File System Interface

Amir H. Payberah
amir@sics.se

Amirkabir University of Technology
(Tehran Polytechnic)

Motivation

Motivation

- For most users, the file system (FS) is the most visible aspect of an OS.

Motivation

- For most users, the file system (FS) is the most visible aspect of an OS.
- Provides mechanism to access data/programs on storage.

Motivation

- For most users, the file system (FS) is the most visible aspect of an OS.
- Provides mechanism to access data/programs on storage.
- The FS consists of two distinct parts:
  - A collection of files.
  - A directory structure that organizes and provides information about all the files in the system.

File Concept

File Concept

- Contiguous logical address space.

File Concept

- Contiguous logical address space.
- Types:
  - Data, e.g., numeric, text data, photos, music, ...
  - Program

File Concept

- Contiguous logical address space.
- Types:
  - Data, e.g., numeric, text data, photos, music, ...
  - Program
- Contents defined by file's creator. Many types:
  - Text file: a sequence of characters.
  - Source file: a sequence of functions.
  - Executable file: a series of code sections that the loader can bring into memory and execute.

File Types: Name and 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, perl, asm	source code in various languages
batch	bat, sh	commands to the command interpreter
markup	xml, html, tex	textual data, documents
word processor	xml, rtf, docx	various word-processor formats
library	lib, a, so, dll	libraries of routines for programmers
print or view	gif, pdf, jpg	ASCII or binary file in a format for printing or viewing
archive	rar, zip, tar	related files grouped into one file, sometimes compressed, for archiving or storage
multimedia	mpeg, mov, mp3, mp4, avi	binary file containing audio or A/V information

File Attributes

- Name: only information kept in human-readable form.

File Attributes

- Name: only information kept in human-readable form.
- Identifier: unique number identifies file within file system.

File Attributes

- Name: only information kept in human-readable form.
- Identifier: unique number identifies file within file system.
- Type: needed for systems that support different types.

File Attributes

- Name: only information kept in human-readable form.
- Identifier: unique number identifies file within file system.
- Type: needed for systems that support different types.
- Location: pointer to file location on device.

File Attributes

- Name: only information kept in human-readable form.
- Identifier: unique 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.

File Attributes

- Name: only information kept in human-readable form.
- Identifier: unique 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.

File Attributes

- Name: only information kept in human-readable form.
- Identifier: unique 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.

File Attributes

- Name: only information kept in human-readable form.
- Identifier: unique 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.

File Operations

- Create

File Operations

- Create
- Write: at write pointer location.

File Operations

- Create
- Write: at write pointer location.
- Read: at read pointer location.

File Operations

- Create
- Write: at write pointer location.
- Read: at read pointer location.
- Reposition within file: seek

File Operations

- Create
- Write: at write pointer location.
- Read: at read pointer location.
- Reposition within file: seek
- Delete

File Operations

- Create
- Write: at write pointer location.
- Read: at read pointer location.
- Reposition within file: seek
- Delete
- Truncate: to erase the contents of a file but keep its attributes.

File Operations

- Create
- Write: at write pointer location.
- Read: at read pointer location.
- Reposition within file: seek
- Delete
- Truncate: to erase the contents of a file but keep its attributes.
- Open(Fi): move the content of entry Fi from disk to memory.

File Operations

- Create
- Write: at write pointer location.
- Read: at read pointer location.
- Reposition within file: seek
- Delete
- Truncate: to erase the contents of a file but keep its attributes.
- Open(Fi): move the content of entry Fi from disk to memory.
- Close(Fi): move the content of entry Fi in memory to directory structure on disk.

Open Files (1/2)

- Open(Fi): move the content of a file to memory
- Search the directory structure on disk for the file.

Open Files (1/2)

- Open(Fi): move the content of a file to memory
- Search the directory structure on disk for the file.
- To avoid the constant searching: `open()` system should be called before a file is first used.
  - Open-file table: tracks open files
  - Per-process table and system-wide table

Open Files (1/2)

- Open(Fi): move the content of a file to memory
- Search the directory structure on disk for the file.
- To avoid the constant searching: `open()` system should be called before a file is first used.
  - Open-file table: tracks open files
  - Per-process table and system-wide table
- When the file is no longer being actively used, it is closed by the process, and the OS removes its entry from the open-file table.

Open Files (2/2)

- Several information are associated with an open file:

Open Files (2/2)

- Several information are associated with an open file:
- File pointer: pointer to last read/write location, per process that has the file open.

Open Files (2/2)

- Several information are associated with an open file:
- File pointer: pointer to last read/write location, per process that has the file open.
- File-open count: counter of the number of times a file is open to allow removal of data from open-file table when last processes closes it.

Open Files (2/2)

- Several information are associated with an open file:
- File pointer: pointer to last read/write location, per process that has the file open.
- File-open count: counter of the 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.

Open Files (2/2)

- Several information are associated with an open file:
- File pointer: pointer to last read/write location, per process that has the file open.
- File-open count: counter of the 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 (1/2)

- File locks allow one process to lock a file and prevent other processes from gaining access to it.

Open File Locking (1/2)

- File locks allow one process to lock a file and prevent other processes from gaining access to it.
- Similar to reader-writer locks.
  - Shared lock similar to reader lock: several processes can acquire concurrently
  - Exclusive lock similar to writer lock: only one process can acquire it

Open File Locking (2/2)

- Mandatory: the OS will prevent access until the exclusive lock is released.

Open File Locking (2/2)

- Mandatory: the OS will prevent access until the exclusive lock is released.
- Advisory: the OS will not prevent applications from acquiring access to the file, and the application must be written so that it manually acquires the lock before accessing the file.

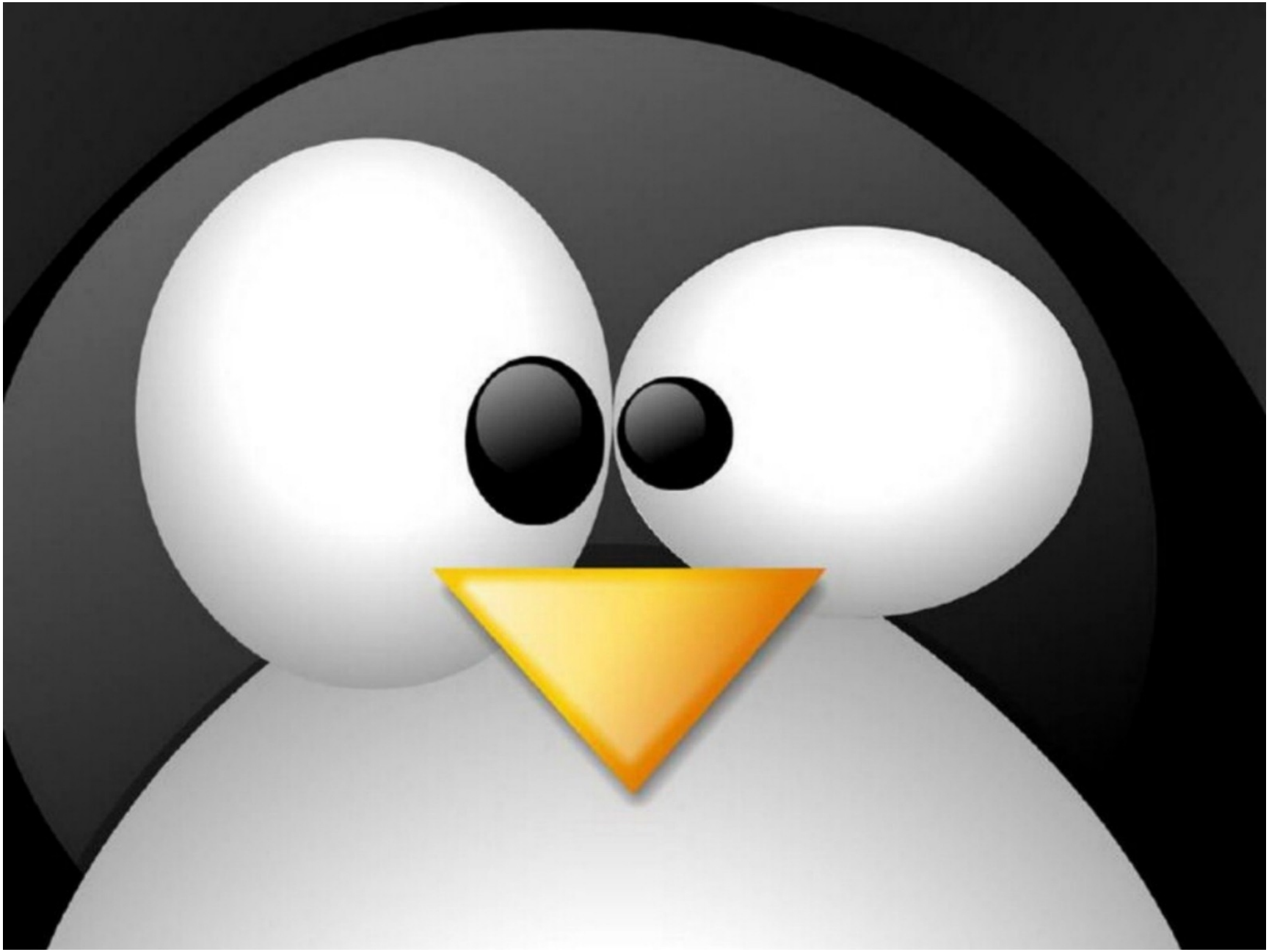
File Structure

- Files must conform to structures that are understood by the OS.
  - E.g., the OS requires that an executable file have a specific structure so that it can determine where in memory to load the file and what the location of the first instruction is.

