

Chapter 10

File System Interface

Chapter 10: File System Interface

- 10.1 File Concept
- 10.2 Access Methods
- 10.3 Directory Structure
- 10.4 File-System Mounting
- 10.5 File Sharing (skip)
- 10.6 File Protection

Objectives

- To explain the function of file systems
- To describe the interfaces to file systems
- To discuss file-system design tradeoffs, including access methods, file sharing, file locking, and directory structures
- To explore file-system protection

10.1 File Concept

Introduction

- Computers can store information on various storage media
- The operating system abstracts from the physical properties of its storage devices to define a logical storage unit called a **file**
- Files are **mapped** by the operating system onto nonvolatile physical devices
- A file is a **named collection of related information** that is recorded on secondary storage
- For a user's perspective, a file is the **smallest allotment** of logical secondary storage (i.e., data supposedly cannot be written to secondary storage unless written in a file)
- Files commonly represent programs (both source and object code) and data
- Data file contents may be numeric, alphabetic, alphanumeric, or binary
- Files may be **free form** (e.g., text files) or **rigidly formatted**
- In general, a file is a **sequence** of bits, bytes, lines, or records, whose meaning is defined by the file's creator and user

File Structure

- A file has a certain **defined** structure, which depends on its type
- A **text file** is a sequence of characters organized into lines
- A **source code file** is a sequence of declarations, statements, and subroutine definitions
- An **object code file** and an **executable file** each contain a sequence of bytes organized into headers and tables of data and code understandable by a system linker and loader
- A file is **named** for the convenience of its human users, and is referred to by its name
- After a file is created, it becomes **independent** of the process that created it; other processes may read it or edit it

File Attributes

- **Name** – the only information kept in human-readable form
- **Identifier** – unique tag (a number) that identifies file within the file system
- **Type** – used by systems that support different types of files
- **Location** – pointer to file location on a device
- **Size** – current file size in bytes, words, or blocks
- **Protection** – access controls for who can read, write, and execute
- **Time, date, and user identification** – used for documenting file creation, last modification, and last use
- Information about all files is kept in the directory structure, which also is located on secondary storage

File Operations

- A file is an **abstract data type** with operations to
 - **Create**
 - **Write**
 - **Read**
 - **Reposition within file**
 - **Delete**
 - **Truncate (i.e., erase the contents but keep the file)**

Open Files

- Several pieces of data are needed to manage open files:
 - **File pointer:** pointer to last read/write location; unique to **each process** that has the file open
 - **File-open count:** count of number of **simultaneous** opens for a file to allow removal of the file entry from the open-file table when the last process closes it
 - **Disk location of the file:** Information kept in memory and used to locate the file on disk
 - **Access rights:** Access mode information stored for each process

Open File Locking

- Provided by some operating systems and file systems
- Mediates access to a file
- Mandatory or advisory:
 - **Mandatory** – access is denied depending on locks held and requested
 - **Advisory** – processes can find status of locks and decide what to do

File Types – Name, Extension

file type	usual extension	function
executable	exe, com, bin or none	ready-to-run machine-language program
object	obj, o	compiled, machine language, not linked
source code	c, cc, java, pas, asm, a	source code in various languages
batch	bat, sh	commands to the command interpreter
text	txt, doc	textual data, documents
word processor	wp, tex, rtf, doc	various word-processor formats
library	lib, a, so, dll	libraries of routines for programmers
print or view	ps, pdf, jpg	ASCII or binary file in a format for printing or viewing
archive	arc, zip, tar	related files grouped into one file, sometimes compressed, for archiving or storage
multimedia	mpeg, mov, rm, mp3, avi	binary file containing audio or A/V information

10.2 Access Methods

Access Methods

□ Sequential Access

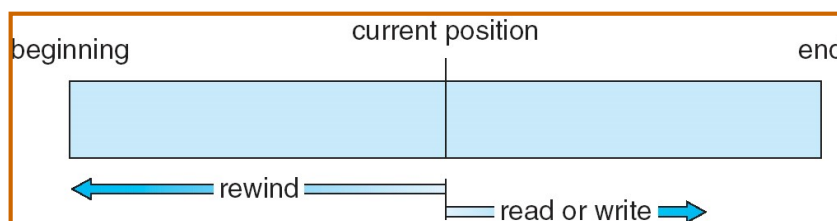
read next
write next
reset
skip forward
skip backward

□ Direct Access

read n
write n
position to n

n = relative block number

Sequential-access File

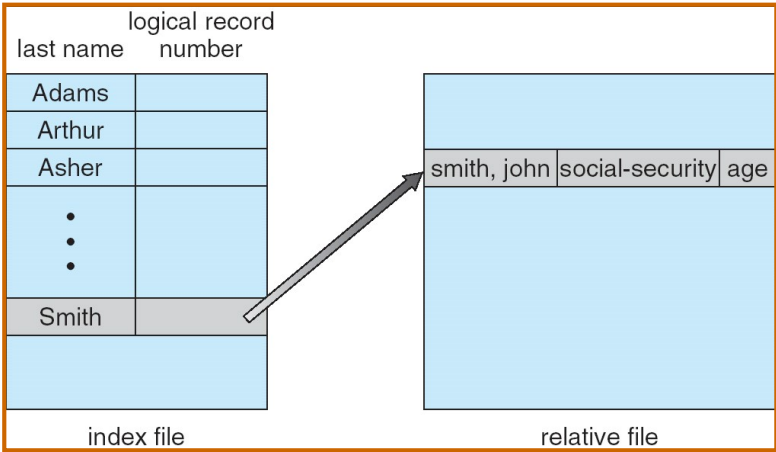


Simulation of Sequential Access on a Direct-access File

sequential access	implementation for direct access
<i>reset</i>	<i>cp</i> = 0;
<i>read next</i>	<i>read cp</i> ; <i>cp</i> = <i>cp</i> + 1;
<i>write next</i>	<i>write cp</i> ; <i>cp</i> = <i>cp</i> + 1;

cp = current position

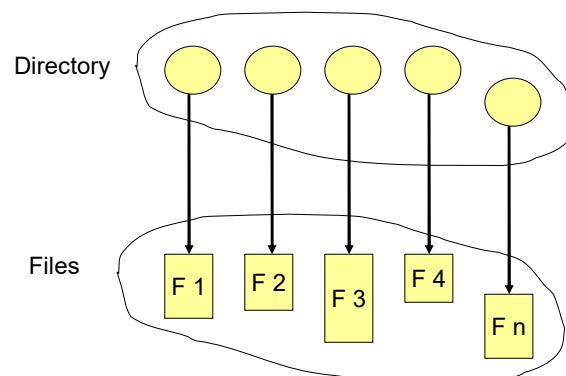
Example of Index and Relative Files



10.3 Directory Structure

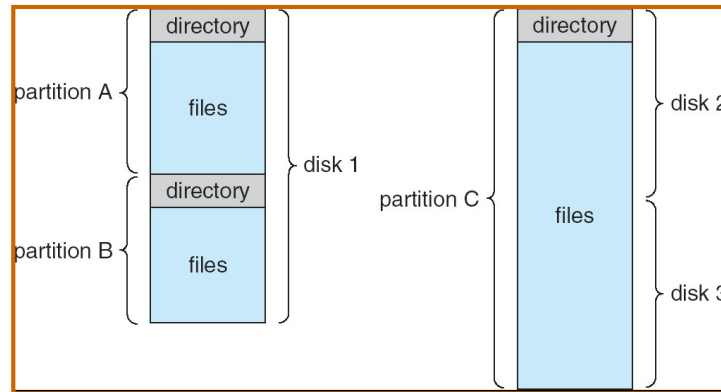
Directory Structure

- A collection of nodes containing information about all files



Both the directory structure and the files reside on disk.
Backups of these two structures may be kept on tapes.

