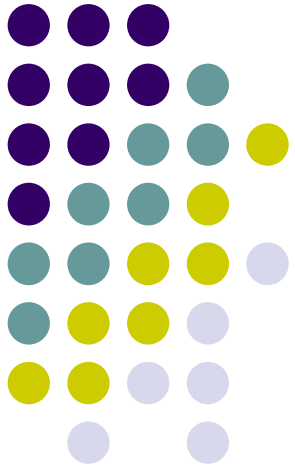
Chapter 10: File System

WHAT'S AHEAD:

- File Concept
- Access Methods
- Disk and Directory Structure
- File-System Mounting
- File Sharing
- Protection

WE AIM:

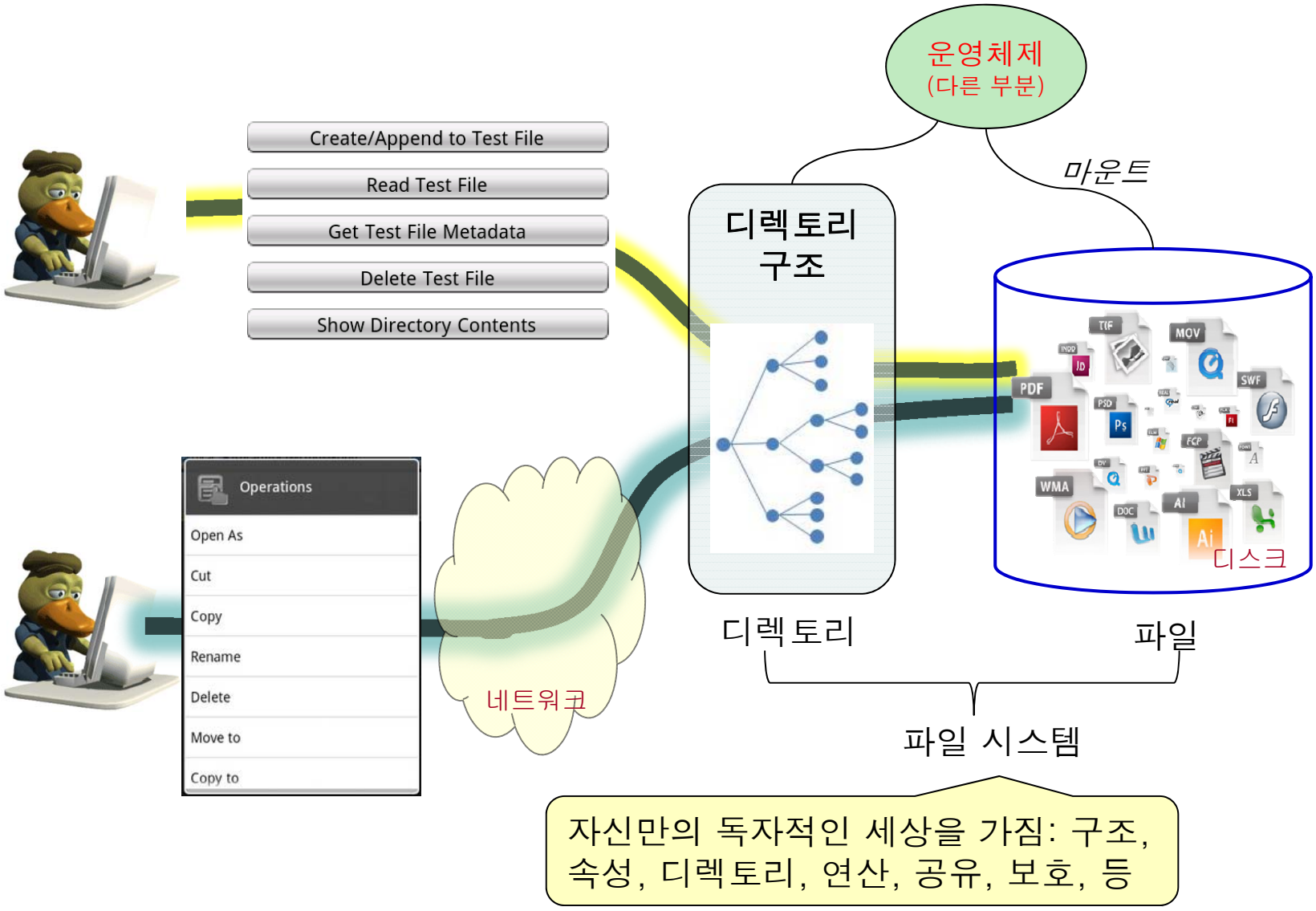
- To explain the function of file systems
- To describe the interfaces to file systems
- To discuss file-system design tradeoffs
- To explore file-system protection



Note: These lecture materials are based on the lecture notes prepared by the authors of the book titled *Operating System Concepts*, 9e (Wiley)