File Structure

- Files must conform to structures that are understood by the OS.
  - E.g., the OS requires that an executable file have a specific structure so that it can determine where in memory to load the file and what the location of the first instruction is.

- Support multiple file structures?
  - The size of the OS could be big, since it needs to contain the code to support these file structures.
  - On the other hand, severe problems may result if the OS does not support some file structures.



Files and Their Metadata

- The `stat` structure: the metadata of a file.
- Defined in `<bits/stat.h>`, which is included from `<sys/stat.h>`.

```
struct stat {
  dev_t st_dev;          /* ID of device containing file */
  ino_t st_ino;          /* inode number */
  mode_t st_mode;        /* permissions */
  nlink_t st_nlink;      /* number of hard links */
  uid_t st_uid;          /* user ID of owner */
  gid_t st_gid;          /* group ID of owner */
  dev_t st_rdev;         /* device ID (if special file) */
  off_t st_size;         /* total size in bytes */
  blksize_t st_blksize; /* blocksize for filesystem I/O */
  blkcnt_t st_blocks;    /* number of blocks allocated */
  time_t st_atime;       /* last access time */
  time_t st_mtime;       /* last modification time */
  time_t st_ctime;       /* last status change time */
};
```

The Stat Family

- `stat()` returns information about the file denoted by the path `path`.
- `fstat()` returns information about the file represented by the file descriptor `fd`.

```
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>

int stat(const char *path, struct stat *buf);
int fstat(int fd, struct stat *buf);
```

Open a File

- `open()` maps the file to a file descriptor.

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

int open(const char *name, int flags);
```

Read and Write

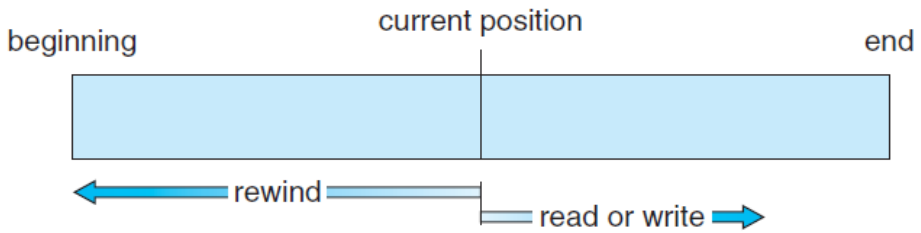
- `read()` and `write()` to read and write from/to a file.

```
#include <unistd.h>

ssize_t read(int fd, void *buf, size_t len);