A Typical File-system Organization



Operations Performed on Directory

- ❑ Search for a file
- ❑ Create a file
- ❑ Delete a file
- ❑ List a directory
- ❑ Rename a file
- ❑ Traverse the file system

Goal: Organize a directory (logically) based on the following criteria:

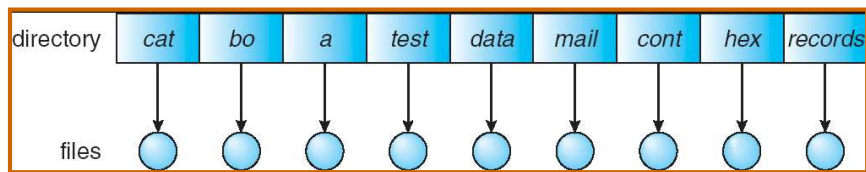
- **Efficiency** – locating a file quickly
- **Naming** – convenient to users
 - Two users can have **same name** for different files
 - The same file can have several **different names**
- **Grouping** – logical grouping of files by properties, (e.g., all Java programs, all games, ...)

Four Possible Approaches

- Single-level Directory
- Two-level Directory
- Tree-structured Directories
- Acyclic Graph Directories

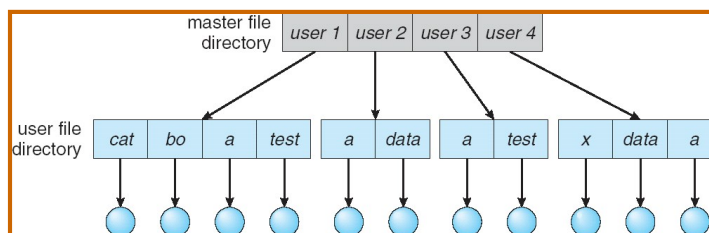
Single-Level Directory

- A single directory maintained for all users
- Advantages
 - Efficiency
- Disadvantages
 - Naming - collisions
 - Grouping – not possible

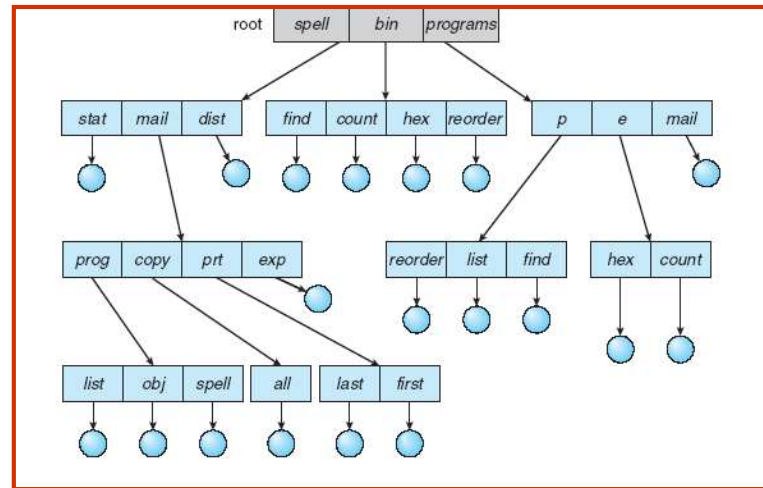


Two-Level Directory

- Separate directory for each user
- Advantages
 - Efficient searching
 - Naming - same file name in a different directory; path name
- Disadvantages
 - Naming - collisions
 - Grouping capability – not possible except by user



Tree-Structured Directories



Tree-Structured Directories (Cont)

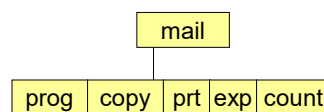
- Advantages
 - Efficient searching – current working directory
 - Naming - same file name in a different directory
 - Grouping capability
- Disadvantages
 - Structural complexity

Tree-Structured Directories (Cont)

- **Absolute** or **relative** path name
- Creating a new file is done in current directory
- Delete a file
`rm <file-name>`
- Creating a new subdirectory is done in current directory
`mkdir <dir-name>`

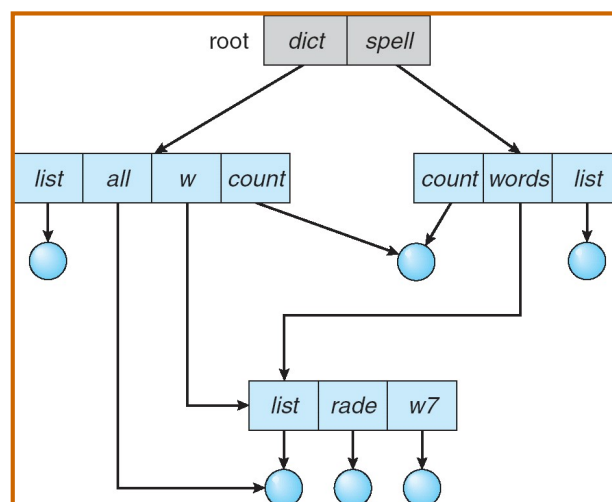
Example: if in current directory `/mail`

`mkdir count`



Acyclic-Graph Directories

- Have shared subdirectories and files



Acyclic-Graph Directories (Cont.)

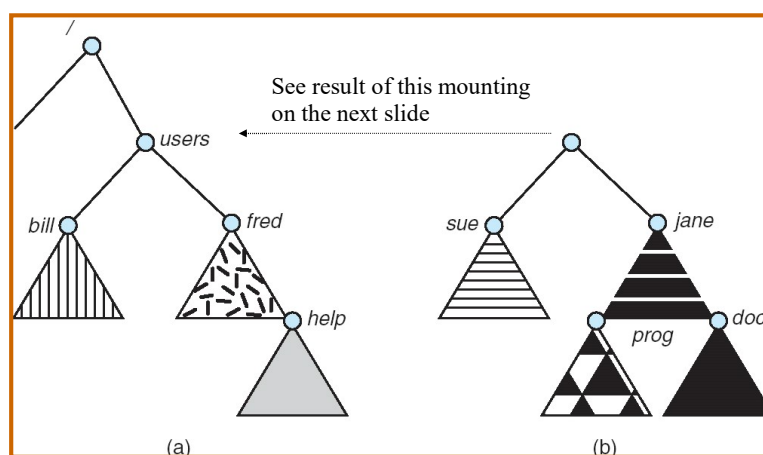
- Same advantages as tree-structured directory
 - In addition, the same file or directory may have a reference that appears in two or more directories
- Disadvantage is that its structure is more complex
 - The same file or directory may be referred to by many names
 - Need to be cautious of dangling pointers when files are deleted
- Solutions to deletion
 - Just delete the link
 - Preserve the file until all links (i.e., references) are deleted
- This requires a new directory entry type called a **link**
 - Keep a count of the number of references

