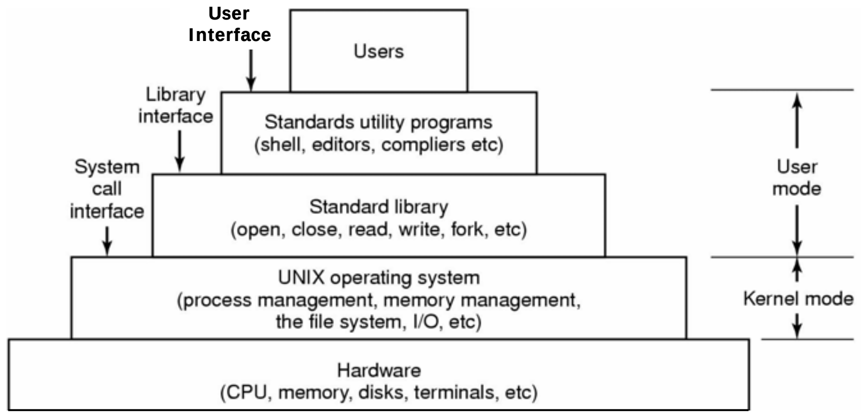
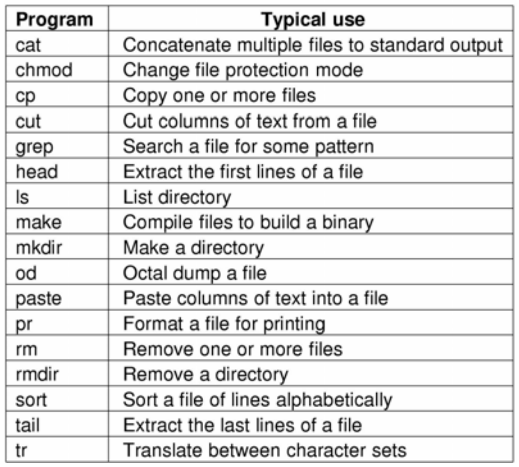
Chapter 10

Case Study 1: UNIX and LINUX

UNIX


User Interface
Users
Library interface
Standards utility programs (shell, editors, compliers etc)
System call interface
Standard library (open, close, read, write, fork, etc)
UNIX operating system (process management, memory management, the file system, I/O, etc)
Hardware (CPU, memory, disks, terminals, etc)
User mode
Kernel mode

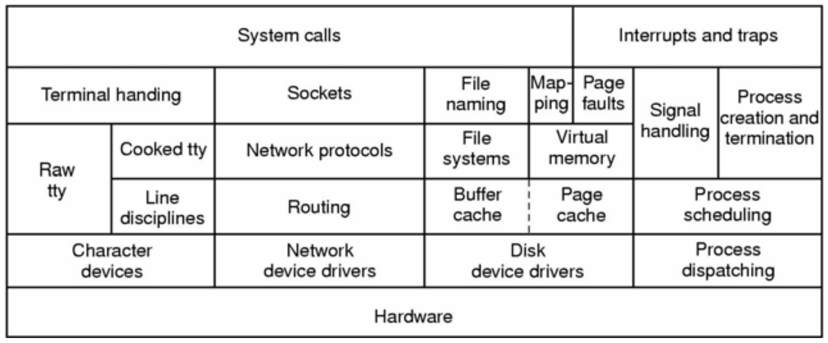
UNIX Utility Programs



Program	Typical use
cat	Concatenate multiple files to standard output
chmod	Change file protection mode
cp	Copy one or more files
cut	Cut columns of text from a file
grep	Search a file for some pattern
head	Extract the first lines of a file
ls	List directory
make	Compile files to build a binary
mkdir	Make a directory
od	Octal dump a file
paste	Paste columns of text into a file
pr	Format a file for printing
rm	Remove one or more files
rmdir	Remove a directory
sort	Sort a file of lines alphabetically
tail	Extract the last lines of a file
tr	Translate between character sets

A few of the more common UNIX utility programs required by POSIX

UNIX Kernel



Approximate structure of generic UNIX kernel

Processes in UNIX

```
pid = fork( );                /* if the fork succeeds, pid > 0 in the parent */
if (pid < 0) {
     handle_error( );         /* fork failed (e.g., memory or some table is full) */
} else if (pid > 0) {
                              /* parent code goes here. /*/
} else {
                              /* child code goes here. /*/
}
```

