

ECE30021/ITP30002 Operating System

File System

(OSC: Ch. 10)

This lecture note is taken from the instructor's resource of Operating System Concept, 9/e and then partly edited/revised by Shin Hong.

File Concept

- 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 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:
 - **Mandatory** – access is denied depending on locks held and requested
 - **Advisory** – processes can find status of locks and decide what to do

File Structure

- None - sequence of words, bytes
- Simple record structure
 - Lines
 - Fixed length
 - Variable length
- Complex Structures
 - Formatted document
 - Relocatable load file
- Can simulate last two with first method by inserting appropriate control characters
- Who decides:
 - Operating system
 - Program

Access Methods

- **Sequential Access**

```
read next
write next
reset
no read after last write
    (rewrite)
```

- **Direct Access** – file is fixed length [logical records](#)

```
read n
write n
position to n
    read next
    write next
rewrite n
```

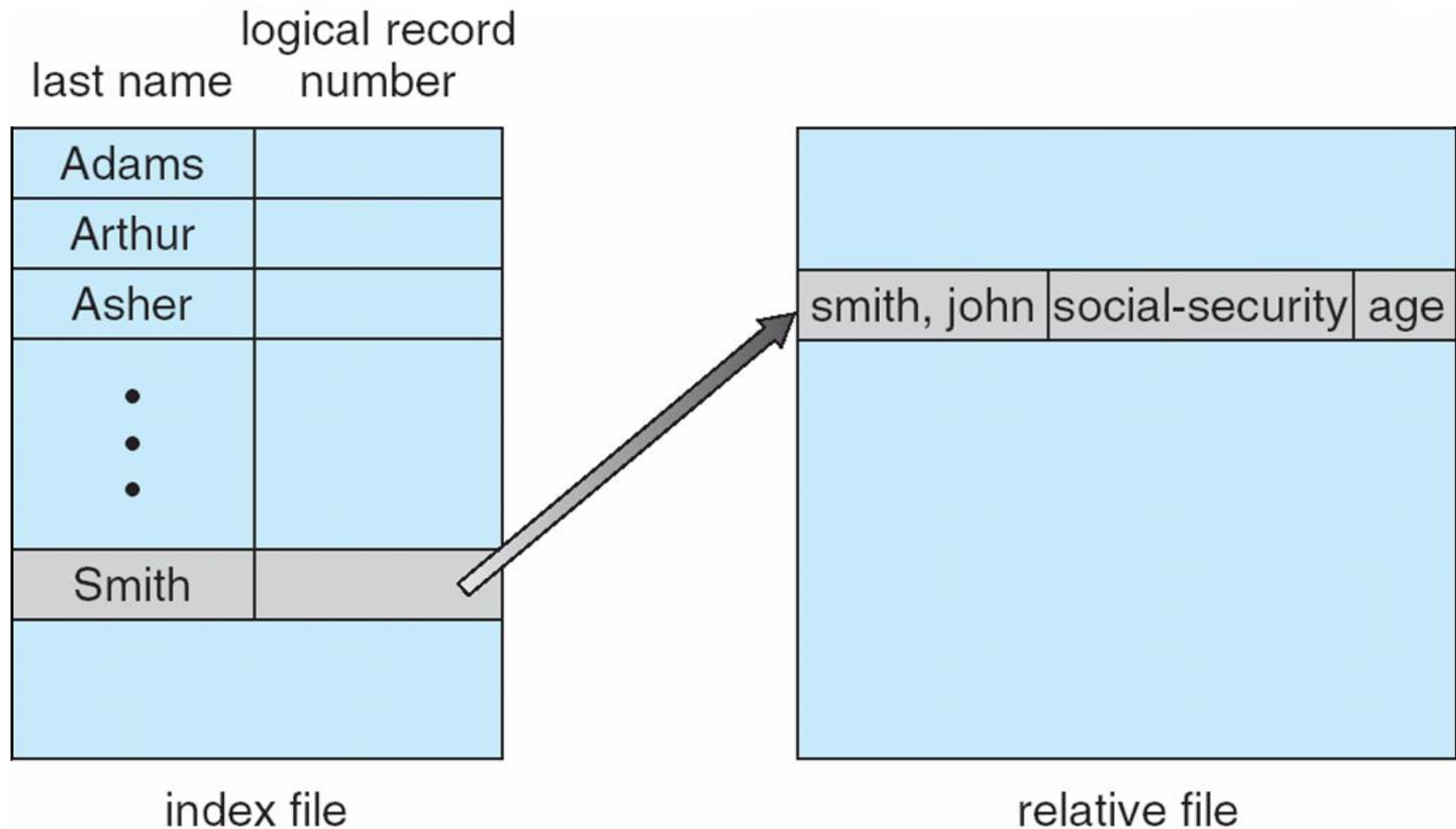
n = [relative block number](#)