10.4 File System Mounting

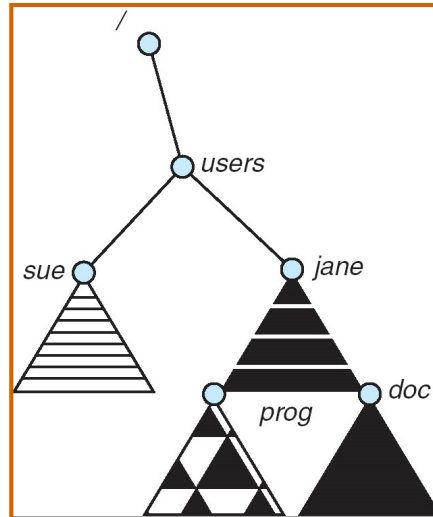
File System Mounting

- A file system must be **mounted** before it can be accessed
- An unmounted file system is mounted at a **mount point** (see the next two slides)

(a) Existing. (b) Unmounted Partition



Mount Point



Common Mount Points

- ❑ Unix or Linux – any directory
 - ❑ `/` - root directory
 - ❑ `/floppy` – floppy drive
 - ❑ `/cdrom` – cdrom drive
 - ❑ `/dev` – device “drivers”
 - ❑ `/home` – user directories
- ❑ Windows XP – drive letters (examples only)
 - ❑ A:, B: - floppy drive
 - ❑ C: - hard drive
 - ❑ D: - CD-ROM drive
 - ❑ E: - USB drive
 - ❑ F: - network drive

10.5 File Sharing (Optional, Skip over)

10.6 File Protection

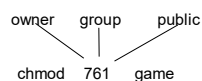
Protection

- File owner/creator should be able to control:
 - what can be done to a file
 - by whom
- Types of access
 - **Read**
 - **Write**
 - **Execute**
 - **Append**
 - **Delete**
 - **List**

Access Lists and Groups

- Mode of access: read, write, execute
- Three classes of users

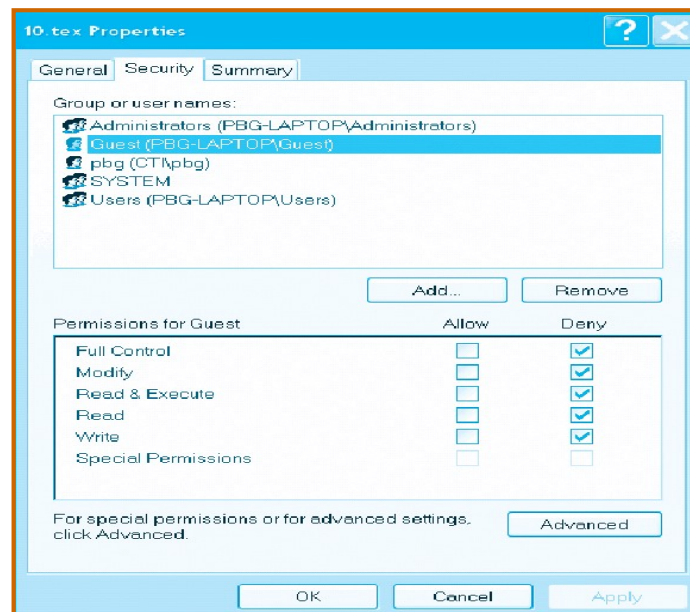
a) owner access	7	RWX	⇒	1 1 1
b) group access	6	RWX	⇒	1 1 0
c) public access	1	RWX	⇒	0 0 1
- A system administrator creates a group (unique name), say G, and adds some users to the group.
- For a particular file (say *game*) or subdirectory, define an appropriate access.



Attach a group to a file

chgrp G game

Windows XP Access-control List Management



A Sample UNIX Directory Listing

-rw-rw-r--	1	pbq	staff	31200	Sep 3 08:30	intro.ps
drwx-----	5	pbq	staff	512	Jul 8 09:33	private/
drwxrwxr-x	2	pbq	staff	512	Jul 8 09:35	doc/
drwxrwx---	2	pbq	student	512	Aug 3 14:13	student-proj/
-rw-r--r--	1	pbq	staff	9423	Feb 24 2003	program.c
-rwxr-xr-x	1	pbq	staff	20471	Feb 24 2003	program
drwx--x--x	4	pbq	faculty	512	Jul 31 10:31	lib/
drwx-----	3	pbq	staff	1024	Aug 29 06:52	mail/
drwxrwxrwx	3	pbq	staff	512	Jul 8 09:35	test/

End of Chapter 10

Chapter 11

File System Implementation

Chapter 11: File System Implementation

- 11.1 File-System Structure
- 11.2 File-System Implementation
- 11.3 Directory Implementation
- 11.4 Allocation Methods
- 11.5 Free-Space Management
- 11.6 Efficiency and Performance
- 11.7 Recovery
- 11.8 Log-Structured File Systems (skip)
- 11.9 NFS

Objectives

- To describe the details of implementing local file systems and directory structures
- To describe the implementation of remote file systems
- To discuss block allocation and free-block algorithms and trade-offs

11.1 File-System Structure

Advantages of Disk Storage

- Disks provide the bulk of secondary storage on which a file system is maintained
- This occurs because of two characteristics
 - A disk can be written in place; it is possible to read a block from the disk, modify the block, and write it back into the same place
 - A disk can access directly any given block of information it contains
 - ▶ It is simple to access any file either sequentially or randomly
 - ▶ Switching from one file to another requires only moving the read-write heads and waiting for the disk to rotate

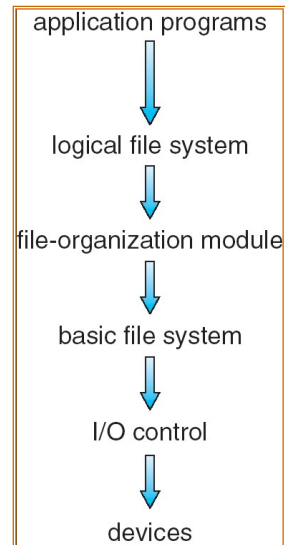


File System

- A file system provides efficient and convenient access to the disk
- The operating system imposes one or more file systems on a disk
- A file system poses two different design problems
 - Defining how the file system should look to the user
 - ▶ A file and its attributes
 - ▶ The operations allowed on a file
 - ▶ The directory structure for organizing files
 - Creating algorithms and data structures to map the logical file system onto the physical secondary-storage devices

Layered File System

- Application programs
- Logical file system
 - Manages the directory structure
 - Maintains the file structure via file control blocks
- File-organization module
 - Maps the logical blocks to the physical blocks
 - Manages the free space/blocks on the disk
- Basic file system
 - Issues generic commands to the appropriate device driver to read and write physical blocks on the disk
- I/O control
 - Consists of device drivers and interrupt handlers



