kpte;
        int aa, i;
        struct buf *save_bp = bp;

        /* if caller did not have buf >> allocate one for him */
        if (bp == NULL) {
                s = spl6();
                while (bswlist.av_forw == NULL) {
                        bswlist.b_flags |= B_WANTED;
                        sleep((caddr_t)&bswlist, PRIBIO+1);
                }
                bp = bswlist.av_forw;
                bswlist.av_forw = bp->av_forw;
                splx(s);
                bp->b_flags = 0;
        }

nextiov:
        iov = uio->uio_iov;
        if (uio->uio_iovcnt == 0) {
                error = 0;
                goto physio_exit;
        }
        /*  The uio data may be in kernel space, so don't check access if so */
        if (uio->uio_seg != UIOSEG_KERNEL &&
                useracc(iov->iov_base,(u_int)iov->iov_len,
                rw==B_READ?B_WRITE:B_READ) == NULL) {
                error = EFAULT;
                goto physio_exit;
        }
        s = spl6();
        while (bp->b_flags&B_BUSY) {
                bp->b_flags |= B_WANTED;
                sleep((caddr_t)bp, PRIBIO+1);
        }
        splx(s);
        bp->b_error = 0;
        bp->b_proc = u.u_procp;
        base = iov->iov_base;
        if (u.u_pcb.pcb_flags & MULTIPLE_MAP_MASK) {
                if (((int)base & 0xf0000000) != 0xf0000000)
                        base = (caddr_t) ((int)base & 0x0fffffff);
        }
```

```
        while (iov->iov_len > 0) {
                bp->b_flags = B_BUSY | B_PHYS | rw;
                bp->b_dev = dev;
                bp->b_blkno = btodb(uio->uio_offset);
                bp->b_offset = uio->uio_offset;
                bp->b_bcount = iov->iov_len;
                (*mincnt)(bp);
                ccount = bp->b_bcount;
        /*  If the uio data is in kernel space, don't go through the mapping */
                if(uio->uio_seg == UIOSEG_KERNEL)
                  { bp->b_un.b_addr = base;
                    goto do_physio;
                  }
                u.u_procp->p_flag |= SPHYSIO;
                vslock(base, ccount);
/*
 * Allocate kernel address space for mapping in the users buffer.
 */

/* calculate number of pages needed */
                npf = btoc(ccount + ((int)base & CLOFSET));
/* allocate kernel pte's */
                while ((aa = rmalloc(kernelmap, npf)) == 0) {
                        kmapwnt++; /* should never happen */
                        printf("oops - kernelmap should be bigger\n");
                        sleep((caddr_t)kernelmap, PRIBIO+1);
                }
/*get address of pte's for user's buffer */
                upte= (long *)vtopte(u.u_procp,btop(base));
/* calculate kernel logical address for reference thru ptes */
                bp->b_un.b_addr = (caddr_t)kmxtob(aa) + ((int)base & CLOFSET);
/* copy user's ptes into the kernel's ptes */
                for (kpte = (long *)&Sysmap[btop(bp->b_un.b_addr)], i = npf; i>0
                        *kpte = *upte++;
                        ((struct pte *)kpte)->pg_v = 1;
                        ((struct pte *)kpte)->pg_prot = PG_RW;
                }
                PURGE_TLB_SUPER;
```

```
        do_physio:
                physstrat(bp, strat, PRIBIO);
                if(uio->uio_seg != UIOSEG_KERNEL) {
/* free the kernel logical address space */
                        rmfree(kernelmap, npf, aa);
                        vsunlock(base, ccount, rw);
                        u.u_procp->p_flag &= ~SPHYSIO;
                }
                (void) spl6();
                if (bp->b_flags&B_WANTED)
                        wakeup((caddr_t)bp);
                splx(s);
                ccount -= bp->b_resid;
                base += ccount;
                iov->iov_len -= ccount;
                uio->uio_resid -= ccount;
                uio->uio_offset += ccount;
                /* temp kludge for tape drives */
                if (bp->b_resid || (bp->b_flags&B_ERROR))
                        break;
        }
        PURGE_DCACHE;
        bp->b_flags &= ~(B_BUSY|B_WANTED|B_PHYS);
        error = geterror(bp);
        /* temp kludge for tape drives */
        if (bp->b_resid || error)
                goto physio_exit;
        uio->uio_iov++;
        uio->uio_iovcnt--;
        goto nextiov;

physio_exit:
        /* if we allocated buf for caller, then deallocate it */
        if (save_bp == NULL) {
                s = spl6();
                bp->b_flags &= ~(B_BUSY|B_WANTED|B_PHYS|B_PAGET|B_UAREA|B_DIRTY)
                bp->av_forw = bswlist.av_forw;
                bswlist.av_forw = bp;
                if (bswlist.b_flags & B_WANTED) {
                        bswlist.b_flags &= ~B_WANTED;
                        wakeup((caddr_t)&bswlist);
                        wakeup((caddr_t)&proc[2]);
                }
                splx(s);
        }
        return(error);
}
```

```
#define NETMAXPHYS (8 * 1024)
#define MAXPHYS (64 * 1024)

unsigned
minphys(bp)
	struct buf *bp;
{
	if (my_site_status & CCT_SLWS) {
		if (bp->b_bcount > NETMAXPHYS)
			bp->b_bcount = NETMAXPHYS;
		return;
	}

	if (bp->b_bcount > MAXPHYS)
		bp->b_bcount = MAXPHYS;
}
```

Hpib_strategy Driver Routine

The hpib_strategy is the common DIL strategy for both the gpio and hpib drivers. It just sets up the buffer pointer (bp) for the transfer and queues up the transfer with the enqueue() routine. The enqueue() routine will just make a call to hpib_transfer() when it is queued.

```
/* HPUX_ID: @(#)dil_hpib.c      49.3            87/10/14 */

/*
(c) Copyright 1983, 1984, 1985, 1986, 1987 Hewlett-Packard Company.
(c) Copyright 1979 The Regents of the University of Colorado, a body corporate
(c) Copyright 1979, 1980, 1983 The Regents of the University of California
(c) Copyright 1980, 1984 AT&T Technologies.  All Rights Reserved.
The contents of this software are proprietary and confidential to the Hewlett-
Packard Company, and are limited in distribution to those with a direct need
to know.  Individuals having access to this software are responsible for main-
taining the confidentiality of the content and for keeping the software secure
when not in use.  Transfer to any party is strictly forbidden other than as
expressly permitted in writing by Hewlett-Packard Company.  Unauthorized trans-
fer to or possession by any unauthorized party may be a criminal offense.

                    RESTRICTED RIGHTS LEGEND

          Use,  duplication,  or disclosure by the Government  is
          subject to restrictions as set forth in subdivision (b)
          (3)  (ii)  of the Rights in Technical Data and Computer
          Software clause at 52.227-7013.