핵심 요점

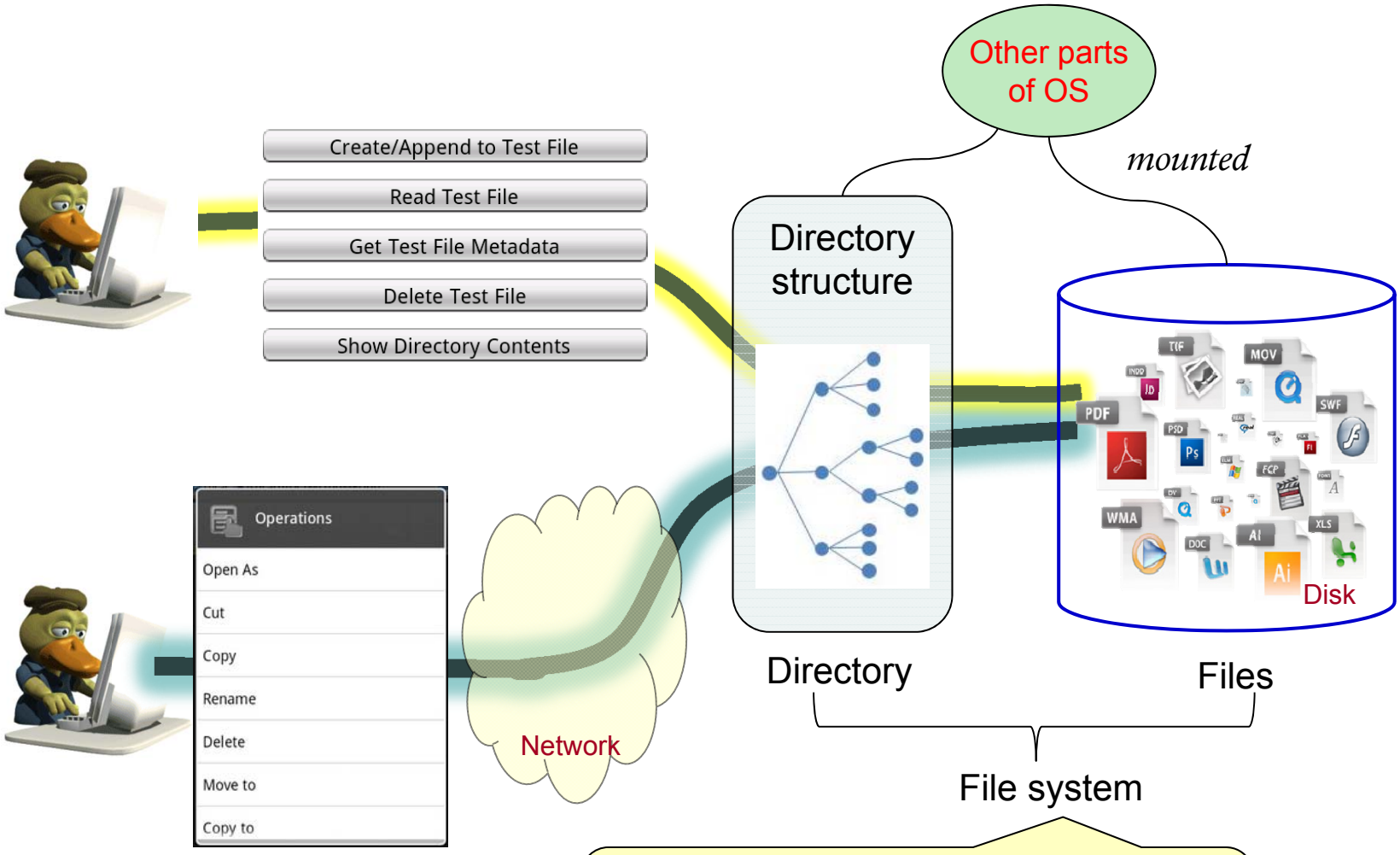
파일: 사용자가 접근할 수 있는 데이터



자신만의 독자적인 세상을 가짐: 구조, 속성, 디렉토리, 연산, 공유, 보호, 등

Core Ideas

File: Data Users Can Access



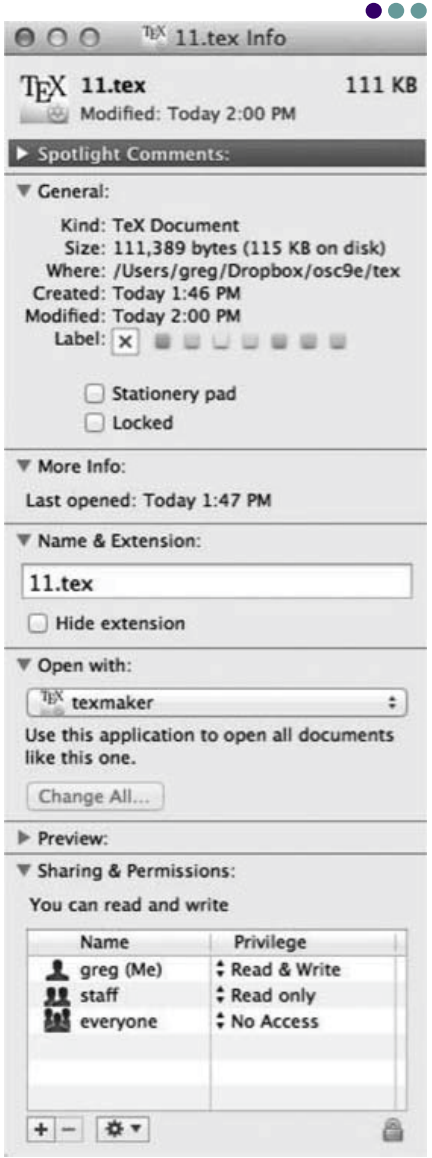
I have my own world of concern: Structures, attributes, directories, operations, sharing, protection, etc.

File Concept

- File
  - a collection of data or information with a name which is known to a user or application
  - a named container of data or information
  - defined in the contiguous logical address space
- Types:
  - Data
    - numeric
    - character
    - binary
  - Program
- Contents defined by file's creator
  - Many types
    - Consider text file, source file, executable file

File Attributes

- **Name** – only information kept in human-readable form
- **Identifier** – unique tag (number) identifies file within file system
- **Type** – needed for systems that support different types
- **Location** – pointer to file location on device
- **Size** – current file size
- **Protection** – controls who can do reading, writing, executing
- **Time, date, and user identification** – data for protection, security, and usage monitoring
- Information about files are kept in the directory structure, which is maintained on the disk
- Many variations, including extended file attributes such as file checksum
- Information kept in the directory structure



File Info Window on Mac OS X

File Operations

- File is an **abstract data type**
- **Create**
- **Write** – at **write pointer** location
- **Read** – at **read pointer** location
- **Reposition within file** - **seek**
- **Delete**
- **Truncate**
- ***Open*(F_i)** – search the directory structure on disk for entry F_i, and move the content of entry to memory
- ***Close*** (F_i) – move the content of entry F_i in memory to directory structure on disk

Open Files

- Several pieces of data are needed to manage open files:
  - **Open-file table**: tracks open files
  - File pointer: pointer to last read/write location, per process that has the file open
  - **File-open count**: counter of number of times a file is open – to allow removal of data from open-file table when last processes closes it
  - Disk location of the file: cache of data access information
  - Access rights: per-process access mode information

Open File Locking

- Provided by some operating systems and file systems
  - Similar to reader-writer locks
  - **Shared lock** similar to reader lock – several processes can acquire concurrently
  - **Exclusive lock** similar to writer lock
- Mediates access to a file
- Mandatory or advisory locking schemes:
  - **Mandatory** – access is denied depending on locks held and requested
    - OS ensures locking integrity by enforcing the given locking mechanism (Windows OS)
  - **Advisory** – processes can find status of locks and decide what to do
    - It is up to software developers to ensure that locks are appropriately acquired and released (Unix)

File Types – Name, Extension

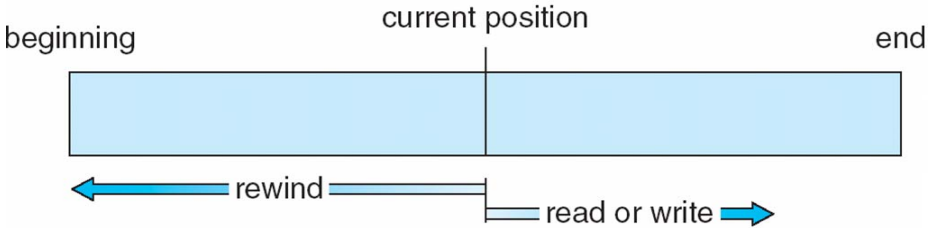
file type	usual extension	function
executable	exe, com, bin or none	ready-to-run machine-language program
object	obj, o	compiled, machine language, not linked
source code	c, cc, java, pas, asm, a	source code in various languages
batch	bat, sh	commands to the command interpreter
text	txt, doc	textual 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