Layered File System (continued)

A file control block (FCB) contains information about the file, including ownership, permissions, and location of the file contents

file permissions
file dates (create, access, write)
file owner, group, ACL
file size
file data blocks or pointers to file data blocks

Layered File System (continued)

- Advantage of a layered structure: Duplication of code is minimized
 - I/O control and basic file-system code can be used by multiple file systems
 - Each file system can have its own logical file system and file-organization modules
- Many kinds of file systems are in use today
 - CD-ROMs are written in the ISO 9660 format
 - UNIX uses the UNIX file system (UFS)
 - Windows NT, 2000, and XP support
 - FAT, FAT32, and NTFS
 - See: <http://www.ntfs.com/#ntfs%20fat>
 - See: http://www.ntfs.com/ntfs_vs_fat.htm
 - CD-ROM, DVD, and floppy-disk file-system formats
 - Linux supports over 40 different file systems
 - Main file system is the extended file system (versions ext2 and ext3)
 - There are also distributed file systems in which a file system on a server computer can be mounted by one or more client computers

11.2 File-System Implementation

Overview

- Several on-disk and in-memory structures are used to implement a file system
- These structures vary depending on the operating system and the file system, but some general principles apply
- For on-disk structures, the file system may contain
 - Information about how to boot an operating system stored there
 - The total number of blocks
 - The number and location of free blocks
 - The directory structure
 - Individual files
- In-memory information is used for both file-system management and performance improvement via caching
 - The data are loaded at mount time and discarded at dismount

Overview (on-disk structures)

- Boot control block (per volume)
 - Can contain information needed by the system to boot an operating system
 - It is typically the first block of a volume
 - In UFS, it is called the boot block
 - In NTFS, it is the partition boot sector
- Volume control block (per volume)
 - Contains volume or partition details such as the number of blocks in the partition, size of the blocks, free block count and free-block pointers, and free FCB count and FCB pointers
 - In UFS, this is called a superblock
 - In NTFS, it is stored in the master file table

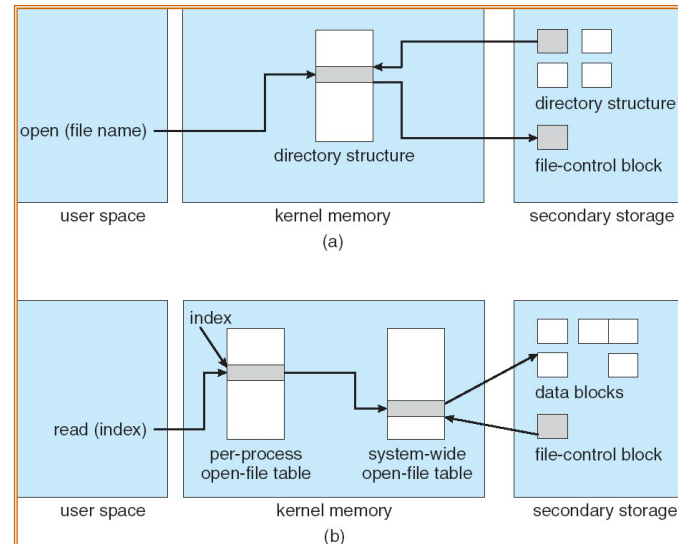
Overview (on-disk structures)

- A directory structure per file system is used to organize the files
 - In UFS, this includes the file names and associated inode numbers
 - In NTFS, it is stored in the master file table
- A per-file File Control Block (FCB) contains many details about the file
 - In UFS, this is called the inode
 - In NTFS, this information is stored in the master file table, with one row per file

Overview (in-memory structures)

- In-memory mount table
 - Contains information about each mounted volume
- In-memory directory-structure cache
 - Holds the directory information of recently accessed directories
- System-wide open-file table
 - Contains a copy of the FCB of each open file, as well as other information
- Per-process open-file table
 - Contains a pointer to the appropriate entry in the system-wide open-file table

In-Memory File System Structures



Steps for creating a new file

- 1) An application program calls the logical file system, which knows the format of the directory structures
 - 2) The logical file system allocates a new file control block (FCB)
 - If all FCBs are created at file-system creation time, an FCB is allocated from the free list
 - 3) The logical file system then
 - a) Reads the appropriate directory into memory
 - b) Updates the directory with the new file name and FCB
 - c) Writes the directory back to the disk
- UNIX treats a directory exactly the same as a file by means of a type field in the inode
 - Windows NT implements separate system calls for files and directories and treats directories as entities separate from files

Steps for opening a file using open()

- 1) The function first searches the system-wide open-file table to see if the file is already in use by another process
 - If it is, a per-process open-file table entry is created pointing to the existing system-wide open-file table
 - This algorithm can have substantial overhead; consequently, parts of the directory structure are usually cached in memory to speed operations
- 2) Once the file is found, the FCB is copied into a system-wide open-file table in memory
 - This table also tracks the number of processes that have the file open
- 3) Next, an entry is made in the per-process open-file table, with a pointer to the entry in the system-wide open-file table
- 4) The function then returns a pointer/index to the appropriate entry in the per-process file-system table
 - All subsequent file operations are then performed via this pointer
 - UNIX refers to this pointer as the file descriptor
 - Windows refers to it as the file handle

Steps for closing a file using close()

- 1) The per-process table entry is removed
- 2) The system-wide entry's open count is decremented
- 3) When all processes that have opened the file eventually close it
 - Any updated metadata is copied back to the disk-based directory structure
 - The system-wide open-file table entry is removed


Partitions and mounting

- Each partition can be either raw, containing no file system, or formatted as a file system (i.e., cooked)
- A raw partition is used where no file system is appropriate
 - UNIX swap space can use a raw partition
 - Some databases use raw partitions and format the data to suit their needs
 - Raw partitions can also be used to implement RAID systems (Redundant Array of Inexpensive Disks)
- Boot information can be stored in a separate boot partition
 - It has its own format because the file-system device drivers are not loaded yet
 - Boot information is usually a sequential series of blocks loaded as an image into memory
 - The boot image can contain code to perform a dual boot if multiple operating systems are stalled on the same disk

Partitions and mounting (continued)