                     HEWLETT-PACKARD COMPANY
                        3000 Hanover St.
                      Palo Alto, CA  94304
*/

hpib_strategy(bp, uio)
register struct buf *bp;
struct uio *uio;
{
        register struct iobuf *iob = bp->b_queue;
        register struct isc_table_type *sc = bp->b_sc;
        register struct dil_info *info = (struct dil_info *) bp->dil_packet;

        bp->b_flags |= B_DIL; /* mark the buffer */
        bp->b_error = 0; /* clear errors */

        /* set up any buffer stuff */
        bp->b_resid = bp->b_bcount;
        iob->b_xaddr = bp->b_un.b_addr;
        iob->b_xcount = bp->b_bcount;

        info->dil_timeout_proc = hpib_transfer_timeout;
        bp->b_action = hpib_transfer;
        enqueue(iob, bp);
```

Hpib_transfer Routine

The hpib_transfer routine determines what action is to be taken. In this case, we want to do a transfer. We enter into the Finite State Machine (START_FSM is a macro with assembly code that is to ensure that only one process is in the FSM for a given select code at a time). When we are able to perform our transfer, the routine determines what type of transfer can be used. If we had asked for termination on pattern or the there is only 1 byte to transfer (two in word mode), we select MUST_INTR control. Otherwise we will select MAX_OVERLAP, which means DMA will be tried. We will then call the driver routine for transfer (in this case gpio_driver).

```
/* HPUX_ID: @(#)dil_hpib.c      49.3            87/10/14 */
```



```
enum dil_transfer_state {get_dil_sc = 0,
                         check_transfer,
                         re_get_dil_sc,
                         do_transfer,
                         end_transfer,
                         tfr_timedout,
                         tfr_defaul};

hpib_transfer(bp)
register struct buf *bp;
{
        register struct iobuf *iob = bp->b_queue;
        register struct dil_info *info = (struct dil_info *) bp->dil_packet;
        register struct isc_table_type *sc = bp->b_sc;
```

```
        register unsigned char state = 0;
        register enum transfer_request_type control;
        register int x;
```

```
        state = iob->dil_state;