File Structure

- File structure – A premise
  - Assumes it will ultimately be understood by related programs
- Various file structures
  - None - sequence of words, bytes
  - Simple record structure
    - Lines
    - Fixed length
    - Variable length
  - Complex Structures
    - Formatted document
    - Relocatable load file
- Who decides or understands:
  - Operating system
    - E.g., executable file is understood by OS
  - Program
- Files are all transformed into physical blocks of disk
- In Unix
  - All files are defined to be simply streams of bytes → logical record size is one
  - Programs need to understand the file structure

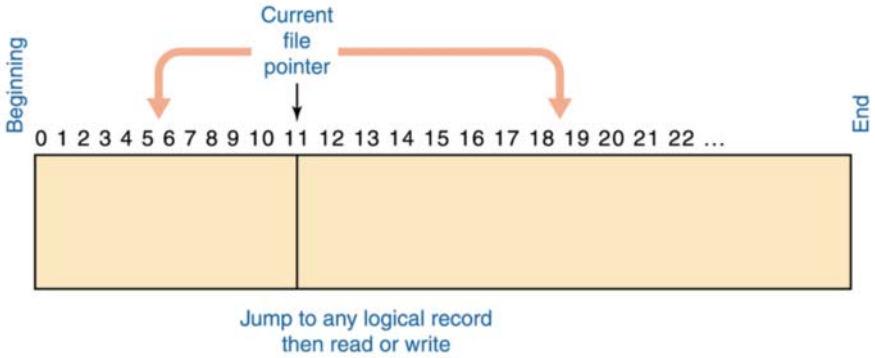
Access Methods

- **Sequential Access**
  - **read next**
  - **write next**
  - **reset**
  - no read after last write



- **Direct Access** – File consists of fixed length logical records
  - **read *n***
  - **write *n***
  - **position to *n***
    - **read next**
    - **write next**
  - **rewrite *n***

n = relative block number



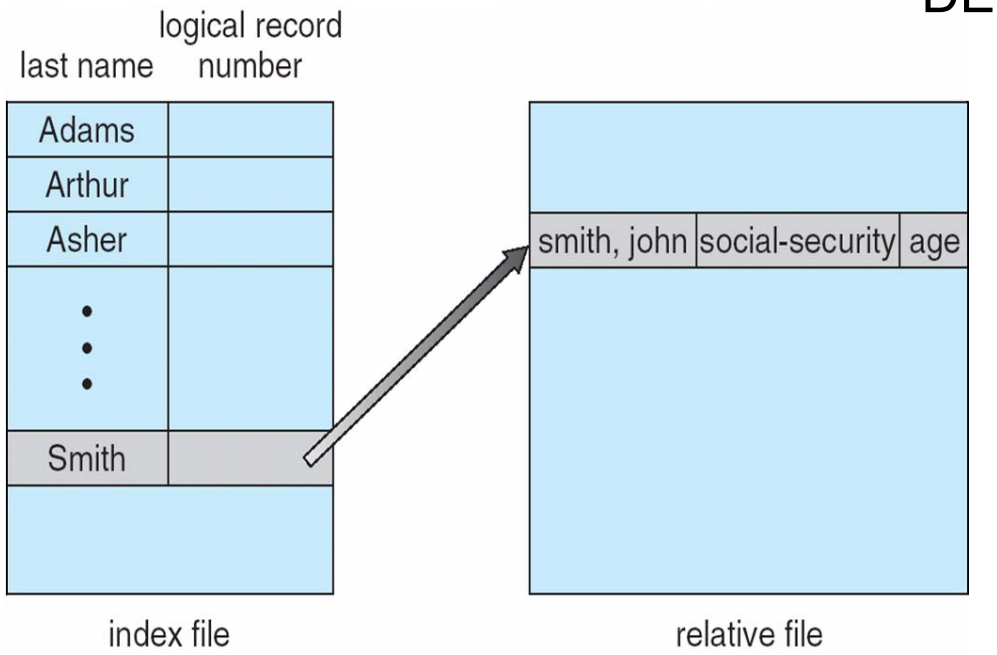
- Relative block numbers allow OS to decide where file should be placed
  - See allocation problem in Ch 11 (Implementing file systems)

Other Access Methods

- Can be built on top of base methods
- Generally involve creation of an **index** for the file
- Keep index in memory for fast determination of location of data to be operated on (consider Universal Product Code (UPC) plus record of data about that item)
- If too large, index (in memory) of the index (on disk)
- IBM indexed sequential-access method (ISAM)
  - Small master index, points to disk blocks of secondary index
  - File kept sorted on a defined key
  - All done by the OS
- VMS operating system provides index and relative files as another example (see next slide)

