# Lecture Topics: 11/29

- File System Interface
  - Files and Directories
  - Access Methods
  - Protection
  - Consistency

# Files

- A user-level abstraction for "a collection of bytes in (non-volatile) storage"
- Files have:
  - A name
  - Possibly a type
  - A location - which device, where on that device
  - Size (and possibly maximum size)
  - Protection - who may read and write this?
  - Time, date, and user identification

## File Operations

- File creation
  - make room for the file
  - enter the new file into the directory
- Writing a file
  - specify the file and the data to write to the file
  - OS keeps track of your location in the file
  - successive writes are placed one after the other in the file

## More File Operations

- Reading a file
  - specify the file and the buffer into which the data should be read
  - OS keeps track of your location in the file
  - Location pointer is often shared between read and write operations
- Repositioning within a file
  - Changes the location pointer
  - Often called "seeking"
  - No actual I/O

## Yet More File Operations

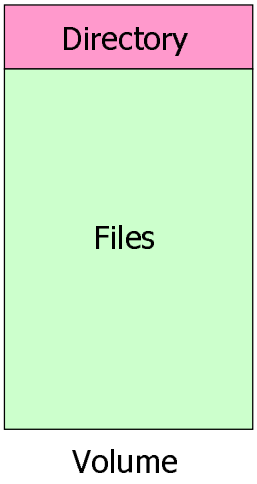
- Deleting a file
  - Find the directory entry and delete it
  - Mark the space used by the file as free
  - Don't actually "erase" the file
- Truncating a file
  - Throw away all the data in the file
  - Keep the attribute information

## Opening and Closing Files

- The above six operations are enough
- But for convenience, we have the notion of the *open file*
- The open system call tells the OS that the specified file will be used by several operations
  - user need not specify name each time
  - OS need not search directories each time
  - Location pointers, etc. need only be maintained for open files

# Directories

- The *directory* lists all of the files in the *volume*
  - A volume is a logical disk - there may be more than one volume per physical disk
- We put the directory at the beginning so we always know where to start




# Single-Level Directories

- In a single-level directory structure, the directory lists all files and their offsets
- Like a table of contents

| | |
|---|---|
| notes410 | 5 |
| Spring99 | 12 |
| todo | 55 |
| ideas | 59 |
| notes 410 | |
| Spring99 | |
| todo | |
| ideas | |
| | |

## Two-Level Directories

- Single-level directories suffer from *name collision*
  - If you and I both name a file "prog1.c" then one file will overwrite the other
- Split up the space into top-level directories for each user
- Keep a directory for each user's files, and a directory of the user directories

## Tree-Structured Directories

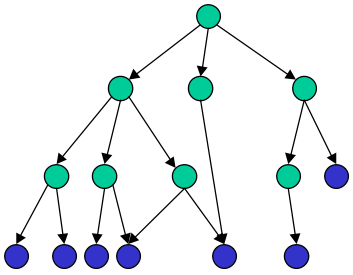
- Let directories contain subdirectories
- Arrange files in a tree
- To name a file, specify a list of directories from the top down, plus the name of the file itself
  - This is called a *path name*
- A path beginning at the root is an *absolute path*; if part of the path is implied, it's a *relative path*

## The Current Directory

- Set the *current directory* with setcwd() system call
- All future open() calls interpret path names relative to the current directory
  - Saves on directory lookups
- Initial current directory is often set at login time, to the user's *home directory*

## Acyclic Graph Directories

- Like tree structures, but some nodes have more than one parent
- Facilitates sharing
- But how should deletion work?
- *hard links* vs. *soft* or *symbolic links*



## General Graph Directories

- Acyclic graphs are usually right, but it's expensive to check for cycles
- Instead, many systems allow any kind of graph
  - A is a subdirectory of B, and B is a subdirectory of A
- Now, applications that traverse the tree have to be careful
- Also, reference counting for deletion may not work

## File Protection

- Protection allows the owner of a file or directory to define *who* may do *what* to that file or directory
  - The *who* is restricted by user or group
  - The *what* is restricted by type of access:
    - read, write, execute (Unix)
    - read, lookup, insert, delete, write, lock, administrative (AFS)

## Access Control Lists

- Access Control Lists (ACLs) are properties of a file or a directory
- They list what operations particular users are allowed to execute
- An AFS-ish example:

```
> fs listacl research
system:anyuser none
gretta all
levy read
```

## Drawbacks of ACLs

- Directory entries are now variable sized
- Listing all users who may read a file can be tedious
  - Workarounds: specify polices for groups
  - Apply ACLs to directories rather than files
  - Allow negative ACLs

## Unix File System Protection

- A simplified ACL scheme
- Only three classifications of users:
  - Owner, Group, and Universe
- Only three operations:
  - Read, Write, Execute
- Now all protection information can be specified with nine bits
  - see these on a Unix system with "ls -l"

## Consistency Semantics

- Suppose two users have the same file open at the same time
- If one of them writes to the file,
  - when should the other user see the changes? immediately? not until reopening?
- Should the location pointer be shared or separate?

## Consistency Policies

- Unix (for local files): writes are immediately visible, location pointer may be shared
- AFS: writes by another user are not visible until the file is closed and reopened
- Immutable-Shared-Files: files may be either shared or writable but not both (locking)