ssize_t write(int fd, const void *buf, size_t count);
```

Access Methods

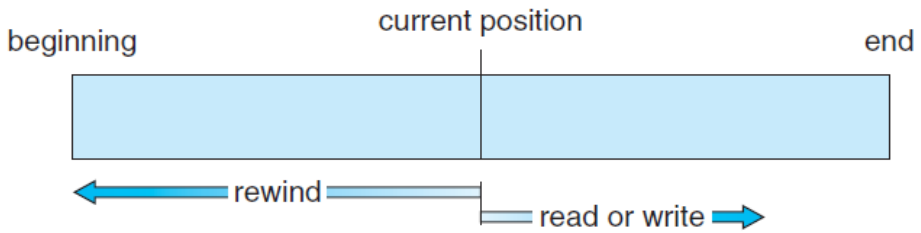
Access Methods - Sequential Access

- The simplest and most common access method.
- Sequential access is based on a tape model of a file.



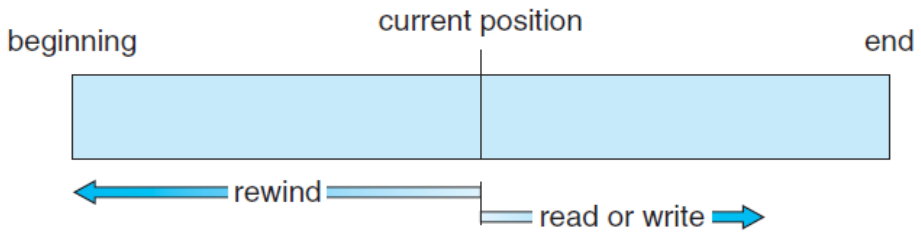
Access Methods - Sequential Access

- The simplest and most common access method.
- Sequential access is based on a tape model of a file.
- Information in the file is processed in order, one record after the other.



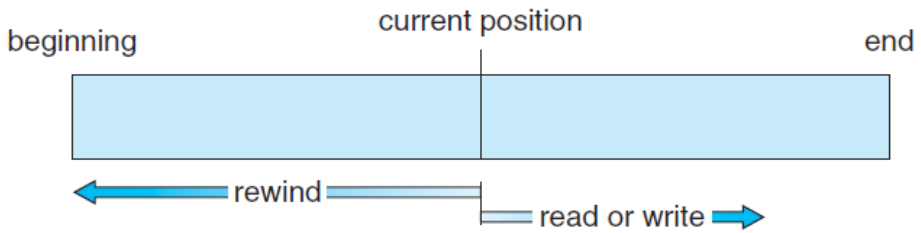
Access Methods - Sequential Access

- The simplest and most common access method.
- Sequential access is based on a tape model of a file.
- Information in the file is processed in order, one record after the other.
- A read operation (`read_next()`): reads the next portion of the file and automatically advances a file pointer.



Access Methods - Sequential Access

- The simplest and most common access method.
- Sequential access is based on a tape model of a file.
- Information in the file is processed in order, one record after the other.
- A read operation (`read_next()`): reads the next portion of the file and automatically advances a file pointer.
- A write operation (`write_next()`): appends to the end of the file and advances to the end of the newly written material.



Access Methods - Direct Access

- A file is made up of fixed-length logical records that allow programs to read and write records rapidly in no particular order.
- It is based on a disk model of a file.

Access Methods - Direct Access

- A file is made up of fixed-length logical records that allow programs to read and write records rapidly in no particular order.
- It is based on a disk model of a file.
- Immediate access to large amounts of information.
  - Databases are often of this type.

Access Methods - Direct Access

- A file is made up of fixed-length logical records that allow programs to read and write records rapidly in no particular order.
- It is based on a disk model of a file.
- Immediate access to large amounts of information.
  - Databases are often of this type.
- `read(n)` rather than `read_next()`.
  - `n` is the block number.

Access Methods - Direct Access

- A file is made up of fixed-length logical records that allow programs to read and write records rapidly in no particular order.
- It is based on a disk model of a file.
- Immediate access to large amounts of information.
  - Databases are often of this type.
- `read(n)` rather than `read_next()`.
  - `n` is the block number.
- `write(n)` rather than `write_next()`.

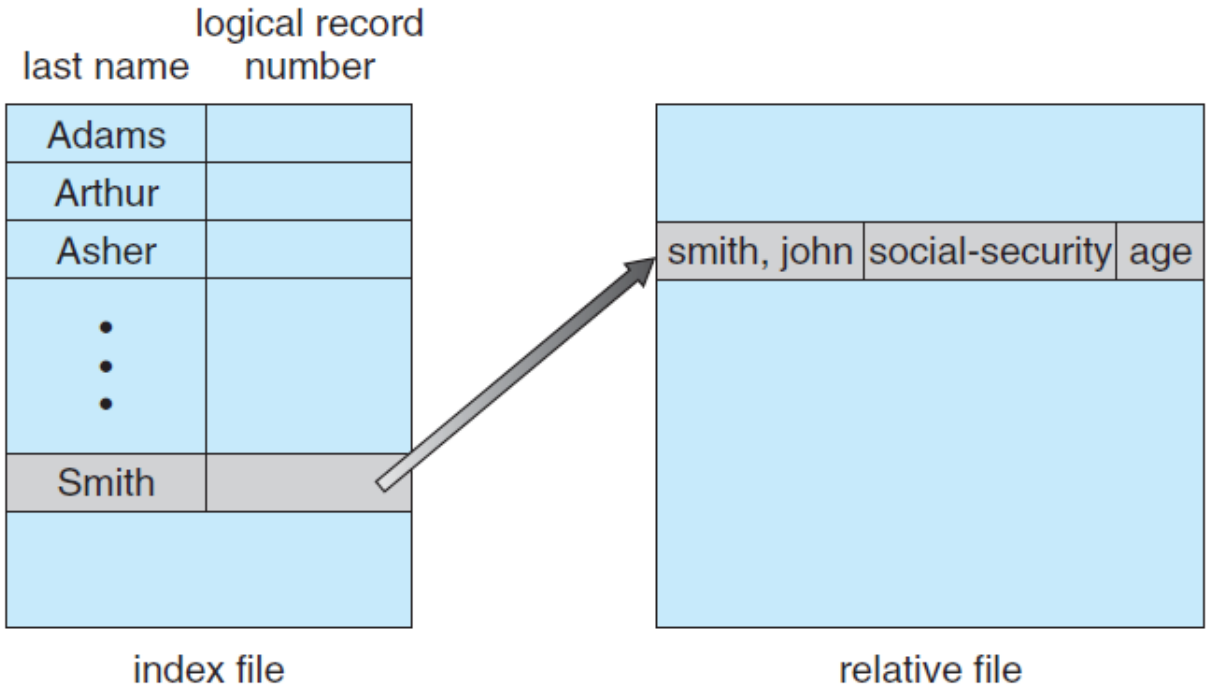
Simulation of Sequential Access on Direct-Access File

sequential access	implementation for direct access
reset	cp = 0;
read_next	read cp ; cp = cp + 1;
write_next	write cp; cp = cp + 1;

Other Access Methods

- Can be built on top of base methods.
- Creation of an index for the file: contains pointers to the various blocks
- Keep index in memory for fast determination of location of data.