Example of Index and Relative Files

DEC VMS OS



The primary index file

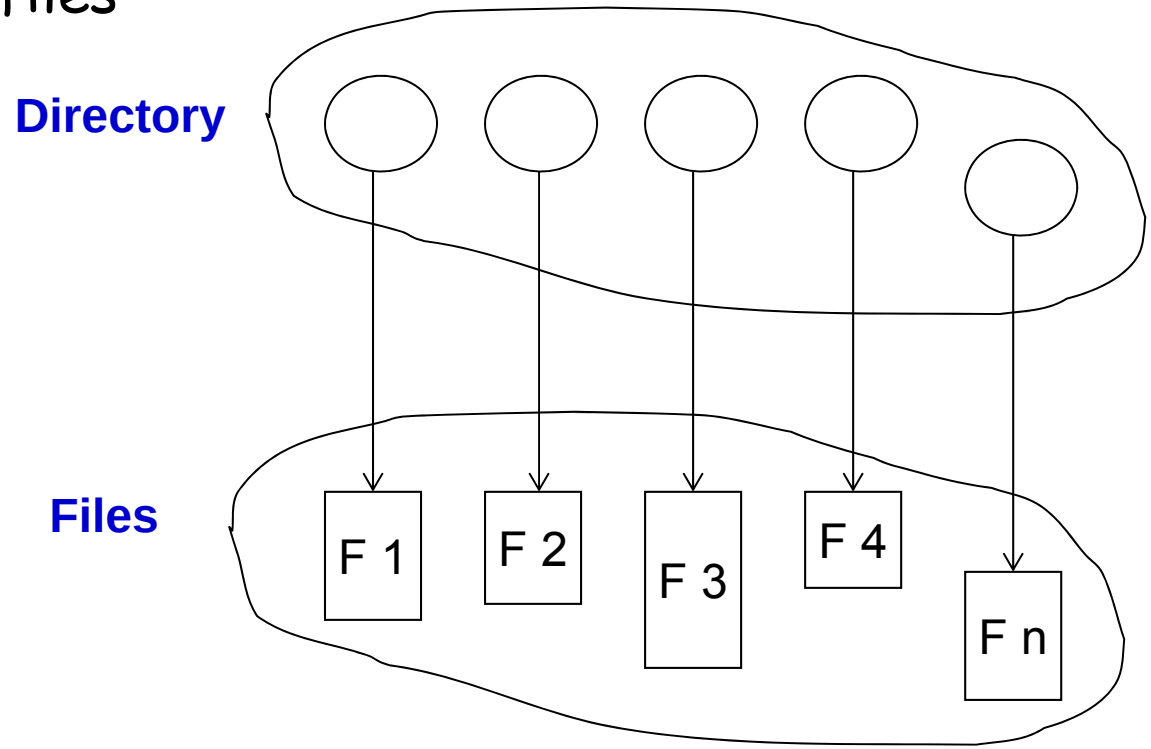
The secondary index file

Indexed Access

- An index is created that contains a key value for indexing and pointers to related blocks.
- For large files, the index file itself may become too large to be kept in memory.
- One solution is to create an index for the index file. The primary index file would contain pointers to secondary index files, which would point to the actual data items.

Directory and Disk Structure

- Directory
  - A collection of nodes containing information about all files

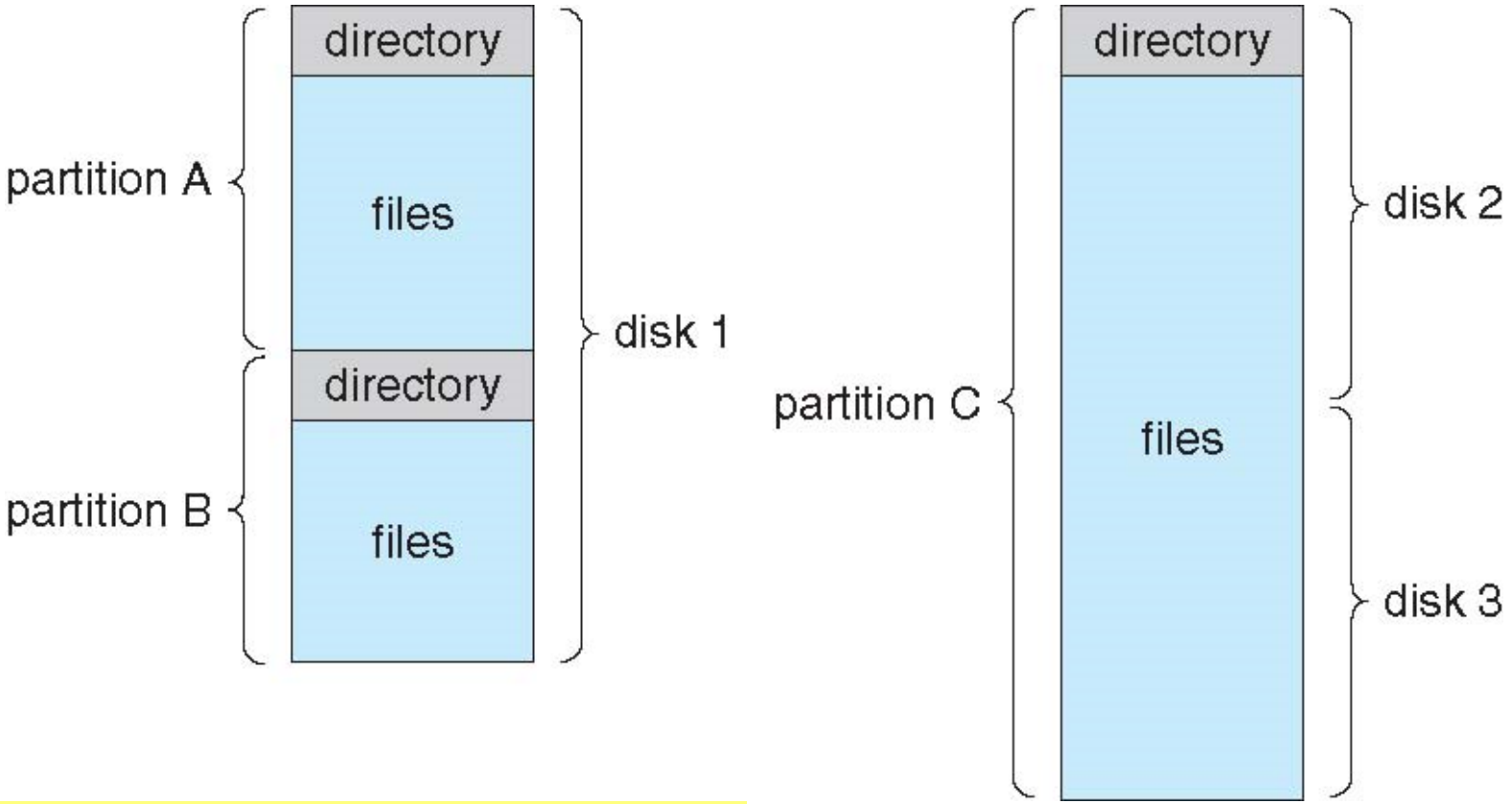


Both the directory structure and the files reside on disk

Directory and Disk Structure (Cont.)

- Disk can be subdivided into **partitions**
- Disks or partitions can be **RAID** protected against failure
- Disk or partition can be used **raw** (without a file system), or **formatted** with a file system
- Partitions also known as minidisks, slices
- Entity containing file system known as a **volume**
- Each volume containing file system also tracks that file system's info in **device directory** or **volume table of contents**
- As well as **general-purpose file systems** there are many **special-purpose file systems**, frequently all within the same operating system or computer

A Typical File-system Organization



directory: records information, such as name, location, size, type, etc., for all files on that volume.