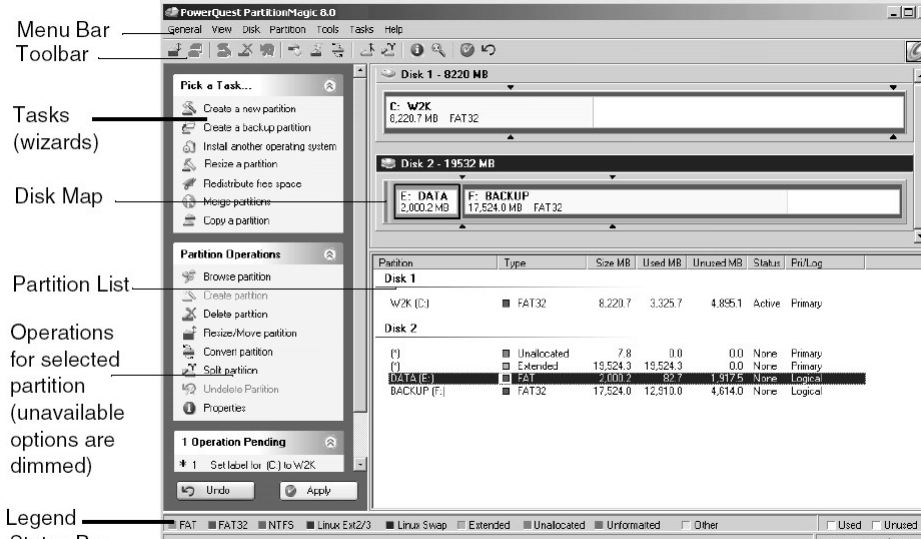
- The root partition is mounted at boot time
 - It contains the operating-system kernel and possibly other system files
- Other volumes can be automatically mounted at boot time or manually mounted later
- As part of a successful mount operation, the operating system verifies that the storage device contains a valid file system
 - It asks the device driver to read the device directory and verify that the directory has the expected format
 - If the format is invalid, the partition must have its consistency checked and possibly corrected
 - Finally, the operating system notes in its in-memory mount table structure that a file system is mounted along with the type of the file system
 - ▶ Microsoft Windows-based systems mount each volume in a separate name space denoted by a letter and a colon (e.g., C:)
 - The operating system places a pointer to the file system in the field of the device structure corresponding to the drive letter
 - ▶ UNIX allows file systems to be mounted at any directory
 - Mounting is implemented by setting a flag in the in-memory copy of the inode for the directory

Example Partitioning of a Disk



Disk	Partition	Type	Size MB	Used MB	Unused MB	Status	Pri/Log
1	C: root	NTFS	25,640.5	23,501.2	2,139.2	Active	Primary
1	*	Extended	12,521.2	12,521.2	0.0	None	Primary
1	*	Linux Ext2	12,004.4	12,004.4	0.0	None	Logical
1	*SWAPSPACE2	Linux Swap	516.8	0.0	516.8	None	Logical

Example Disk Partitioning Software



Menu Bar

Toolbar

Tasks (wizards)

Disk Map

Partition List

Operations for selected partition (unavailable options are dimmed)

Legend

Status Bar

1 Operation Pending

* 1 Set label for (C:) to W2K

Undo Apply

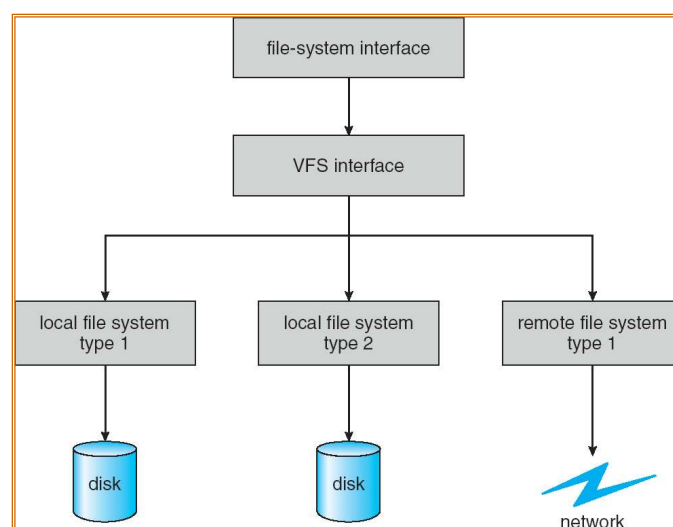
Legend: FAT, FAT32, NTFS, Linux Ext2/3, Linux Swap, Extended, Unallocated, Unformatted, Other

Status Bar: 1 operation pending

Virtual File Systems

- Virtual File Systems (VFS) provide an object-oriented way of implementing file systems.
- VFS allows the same system call interface (the API) to be used for different types of file systems.
- The API is to the VFS interface, rather than any specific type of file system.

Schematic View of Virtual File System



11.3 Directory Implementation

Directory Implementation

- **Selection of directory allocation and directory management algorithms significantly affects the efficiency, performance, and reliability of the file system**
- **One Approach: Direct indexing of a linear list**
 - Consists of a list of file names with pointers to the data blocks
 - Simple to program
 - Time-consuming to search because it is a linear search
 - Sorting the list allows for a binary search; however, this may complicate creating and deleting files
- **Another Approach: List indexing via a hash function**
 - Takes a value computed from the file name and returns a pointer to the file name in the linear list
 - Greatly reduces the directory search time
 - **Can result in collisions** – situations where two file names hash to the same location
 - Has a fixed size (i.e., fixed number of entries)
- **A Third Approach: Non-linear structure such as a B-tree (i.e., (a,b) tree)**

11.4 Allocation Methods

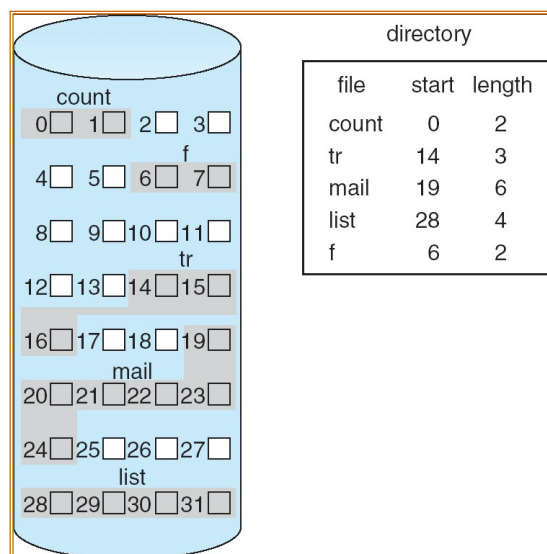
Allocation Methods

- Allocation methods address the problem of allocating space to files so that disk space is utilized effectively and files can be accessed quickly
- Three methods exist for allocating disk space
 - **Contiguous allocation**
 - **Linked allocation**
 - **Indexed allocation**

Contiguous Allocation

- Requires that each file occupy a set of contiguous blocks on the disk
- Accessing a file is easy – only need the starting location (block #) and length (number of blocks)
- Problems
 - Finding space for a new file (first fit, best fit, etc.)
 - External fragmentation (free space is broken into small unusable chunks)
 - Need for compaction, which requires file system down time
 - Determining space for a file, especially if it needs to grow

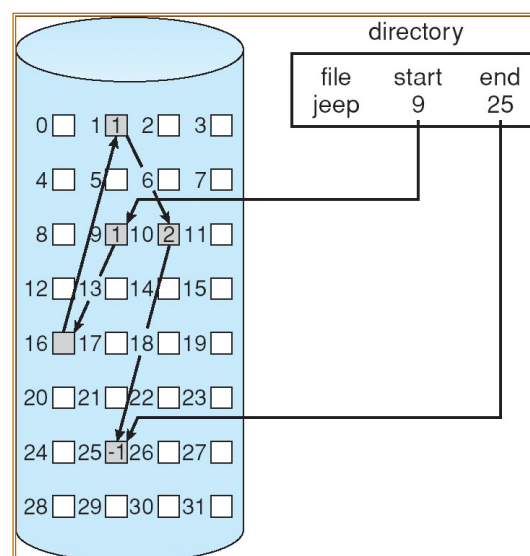
Contiguous Allocation (continued)