Example of Index and Relative Files



Disk Structure

Disk Structure (1/2)

- Disk can be subdivided into partitions.

Disk Structure (1/2)

- Disk can be subdivided into partitions.
- Disks or partitions can be RAID protected against failure.

Disk Structure (1/2)

- 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.

Disk Structure (1/2)

- 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.
- Partitioning is useful for limiting the sizes of individual file systems.

Disk Structure (2/2)

- Entity containing file system known as a volume.

Disk Structure (2/2)

- Entity containing file system known as a volume.
- The volume may be a subset of a device, a whole device, or multiple devices linked together into a RAID set.
  - Each volume can be thought of as a virtual disk.

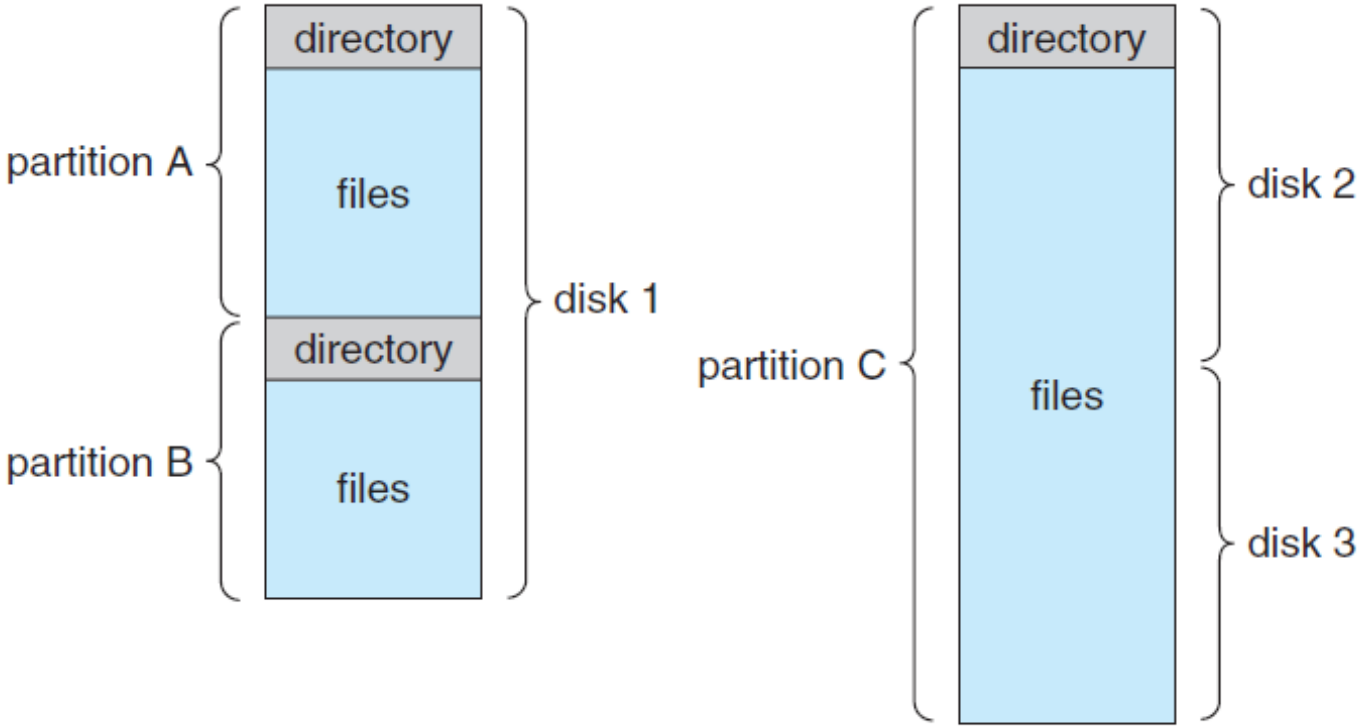
Disk Structure (2/2)

- Entity containing file system known as a volume.
- The volume may be a subset of a device, a whole device, or multiple devices linked together into a RAID set.
  - Each volume can be thought of as a virtual disk.
- Each volume contains information about the files in the system.
  - This information is kept in entries in a device directory or volume table of contents.

Disk Structure (2/2)

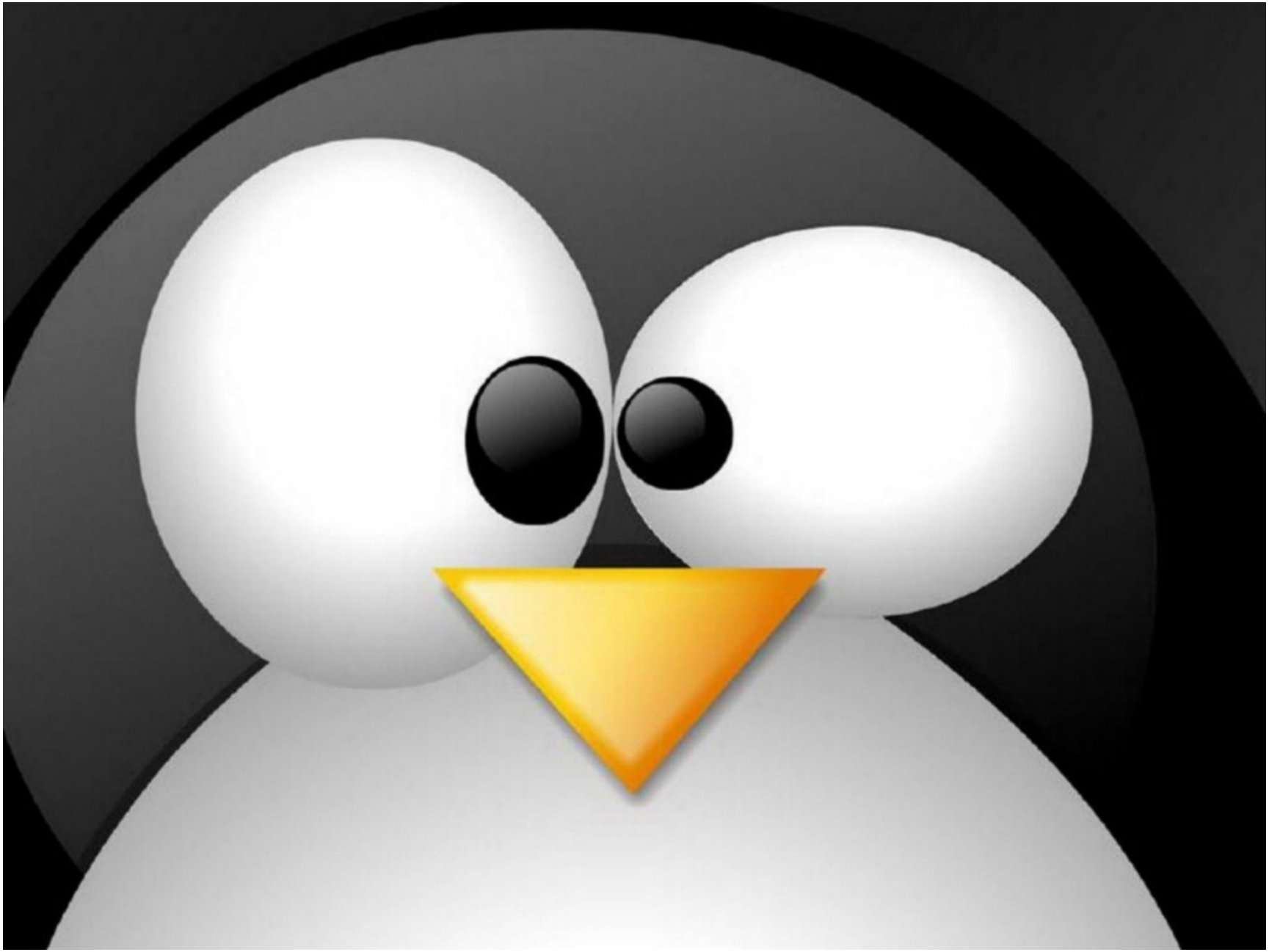
- Entity containing file system known as a volume.
- The volume may be a subset of a device, a whole device, or multiple devices linked together into a RAID set.
  - Each volume can be thought of as a virtual disk.
- Each volume contains information about the files in the system.
  - This information is kept in entries in a device directory or volume table of contents.
- The device directory, (known as the directory), records information such as name, location, size, and type for all files on that volume.

A Typical File-system Organization



Types of File Systems

- Systems may have some general-purpose and some special-purpose file systems.
- 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



Partition Table

- MBR (Master Boot Record)
  - The first sector
  - 512 bytes (446 bytes: boot loader such as GRUB, 64 bytes: partition table, 2 bytes: special code).
- The partition table has enough room for four partitions.
  - One of the four can be used as an extended partition.

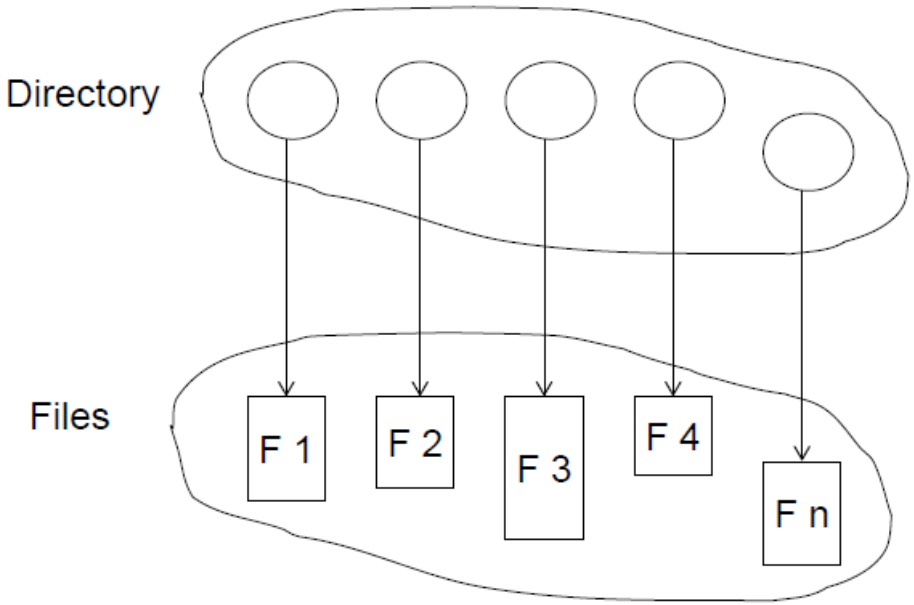
fdisk and mkfs

- `fdisk` and `cfisk` to partition a disk.
- `mkfs.ext4` to format a partition.
  - `mkfs.bfs`, `mkfs.cramfs`, `mkfs.ext2`, `mkfs.ext3`, `mkfs.ext4dev`, `mkfs.minix`, `mkfs.msdos`, `mkfs.vfat`, `...`

Directory Structure

Directory Structure

- The directory can be viewed as a symbol table that translates file names into their directory entries.
- Both the directory structure and the files reside on disk.



Operations Performed on Directory

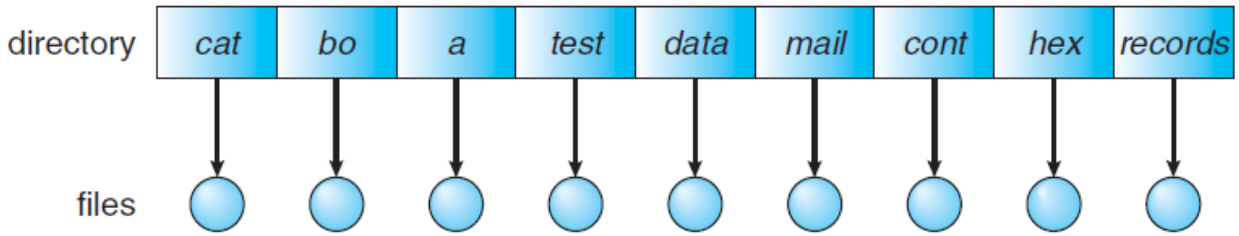
- Search for a file
- Create a file
- Delete a file
- List a directory
- Rename a file
- Traverse the file system

Directory Organization