- Relative block numbers allow OS to decide where file should be placed
 - See [allocation problem](#) in Ch 12

Other Access Methods

- Can be built on top of base methods
- General involve creation of an **index** for the file
- Keep index in memory for fast determination of location of data to be operated on (consider UPC code 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)

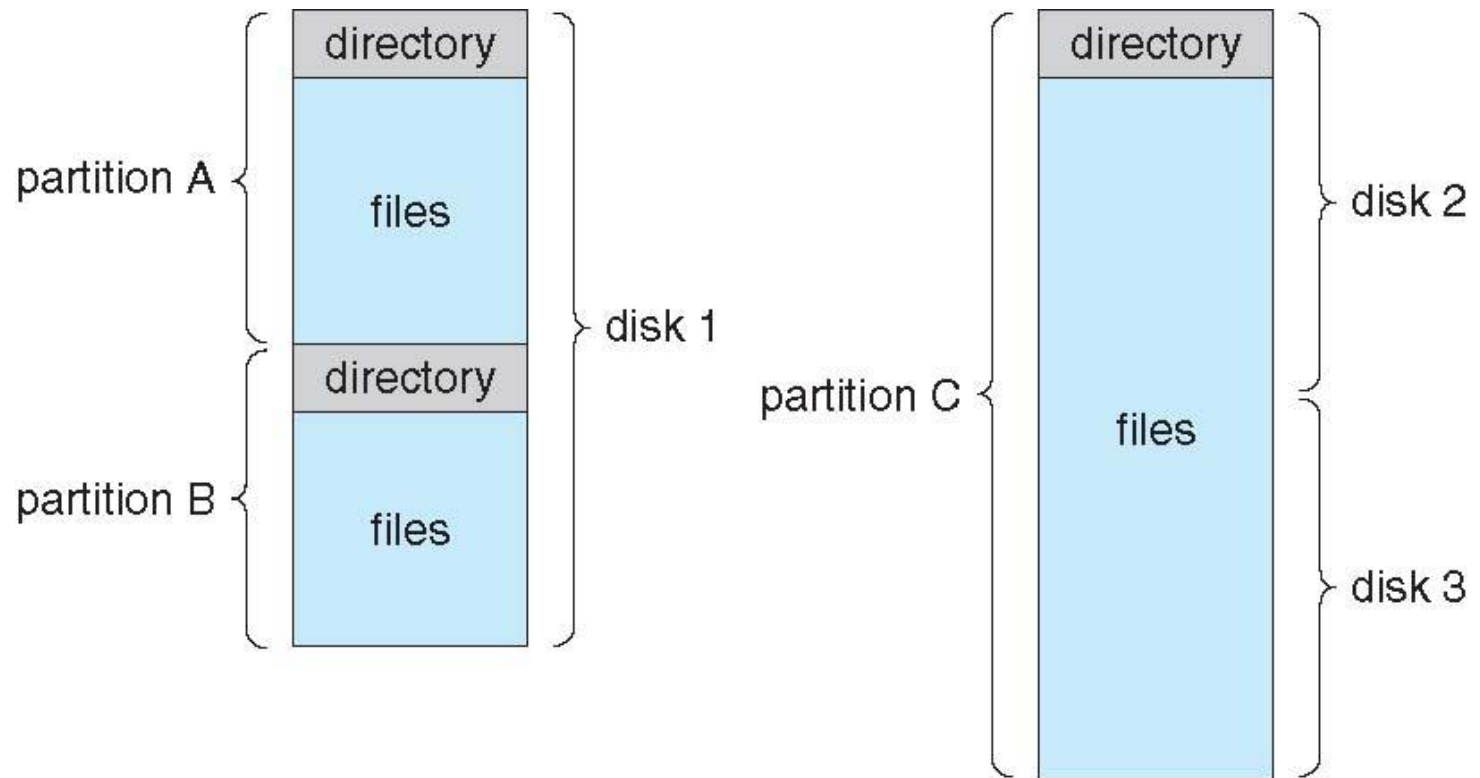
Example of Index and Relative Files



Disk Structure

- 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



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
- Consider Solaris has
 - 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 place of another
 - procfs – kernel interface to process structures
 - 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