Linked Allocation

- Solves the problems of contiguous allocation
- Each file is a linked list of disk blocks: blocks may be scattered anywhere on the disk
- The directory contains a pointer to the first and last blocks of a file
- Creating a new file requires only creation of a new entry in the directory
- Writing to a file causes the free-space management system to find a free block
 - This new block is written to and is linked to the end of the file
- Reading from a file requires only reading blocks by following the pointers from block to block
- Advantages
 - There is no external fragmentation
 - Any free blocks on the free list can be used to satisfy a request for disk space
 - The size of a file need not be declared when the file is created
 - A file can continue to grow as long as free blocks are available
 - ▶ It is never necessary to compact disk space for the sake of linked allocation (however, file access efficiency may require it)

Linked Allocation (continued)



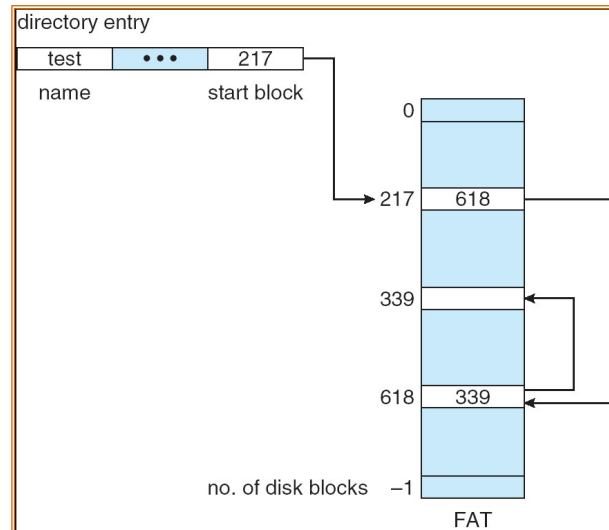
Linked Allocation (continued)

- Disadvantages
 - Can only be used effectively for sequential access of files
 - ▶ Each access to a file block requires a disk access, and some may also require a disk seek
 - ▶ It is inefficient to support direct access capability
 - Disk space is required to store the block pointers
 - ▶ One solution is the clustering of a certain constant number of blocks (e.g., 4)
 - Relies on the integrity of the links – an error might result in a pointer value becoming corrupt and then pointing into the free-space list or to the blocks of another file
 - ▶ A partial solution is a doubly linked list or storing a relative block number or the file name in each block (these schemes all require space and algorithm overhead)

Linked Allocation (continued)

- One variation on linked allocation is the file allocation table (FAT) used by MS-DOS
 - A section of the disk on each volume contains the FAT
 - The FAT has one entry for each disk block and is indexed by block number
 - The directory entry contains the block number of the first block of the file
 - The table entry indexed by the block number contains the block number of the next block in the file
 - The last block contains a special end-of-file value as the table entry
 - Unused blocks are indicated by a zero table value
 - To allocate a new block to a file
 - ▶ Find the first zero-valued table entry
 - ▶ Replace the previous end-of-file value with the address of the new block
 - Disadvantage – can result in a significant number of disk head seeks
 - Advantage – random-access time is improved because the FAT can be checked

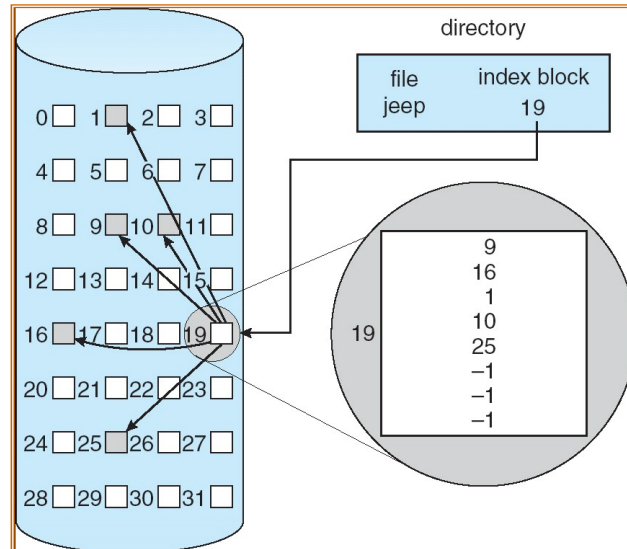
File-Allocation Table (FAT)



Indexed Allocation

- Solves the problems of linked allocation by bringing all the pointers (for a file's blocks) together into one location called the index block
- Each file has its own index block, which is an array of disk-block addresses
- Each entry in the index block points to the corresponding block of the file
- The directory contains the address of the index block
- Finding and reading a specific block in a file only requires the use of the pointer in the index block
- When a file is created
 - The pointer to the index block is set to nil
 - When a new block is first written, it is obtained from the free-space management system and its address is put in the index block
- Supports direct access without suffering from external fragmentation
- Requires the additional space of an index block for each file
- Disadvantages
 - Suffers from some of the same performance problems as linked allocation
 - Index blocks can be cached in memory; however, data blocks may be spread all over the disk volume

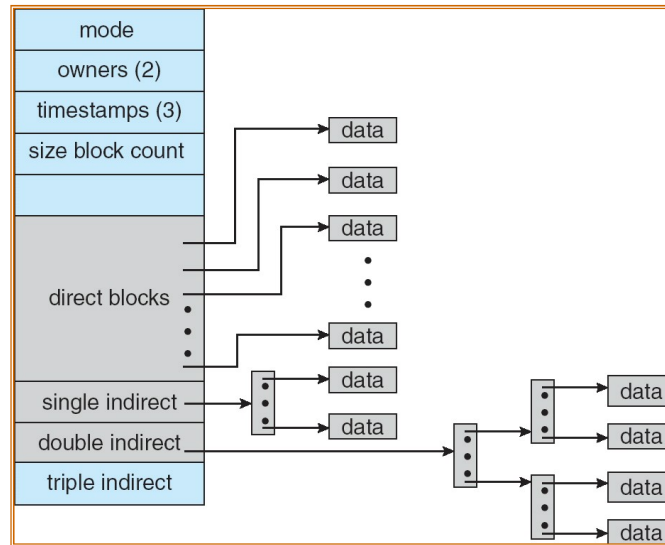
Example of Indexed Allocation



Sizing of the Index Block

- Sizing of the index block is accomplished using various mechanisms
- Approach #1: Linked scheme
 - Several index blocks can be linked together
- Approach #2: Multilevel index scheme
 - A first-level index block points to a set of second-level index blocks
- Approach #3: Combined scheme
 - Used in the UNIX file system (UFS)
 - A specific set of pointers points directly to file blocks
 - Three special pointers point to a single indirect block, a double indirect block, and a triple indirect block
 - Using such an approach, the number of blocks that can be allocated to a file exceeds the amount of addressable space by many operating systems
 - Files can be terabytes in size