POSIX

Signal	Cause
SIGABRT	Sent to abort a process and force a core dump
SIGALRM	The alarm clock has gone off
SIGFPE	A floating-point error has occurred (e.g., division by 0)
SIGHUP	The phone line the process was using has been hung up
SIGILL	The user has hit the DEL key to interrupt the process
SIGQUIT	The user has hit the key requesting a core dump
SIGKILL	Sent to kill a process (cannot be caught or ignored)
SIGPIPE	The process has written to a pipe which has no readers
SIGSEGV	The process has referenced an invalid memory address
SIGTERM	Used to request that a process terminate gracefully
SIGUSR1	Available for application-defined purposes
SIGUSR2	Available for application-defined purposes

System Calls for Process Management

System call	Description
pid = fork( )	Create a child process identical to the parent
pid = waitpid(pid, &statloc, opts)	Wait for a child to terminate
s = execve(name, argv, envp)	Replace a process' core image
exit(status)	Terminate process execution and return status
s = sigaction(sig, &act, &oldact)	Define action to take on signals
s = sigreturn(&context)	Return from a signal
s = sigprocmask(how, &set, &old)	Examine or change the signal mask
s = sigpending(set)	Get the set of blocked signals
s = sigsuspend(sigmask)	Replace the signal mask and suspend the process
s = kill(pid, sig)	Send a signal to a process
residual = alarm(seconds)	Set the alarm clock
s = pause( )	Suspend the caller until the next signal

s is an error code
pid is a process ID
residual is the remaining time from the previous alarm

POSIX Shell

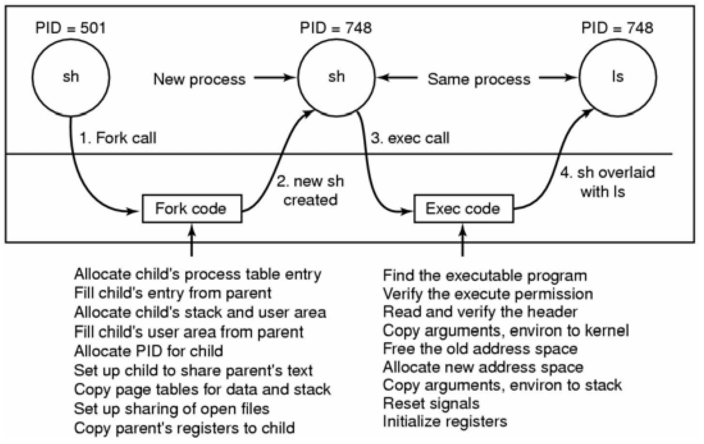
```
while (TRUE) {                               /* repeat forever /*/
     type_prompt( );                         /* display prompt on the screen */
     read_command(command, params);          /* read input line from keyboard */

     pid = fork( );                          /* fork off a child process */
     if (pid < 0) {
          printf("Unable to fork0);          /* error condition */
          continue;                          /* repeat the loop */
     }

     if (pid != 0) {
          waitpid (−1, &status, 0);          /* parent waits for child */
     } else {
          execve(command, params, 0);        /* child does the work */
     }
}
```

A highly simplified shell

Threads in POSIX

Thread call	Description
pthread_create	Create a new thread in the caller's address space
pthread_exit	Terminate the calling thread
pthread_join	Wait for a thread to terminate
pthread_mutex_init	Create a new mutex
pthread_mutex_destroy	Destroy a mutex
pthread_mutex_lock	Lock a mutex
pthread_mutex_unlock	Unlock a mutex
pthread_cond_init	Create a condition variable
pthread_cond_destroy	Destroy a condition variable
pthread_cond_wait	Wait on a condition variable
pthread_cond_signal	Release one thread waiting on a condition variable

The *ls* Command


PID = 501
PID = 748
PID = 748
sh
New process
sh
Same process
ls
1. Fork call
3. exec call
2. new sh created
4. sh overlaid with ls
Fork code
Exec code
Allocate child's process table entry
Fill child's entry from parent
Allocate child's stack and user area
Fill child's user area from parent
Allocate PID for child
Set up child to share parent's text
Copy page tables for data and stack
Set up sharing of open files
Copy parent's registers to child
Find the executable program
Verify the execute permission
Read and verify the header
Copy arguments, environ to kernel
Free the old address space
Allocate new address space
Copy arguments, environ to stack
Reset signals
Initialize registers