- The directory itself can be organized in many ways.
  - Single-level directories
  - Two-level directories
  - Tree-level directories
  - Acyclic-graph directories
  - General graph directories

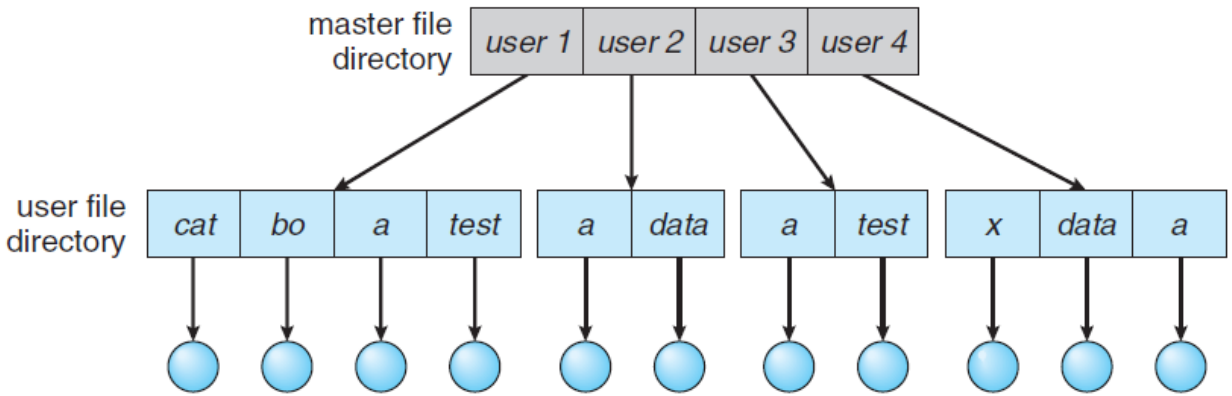
Single-Level Directory

- A single directory for all users.
- Naming problem: they must have unique names.
- Grouping problem



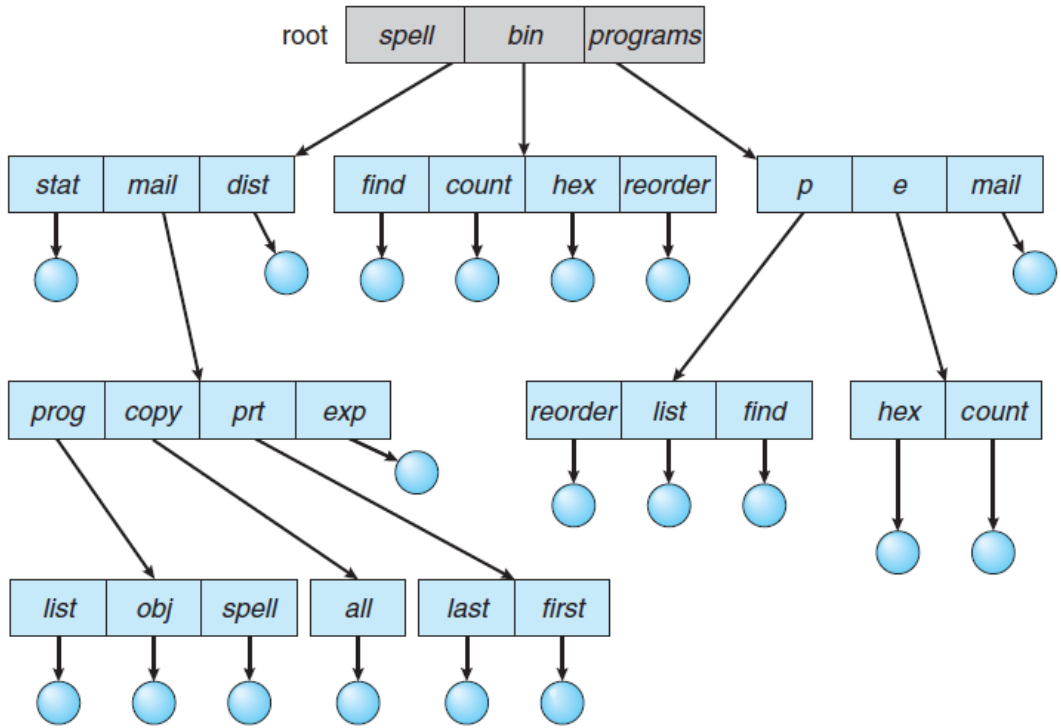
Two-Level Directory

- Separate directory for each user.
- Can have the same file name for different users.
- Efficient searching
- Path name: two level path, e.g., `/userB/file.txt`
- No grouping capability



Tree-Structured Directories (1/2)

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



Tree-Structured Directories (2/2)

- Two types of path names:
  - Absolute path name: begins at the root and follows a path down to the specified file.
  - Relative path name: a path from the current directory.

Tree-Structured Directories (2/2)

- Two types of path names:
  - Absolute path name: begins at the root and follows a path down to the specified file.
  - Relative path name: a path from the current directory.
- Deleting a directory?

Tree-Structured Directories (2/2)

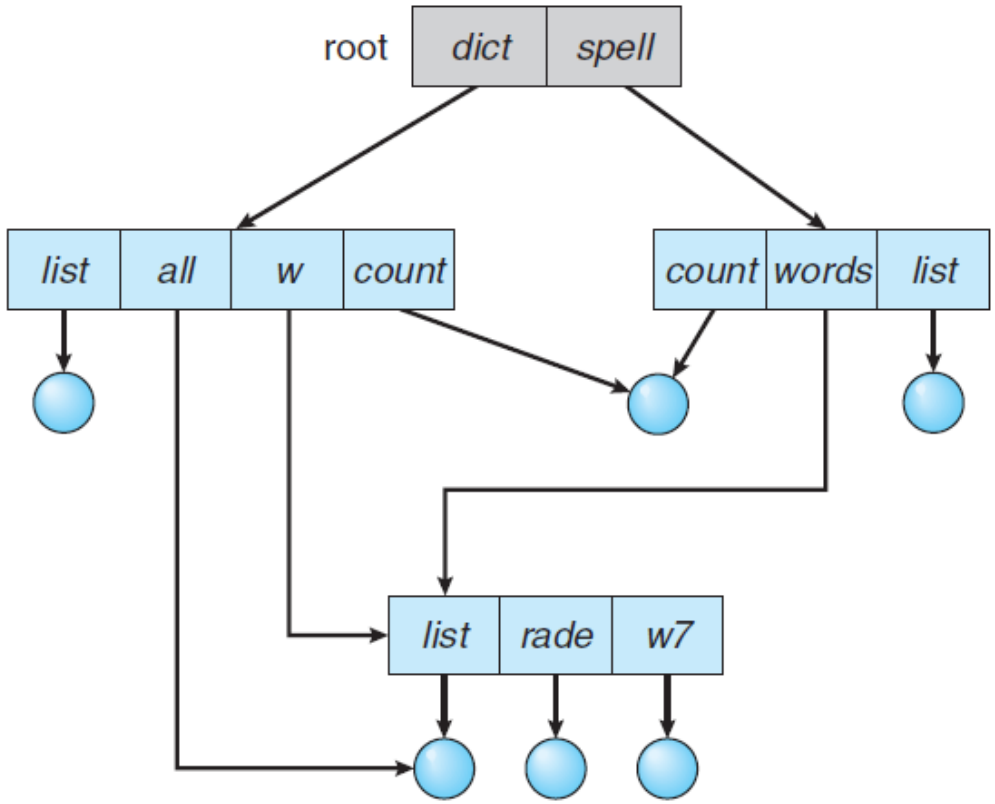
- Two types of path names:
  - Absolute path name: begins at the root and follows a path down to the specified file.
  - Relative path name: a path from the current directory.
- Deleting a directory?
- Simply delete empty directories.

Tree-Structured Directories (2/2)

- Two types of path names:
  - Absolute path name: begins at the root and follows a path down to the specified file.
  - Relative path name: a path from the current directory.
- Deleting a directory?
- Simply delete empty directories.
- If a directory is not empty.
  - The user must first delete all the files in that directory.
  - Or using an option to delete a directory as in Linux `rm`

Acyclic-Graph Directories (1/3)

- Have shared subdirectories and files
- With a shared file, only one actual file exists, so any changes made by one person are immediately visible to the other.



Acyclic-Graph Directories (2/3)

- Two approaches to implement shared files:

- Link
  - Another name (pointer) to an existing file.
  - Resolve the link: follow pointer to locate the file.

Acyclic-Graph Directories (2/3)

- Two approaches to implement shared files:
- Link
  - Another name (pointer) to an existing file.
  - Resolve the link: follow pointer to locate the file.
- Duplicate all information about the file.
  - Both entries are identical and equal.
  - Consistency?

Acyclic-Graph Directories (3/3)

- Deletion possibilities?

Acyclic-Graph Directories (3/3)

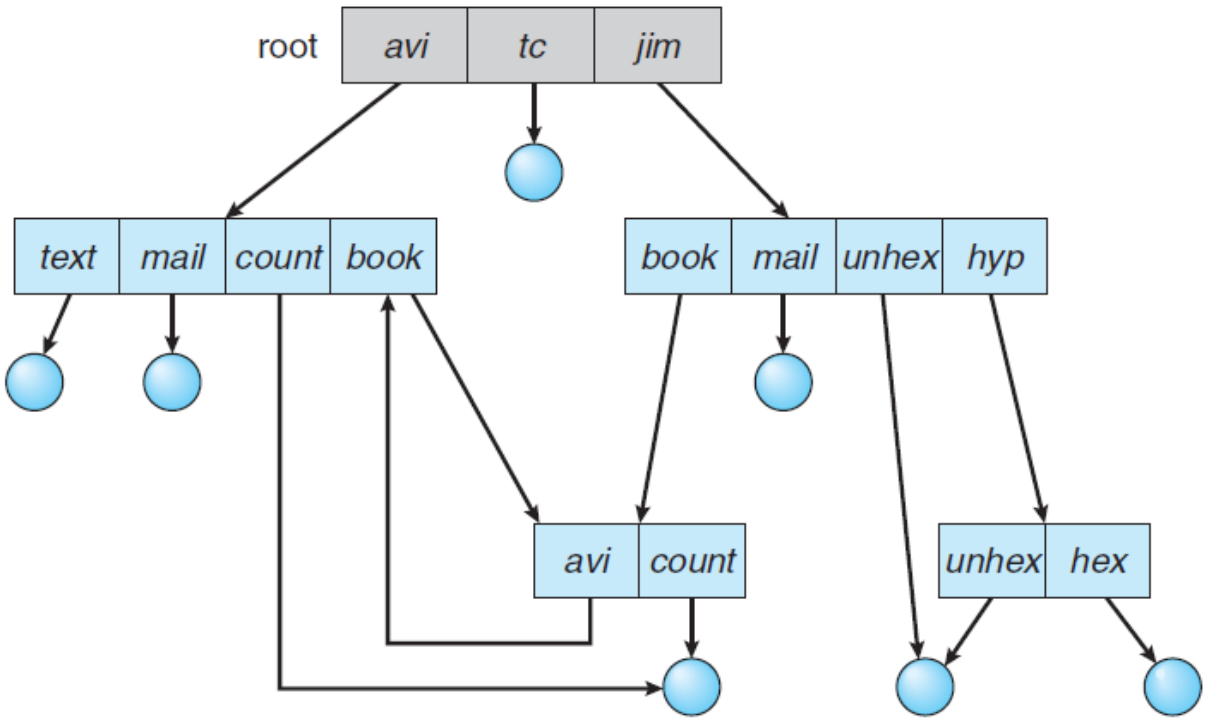
- Deletion possibilities?

- Remove the file content whenever anyone deletes it.
  - Dangling pointers: pointing to the nonexistent file.
  - What if the remaining file pointers contain actual disk addresses?
  - Easy with soft-links (symbolic links)

Acyclic-Graph Directories (3/3)

- Deletion possibilities?
