The BSD 4.4 Operating System Internals

Tad Hunt
(tad@csh.rit.edu)
Computer Science House

vnode-7.fig

Origins of Unix

Ken Thompson of Bell Laboratories invented Unix in 1969

Dennis Ritchie joined shortly thereafter and invented the C language and contributed to the design and implementation of the Second Edition.

In 1977, 1BSD was spawned from the Sixth Edition

The CSRG At the University of California at Berkeley added VM, TCP/IP, and other features

BSDI, FreeBSD, NetBSD, and OpenBSD provide support today

origins.fig

Contributions of BSD to Unix

Reliable signals

Virtual Memory, demand paging, page replacement

TCP/IP

The Berkeley Socket API

Job control

Terminal independence

Multiple file-access permission groups

Filesystem interfaces

etc...

contributions.fig

BSD Design Goals

Early:

Virtual Memory support
Device drivers for third-party (non DEC) peripherals
Terminal independent support
High speed filesystem
IPC facilities
Networking support

Current:

Portability (supports MIPS, SPARC, m68k, 80x86, etc)

Replacement of the VM system

ISO networking protocol suite

POSIX compatible terminal driver

Vnode interface to filesystems

design-goals.fig

Responsibilities of the Kernel

Schedule Processes

Manage access to resources

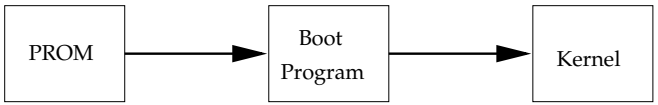
Provide a portable interface for processes

Support a wide variety of devices through a common interface

Make adding support for new devices as simple as possible

kernel-responsibilities.fig

Machine startup



Flow of control during the boot process

The PROM boots a "boot" program

The boot program then boots the kernel from disk,tape, network or etc.

The kernel is generally a standard executable recognized by the BSD OS (a.out, elf, coff, etc)

The boot program zeros the kernels BSS, sets it up with a temporary stack, and passes control to the "start" routine in "locore.s"

machine-startup-1.fig

Machine startup

Locore start routine

Control passed here from the boot program

```
start:
    disable interrupts
    initialize the stack pointer
    mach_init()
    disable interrupts
    switch to kernel stack
    main()

    /* process[1] (/sbin/init) is
       now set to run when the
       processor returns to
       user mode
    */

    restore registers
    return to user mode
```

setup exception vector table
setup VM for proc[0]
setup the null process
size memory
setup exception handlers
allocate space for system data structures
initialize the virtual memory subsystem.

Initialize subsystems
do autoconfiguration
start scheduler
mount root
start init and page daemon

machine-startup-2.fig

Kernel organization

Division of labor

Runs until blocked or done

Can block to wait for a resource
Runs on a per-process kernel stack

Provides services to processes in response to system calls or traps

Top Half

Bottom Half

Never scheduled
Cannot block
Runs on kernel stack in kernel address space

handles interrupts and traps

kernel-organization-1.fig

Kernel organization — Top Half

Provides servies to processes in response to system calls or traps

Essentially it's a library of routines shared by all processes

Executed in supervisor mode and has access to both kernel data structures and the context or user level processes

Process context is stored in two places:

The process structure stores info needed even if the process is swapped out

The user structure stores everything else

Kernel organization — Process structure

Contains:

identifiers associated with the process

the process's rights and privileges

the process's descriptors

it's virtual memory map

pending external events and associated actions

maximum and current resource utilization

etc...

Kernel organization — User structure

Contains:

The Process Control Block (PCB)

process accounting and statistics

additional information for debugging and creating a core dump

Kernel organization — Entry from user process

Hardware Interrupt

Caused by external events.
Occur asynchronously
Causes entry into the bottom half of the kernel
May or may not be related to the currently running process

Hardware trap

Synchronous or asynchronous
Related to the currently running process
Example: Illegal arithmetic operation such as divide by zero

Software initiated trap

Used by the system to force the scheduling of an event such as process rescheduling

Implemented as a hardware generated interrupt or as a flag that is checked when the kernel returns to user space

Kernel organization

Kernel entry

execution path

Save machine state

Call Syscall(), Trap(), or the appropriate device-driver interrupt handler

restore machine state
return to user mode

Syscall()

Passed the system call number and the exception frame

Trap()

Passed the type of trap, the relevant flotain point and virtual address information, and an exception frame

Device Interrupt Handler

Passed the unit (board) number

Process Scheduling

Uses multilevel feedback queues

Processes are placed in different queues according to their priority

The queues are scanned from highest to lowest priority to choose the process to run

The process in the first non-empty queue is chosen
If multiple processes exist at that priority, they're run round-robin

When a process blocks, it's removed from the run queues

If a process uses up its quantum, it's placed on the end of the queue it came from

Process Scheduling

The quantum is set to 0.1 second. This hasn't changed for over 15 years.

The process priority is adjusted dynamically

Process priority is adjusted based on resource usage

Processes are moved between run queues based on changes in their scheduling priority

If a process reaches a higher priority than the one running, execution is immediately switched to that process upon the kernel return to user mode.

