CMSC 412

Filesystems: Interfaces

Announcements

- Reading
  - Today : Chapter 11
  - Next time : Chapter 12

File Abstraction

- What is a file?
  - A *named* collection of information stored on secondary storage
- Properties of a file
  - non-volatile (persistent)
  - can read, store, or update it
  - has *meta-data* to describe attributes of the file
  - May be *structured* or *unstructured*

File Attributes

- **name**: a way to describe the file
- **type**: some information about what is stored in the file
- **location**: how to find the file on disk
- **size**: number of bytes
- **protection**: access control
  - may be different for read, write, execute, append, etc.
- **time**: access, modification, creation
- **version**: how many times the file changed

File Operations

- Files are an *abstract data type (ADT)*
  - **Interface**: what can I do with them?
  - **Implementation**: how do I implement them?
- Operations
  - Create, Open, Read, Write, Fsync, Seek, Delete, Truncate, Close, Read-Meta-Data, Write-Meta-Data

Create, Open

- create
  - assign it a name
  - check permissions
- open
  - check permissions
  - check that the file exists
  - lock the file (if we don't what to permit other users a the same time)
  - may provide *file pointer* for access

Write

- Indicate what file to write (either name of handle)
- Provide data to write
- Specify where to write the data within the file
  - generally this is implicit (file pointer)
  - could be explicit (direct access)

Read

- Indicate what file to read (either name or handle)
- Provide place to put information read
- Indicate how much to read
- Specify where to write the data within the file
  - generally this is implicit (file pointer)
  - could be explicit (direct access)

Fsync, Close

- fsync:
  - synchronize disk version with in-core version
  - ensures any previous writes to the file are stored on disk
- close
  - unlock the file (if locked when opened)
  - update meta data about time
  - free system resources (file descriptors, buffers)

Seek, Delete, Truncate

- seek
  - move the implicit file pointer to a new offset in the file
- delete
  - remove named file
- truncate
  - remove the data in the file from the current position to end

5

Read-MD, Write-MD

- read meta data
  - get file size, time, owner, etc.
- write meta data
  - change file size, time, owner, etc.

Filesystems

- Provides a *namespace* for files via directories
- Can store files of variable size
- Provides protection by restricting access to files based on permissions

Directory

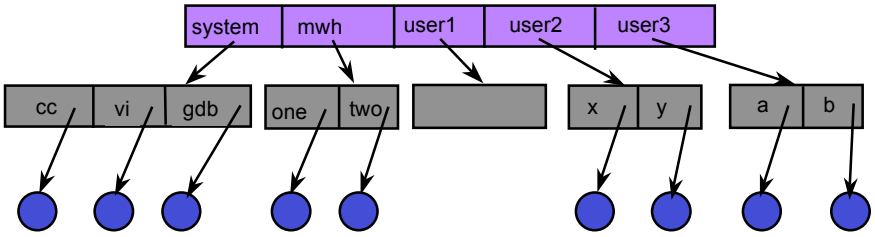
- Collection of files
- Operations
  - Search for a file
  - Create a file
  - Delete a file
  - List a directory
  - Rename a file
  - Traverse the file system

Single Directory Structure

- All files are in a single global namespace
- Simple, but having all of the files in one name space is awkward
  - lots of files to sort through
  - different users would have to coordinate file names
  - each file has to have a unique name

Two-level Directory Structure

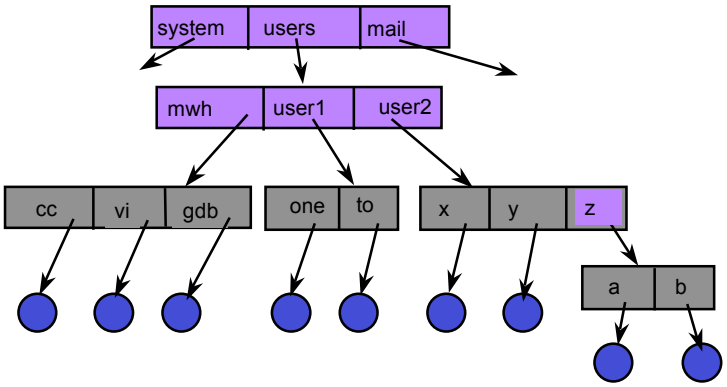
- Top level is users, second level is files per user
  - Less awkward, but still not much control