Types of File Systems

- We mostly talk of general-purpose file systems
- But systems frequently have many file systems, some general- and some special- purpose
- Solaris may have dozens of file systems of different types
  - tmpfs – memory-based volatile FS for fast, temporary I/O
  - objfs – interface into kernel memory to get kernel symbols for debugging
  - ctfs – contract file system for managing daemons
  - lofs – loopback file system allows one FS to be accessed in different directory name
  - procfs – kernel interface to process structures, /proc
  - ufs, zfs – general purpose file systems

Operations Performed on Directory

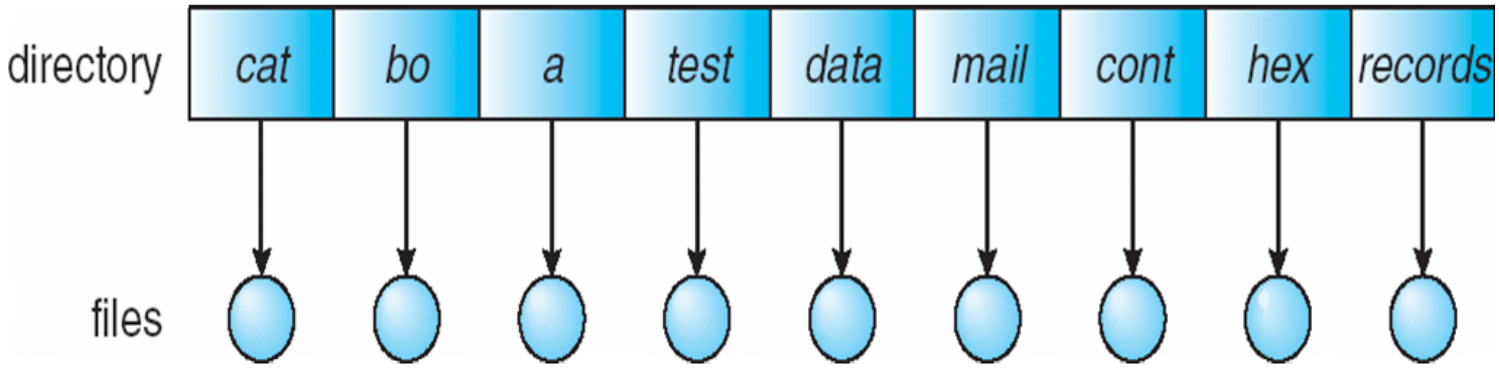
- Search for a file
- Create a file
- Delete a file
- List a directory
- Rename a file
- Traverse the file system

Directory Structures

- Organize the directory (logically) to obtain
  - 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, …)
- Different structures
  - Single-level directory
  - Two-level directory
  - Tree-structured directory
  - Acyclic-graph directory
  - General graph directory

Single-Level Directory

- A single directory for all users

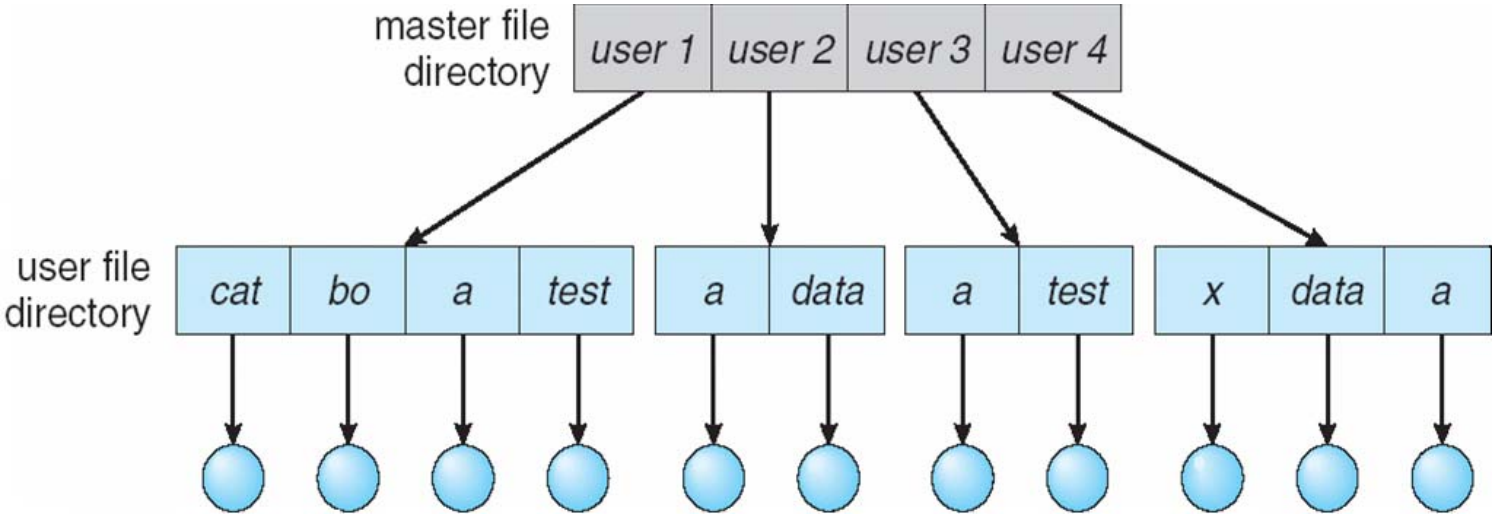


Naming problem

Grouping problem

All files are contained in the same directory.