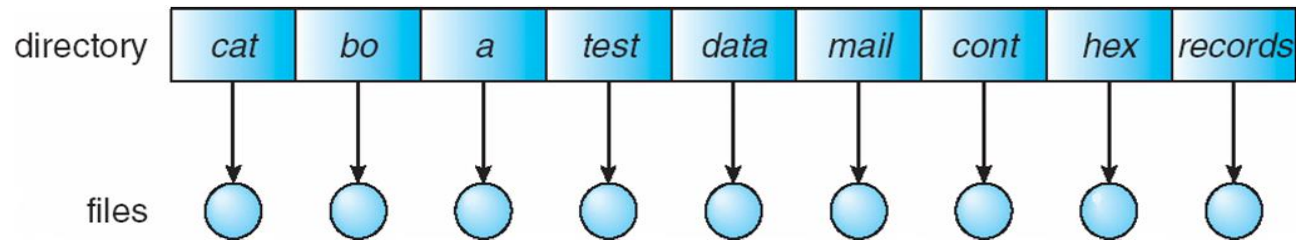
Directory Organization

The directory is organized logically to obtain

- Efficiency – locating a file quickly
- Naming – convenient to users
 - Two users can have the 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, ...)

Single-Level Directory

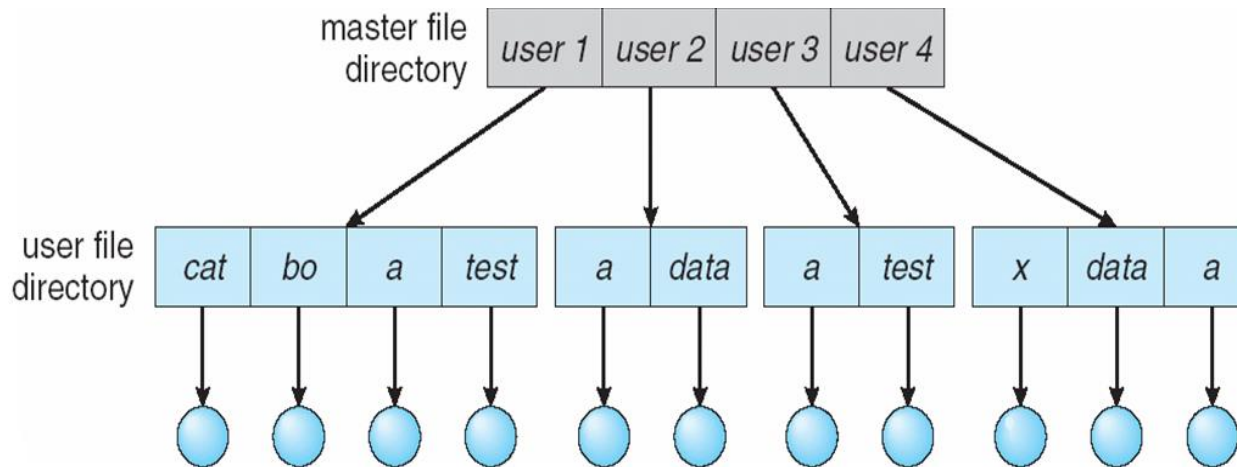