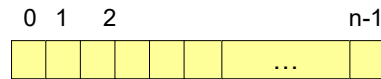
Combined Scheme: UNIX (4K bytes per block)



11.5 Free-Space Management

Bit Vector Approach

- Free-space blocks are tracked via a bit vector (where $n = \text{\#blocks}$)

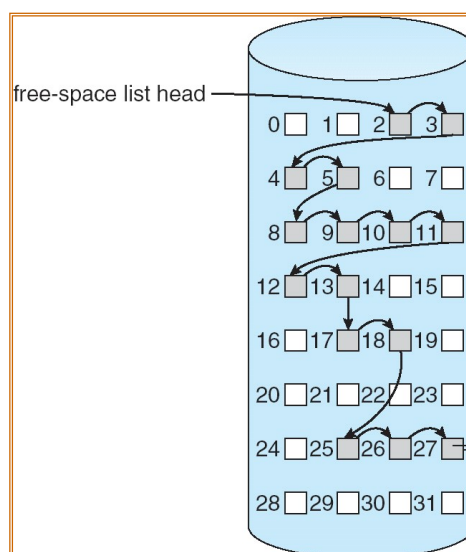


$$\text{bit}[i] = \begin{cases} 0 \Rightarrow \text{block}[i] \text{ allocated} \\ 1 \Rightarrow \text{block}[i] \text{ free} \end{cases}$$

- Calculating the number of a free block:

(number of 0-value words) * (number of bits per word) + offset of first 1 bit in a word

Linked List Approach



Free-Space Management Approaches

- Bit vector
 - Advantage: easy to get contiguous blocks for a file
 - Disadvantage: requires extra space to store the bit vector
- Linked list (free list)
 - Link together all the free disk blocks and keep a pointer to the first free block
 - Advantage: no waste of space
 - Disadvantage: cannot get contiguous blocks easily for a file
- Grouping (modification of free list approach)
 - Store the addresses of n free blocks in the first free block
 - The first $n - 1$ of these blocks are actually free
 - The last block contains the addresses of another n free blocks
- Counting
 - Take advantage of the fact that several contiguous blocks may be allocated or freed simultaneously
 - Keep the address of the first free block and the number n of free contiguous blocks that follow the first block

11.6 Efficiency and Performance

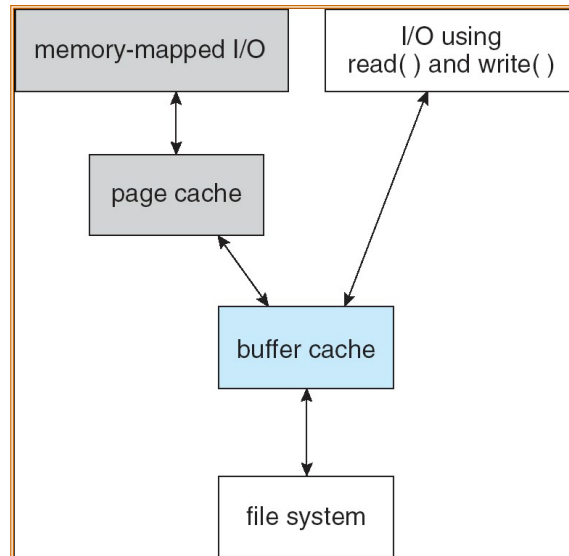
Efficiency and Performance

- Need to consider the effect of various block allocation and directory management approaches on performance and efficient disk use
- Efficiency dependent on:
 - Disk allocation and directory algorithms
 - Types of data kept in each file's directory entry
- Performance
 - Disk cache – separate section of main memory for frequently used blocks
 - Techniques to optimize sequential access
 - Free-behind: remove a page from the cache as soon as the next page is requested
 - Read-ahead: read requested page and several subsequent pages into the cache
 - Improve PC performance by dedicating section of memory as virtual disk, or RAM disk

Page Cache

- A **page cache** caches pages rather than disk blocks using virtual memory techniques
- Memory-mapped I/O uses a page cache
- Routine I/O through the file system uses the buffer (disk) cache
- This leads to the following figure (See next slide)

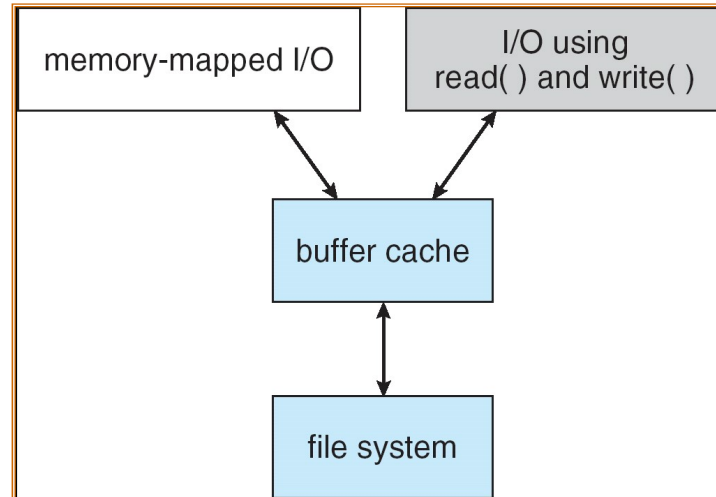
I/O Without a Unified Buffer Cache



Unified Buffer Cache

- A unified buffer cache uses the same page cache to cache both memory-mapped pages and ordinary file system I/O (See next slide)

I/O Using a Unified Buffer Cache



11.7 – Recovery

Recovery Techniques

- Consistency checking – compares data in directory structure with data blocks on disk, and tries to fix inconsistencies
- Use system programs to **back up** data from disk to another storage device (floppy disk, magnetic tape, other magnetic disk, optical)
- Recover lost file or disk by **restoring** data from backup

11.8 – Log-Structured File Systems (skip)

11.9 - NFS

The Sun Network File System (NFS)

- An implementation and a specification of a software system for accessing remote files across LANs (or WANs)
- The implementation is part of the Solaris and SunOS operating systems running on Sun workstations
 - Uses either the TCP or UDP protocol depending on the interconnecting network

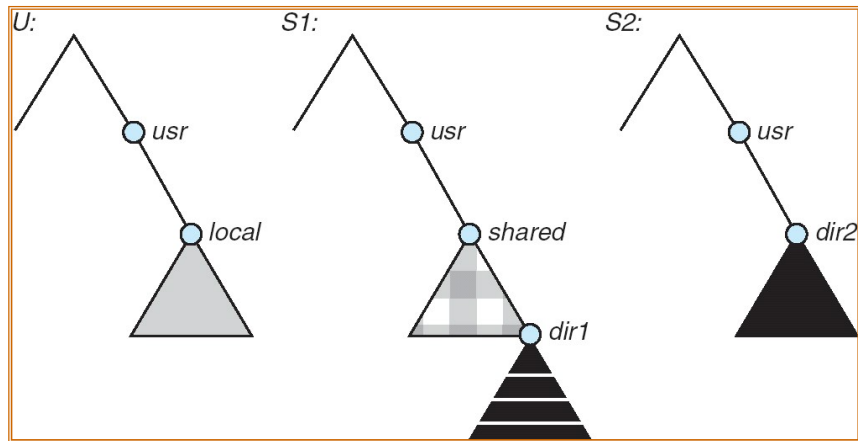
NFS (Cont.)