Tree Directories

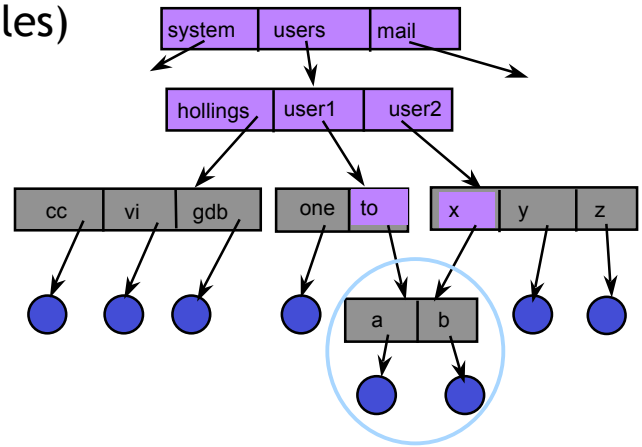
- Create a tree of files
  - Each directory can contain files or directory entries
  - Thus, each non-leaf in the tree is a directory
- Each process has a current directory
  - can name files *relative* to that directory
  - can change current directory as needed

Tree Directories



Acylic Graph Directories

- Users can share directories (and/or files)



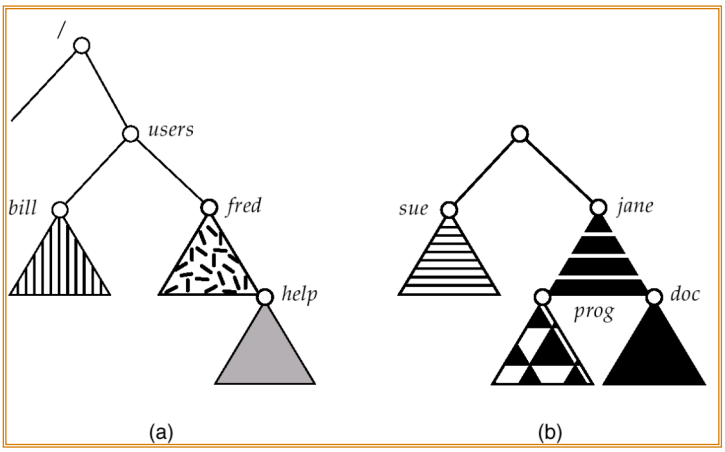
Acylic Graph Issues

- Same file may have several names
  - absolute path name differs, but file is the same
  - similar to memory aliases in prog. languages
- Deletion
  - if one user deletes a file
    - It is deleted for all users, since the directory is shared
  - if one user "deletes" their shared directory
    - The directory stays until the last user deletes it
    - Maintains a reference count to determine this
- Programs to walk the DAG need to be aware
  - disk usage utilities, backup utilities

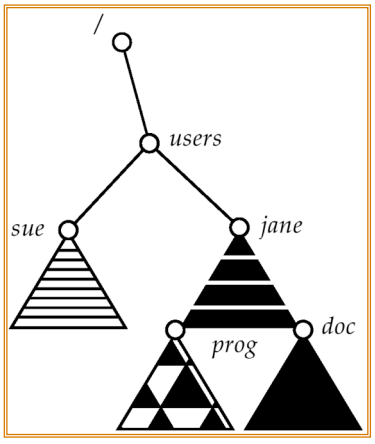
File System Mounting

- A file system must be mounted before it can be accessed
- A unmounted file system is mounted at a mount point

Mounted vs. Unmounted



Mount Point



11

OS Awareness of File Contents

- Needs to know about some types of files
  - Directories, executables
- What about others?
  - Example: word processing file vs. spreadsheet
  - Advantages:
    - OS knows what application to run
    - Automatic make (tops-20)
      - if source changed, re-compile before running
  - Problems:
    - to add new type, may need to extend OS
    - OS vs. application features are blurred
    - what if a file is several types
      - consider a compressed postscript file

Example of File Types

- Macintosh
  - has a file type that is part of file meta-data
  - also has an application associated with each file type
- Windows 95/NT
  - has a file type in the extension of the file name
  - has a table (per user) to map extensions to applications
- Unix
  - can use last part of filename like an extension
  - applications can decide what (if anything) to do with it

12

File Protection

- File sharing implies need of **protection**:
  - How to give access to some users and not others?
- Access types:
  - read, write, execute, append, delete, list
  - rename: often based on protection of directory
  - copy: usually the same as read

Access Policies

- Access lists
  - list for each user for each file the permitted operations
- Groups
  - enumerate users in a list called a group
  - same protection to all members of the group
  - depending on system:
    - files may be in one or many groups
    - users may be in one or many groups
- Per-file passwords
  - tedious and a security problem