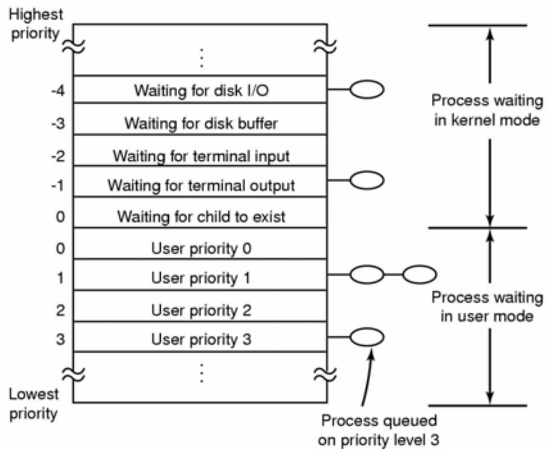
Steps in executing the command *ls* type to the shell

Flags for Linux clone

Flag	Meaning when set	Meaning when cleared
CLONE_VM	Create a new thread	Create a new process
CLONE_FS	Share umask, root, and working dirs	Do not share them
CLONE_FILES	Share the file descriptors	Copy the file descriptors
CLONE_SIGHAND	Share the signal handler table	Copy the table
CLONE_PID	New thread gets old PID	New thread gets own PID

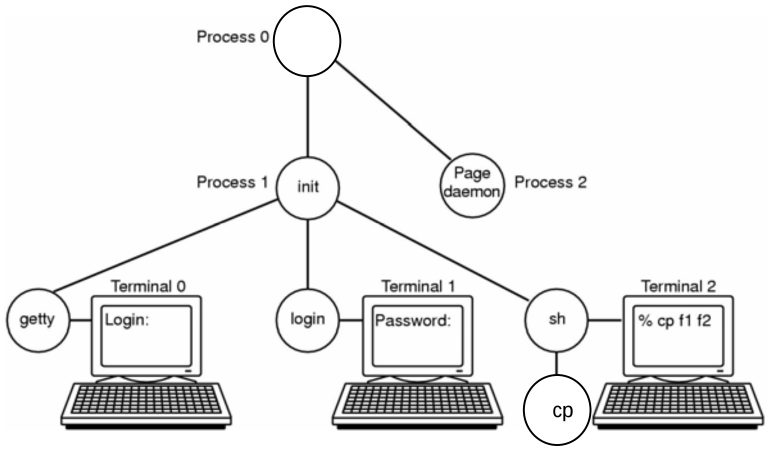
Bits in the sharing_flags bitmap

UNIX Scheduler


Highest priority
-4
-3
-2
-1
0
0
1
2
3
Lowest priority
Waiting for disk I/O
Waiting for disk buffer
Waiting for terminal input
Waiting for terminal output
Waiting for child to exist
User priority 0
User priority 1
User priority 2
User priority 3
Process waiting in kernel mode
Process waiting in user mode
Process queued on priority level 3

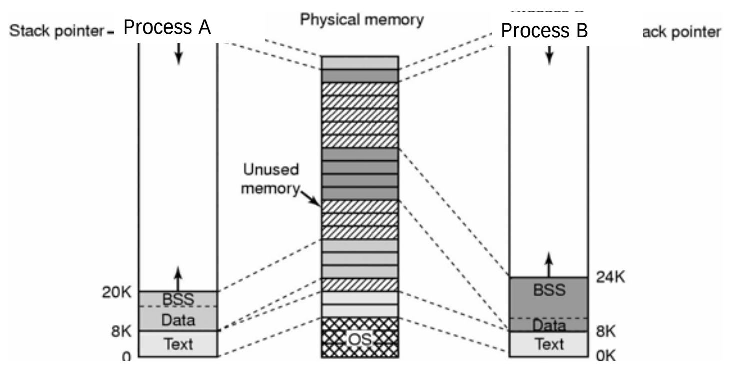
The UNIX scheduler is based on a multilevel queue structure

Booting UNIX



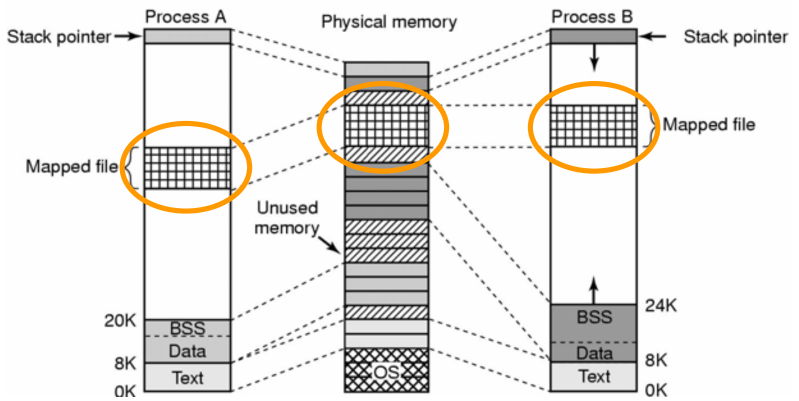
The sequences of processes used to boot some systems