- A single directory for all users



- Naming problem
- Grouping problem

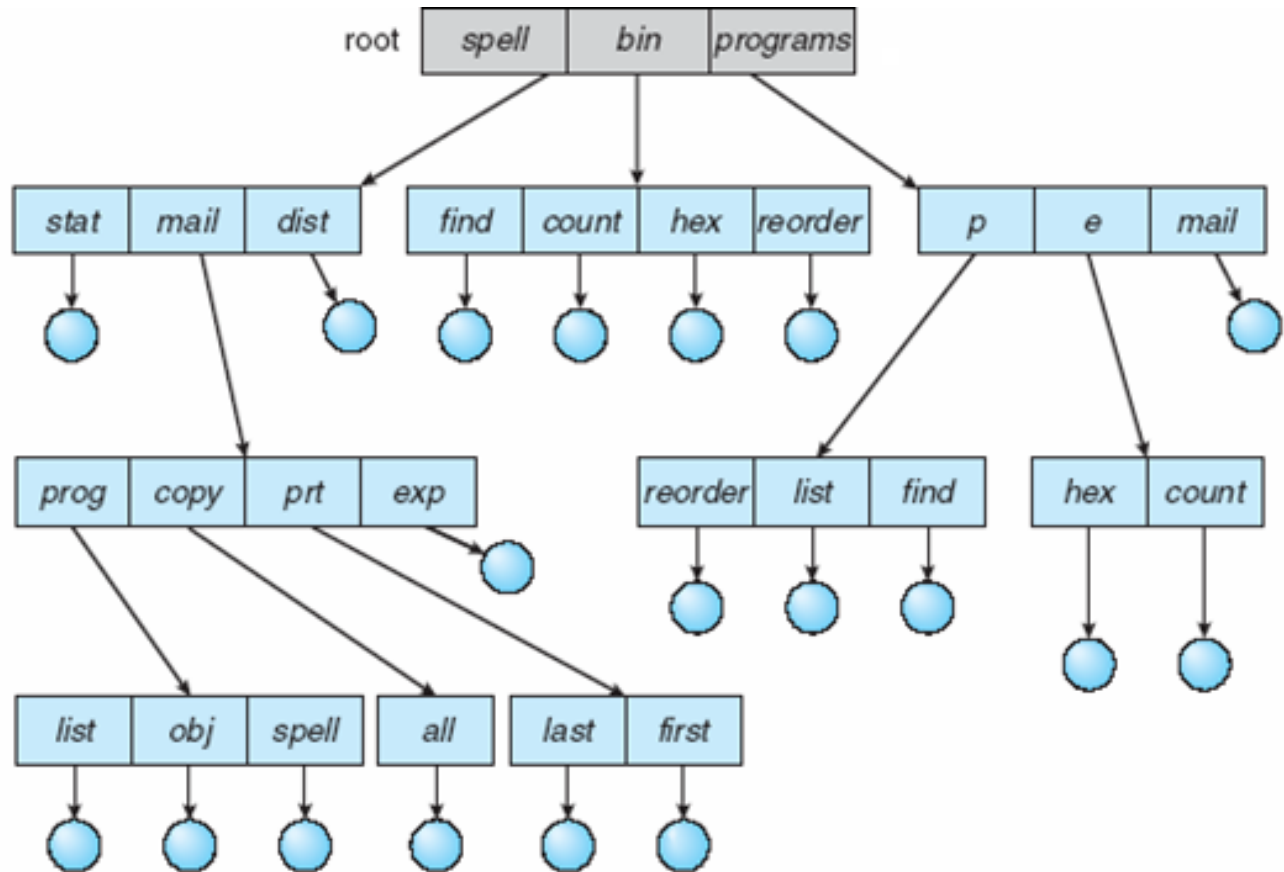
Two-Level Directory

- Separate directory for each user



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

Tree-Structured Directories

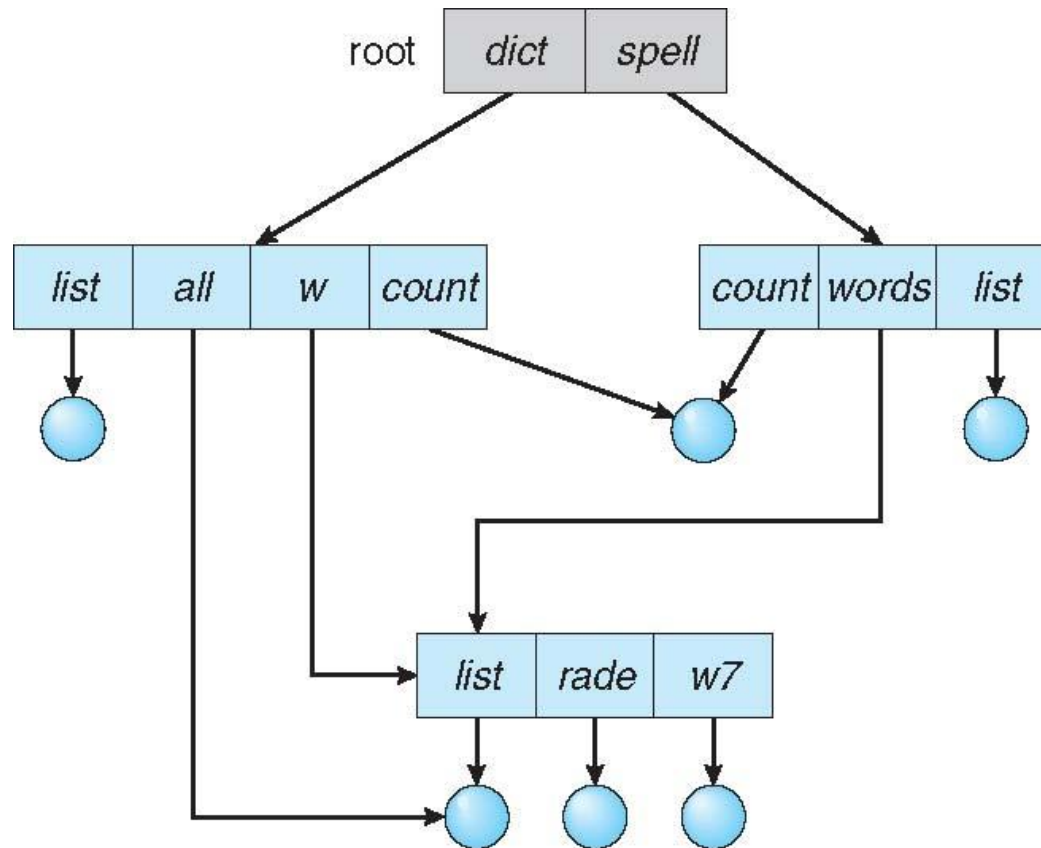


Tree-Structured Directories (Cont.)