UNIX File Protection

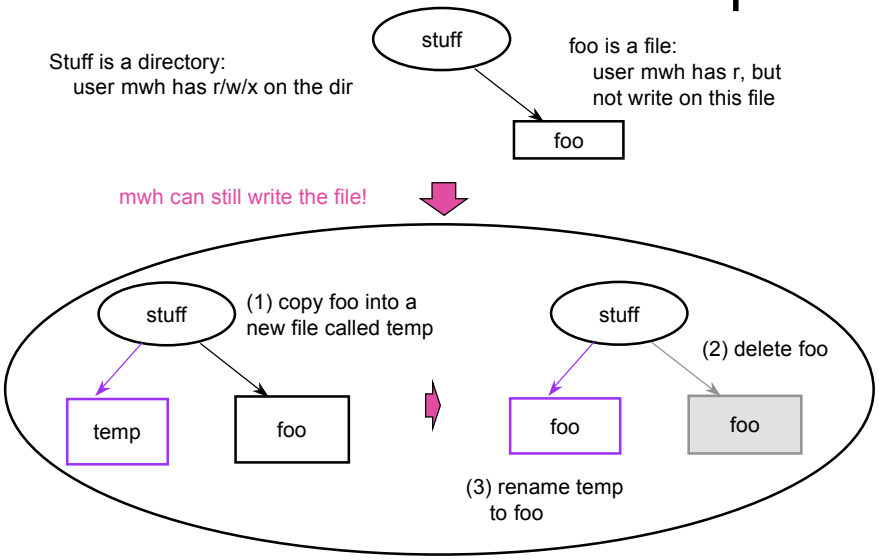
- Each file has three classifications
  - user: the user who owns the file
  - group: a named group of other users
  - world: all others
- Each file has three access types:
  - read, write, execute
- Three additional bits
  - Sticky bit
    - leave executable in memory after is done
  - Setuid, Setgid
    - run the program with the uid/gid of the file's owner
    - used to provide extra privilege to some processes
      - example: passwd command

UNIX Directory Protection

- Permissions interpreted differently
  - read: list the files
  - execute: see the attributes of the files
  - write: delete or create a file in the directory
  - sticky bit: can only modify directory entries owned by yourself
  - setgid: new files will have this group id
  - setuid: new files will have this user id

14

UNIX File Protection Example



AFS File Protection

- Each Directory has an ACL (but no file ACL)
  - applies to all files in a directory, and subdirectories (but can override)
  - 7 file access types:
    - read, write, lookup, delete, insert, lock (k), administer
  - ACLs apply to a user or a group
  - ACL may contain negative rights
    - "everyone but Joe Smith may read this file"
- Groups
  - each user can create a fixed number of groups
    - users can administrate their own groups
- Cells: collections of computers
  - csic, wam, … creates networked namespace

15

Remote File Systems

- Uses networking to allow file system access between systems
  - Manually via programs like FTP
  - Automatically 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 file sharing protocol
  - **CIFS** is standard Windows protocol
  - Standard operating system file calls are translated into remote calls

Effect of Updates to Shared Files

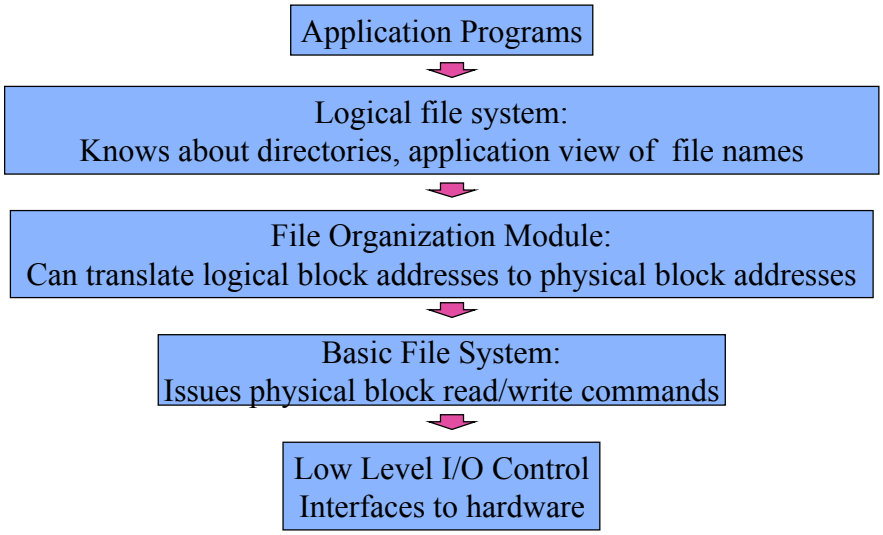
- UNIX
  - writes are visible immediately
  - have a *mode* to permit processes to share file pointers
- AFS
  - open/close semantics
    - "copy" the file on open
    - write-back on close
- Immutable files
  - once "published", the file never changes
    - usually done by attaching a version # to the filename
  - new versions of the file are given a new name

16

Raw Disks vs. Filesystems

- Can implement file systems with *raw disks*, which can be viewed as:
  - a linear array of fixed sized units of allocation, called blocks
    - assume that blocks are error free (for now)
    - typical block size is 512 to 4096 bytes
  - can update a block in place, but must write the entire block
  - can access any block in any desired order
    - blocks must be read as a unit
    - for performance reasons may care about "near" vs. "far" blocks (but that is covered in a future lecture)

File System Implementation



17