        try
                START_FSM;
re_switch:
                switch ((enum dil_transfer_state)iob->b_state) {
                        :
                        :
                case do_transfer:
                        END_TIME
                        iob->b_state = (int)end_transfer;
                        DIL_START_TIME(hpib_transfer_timeout)
                        if (state & D_RAW_CHAN)
                                (*sc->iosw->iod_save_state)(sc);
                        else
                                (*sc->iosw->iod_preamb)(bp, 0);
                        END_TIME
                        /* set the speed for the process */
                        if (state & (READ_PATTERN | USE_INTR)) {
                                control = MUST_INTR;
                                sc->pattern = iob->read_pattern;
                        }
                        else if (bp->b_bcount == 1)
                                control = MUST_INTR;
                        else if (dma_here != 2)
                                control = MUST_INTR;
                        else if (state & USE_DMA)
                                control = MAX_OVERLAP;
                        else
                                control = MAX_OVERLAP;
                        /* set up transfer control info here */
                        sc->tfr_control = state;
                        DIL_START_TIME(hpib_transfer_timeout)
                        (*sc->iosw->iod_tfr)(control, bp, hpib_transfer);
                        break;
                        :
                        :
                case tfr_timedout:
                        escape(TIMED_OUT);
                default:
                        panic("bad dil transfer state");
                }
                END_FSM;
        recover {
                ABORT_TIME;
                iob->b_state = (int) tfr_defaul;
                if (escapecode == TIMED_OUT)
                        bp->b_error = EIO;
                (*sc->iosw->iod_abort_io)(bp);
                HPIB_status_clear(bp);
                iob->term_reason = TR_ABNORMAL;
                dil_drop_selcode(bp);
                queuedone(bp);
                dil_dequeue(bp);
        }
```

Gpio_driver routine

This routine determines what type of transfer to perform and then kicks of the transfer. It will select INTR_TRANSFER if:

1) 1 byte transfer
2) 2 byte transfer in word mode
3) using READ_PATTERN for termination
4) requested INTR XFER (via io)speed_ctl
5) try_dma routine failed

otherwise the transfer type is DMA_TFR

If DMA_TFR
 set transfer width to 8 or 16 bit
 calls dma_build_chain to build a chain of DMA requests

DMA_build_chain routine

The routine will set up dma channel and card for dma transfer (and selects transfer mode (8 or 16 bit).

It builds a chain of dma transactions for the transfer. These are usually 4k (1 page/chain or less), but will chain transactions for contiguous pages.

The information in the chain:
 address of i/o card
 interrupt level for DMA card (7 except for last link, then i/o cards IRL)

After building the chain, we then return to gpio_dma() routine.

Gpio_dma Routine

This routine called the dma_build_chain routine. When the chain is built, it sets up where to go on completion of dma transaction. This is gpio_do_isr(). We then call dma_start() to actually start the transfer.

Dma_start Routine

This routine sets up the first DMA transaction (first link in the chain) and returns to the driver. The driver will sleep awaiting completion of the transfers. The DMA transfer is started by writing the address of the buffer and the count into the DMA channel and arming the channel.

When the channel transfers all the bytes for that link, it generates a level 7 interrupt. This interrupt jumps to very specific code which is all in assembly. The first thing this code checks is if the interrupt was caused by one of the DMA channels. If so, then it updates the link to the next in the chain, arms the DMA channel, and returns. If it is the last link in the chain, then it sets up interrupt level to that of the card, so that following the last link, we enter the gpio card's isr routine.

Gpio_do_isr Routine

This routine will transfer the last byte (on read) from the gpio card. It is also the routine used if the transfer type is not dma. It just executes the transfer, then returns to the process that was interrupted. If it transfered the last byte, then it will wake up the driver routine so that the driver can complete the transaction and return to user code.

A Brief History of File Transfers
The First Generation
Sneakernet

Carry the file from one machine to another on tape (or punched cards)