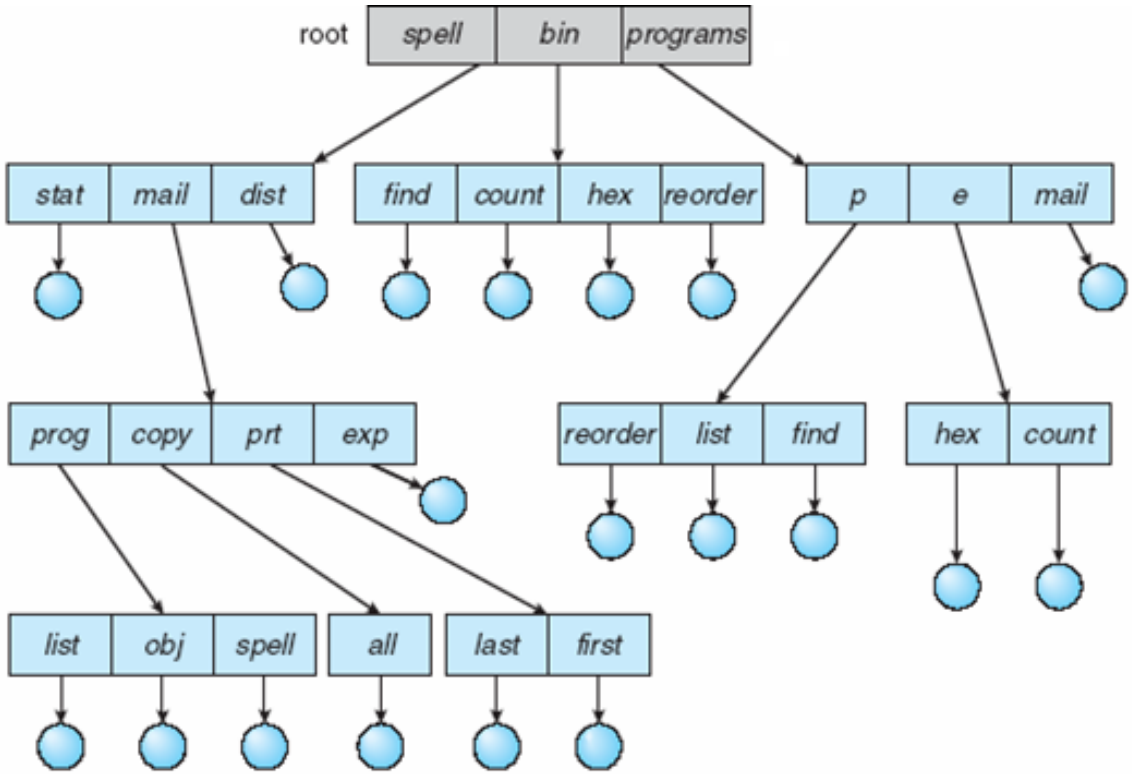
Two-Level Directory



- Separate directory for each user
  - Path name
  - Can have the same file name for different user
  - Efficient searching
  - No grouping capability

Tree-Structured Directories

- Efficient searching
- Grouping capability
- Current directory (working directory)
  - **`cd /spell/mail/prog`**
  - **`type list`**



Tree-Structured Directories (Cont)

- Absolute or relative path name
- Creating a new file is done in current directory
- Delete a file

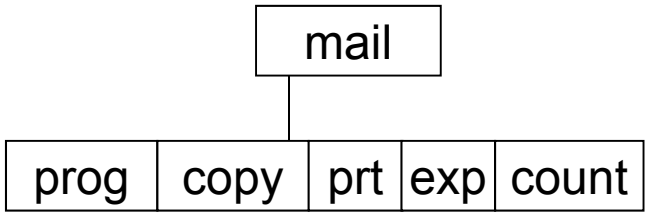
```
rm <file-name>
```

- Creating a new subdirectory is done in current directory

```
mkdir <dir-name>
```

Example: if in current directory **/mail**

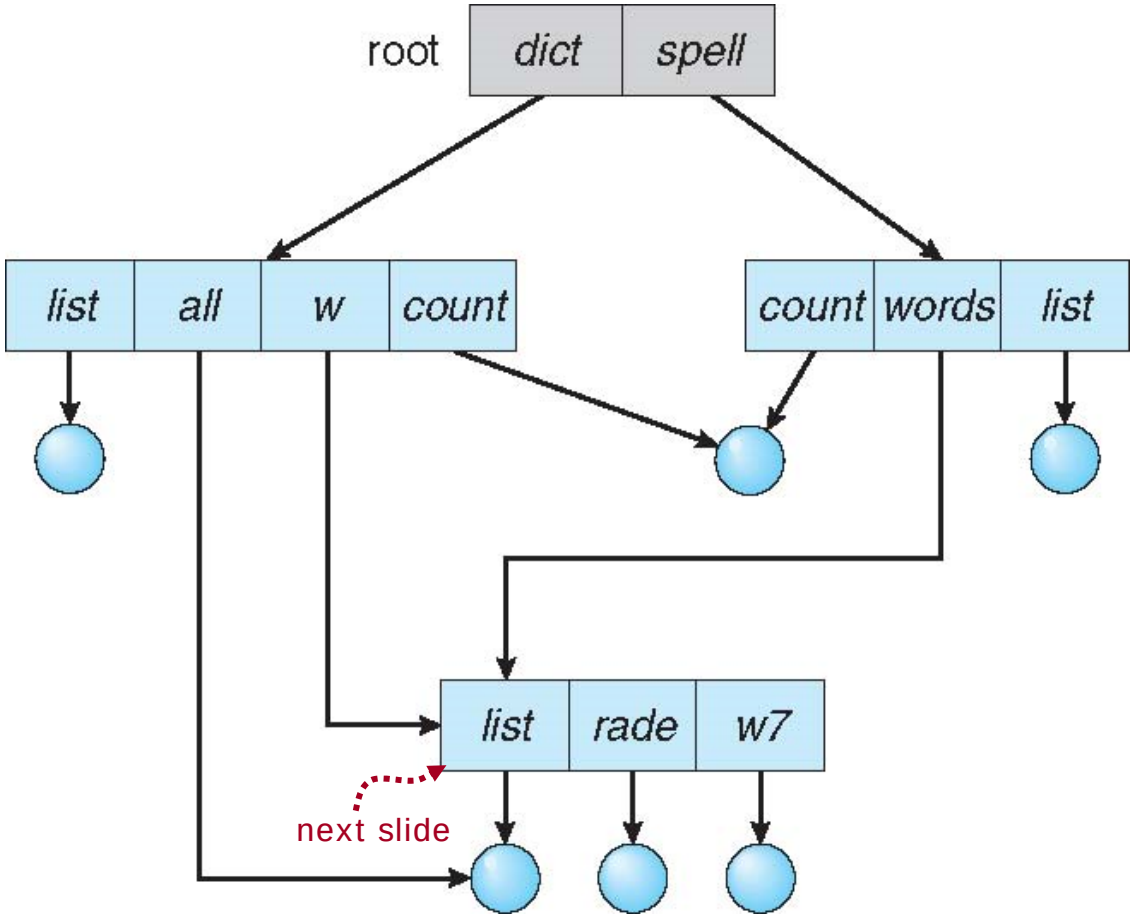
```
mkdir count
```



Deleting "mail" ⇒ deleting the entire subtree rooted by "mail"

Acyclic-Graph Directories

- Have shared subdirectories and files



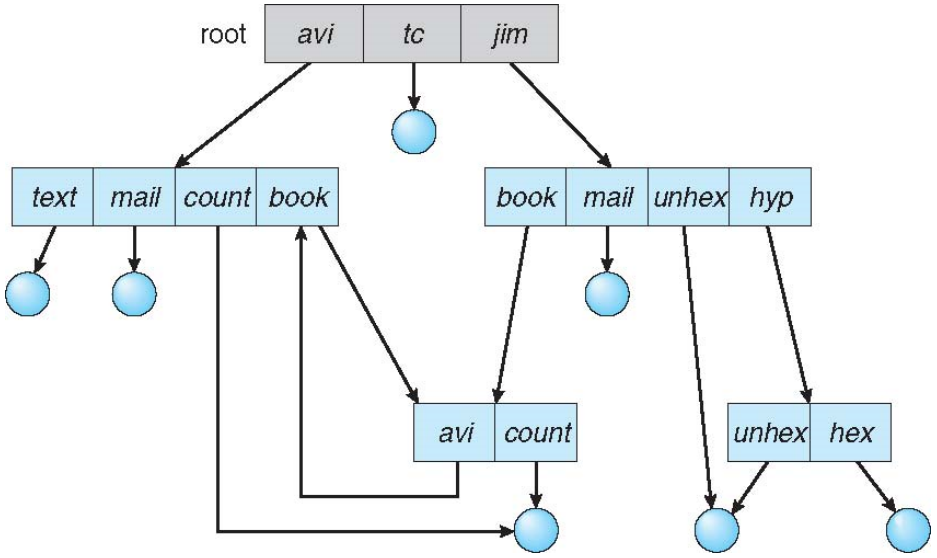
Acyclic-Graph Directories (Cont.)