Processes blocked waiting for I/O for 1 or more seconds have their priorities raised, and processes that use a lot of CPU time have their priorities lowered.

Process Scheduling

$$p_userpri = PUSER + \left[\frac{p_estcpu}{4} \right] + 2 \times p_nice \qquad \text{EQ. 1}$$

$$p_estcpu = \left[\frac{(2 \times Load)}{(2 \times Load + 1)} \right] p_estcpu + p_nice \qquad \text{EQ. 2}$$

p_userpri	the new process priority. Range = PUSER to 127
p_estcpu	incremented each time the system clock ticks and the process is still executing. Additionally, adjusted once per seond via a digital decay filter (EQ 2). (90% of the CPU time is forgotten about after some time T.)
p_nice	user controlled weighting factor
PUSER	Base priority for user mode execution. Value is 50.
Load	the current system load

Process Scheduling

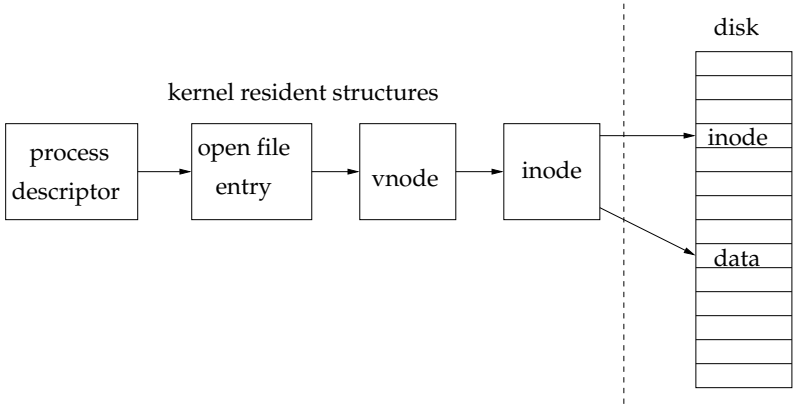
$$p_estcpu = \left[\frac{(2 \times Load)}{(2 \times Load + 1)} \right]^{p_slptime} \times p_estcpu \qquad \text{EQ. 3}$$

This value for p_estcpu is used by EQ. 1 when a process is awakened after being in a SSLEEP or SSTOP state.

Load is the current load average, not the load average at the time the process went to sleep.

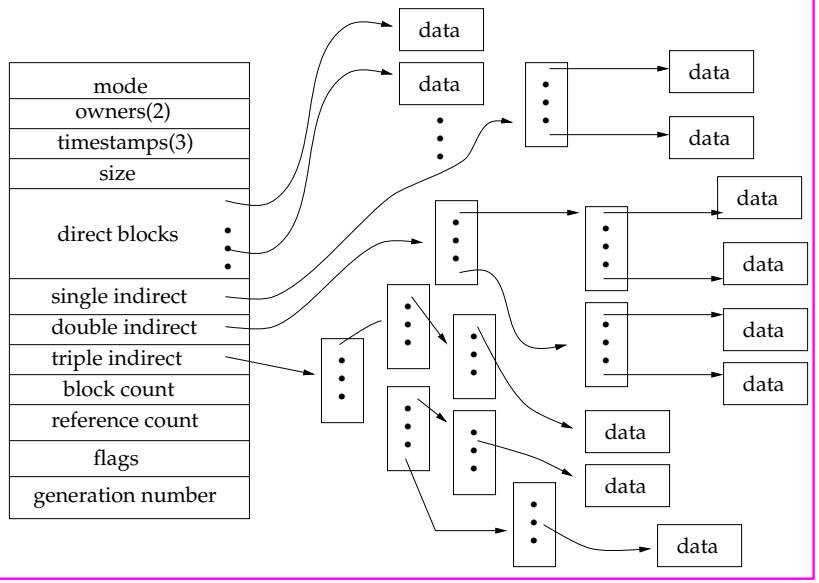
proc-scheduling-4.fig

Kernel file table layout



kernel-file-table.fig

Structure of an inode



inode-structure.fig

File flags and security

In addition to the standard file permissions, BSD supports some useful extensions:

two new system calls, chflags(2) and fchflags(2) set these flags.

append-only	the file can only be appended to
immutable	the file cannot be changed
don't dump	don't back this file up

file-flags-1.fig

Levels of system security

-1 Permanently insecure mode
always run the system in level 0.
(Must be compiled into kernel)

0 Insecure mode
Immutable and append-only flags may be turned off
devices may be read or written to depending on their permissions

1 Secure mode
The superuser-settable immutable and append-only flags cannot be cleared. Disks and kernel memory (/dev/mem and /dev/kmem) are are read only.

2 Highly secure mode
Same as secure mode, except that disks are always read only even if they're not mounted. This disallows even a superuser process from tampering with filesystems.