The Second Generation
Copynet

- Copy the file via a network (Ethernet, RS232, telephone, etc...) from one machine to another
  - uucp
    - `uucp remote!~uucp/file ~uucp/file`
  - rcp
    - `rcp remote:file file`
- Usually implemented as an application program. Kernel does not know about remote files

Problems with Second Generation File Access

- Requires special mechanisms to access remote files
  - Not transparent
  - Remote files are normally inaccessible to programs and must be manually copied over first
- Need to make a local copy of a file to use it
  - Wastes local disc space
  - Easy to forget about the copy and leave it around
  - Local copy can get out of date
- Not integrated with system
- Either requires password every time a file is copied or presents a large security hole
- Shared packages must be duplicated on all machines
  - Wastes space
  - Major system administration problems keeping all copies up to date

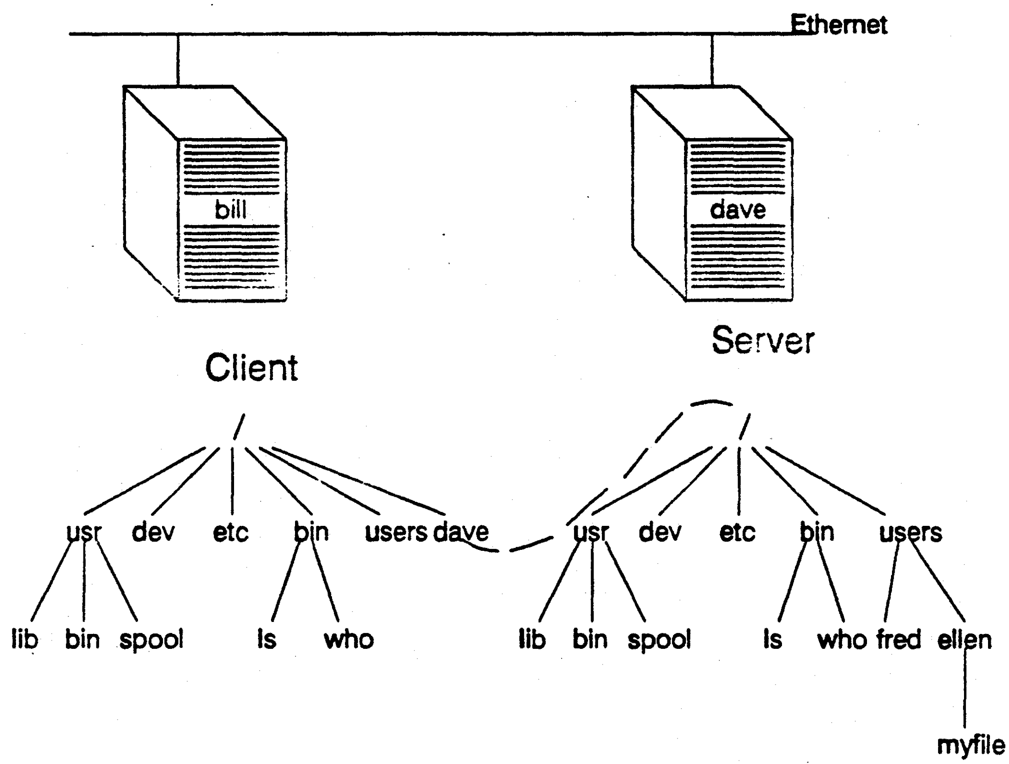
The Third Generation Remote File Systems

- Three UNIX remote file systems at HP
  - RFA (Remote File Access)
    - Developed at Hewlett-Packard
  - NFS (Network File System)
    - Developed at SUN Microsystems
  - RFS (Remote File System)
    - Developed at AT&T

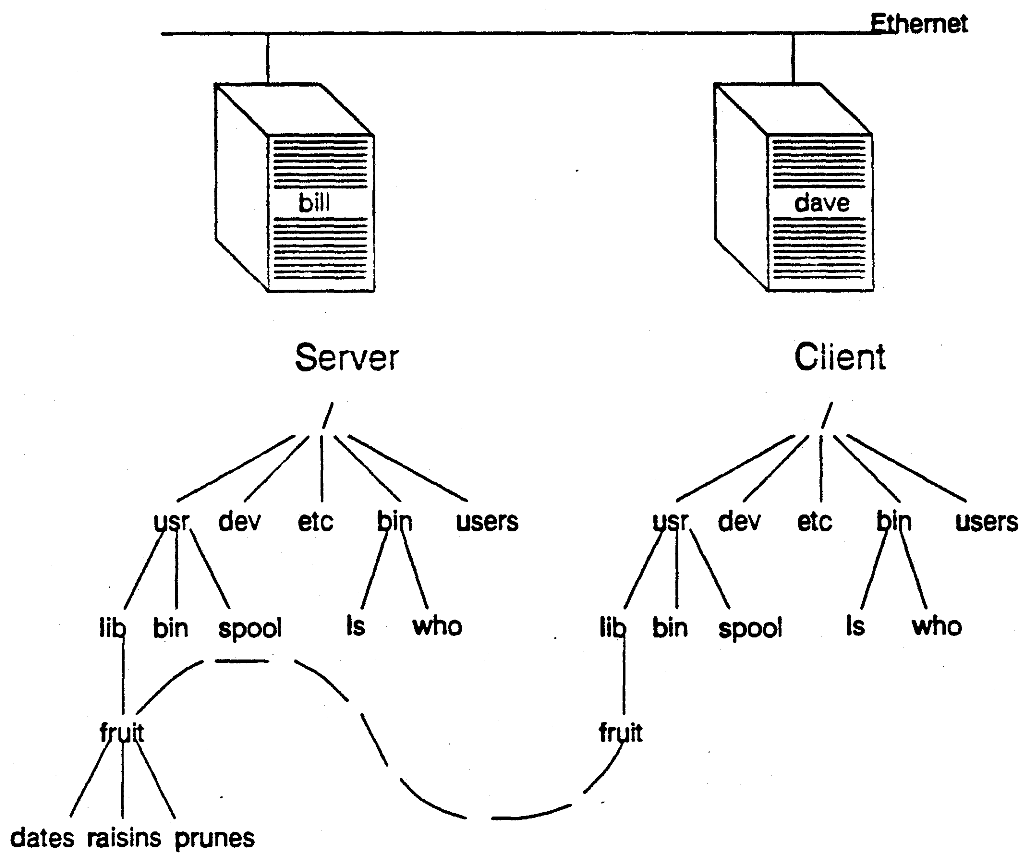
Characteristics of a Remote File System

- Provides access to file systems on a remote machine in a manner identical to local file systems
  - Same commands
    - `vi /localfile`
    - `vi /remote/remotefile`
  - Same system calls
  - Applications need not know about the remote file system (unless they want to)
- Client-Server model
  - Server performs actions dealing with remote files on behalf of the client
  - Client does not make a local copy of the file
- Integrated into the system
- Provides protection
  - Control over which files are available over the network
  - Protect available files from unauthorized access
- Implemented in the form of a remote mount

Mounting a Complete Remote Machine's File System



Mounting a Package from a Remote Machine



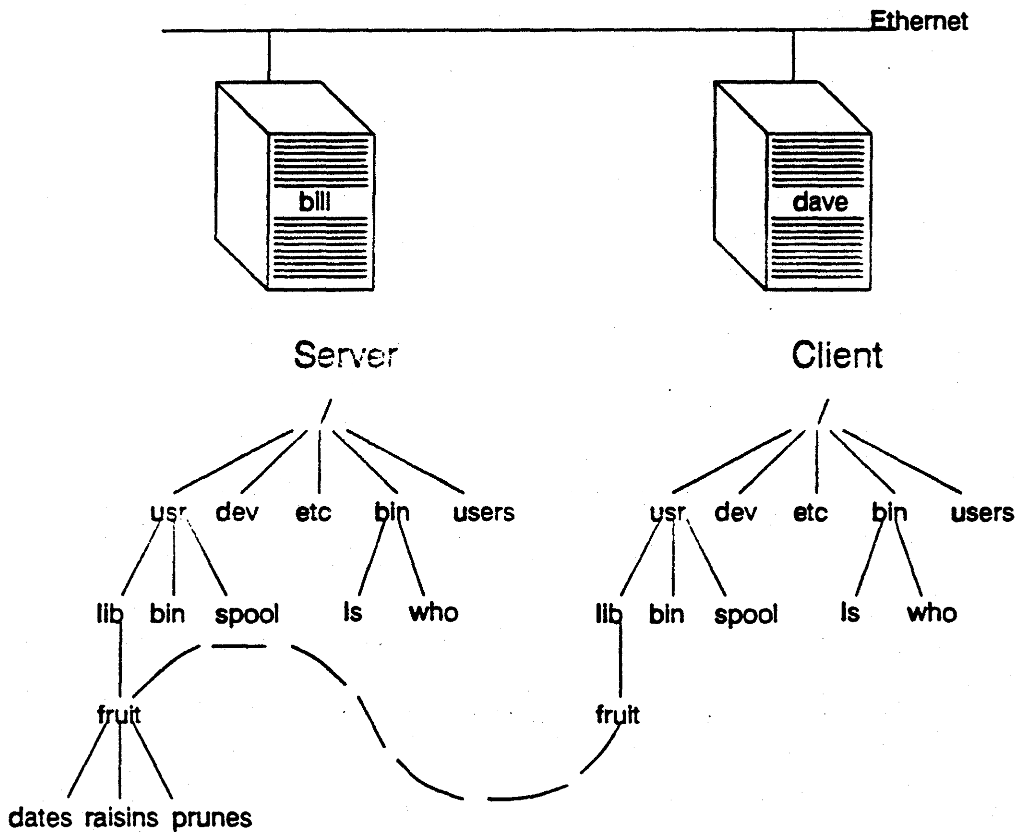
Remote File Systems Solve Second Generation Problems

- No special mechanisms to access remote files
  - Transparent file access
  - Any program that works on local files will work on remote files
- No need to copy files locally
  - No wasted disc space
  - No left over copies
  - Copies don't get out of date
- Fully integrated with system
- Password protection integrated in
- Provides mechanism for sharing packages
  - No wasted disc space
  - Simplifies system administration—no need to keep copies up to date

The Network File System (NFS)

- Developed by SUN Microsystems
- The "Industry Standard" Remote File System
- Goals:
  - Simple
  - General purpose
- Export/mount interface
- Stateless System
- Includes *Yellow Pages* to provide consistent userids among machines
- Available from HP in early '88 on 300s and 800s

Export and Mount in NFS



File /etc/exports:

/usr/lib/fruit   dave

$ ls /usr/lib/fruit

mount bill:/usr/lib/fruit
/usr/lib/fruit

$ ls /usr/lib/fruit
dates   prunes   raisins

A Stateless System

The NFS server does not keep track of which clients are accessing it

- Advantages:
  - Easy to recover from client failure
  - If server or network fails temporarily, client can continue once server is again available
- Disadvantages:
  - Does not provide full UNIX semantics
    - Synchronization not guaranteed in the event of concurrent access
    - Unlinked open files are no longer accessible
  - But:
    - Most programs run without any problems

The Yellow Pages

The *Yellow Pages* (YP) is a rudimentary distributed data base used primarily for sharing certain system administration files such as the password file among cooperating machines

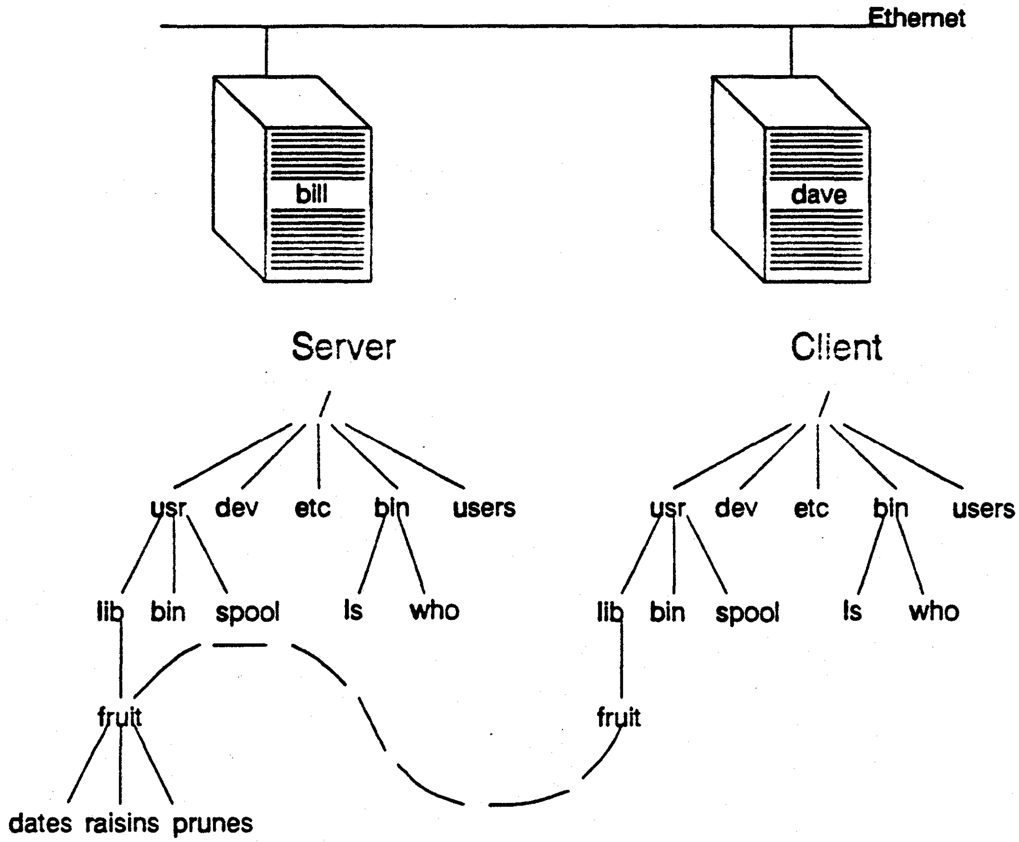
NFS already provides sharing of files. Why do we need the Yellow Pages too?

- YP allows configuration of multiple servers, eliminating reliance on a single point.
  - Can still run if server fails
  - Certain files needed at boot time before NFS is up
- YP permits customization on a machine by machine basis, permitting local overrides. For example, each machine can have its own superuser password.
- Putting these capabilities in an application program cuts down on the kernel size.

The Remote File System (RFS)

- Developed by AT&T
- The "System V.3 Standard" Remote File System
- Goal:
  - Full UNIX semantics
- Advertise/mount interface
  - Advertise is symbolic; hides advertising machine
- Stateful system
- User ID mapping
- Remote device access
- HP is currently porting RFS

Advertise and Mount in RFS