Handling Memory



- Process A's virtual address space
- Physical memory
- Process B's virtual address space

Sharing Files



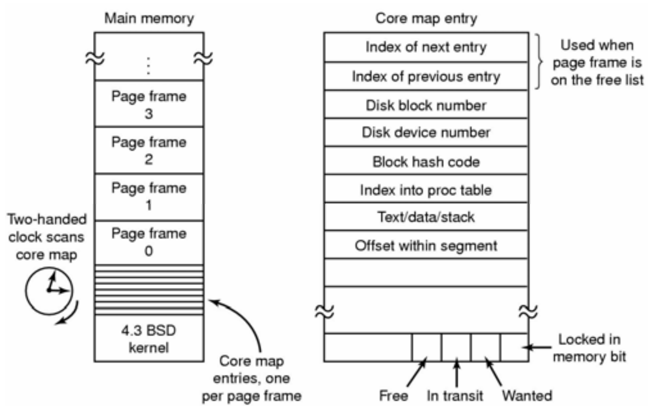
A new file mapped simultaneously into two processes

System Calls for Memory Management

System call	Description
s = brk(addr)	Change data segment size
a = mmap(addr, len, prot, flags, fd, offset)	Map a file in
s = unmap(addr, len)	Unmap a file

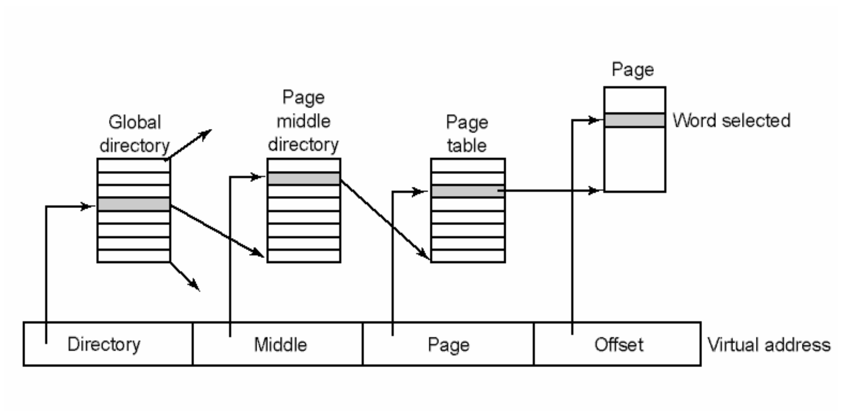
- **s** is an error code
- **b** and **addr** are memory addresses
- **len** is a length
- **prot** controls protection
- **flags** are miscellaneous bits
- **fd** is a file descriptor
- **offset** is a file offset

Paging in UNIX



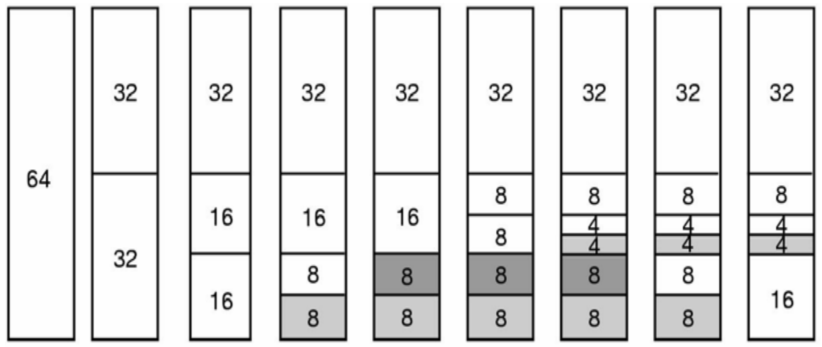
The core map has an entry for each page

Paging in Linux (1)