Device driver organization

Device drivers are divided into three main sections

Autoconfiguration and initialization

Probes for the given device to determine if it's present.
initializes the device and device state info if it's present
Called once when the system is initialized.

Routines for servicing I/O requests (the top half)

Invoked because of system calls or the virtual memory subsystem. Executes synchronously in the top half of the kernel.
Permitted to block by calling sleep()

Interrupt service routines (bottom half)

Invoked when the system receives an interrupt from the device.
Cannot depend on any per-process state.
can't block.

Device driver organization

The glue routine placed in the exception table does the following:

Saves all volatile registers

Updates statistics on device interrupts

Calls the interrupt service routine with the appropriate unit number parameter

Restores the volatile registers

returns from the interrupt

Device driver organization

Types of device drivers

Block

Examples are disks and tapes

Converts from the user abstraction of a disk as an array of bytes to the structure imposed by the underlying physical medium.

Character

Examples are terminals and serial ports

Maps the hardware interface into a byte stream.

Also considered the "raw" device interface.

Device driver organization

Block Device Interface

open Called for each open system call on a block special device file, or internally when mounted. Usually verifies the integrity of the associated medium.

strategy Starts a read or write operation, and returns. called with a pointer to a buf structure containing the parameters for the I/O request.

close Called after the final client interested in the device terminates.

dump Dumps physical memory to the device.

psize Returns the size of a disk partition. Used to calculate where to place a crash dump and to determine the size of the swap.

Device driver organization

Character device interface

open open the device

close close the device

ioctl do an I/O control operation

mmap map device offset to memory location

read do an input operation

reset reinitialize device after a bus reset

select poll device for I/O readiness

stop stop output on the device

write do an output operation

The VNODE layer

Contents of a vnode

Flags to lock the vnode and to identify generic attributes

Reference counts on:

- The number of file entries open that reference the vnode.
- The number of pages and buffers associated with the vnode.

A pointer to the mount structure describing the filesystem containing the object represented by the vnode

Information to do file read-ahead

A reference to an NFS lease

A referenct to state about special devices, sockets, and FIFOs

Pointer to the set of vnode operations defined for this object

Pointer to private informatrion needed for the underlying object
(i.e.: an inode for a local filesystem, a nfsnode for NFS)

The VNODE layer

Contents of a vnode

The type of the underlying object
(regular file, directory ,character device, etc)

Clean and dirty buffers associated with the vnode

A count of the number of buffer write operations in progress

The VNODE layer — Vnode operations

Set of operations for hierarchical filesystem operations

Set of operations for storage of variable sized objects using a flat namespace

Pathname Translation

1. Pathname to be translated is copied into a kernel buffer
2. Starting point is determined as either the root directory or relative to the current directory. The appropriate vnode becomes the lookup directory for step 3.
3. The filesystem-specific lookup() operation gets the remaining pathname and the current lookup directory. It returns the appropriate vnode for the next layer or it returns an error
4. If there is no error, the last vnode is the result of the lookup. it is then checked. If the vnode is a directory, it's checked to determine if it's a mount point for another filesystem. If so, the vnode for that filesystem is returned.

The VNODE layer — Exported filesystem services

All filesystems support the update of generic mount options

These flags can be changed while a filesystem is mounted

Flags:

noexec Don't allow execution of any files from this filesystem

nosuid Don't run setuid or setgid executables on this filesystem with the setuid or setgid privileges

nodev don't allow special devices on the filesystem to be opened

The statfs service returns a buffer giving the number of used and free disk blocks and inodes, the filesystem mountpoint, the device, location, or program from which the filesystem is mounted

The getfsstat service returns info about all mounted filesystems

The VNODE layer — Filesystem independent services

inactive() - called by the vnode interface when the final reference to a vnode is cleared.

When the last reference goes away, the vnode is placed on a system wide free list. This allows vnodes to be used where they are needed.

getnewvnode() - picks an available vnode. Used when a file is opened

reclaim() - called to let the underlying filesystem know that the vnode is about to be reused.

vgone() - detaches the underlying special device and attaches the vnode to the deadfs. All further operations result in errors.

revoke() - system call that allows processes access to the vgone() routine.

The VNODE layer — The name cache

Caches positive and negative references to names

Positive caching speeds up operations such as ls(1) where all files in a directory are stat(2)ed.

Negative caching speeds up PATH lookups for shells.

The VNODE layer

Some cool filesystems

The vnode interface allows for stackable filesystems

nullfs simply passes its arguments through to the vnode it is mounted from. Useful to mount the filesystem at it's root somewhere else.

umapfs maps user and group id's. Useful for NFS exported filesystems where the remote users have different's and gid's.

unionfs Provides the union of two filesystems. The one on top always obscures duplications of the one underneath. Useful for building a source distribution from a CDROM.

portalfs passes the pathname component from it's root down to a user process.

References

- *The Design and Implementation of the 4.4BSD Operating System,*
  McKusick, Bostic, Karels, Quarterman, Addison Wesley, 1996
  ISBN 0-201-54979-4
- NetBSD 1.0 - NetBSD 1.2 Source