```
# adv FRUIT /usr/lib/fruit
                dave
```

```
$ ls /usr/lib/fruit

# mount -r FRUIT
         /usr/lib/fruit

$ ls /usr/lib/fruit
dates   prunes   raisins
```

- Advertised name is symbolic—Does not include machine name

A Stateful System

The RFS server keeps track of which clients are accessing it

- Advantages:
  - Provides full UNIX semantics including synchronization
- Disadvantages:
  - Recovery from client failure more difficult
  - Temporary loss of the server or the network means loss of the file access
    - Client must reopen the remote file once the system is available

User ID Mapping

RFS provides a facility for mapping userids (UIDs) from one machine to another

- Set up by system administrator
  - Easy to configure defaults
    - Transparent mapping—each UID maps to itself
    - Given UID mapping—all UIDs map to a single remote UID
  - Can override on a UID by UID basis
  - Each machine can have a different mapping

Why Three Remote File Systems?

HP will be providing three remote file systems, RFA, NFS, and RFS. Why do we need them all?

- RFA is needed for backwards compatibility and to talk to S500s
- NFS is the current industry standard, and is available from more vendors than any other remote UNIX file system
- RFS is needed for AT&T System V.3 compatibility

The Fourth Generation
Distributed Systems
A Future Vision

- One system view
  - The same file system seen from all sites
  - File locations fully hidden
  - Not a remote mount based model
- Other resources also distributed in a transparent manner