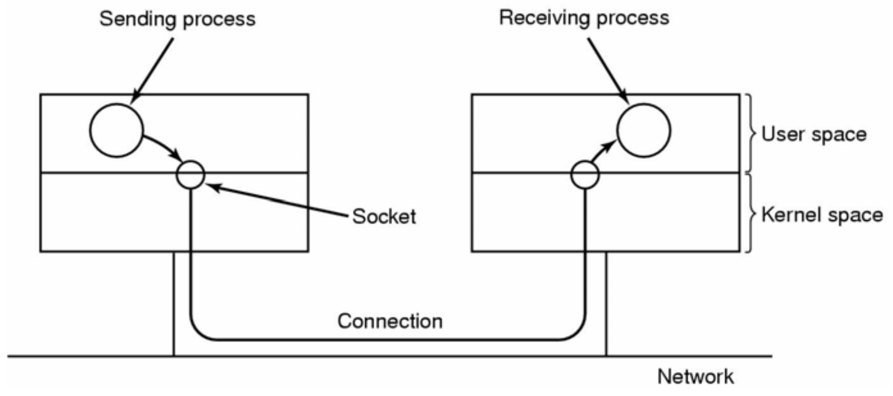
Linux uses three-level page tables

Paging in Linux (2)



Buddy algorithm

Networking



Use of sockets for networking

Terminal Management

Function call	Description
s = cfsetospeed(&termios, speed)	Set the output speed
s = cfsetispeed(&termios, speed)	Set the input speed
s = cfgetospeed(&termios, speed)	Get the output speed
s = cfgtetispeed(&termios, speed)	Get the input speed
s = tcsetattr(fd, opt, &termios)	Set the attributes
s = tcgetattr(fd, &termios)	Get the attributes

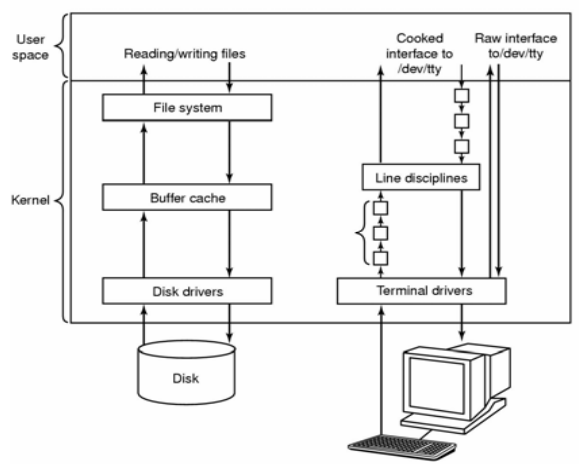
The main POSIX calls for managing the terminal

UNIX I/O (1)

Device	Open	Close	Read	Write	Ioctl	Other
Null	null	null	null	null	null	...
Memory	null	null	mem_read	mem_write	null	...
Keyboard	k_open	k_close	k_read	error	k_ioctl	...
Tty	tty_open	tty_close	tty_read	tty_write	tty_ioctl	...
Printer	lp_open	lp_close	error	lp_write	lp_ioctl	...

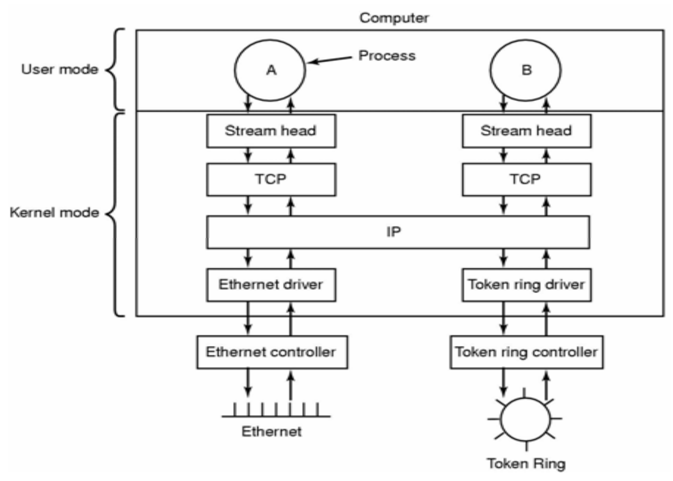
Some of the fields of a typical *cdevsw* table

UNIX I/O (2)



The UNIX I/O system in BSD

Streams



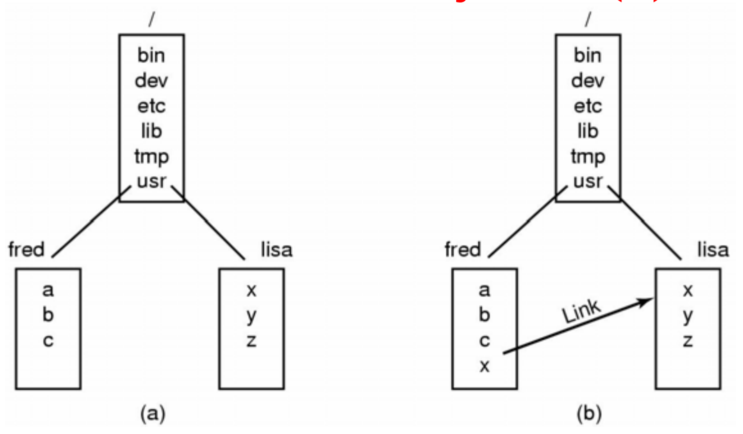
An example of streams in System V

The UNIX File System (1)