- Interconnected workstations are viewed as a set of independent machines with independent file systems, which allows sharing among these file systems in a transparent manner
 - A remote directory is mounted over a local file system directory
 - ▶ The mounted directory looks like an integral subtree of the local file system, replacing the subtree descending from the local directory
 - Specification of the remote directory for the mount operation is nontransparent; the host name of the remote directory has to be provided
 - ▶ Files in the remote directory can then be accessed in a transparent manner
 - Subject to access-rights accreditation, potentially any file system (or directory within a file system), can be mounted remotely on top of any local directory

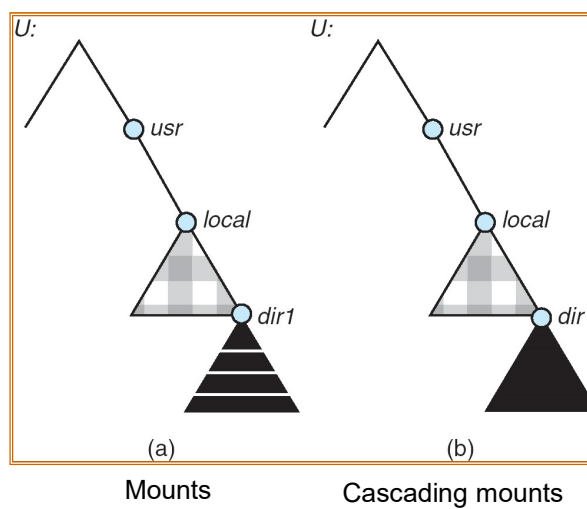
NFS (Cont.)

- NFS is designed to operate in a heterogeneous environment of different machines, operating systems, and network architectures; the NFS specification is independent of these media
- This independence is achieved through the use of RPC primitives built on top of an External Data Representation (XDR) protocol used between two implementation-independent interfaces
- The NFS specification distinguishes between the services provided by a mount mechanism and the actual remote-file-access services

Three Independent File Systems



Mounting in NFS



NFS Mount Protocol

- Establishes initial logical connection between server and client
- Mount operation includes name of remote directory to be mounted and name of server machine storing it
 - Mount request is mapped to corresponding RPC and forwarded to mount server running on server machine
 - Export list – specifies local file systems that server exports for mounting, along with names of machines that are permitted to mount them
- Following a mount request that conforms to its export list, the server returns a file handle—a key for further accesses
- File handle – a file-system identifier, and an inode number to identify the mounted directory within the exported file system
- The mount operation changes only the user's view and does not affect the server side

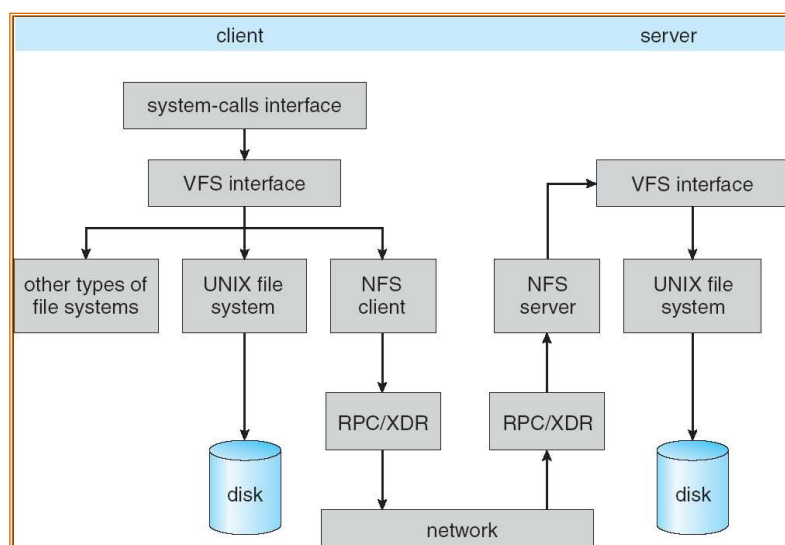
NFS Protocol

- Provides a set of remote procedure calls for remote file operations. The procedures support the following operations:
 - Searching for a file within a directory
 - Reading a set of directory entries
 - Manipulating links and directories
 - Accessing file attributes
 - Reading and writing files
- NFS servers are **stateless**; each request has to provide a full set of arguments (NFS V4 is just coming available – very different, stateful)
- Modified data must be committed to the server's disk before results are returned to the client (lose advantages of caching)
- The NFS protocol does not provide concurrency-control mechanisms

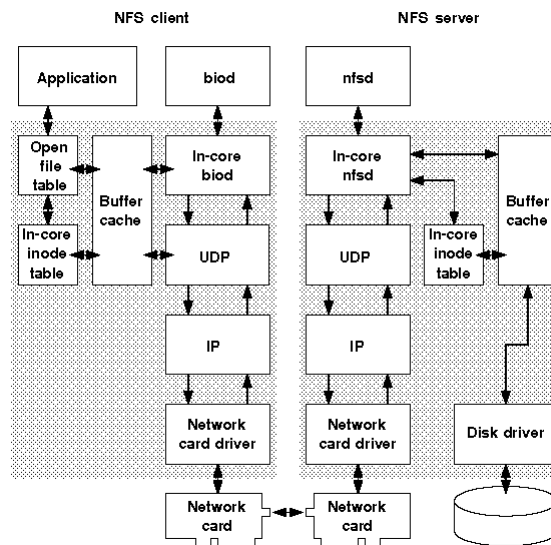
Three Major Layers of NFS Architecture

- UNIX file-system interface (based on the **open**, **read**, **write**, and **close** calls, and **file descriptors**)
- *Virtual File System* (VFS) layer – distinguishes local files from remote ones, and local files are further distinguished according to their file-system types
 - The VFS activates file-system-specific operations to handle local requests according to their file-system types
 - Calls the NFS protocol procedures for remote requests
- NFS service layer – bottom layer of the architecture
 - Implements the NFS protocol

Schematic View of NFS Architecture



Schematic Diagram of NFS



Source: http://osr507doc.sco.com/en/PERFORM/NFS_rsc.html

NFS Path-Name Translation

- Performed by breaking the path into component names and performing a separate NFS lookup call for every pair of component name and directory vnode
- To make lookup faster, a directory name lookup cache on the client's side holds the **vnodes** for remote directory names

NFS Remote Operations

- Nearly one-to-one correspondence between regular UNIX system calls and the NFS protocol RPCs (except opening and closing files)
- NFS adheres to the remote-service paradigm, but employs buffering and caching techniques for the sake of performance
- **File-blocks cache** – when a file is opened, the kernel checks with the remote server whether to fetch or revalidate the cached attributes
 - Cached file blocks are used only if the corresponding cached attributes are up to date
- **File-attribute cache (inode information)** – the attribute cache is updated whenever new attributes arrive from the server
- Clients do not free delayed-write blocks until the server confirms that the data have been written to disk

End of Chapter 11