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

Acyclic-Graph Directories

- Have shared subdirectories and files



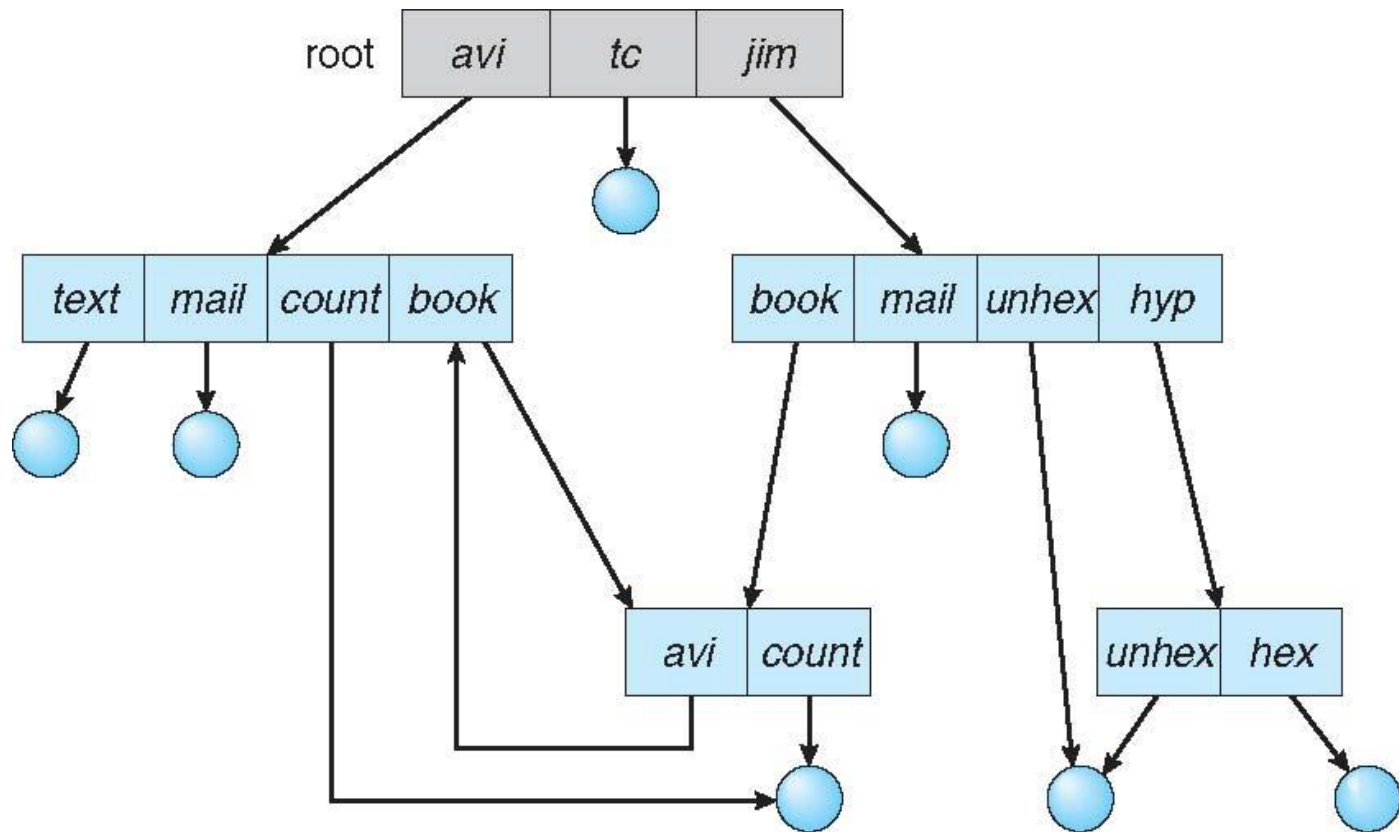
Acyclic-Graph Directories (Cont.)