Directory	Contents
bin	Binary (executable) programs
dev	Special files for I/O devices
etc	Miscellaneous system files
lib	Libraries
usr	User directories

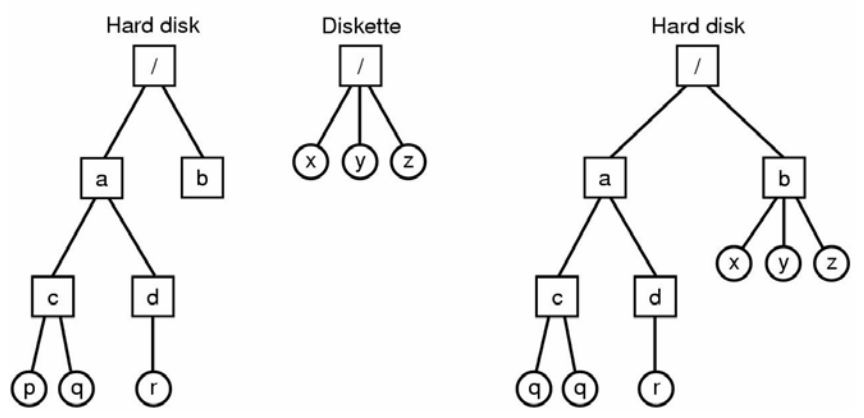
Some important directories found in most UNIX systems

The UNIX File System (2)



(a) Before linking. (b) After linking

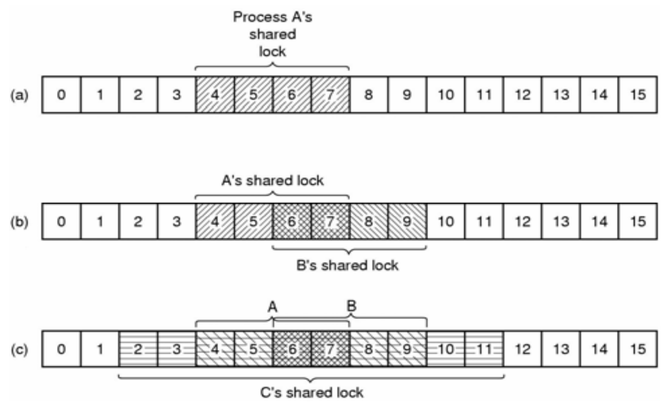
The UNIX File System (3)



(a) (b)

(a) Before mounting. (b) After mounting

Locking Files



(a) File with one lock
(b) Addition of a second lock
(c) A third lock

System Calls for File Management

System call	Description
fd = creat(name, mode)	One way to create a new file
fd = open(file, how, ...)	Open a file for reading, writing or both
s = close(fd)	Close an open file
n = read(fd, buffer, nbytes)	Read data from a file into a buffer
n = write(fd, buffer, nbytes)	Write data from a buffer into a file
position = lseek(fd, offset, whence)	Move the file pointer
s = stat(name, &buf)	Get a file's status information
s = fstat(fd, &buf)	Get a file's status information
s = pipe(&fd[0])	Create a pipe
s = fcntl(fd, cmd, ...)	File locking and other operations

- **s** is an error code
- **fd** is a file descriptor
- **position** is a file offset

The lstat System Call

Device the file is on
I-node number (which file on the device)
File mode (includes protection information)
Number of links to the file
Identity of the file's owner
Group the file belongs to
File size (in bytes)
Creation time
Time of last access
Time of last modification

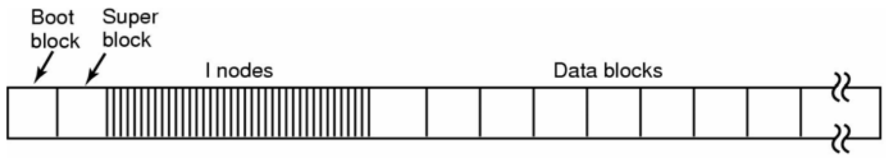
Fields returned by the lstat system call.