  - Transparent remote process execution
  - Process migration
  - Remote and local inter-process communication treated identically
- Every machine feels like home

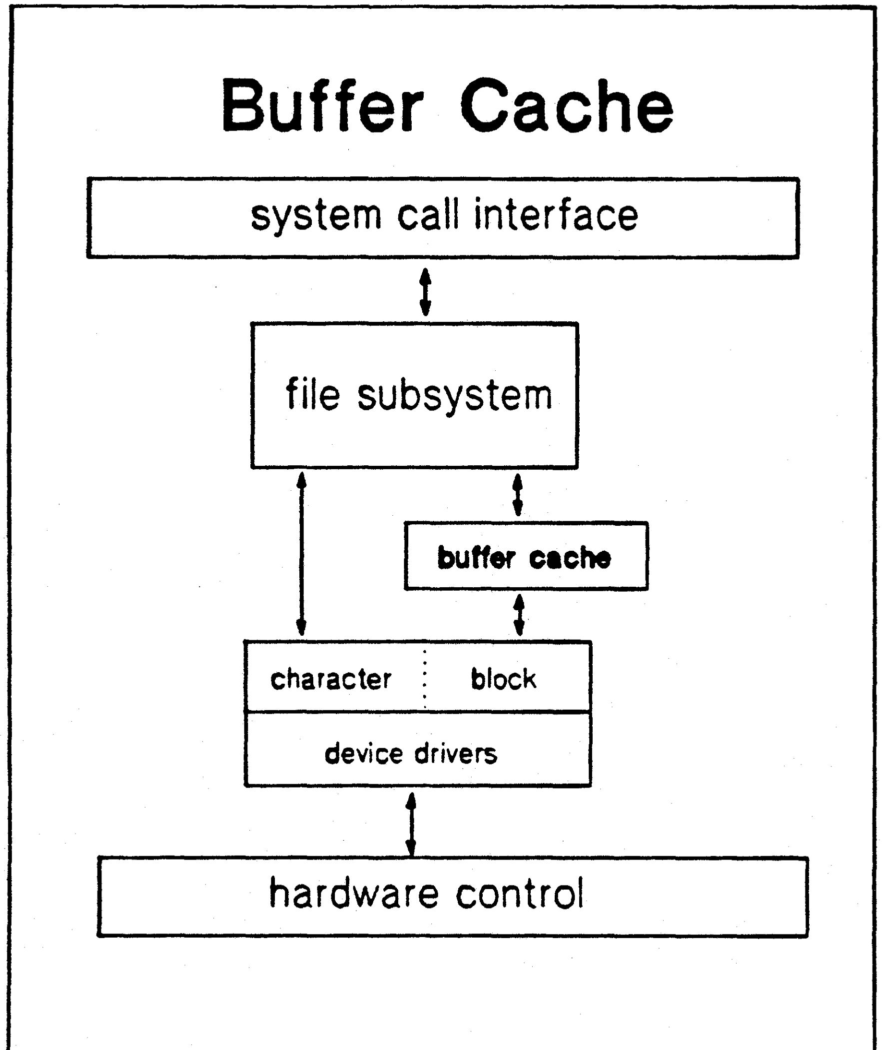
The Discless 300s

- Not a fourth generation system, but has some aspects of it
- Allows multiple Series 300s to share a single set of discs
- All machines in cluster see the same view of the file system
- Common logins and passwords automatically provided
- Not a remote file system
  - Provides sharing within the cluster only
  - Does not use a remote mount model–all machines see the same file system
- Only one logical machine to administer

The Buffer Cache

☐ The Buffer Cache

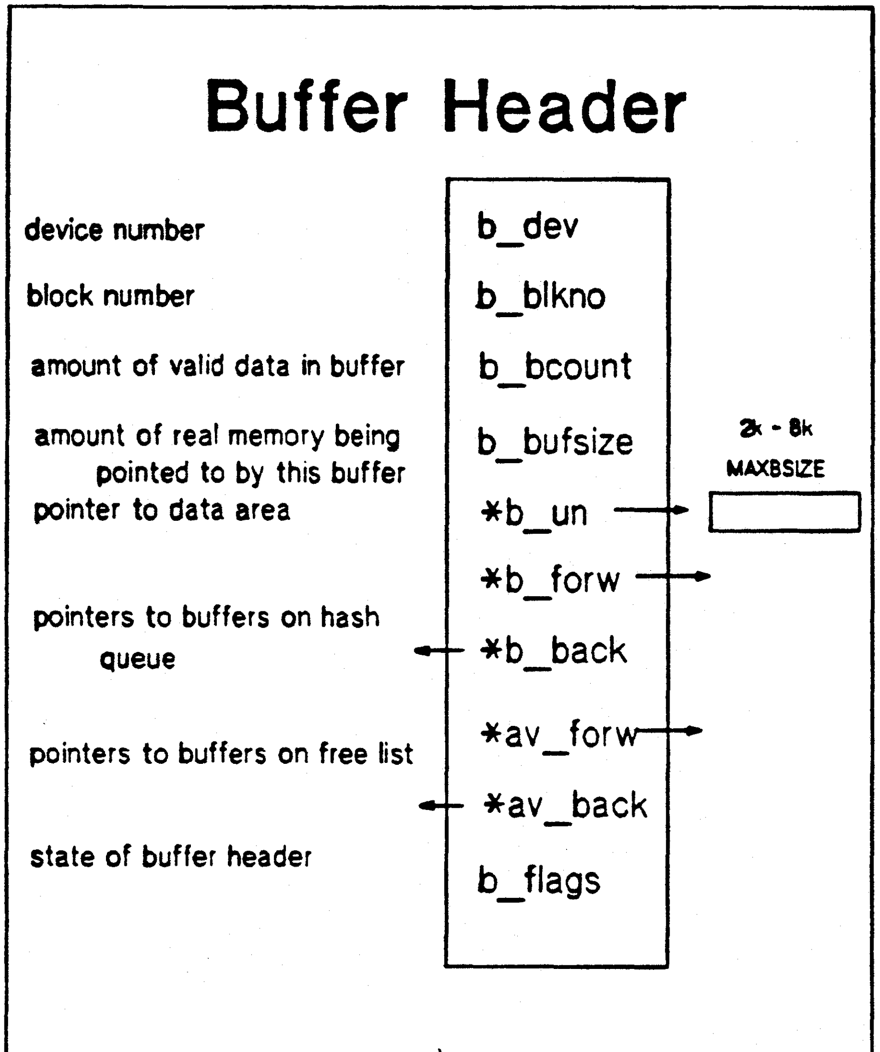
☐ Notes



UFS30010

☐ The Buffer Cache

☐ Notes

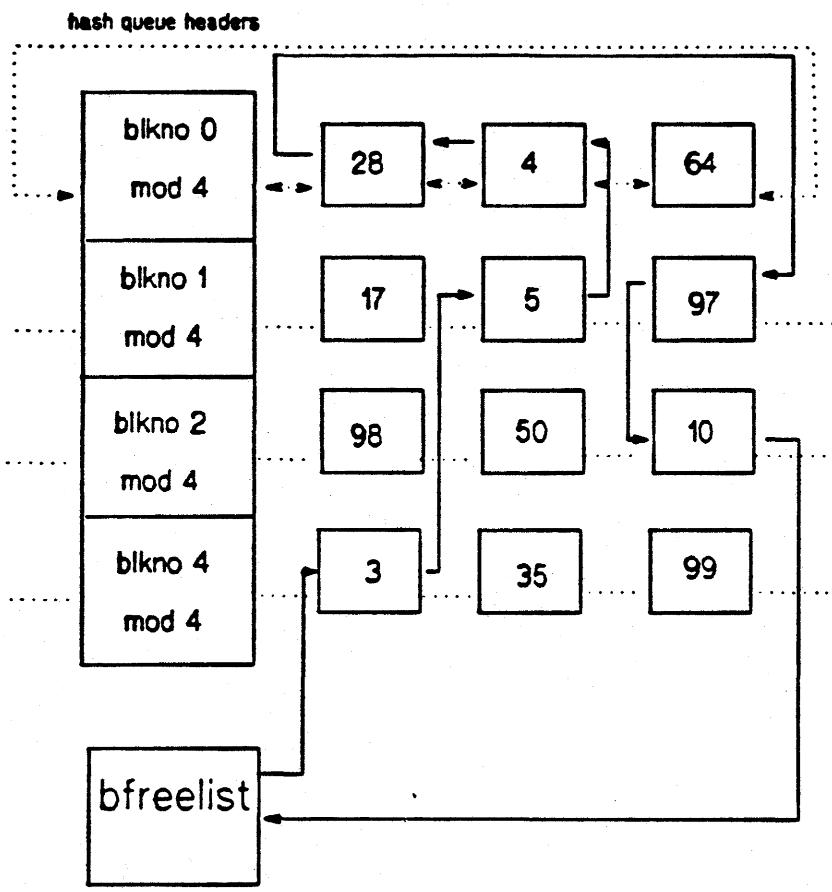


UFS30015

The Buffer Cache

Notes

Buffer Header Usage

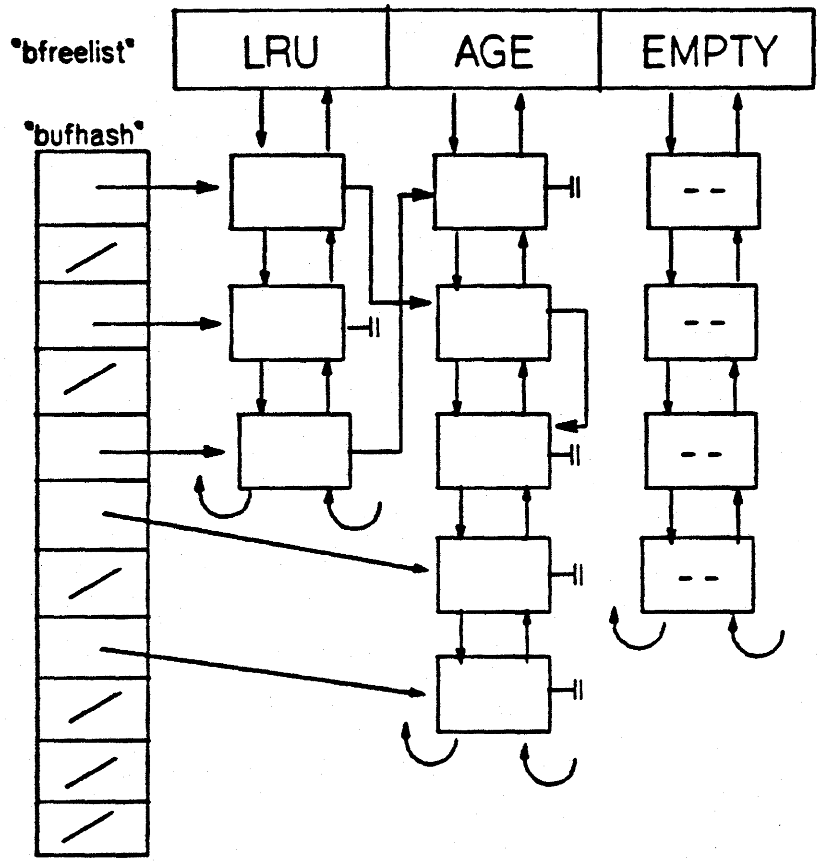


ufa30020

☐ The Buffer Cache

☐ Notes

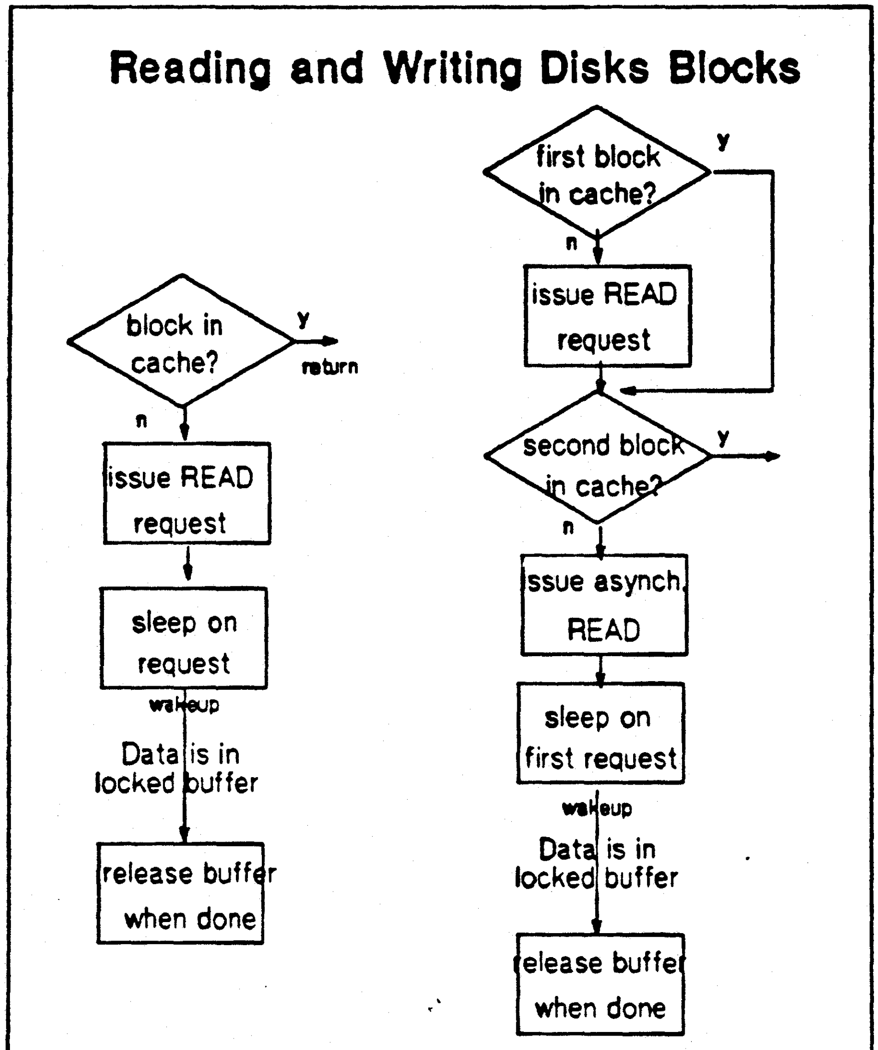
Buffer Management



- Always on exactly one hashchain unless EMPTY
- Always on exactly one freelist unless BUSY

The Buffer Cache

Notes



ufa30050

☐ The Buffer Cache

☐ Notes

Major Routines

- Getting Buffers

  bp = bread(dev,blkno,size)

- Releasing Buffers

  brelse(bp) - release it, no write

  bwrite(bp) - syncronous write

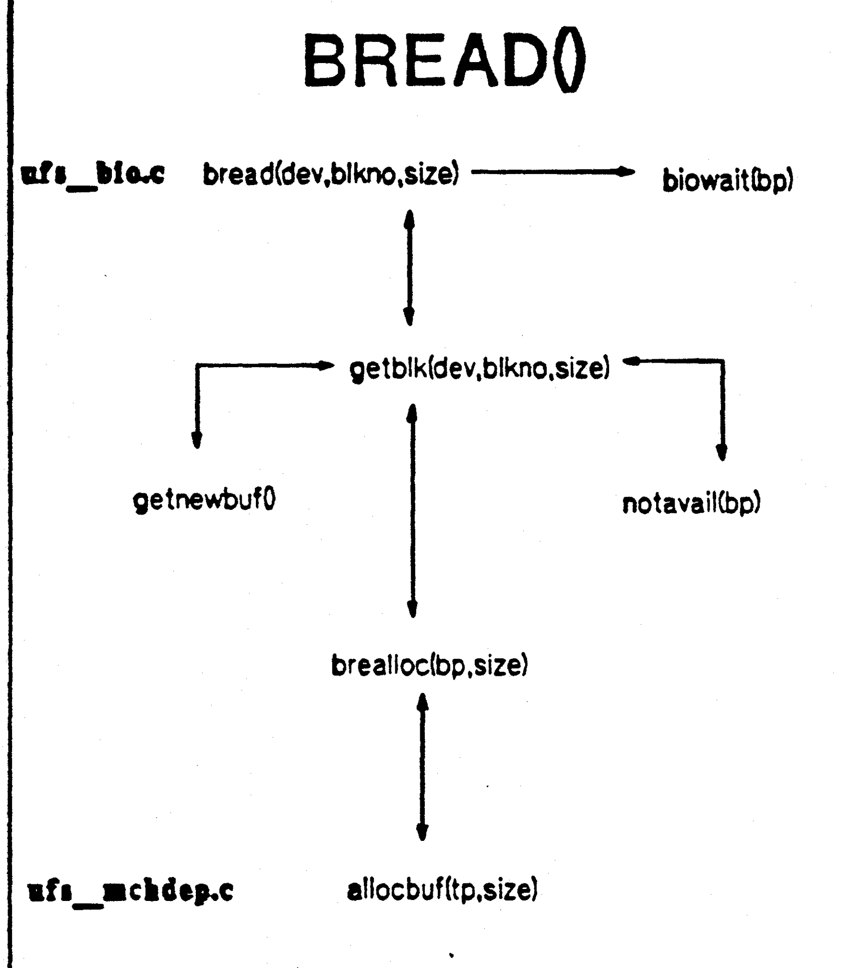
  bdwrite(bp) - delayed write

  bawrite(bp) - asynchronous write

ufs30022

The Buffer Cache

Notes

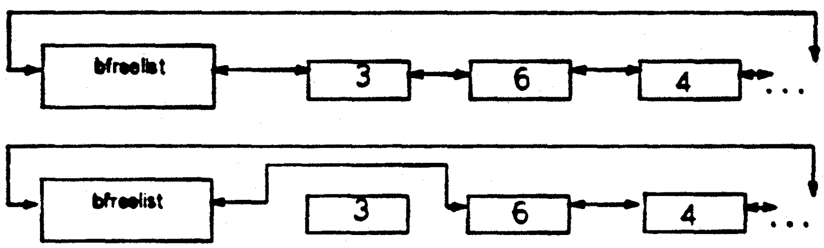


ufs30024

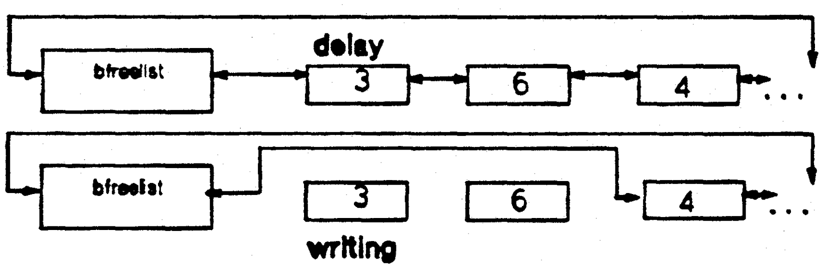