- Two different names (aliasing)
- If **dict** deletes **list** \Rightarrow dangling pointer

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
 - **Resolve the link** – follow pointer to locate the file

General Graph Directory

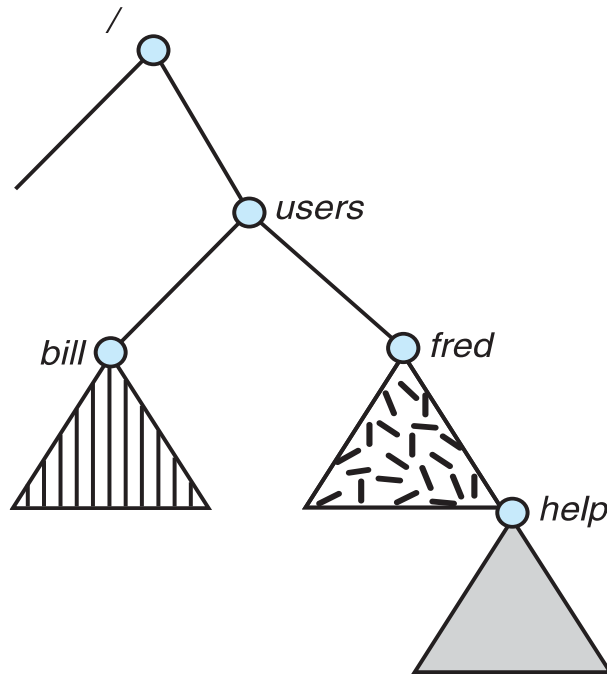


General Graph Directory (Cont.)

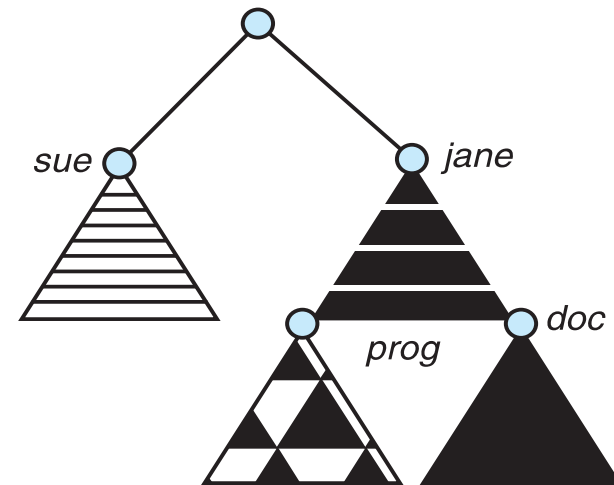
- How do we guarantee no cycles?
 - Allow only links to file not subdirectories
 - **Garbage collection**
 - Every time a new link is added use a cycle detection algorithm to determine whether it is OK

File System Mounting

- A file system must be **mounted** before it can be accessed
- A unmounted file system (i.e., Fig. 11-11(b)) is mounted at a **mount point**

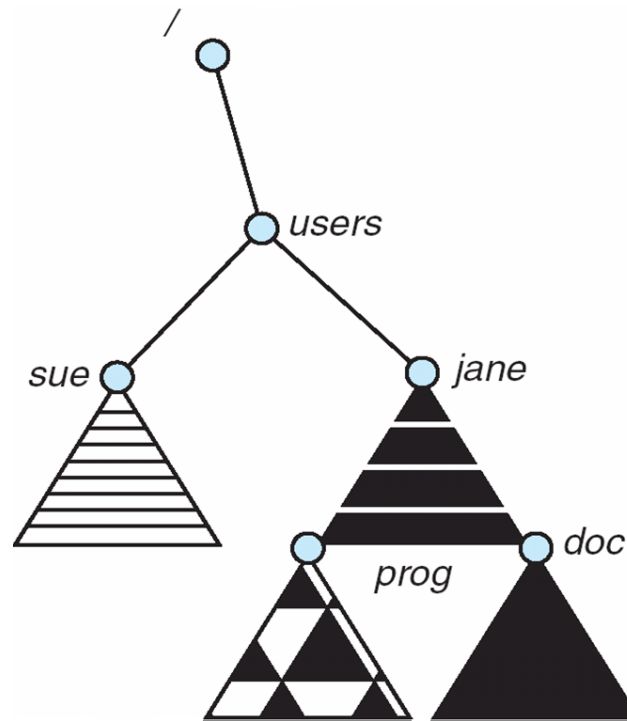


(a)



(b)

Mount Point



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

File Sharing – Consistency Semantics

- Specify how multiple users are to access a shared file simultaneously
 - Similar to Ch 5 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:
 - Writes to an open file visible immediately to other users of the same open file
 - Sharing file pointer to allow multiple users to read and write concurrently
 - AFS has session semantics
 - Writes only visible to sessions starting after the file is closed