- Two different names (aliasing)
- If ***words*** deletes ***list*** ⇒ **dangling pointer** (to non-existent directory or file)

  Solutions:
  - Backpointers, so we can delete all pointers
    Variable size records - a problem
  - Backpointers using a daisy chain organization
  - Entry-hold-count solution
- New directory entry type
  - **Link** – another name (pointer) to an *existing* file or subdirectory
    - a way to share files or subdirectories
    - includes an absolute or a relative path name to a file or subdirectory
  - **Resolve the link** – follow pointer to locate the file

General Graph Directory

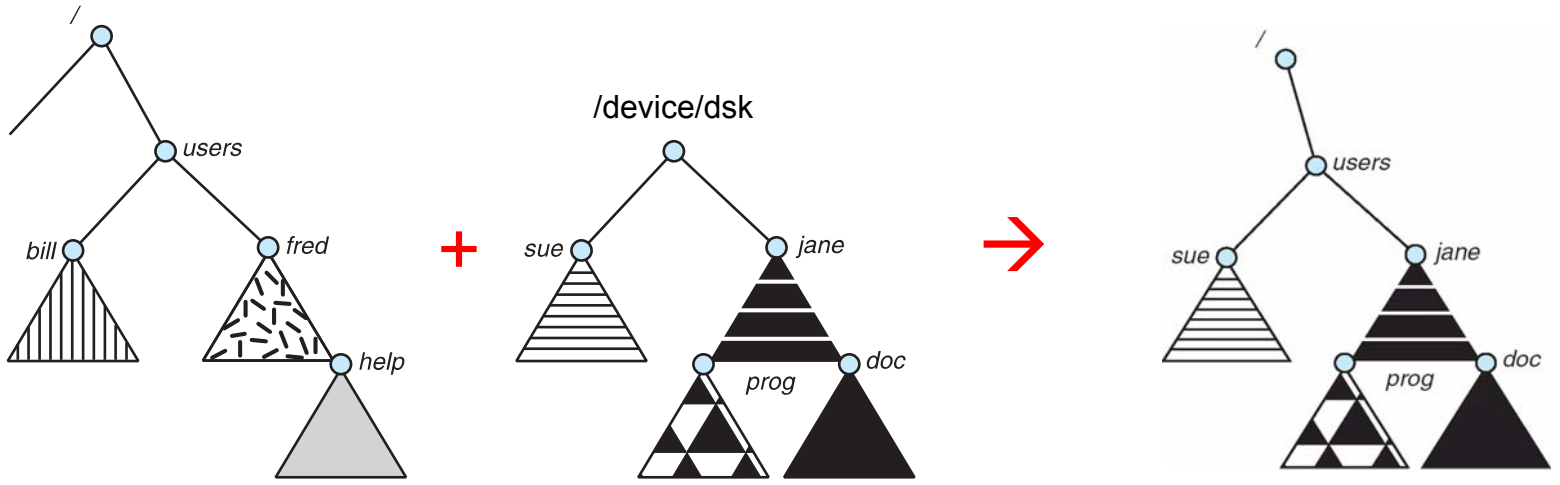
- General graph directory
  - can contain a cycle
  - Acyclic-graph directory may end up with general graph directory if new links are added
- How do we guarantee no cycles?
  - Allow only links to file not subdirectories
  - Garbage collection
  - Every time a new link is added use a cycle detection algorithm to determine whether it is OK



File System Mounting

- A file system must be **mounted** before it can be accessed
- A unmounted file system (i.e., Fig. 10-11(b)) is mounted at a **mount point**
  - mount point: the location within the file structure where the file system is to be attached, typically an empty directory

Semantics: For example, a system may disallow a mount over a directory that contains files; or it may make the mounted file system available at that directory and obscure (make invisible) the directory's existing files until the file system is unmounted, allowing access to the original files in that directory.



(a) Existing system

(b) Unmounted volume

mounting the volume residing on /device/dsk over /users

File Sharing

- Sharing of files on multi-user systems is desirable
- Sharing may be done through a **protection** scheme
- On distributed systems, files may be shared across a network
- Network File System (NFS) is a common distributed file-sharing method
- If multi-user system
  - **User IDs** identify users, allowing permissions and protections to be per-user
    Group IDs allow users to be in groups, permitting group access rights
  - Owner of a file / directory
  - Group of a file / directory

File Sharing – Remote File Systems

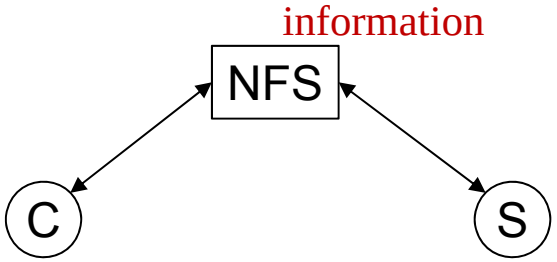
- Uses networking to allow file system access between systems
  - Manually via programs like FTP
  - Automatically, seamlessly using **distributed file systems**
  - Semi automatically via the **world wide web**
- **Client-server** model allows clients to mount remote file systems from servers
  - Server can serve multiple clients
  - Client and user-on-client identification is insecure or complicated
  - **NFS** is standard UNIX client-server file sharing protocol
  - **CIFS** is standard Windows protocol
  - Standard operating system file calls are translated into remote calls
- Distributed Information Systems (**distributed naming services**) such as LDAP, DNS, NIS, Active Directory implement unified access to information needed for remote computing

File Sharing – Failure Modes

- All file systems have failure modes
  - For example, corruption of directory structures or other non-user data, called **metadata**
- Remote file systems add new failure modes, due to network failure, server failure
- Recovery from failure can involve **state information** about status of each remote request
- **Stateless** protocols such as NFS v3 include all information in each request, allowing easy recovery but less security

- Stateless protocol
  - NFS has all info.
  - But NFS does not track which clients have the exported volume mounted → "unknown" client may cut in → insecure