System Calls for Directory Management

System call	Description
s = mkdir(path, mode)	Create a new directory
s = rmdir(path)	Remove a directory
s = link(oldpath, newpath)	Create a link to an existing file
s = unlink(path)	Unlink a file
s = chdir(path)	Change the working directory
dir = opendir(path)	Open a directory for reading
s = closedir(dir)	Close a directory
dirent = readdir(dir)	Read one directory entry
rewinddir(dir)	Rewind a directory so it can be reread

- **s** is an error code
- **dir** identifies a directory stream
- **dirent** is a directory entry

UNIX File System (1)



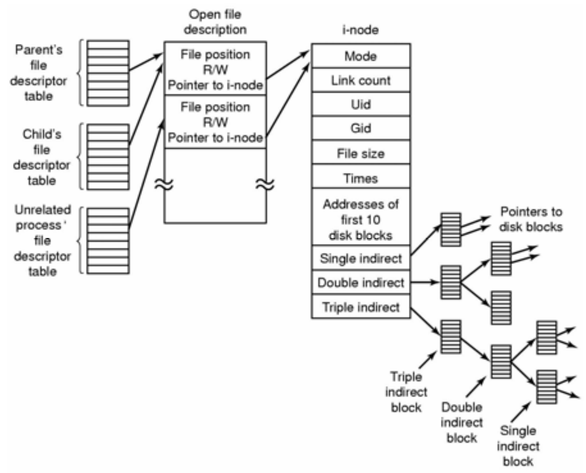
Disk layout in classical UNIX systems

UNIX File System (2)

Field	Bytes	Description
Mode	2	File type, protection bits, setuid, setgid bits
Nlinks	2	Number of directory entries pointing to this i-node
Uid	2	UID of the file owner
Gid	2	GID of the file owner
Size	4	File size in bytes
Addr	39	Address of first 10 disk blocks, then 3 indirect blocks
Gen	1	Generation number (incremented every time i-node is reused)
Atime	4	Time the file was last accessed
Mtime	4	Time the file was last modified
Ctime	4	Time the i-node was last changed (except the other times)

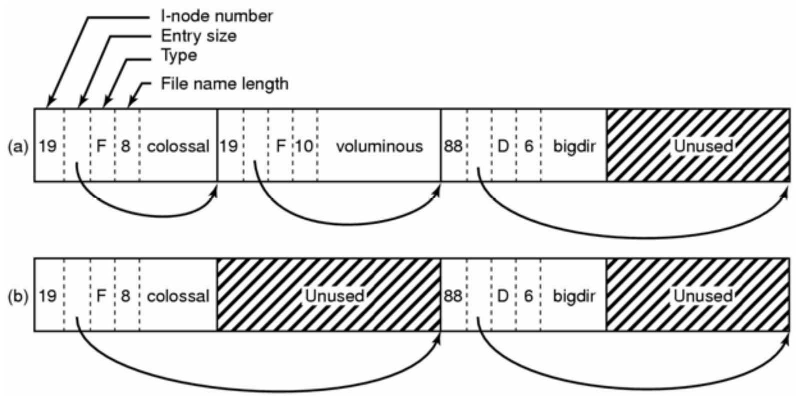
Structure of the i-node

UNIX File System (3)



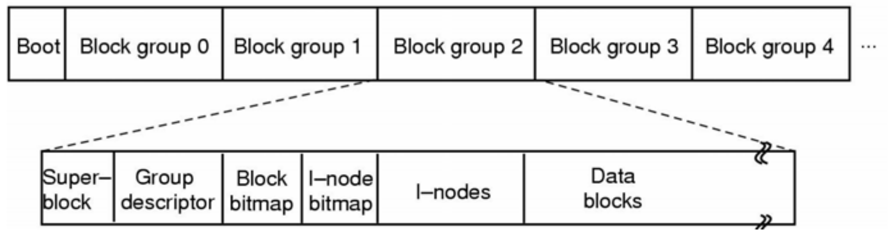
The relation between the file descriptor table, the open file description

UNIX File System (4)



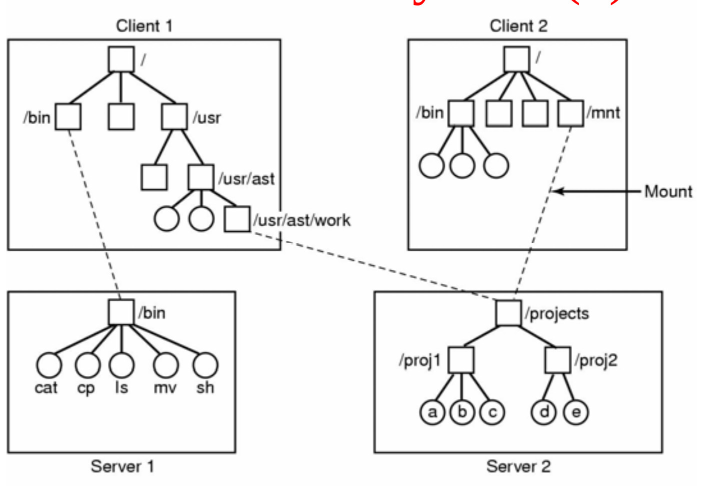
- A BSD directory with three files
- The same directory after the file *voluminous* has been removed