Protection

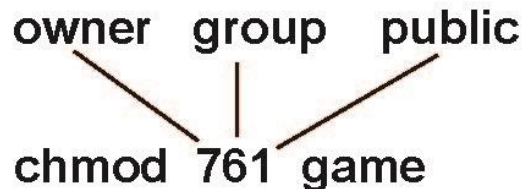
- File owner/creator should be able to control:
 - what can be done
 - by whom
- Types of access
 - **Read**
 - **Write**
 - **Execute**
 - **Append**
 - **Delete**
 - **List**

Access Lists and Groups

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

a) owner access	7	⇒	RWX RWX
b) group access	6	⇒	0 RWX
c) public access	1	⇒	0 0

- 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

ECE3002I/ITP30002 Operating System

File System Implementation

(OSC: Ch. 11.1--11.2)

This lecture note is taken from the instructor's resource of Operating System Concept, 9/e and then partly edited/revised by Shin Hong.

File-System Structure

- File structure
 - Logical storage unit
 - Collection of related information
- **File system** resides on secondary storage (disks)
 - Provided user interface to storage, mapping logical to physical
 - Provides efficient and convenient access to disk by allowing data to be stored, located retrieved easily
- Disk provides in-place rewrite and random access
 - I/O transfers performed in **blocks** of **sectors** (usually 512 bytes)
- **File control block** – storage structure consisting of information about a file
- **Device driver** controls the physical device
- File system organized into layers

File System Layers

- **Device drivers** manage I/O devices at the I/O control layer
 - Given commands like “read drive 1, cylinder 72, track 2, sector 10, into memory location 1060” outputs low-level hardware specific commands to hardware controller
- **Basic file system** given command like “retrieve block 123” translates to device driver
 - Also manages memory buffers and caches (allocation, freeing, replacement)
 - Buffers hold data in transit
 - Caches hold frequently used data
- **File organization module** understands files, logical address, and physical blocks
 - Translates logical block # to physical block #
 - Manages free space, disk allocation

File System Layers (Cont.)

- **Logical file system** manages metadata information
 - Translates file name into file number, file handle, location by maintaining file control blocks (**inodes** in UNIX)
 - Directory management
 - Protection
- Layering useful for reducing complexity and redundancy, but adds overhead and can decrease performance
 - Logical layers can be implemented by any coding method according to OS designer
- Many file systems, sometimes many within an OS

File-System Implementation

- We have system calls at the API level, but how do we implement their functions?
 - On-disk and in-memory structures
- **Boot control block** contains info needed by system to boot OS from that volume
 - Needed if volume contains OS, usually first block of volume
- **Volume control block (superblock, master file table)** contains volume details
 - Total # of blocks, # of free blocks, block size, free block pointers or array
- Directory structure organizes the files
 - Names and inode numbers, master file table

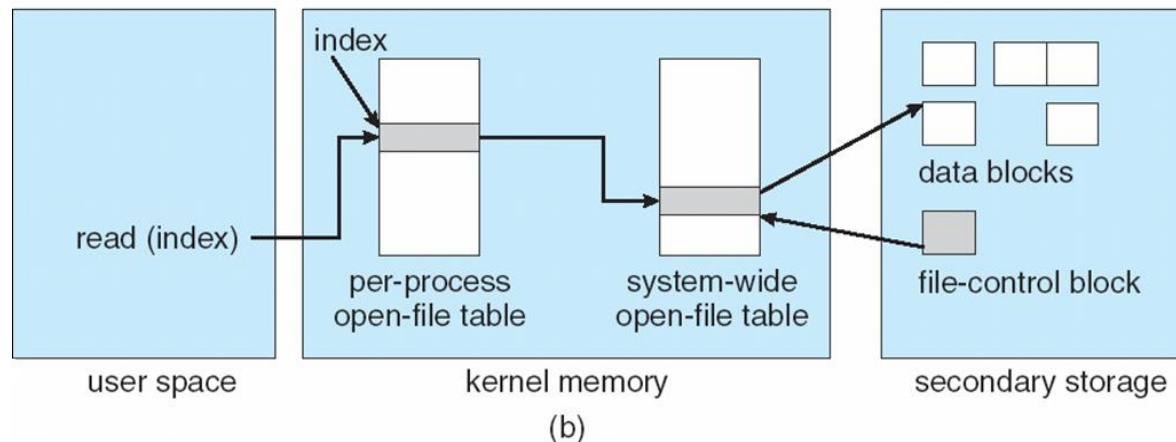
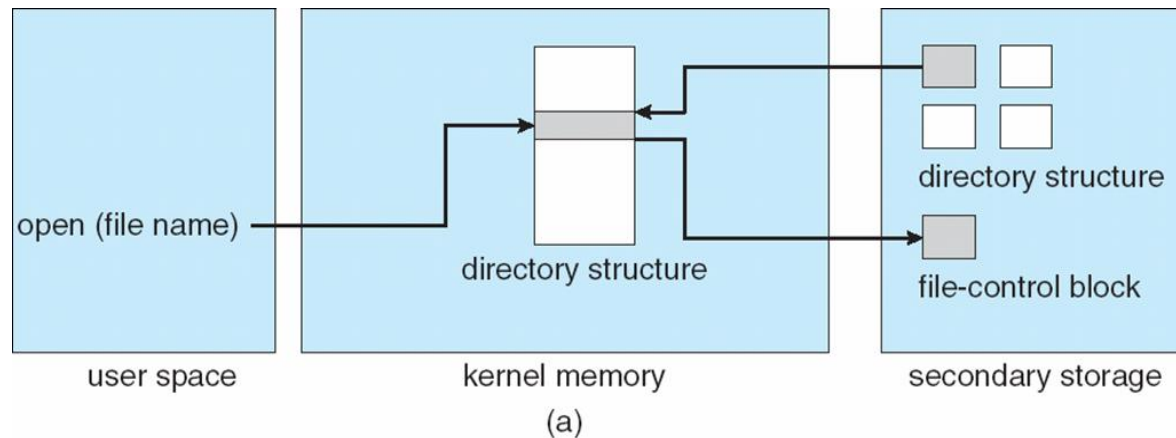
File-System Implementation (Cont.)