(a) stateful



(a) stateless

File Sharing – Consistency Semantics

- Specify how multiple users are to access a shared file simultaneously
  - Similar to Chapter 6 process synchronization algorithms
    - Tend to be less complex due to disk I/O and network latency for remote file systems
  - Andrew File System (AFS) implemented complex remote file sharing semantics
  - Unix file system (UFS) implements:
    - Sharing file pointer to allow multiple users to read and write concurrently
  - AFS has session semantics
    - Writes to an open file are not visible immediately to other users of the same open file
    - Writes only visible to sessions starting after the file is closed. – Already open instances of the file do not reflect these changes.
    - [file] session: the series of accesses between the **`open()`** and **`close()`**

Protection

- File owner/creator should be able to control:
  - what can be done
  - by whom

- Types of access
  - **Read**
  - **Write**
  - **Execute**
  - **Append**
  - **Delete**
  - **List**

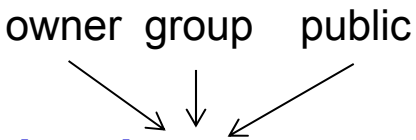
access control ≠ mutual exclusion

Access Lists and Groups

- Mode of access: read, write, execute
- Three classes of users on Unix / Linux

			R W X
a) **owner access**	7	⇒	1 1 1
b) **group access**	6	⇒	1 1 0
c) **public access**	1	⇒	0 0 1

- Ask manager to create a group (unique name), say G, and add some users to the group.
- For a particular file (say *game*) or subdirectory, define an appropriate access.

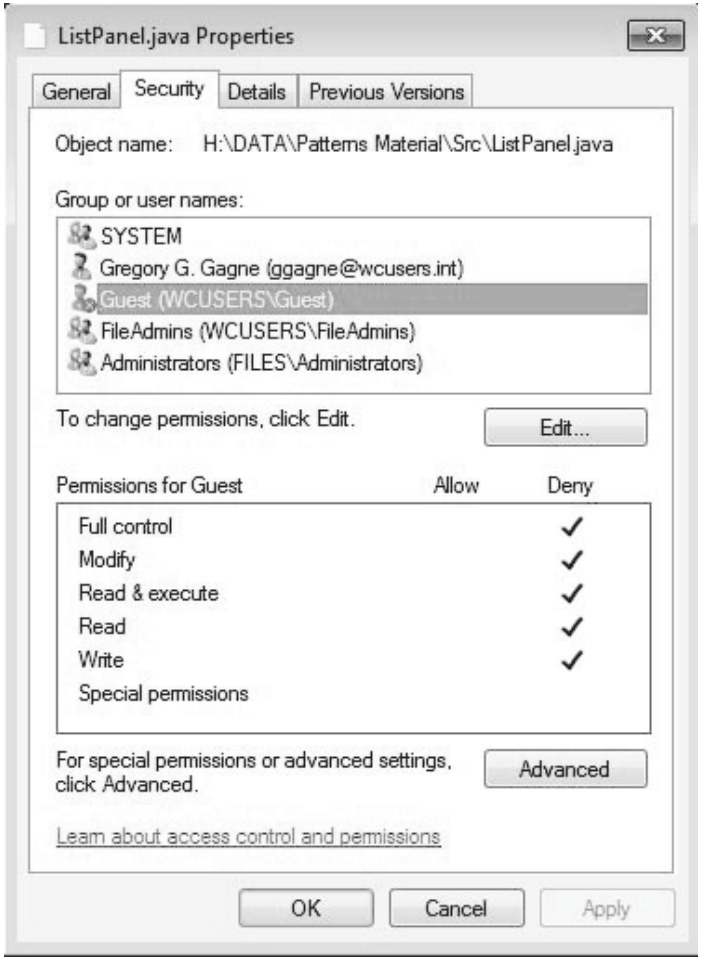


- Attach a group to a file

```
chgrp    G    game
```

Examples

- Windows 7 access-control list management



- A sample UNIX directory listing

```
-rw-rw-r--   1 pbg   staff    31200  Sep 3 08:30   intro.ps
drwx------   5 pbg   staff      512  Jul 8 09.33   private/
drwxrwxr-x   2 pbg   staff      512  Jul 8 09:35   doc/
drwxrwx---   2 jwg   student    512  Aug 3 14:13   student-proj/
-rw-r--r--   1 pbg   staff     9423  Feb 24 2012   program.c
-rwxr-xr-x   1 pbg   staff    20471  Feb 24 2012   program
drwx--x--x   4 tag   faculty    512  Jul 31 10:31  lib/
drwx------   3 pbg   staff     1024  Aug 29 06:52  mail/
drwxrwxrwx   3 pbg   staff      512  Jul 8 09:35   test/
```

Summary

- 파일이란?
  - 사용자가 저장된 데이터에 접근할 수 있는 유일한 수단 제공
  - 파일의 속성
  - 파일에 대한 연산
  - open() system call
  - 파일 구조: byte sequence, collection of record, formatted, loadable, etc.
- 파일에의 접근방법
  - 순차적인 접근
  - 직접적인 접근
  - 인덱스에 의한 접근
- 디스크와 디렉토리의 구조
  - 디스크: partition, volume, raw disk(no fs)
  - 디렉토리: 파일에 대한 정보 포함
  - 디렉토리에 대한 연산
  - 디렉토리의 구조: single level, two level, tree 구조(acyclic, graph)

- File concept
  - provide the only way users can access the stored data
  - file attributes
  - file operations
  - open() system call
  - file structures: byte sequence, collection of record, formatted, loadable, etc.
- Access methods
  - sequential access
  - directed access
  - indexed access
- Disk and directory structure
  - disk: partition, volume, raw disk(no fs)
  - directory: contains info on files
  - operations on directories
  - directory structures: single level, two level, tree-structured (acyclic, graph)

Summary (Cont.)

- 파일시스템 마운팅
  - 파일시스템의 어느 한 곳(mount point, 보통 빈 디렉토리)에 다른 volume을 접속시킴
- 파일의 공유
  - 단일 컴퓨터에서는 비교적 단순함
  - 다른 컴퓨터상의 파일 공유는 NFS 이용
  - 고장시(failure modes): stateful, stateless
  - 일관성 문제(consistency): 공유되는 데이터의 정확성/무결성 보장
- 파일의 보호
  - 접근제어(access control): 상호배제 기법과는 다름
  - Unix/Linux의 access list

- File-system mounting
  - attach a different volume to a mount point (usually a empty directory)
- File sharing
  - rather simple on a singe computer
  - inter-computer file sharing: use NFS
  - failure modes: stateful, stateless
  - consistency semantics: guarantee correctness/integrity of data
- Protection
  - access control: different from mutual exclusion
  - Unix/Linux: access list