- Remove the file content whenever anyone deletes it.
  - Dangling pointers: pointing to the nonexistent file.
  - What if the remaining file pointers contain actual disk addresses?
  - Easy with soft-links (symbolic links)
- Preserve the file until all references to it are deleted.
  - Hard links

General Graph Directory (1/2)



General Graph Directory (2/2)

- How do we guarantee no cycles?

General Graph Directory (2/2)

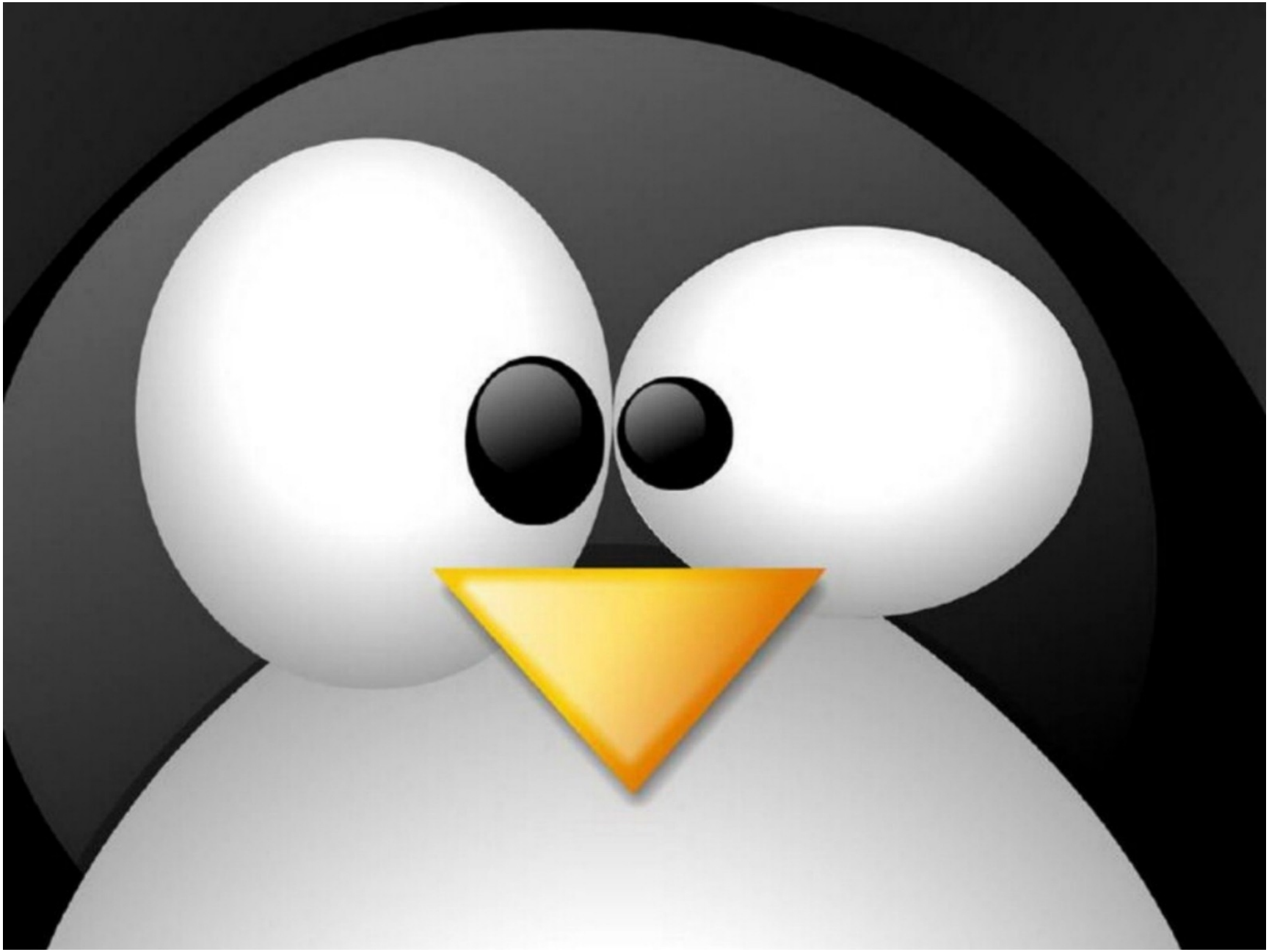
- How do we guarantee no cycles?
- Allow only links to file not subdirectories.

General Graph Directory (2/2)

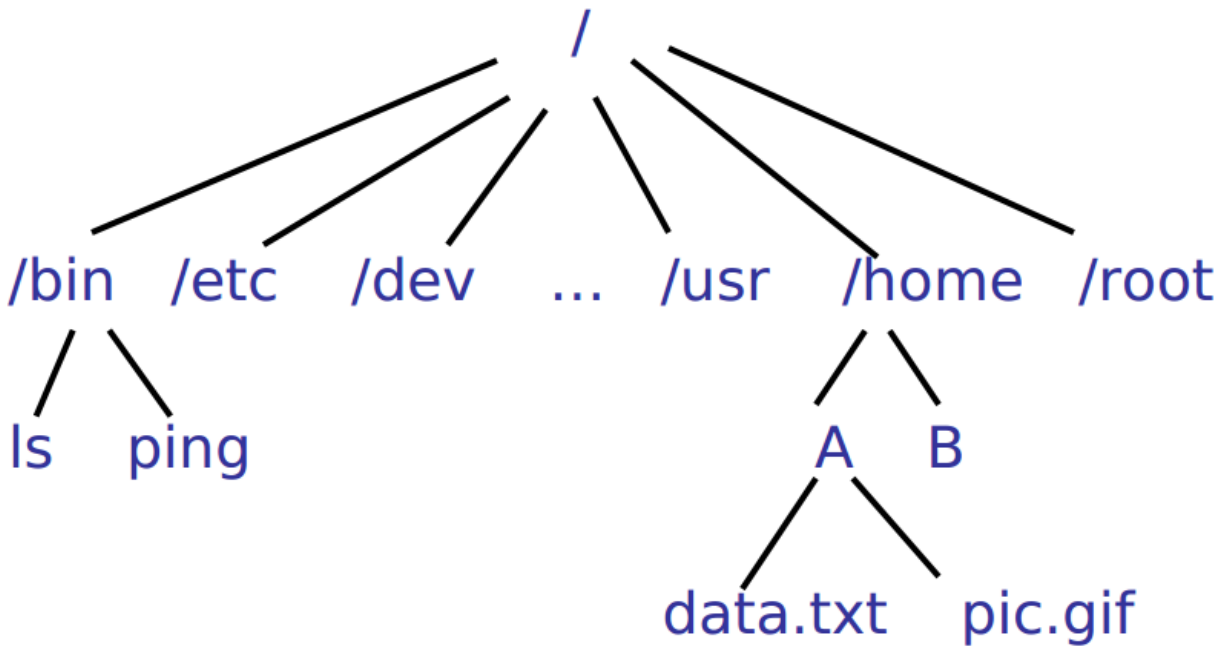
- How do we guarantee no cycles?
- Allow only links to file not subdirectories.
- Garbage collection: determine when the last reference has been deleted and the disk space can be reallocated.

General Graph Directory (2/2)

- How do we guarantee no cycles?
- Allow only links to file not subdirectories.
- Garbage collection: determine when the last reference has been deleted and the disk space can be reallocated.
- Every time a new link is added use a cycle detection algorithm to determine whether it is OK.
  - Easier solution: bypass links during directory traversal.



Linux File System



/bin

- Hold the most commonly used essential user programs.
  - `login`
  - Shells (`bash`, `ksh`, `csh`)
  - File manipulation utilities (`cp`, `mv`, `rm`, `ln`, `tar`)
  - Editors (`ed`, `vi`)
  - File system utilities (`dd`, `df`, `mount`, `umount`, `sync`)
  - System utilities (`uname`, `hostname`, `arch`)
  - GNU utilities (`gzip`, `gunzip`)

/sbin

- Hold essential maintenance or system programs:
  - `fsck, fdisk, mkfs, shutdown, init, ...`

/sbin

- Hold essential maintenance or system programs:
  - `fsck, fdisk, mkfs, shutdown, init, ...`
- The main difference between the programs stored in `/bin` and `/sbin` is that the programs in `/sbin` are executable only by root.

/etc

- Store the system wide configuration files required by many programs:
  - `passwd, shadow, fstab, hosts, ...`

/home and /root

- The /home directory: the home directories for all users.
- The /root directory: the home directories for root user.

/dev

- The special files representing hardware are kept in it.
  - `/dev/hda1`
  - `/dev/ttyS0`
  - `/dev/mouse`
  - `/dev/fd0`
  - `/dev/fifo1`
  - `/dev/loop2`

/tmp and /var

- The /tmp and /var directories: hold temporary files or files with constantly varying content.
- The /tmp directory: files that only need to be used briefly and can afford to be deleted at any time.
- The /var directory: a bit more structured than /tmp.

/usr

- Most programs and files directly relating to users of the system are stored.
- It is in some ways a mini version of the / directory.
  - /usr/bin
  - /usr/sbin
  - /usr/spool

/proc

- It is a virtual file system
- Provided by the kernel
- Provides information about the kernel and processes.

File and Directory Management

- `getcwd()` returns the current working directory.
- `chdir()` changes the current working directory to `path`