Retrieval of a Buffer

Block **NOT** in Cache

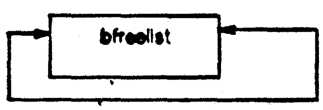
1) Remove first block from Free List



2) block not in hash queue - remove 1st block from free list



3) Free List empty



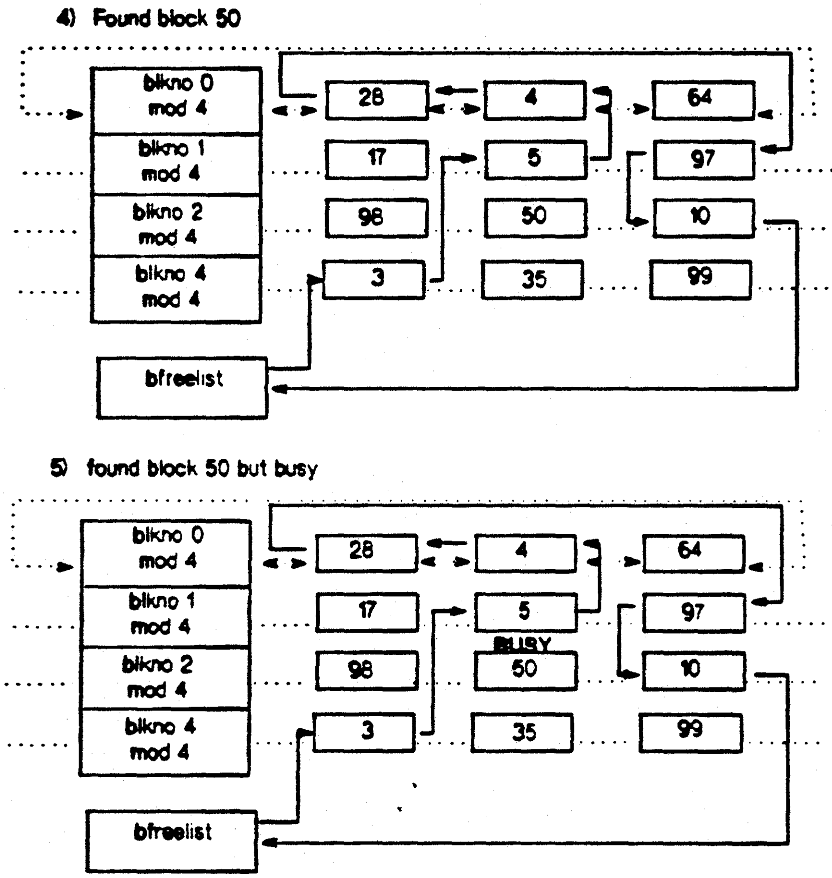
ufa30025

The Buffer Cache

Notes

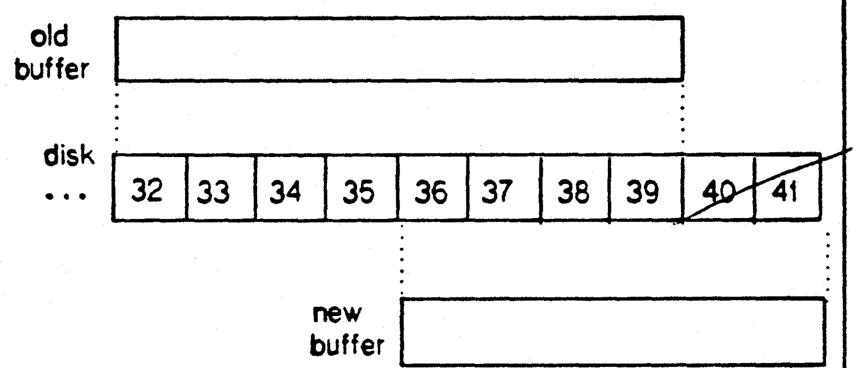
Retrieval of a Buffer

Block **IN** Cache



ufs30026

Overlapping Buffers



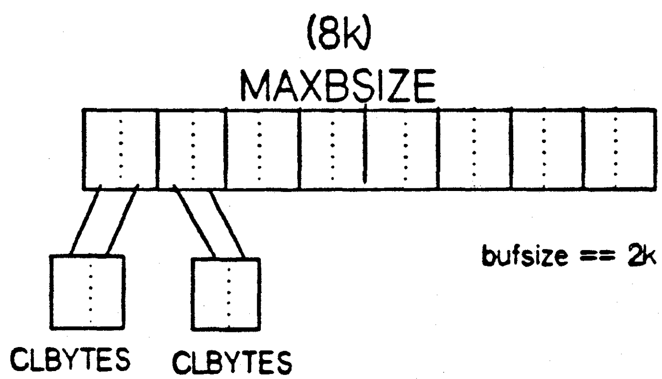
If the "old buffer" is marked **delayed write**, it must be written. The "old buffer" must be marked **INVAL**.

- **Therefore, a disk block is mapped into at most one buffer.**

The Buffer Cache

Notes

allocbuf()



If buffer size is shrinking:

- Take buffer header off EMPTY queue.
- Put excess pages in it and release onto AGE queue.

If buffer size is growing:

- Get a buffer from NEWBUF.
- Transfer pages to new buffer.
- If pages left over return to AGE.
- If no pages left return to EMPTY.
- Repeat until enough pages are allocated to new buffer.

SE 390: Series 300 HP-UX Internals

Monday Afternoon Labs

0. Reboot the system and pay close attention to the messages that are printed out. What't the last line printed by the kernel? What's the first line printed by init(1m)?

1. Using the template provided (ppt.c), print out the values of at least 10 kernel parameters. Verify 2-3 of them with adb(1), and the rest with monitor(1m).

2. Using ppt.c again, write a version of ps(1) that skips most of the garbage (gettys, daemons, etc).

3. Modify top.c so it will run on the 300.

4. Put the system under stress and experiment with nice values. How much do they affect a process when the system is under 1) no stress; 2) moderate stress; 3) heavy stress?

5. Replace /etc/init with a program or script that 1) does something useful, like invoke a shell; 2) moves the real init back into place so that you can reboot and have a normal system.

SE 390: Series 300 HP-UX Internals

Tuesday Afternoon Labs

4 0. Set the sticky bit on a fairly large program and see how this affects startup time.

1. Configure a new kernel and look at the conf.c that is generated. Which
5 parts of it came from /etc/master? Which from the dfile you provided?

1 2. Make the system panic and interpret the resulting stack trace.

2 3. Run a program that will force the system to page and/or swap, and observe the results with monitor(1m).

3 4. Write a program that attaches to some shared memory and then starts malloc(3)ing 1k chunks. How many can you get? What kernel parameter could you change to fix the problem?

SE 390: Series 300 HP-UX Internals

Wednesday Afternoon Labs

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

1. Write a program to figure out which files are in a particular cylinder group on the disk.

2. Write a program that will "stat" a file without using the stat(2) call. (Hint: in what place on the disk is most of the information for a file kept? How can you get there given the file's pathname?)

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

4. Have your partner mess up the disk using disked(1m). 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 the raw disk device.