- Per-file **File Control Block (FCB)** contains many details about the file
 - inode number, permissions, size, dates
 - NFTS stores into in master file table using relational DB structures

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

In-Memory File System Structures

- Mount table storing file system mounts, mount points, file system types

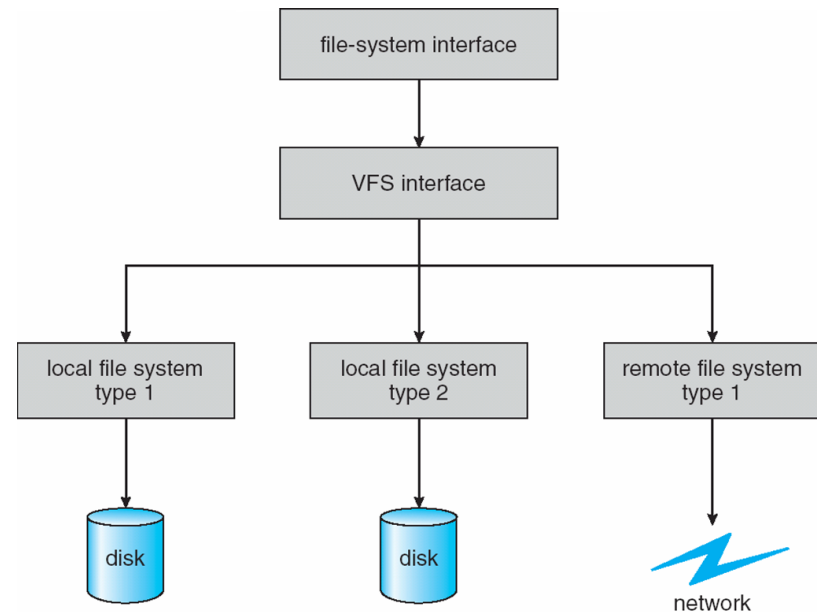


Partitions and Mounting

- Partition can be a volume containing a file system (“cooked”) or **raw** – just a sequence of blocks with no file system
- Boot block can point to boot volume or boot loader set of blocks that contain enough code to know how to load the kernel from the file system
 - Or a boot management program for multi-os booting
- **Root partition** contains the OS, other partitions can hold other Oses, other file systems, or be raw
 - Mounted at boot time
 - Other partitions can mount automatically or manually
- At mount time, file system consistency checked
 - Is all metadata correct?
 - If not, fix it, try again
 - If yes, add to mount table, allow access

Virtual File Systems

- **Virtual File Systems (VFS)** on Unix 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
 - Separates FS generic operations from implementation details
 - Implementation can be one of many file systems types
 - Implements **vnodes** which hold inodes or network file details
 - Then dispatches operation to appropriate FS implementation routines



Virtual File System Implementation

- For example, Linux has four object types:
 - inode, file, superblock, dentry
- VFS defines set of operations on the objects that must be implemented
 - Every object has a pointer to a function table
 - Function table has addresses of routines to implement that function on that object
 - For example:
 - `int open(. . .)`—Open a file
 - `int close(. . .)`—Close an already-open file
 - `ssize_t read(. . .)`—Read from a file
 - `ssize_t write(. . .)`—Write to a file
 - `int mmap(. . .)`—Memory-map a file