```
#include <unistd.h>

char *getcwd(char *buf, size_t size);
int chdir(const char *path);
```

File and Directory Management

- `mkdir()` creates the directory path.

```
#include <sys/stat.h>
#include <sys/types.h>

int mkdir(const char *path, mode_t mode);
```

- `rmdir()` removes a directory from the filesystem.

```
#include <unistd.h>

int rmdir(const char *path);
```

File and Directory Management

- `opendir()` creates a directory stream representing.
- `readdir()` returns the next entry in the directory.
- `closedir()` closes the directory stream.

```
#include <sys/types.h>
#include <dirent.h>

DIR *opendir(const char *name);;
struct dirent *readdir(DIR *dir);
int closedir(DIR *dir);
```

File System Commands (1/3)

- `pwd`: where am I?
- `cd`: changes working directory.
- `ls`: shows the contents of current directory.
- `cat`: takes all input and outputs it to a file or other source.
- `mkdir`: creates a new directory
- `rmdir`: removes empty directory

File System Commands (2/3)

- `mv`: moves files
- `cp`: copies files
- `rm`: removes directory
- `gzip/gunzip`: to compress and uncompress a file
- `tar`: to compress and uncompress a file
- `e2fsck`: check a Linux ext2/ext3/ext4 file system

File System Commands (3/3)

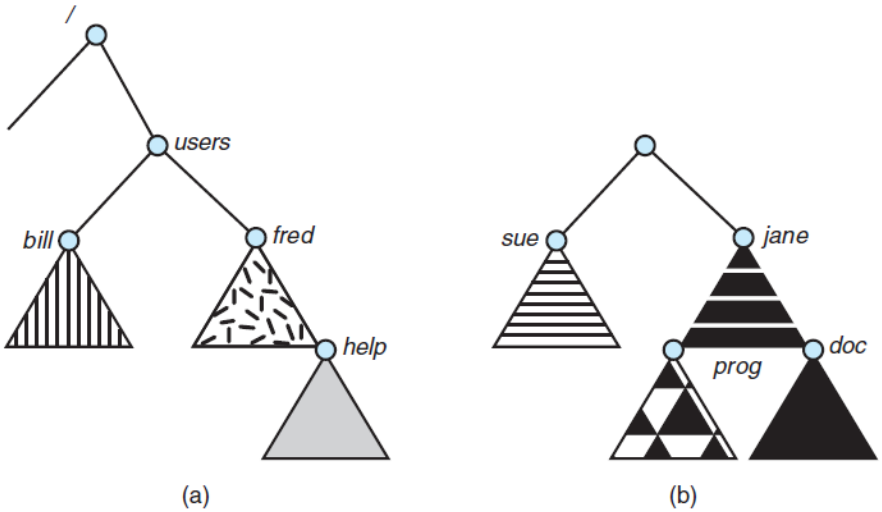
- `dd`: converts and copies a file
- `df`: reports File System disk space usage
- `du`: estimates file space usage
- `ln`: makes links between files
- `file`: determines file type

File System Mounting

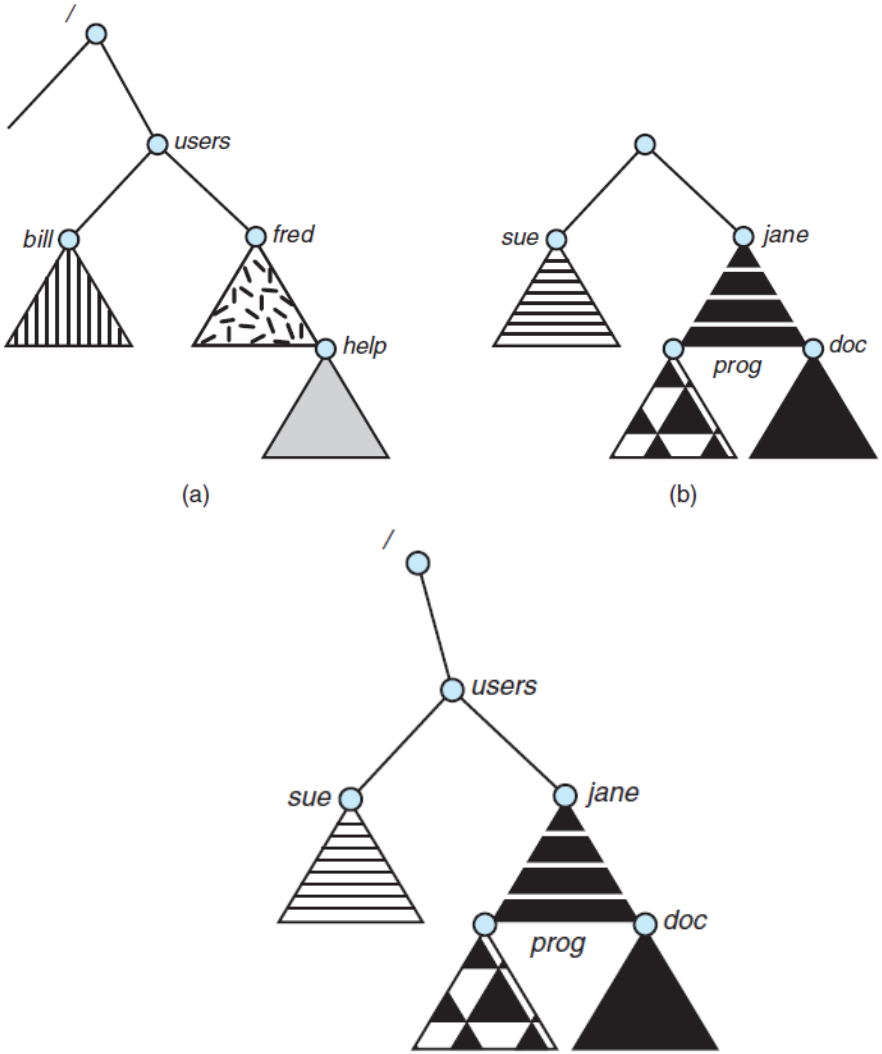
File System Mounting

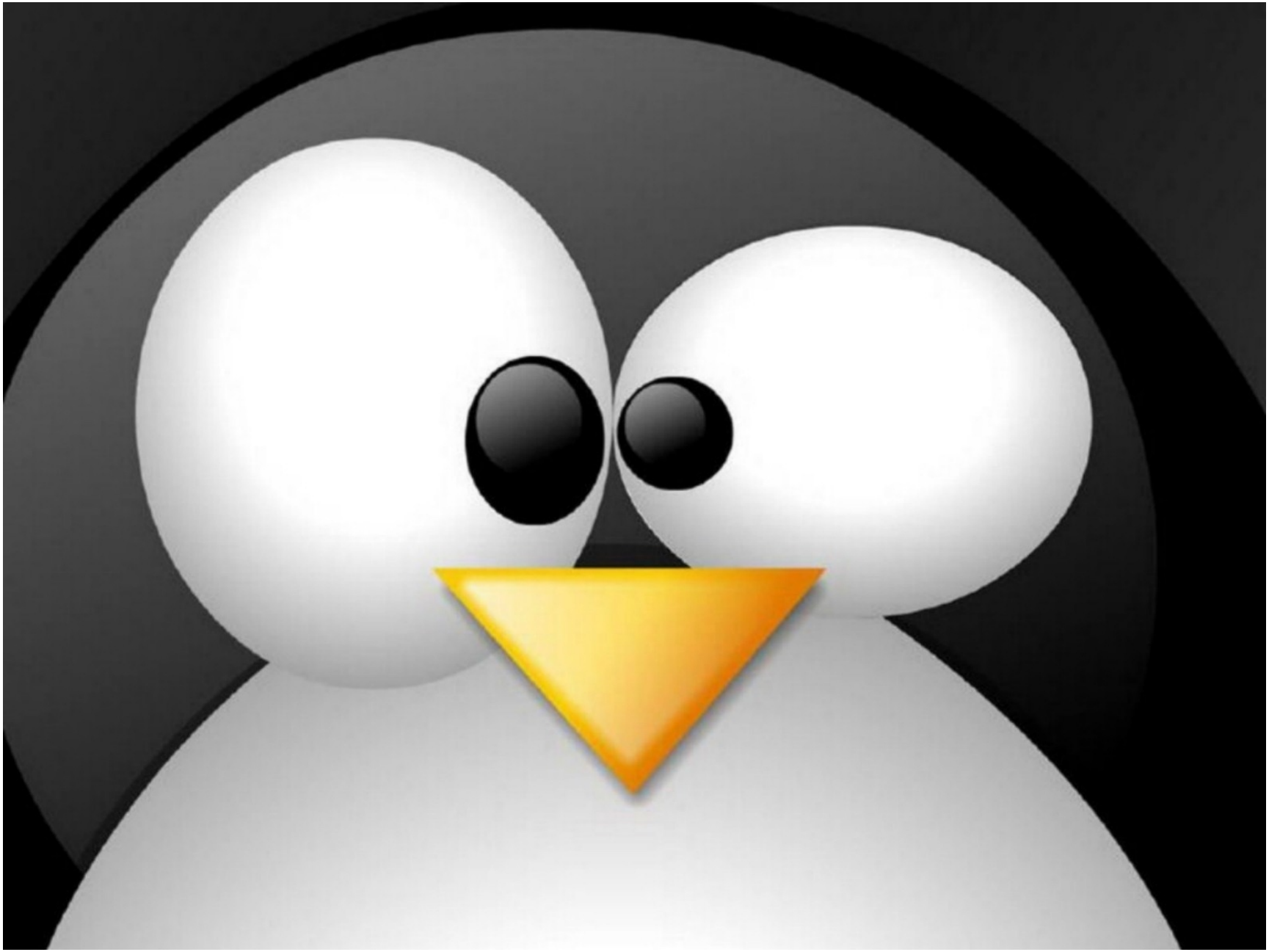
- A file system must be mounted before it can be accessed.
- A unmounted file system is mounted at a mount point.
- Mount point: the location within the file structure where the file system is to be attached.

Mount Point



Mount Point


/
users
bill
fred
help
(a)
sue
jane
prog
doc
(b)
/
users
sue
jane
prog
doc



Mounting File System

- File system are mounted with the `mount` command.
  `mount -t type source mount_point`
- To unmount a file system, the `umount` command is used.
  `umount /dev/<device name> or mount_point`

Mounting Automatically With fstab

- This file lists all the partitions that need to be mounted at boot time and the directory where they need to be mounted.
- /etc/fstab
  - Which devices to be mounted.
  - What kinds of file systems they contain.
  - At what point in the file system the mount takes place.