The Linux File System



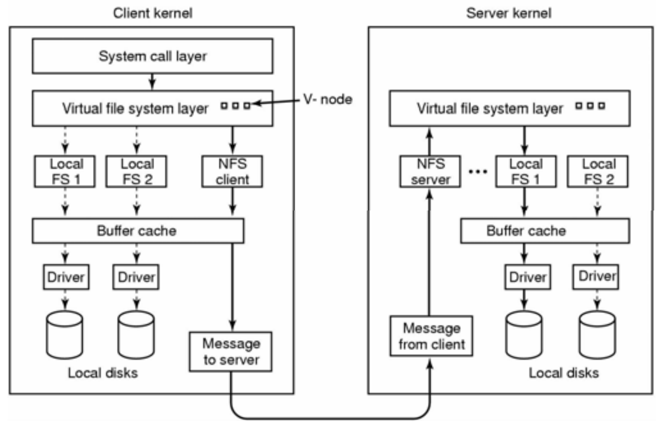
Layout of the Linux Ex2 file system.

Network File System (1)



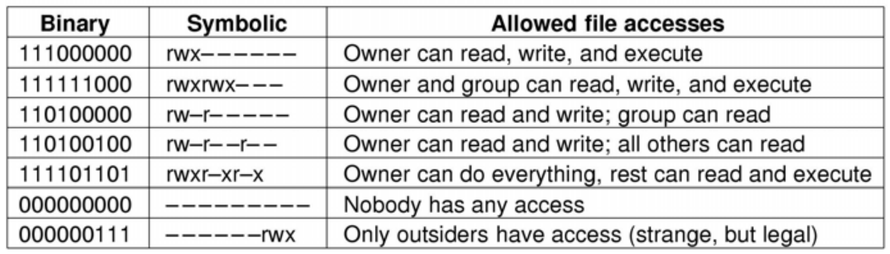
- Examples of remote mounted file systems
- Directories are shown as squares, files as circles

Network File System (2)



The NFS layer structure

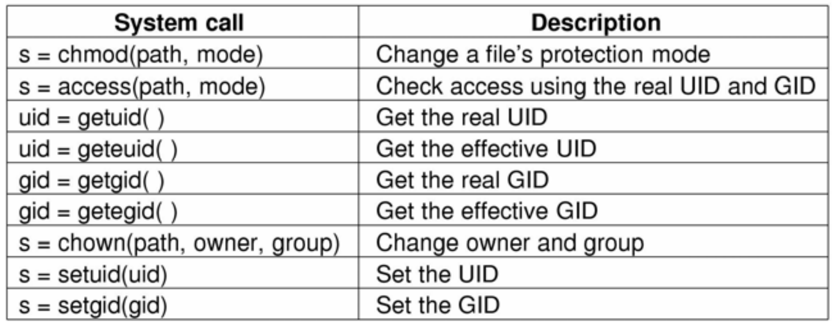
Security in UNIX



Binary	Symbolic	Allowed file accesses
111000000	rwx––––––	Owner can read, write, and execute
111111000	rwxrwx–––	Owner and group can read, write, and execute
110100000	rw–r–––––	Owner can read and write; group can read
110100100	rw–r––r––	Owner can read and write; all others can read
111101101	rwxr–xr–x	Owner can do everything, rest can read and execute
000000000	–––––––––	Nobody has any access
000000111	––––––rwx	Only outsiders have access (strange, but legal)

Some examples of file protection modes

System Calls for File Protection



System call	Description
s = chmod(path, mode)	Change a file's protection mode
s = access(path, mode)	Check access using the real UID and GID
uid = getuid( )	Get the real UID
uid = geteuid( )	Get the effective UID
gid = getgid( )	Get the real GID
gid = getegid( )	Get the effective GID
s = chown(path, owner, group)	Change owner and group
s = setuid(uid)	Set the UID
s = setgid(gid)	Set the GID

- **s** is an error code
- **uid** and **gid** are the UID and GID, respectively