```
# <file system>   <mount point>   <type>  <options>          <dump>  <pass>
UUID=79257dad-... /               ext4    errors=remount-ro  0       1
UUID=2e84fea4-... /home           ext4    defaults           0       2
UUID=7cf4a322-... none            swap    sw                 0       0
```

File Sharing and Protection

File Sharing

- Sharing of files on multi-user systems is desirable.
- Sharing may be done through a protection scheme.
  - User IDs identify user
  - Owner of a file/directory: the user who can change attributes and grant access and who has the most control over the file.
  - Group of a file/directory: a subset of users who can share access to the file.

File Sharing - Remote File Systems (1/2)

- Client-server model allows clients to mount remote file systems from servers:
- Server can serve multiple clients.

File Sharing - Remote File Systems (1/2)

- Client-server model allows clients to mount remote file systems from servers:
- Server can serve multiple clients.
- NFS is standard UNIX client-server file sharing protocol.
- CIFS is standard Windows client-server protocol.

File Sharing - Remote File Systems (1/2)

- Client-server model allows clients to mount remote file systems from servers:
- Server can serve multiple clients.
- NFS is standard UNIX client-server file sharing protocol.
- CIFS is standard Windows client-server protocol.
- Standard OS file calls are translated into remote calls.

File Sharing - Remote File Systems (2/2)

- 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.

File Sharing - Failure Modes

- All file systems have failure modes.
- Remote file systems add new failure modes, due to network and server failure.

File Sharing - Failure Modes

- All file systems have failure modes.
- Remote file systems add new failure modes, due to network and server failure.
- Recovery from failure can involve state: information about status of each remote request.

File Sharing - Failure Modes

- All file systems have failure modes.
- Remote file systems add new failure modes, due to network and 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.

File Sharing - Consistency Semantics

- Specify how multiple users are to access a shared file simultaneously.
- Similar to process synchronization algorithms.

File Sharing - Consistency Semantics

- Specify how multiple users are to access a shared file simultaneously.
- Similar to process synchronization algorithms.
- Unix file system (UFS):
  - 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

File Sharing - Consistency Semantics

- Specify how multiple users are to access a shared file simultaneously.
- Similar to process synchronization algorithms.
- Unix file system (UFS):
  - 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
- Andrew File System (AFS):
  - Writes only visible to sessions starting after the file is closed

Access Lists and Groups

- Mode of access: read, write, execute (rwx)
- Make access dependent on the identity of the user.
- Different users may need different types of access to a file or directory.

Access Control List

- Associate access-control list (ACL) with each file and directory.
  - Specifying user names and the types of access allowed for each user.
- Three classes of users:
  - Owner: the user who created the file.
  - Group: a set of users who are sharing the file and need similar access.
  - Universe: all other users in the system.

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

Summary

- File concept: types, attributes, operations, locks

Summary

- File concept: types, attributes, operations, locks
- Access methods: sequential, direct

Summary

- File concept: types, attributes, operations, locks
- Access methods: sequential, direct
- Disk structure and file system

Summary

- File concept: types, attributes, operations, locks
- Access methods: sequential, direct
- Disk structure and file system
- Directory structure: single-level, two-level, tree-structured, acyclic-graph, general-graph

Summary

- File concept: types, attributes, operations, locks
- Access methods: sequential, direct
- Disk structure and file system
- Directory structure: single-level, two-level, tree-structured, acyclic-graph, general-graph
- Mounting

Summary

- File concept: types, attributes, operations, locks
- Access methods: sequential, direct
- Disk structure and file system
- Directory structure: single-level, two-level, tree-structured, acyclic-graph, general-graph
- Mounting
- File sharing and protection: rwx, owner, group, universe

Questions?

Acknowledgements

Some slides were derived from Avi Silberschatz slides.