

CMSC 412

Spring 2007

Filesystems: Interfaces

Announcements

- Reading
 - Today : Chapter 10
 - Next time : Chapter 11

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?
- Basic operations
 - Create, Read, Write, Seek, Delete, Truncate
- Other operations
 - Open, Close, Sync, Read/write metadata

Create

- Construct a new file
 - assign it a name
 - check permissions (of directory)

Write

- Indicate what file to write
- Provide data to write
- Specify where to write the data within the file
 - generally this is implicit (file pointer)
 - Advances file pointer afterward
 - Supports **sequential access**
 - else explicit **direct access**
 - Specify **relative block number**
 - Or use some kind of index file

Read

- Indicate what file to read
- Provide place to put information read
- Indicate how much to read
- Specify where to write the data within the file
 - Implicit or explicit, as with writing

Delete, Truncate, Seek

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

Read-MD, Write-MD, Fsync

- read meta data
 - get file size, time, owner, etc.
- write meta data
 - change file size, time, owner, etc.
- fsync:
 - synchronize disk version with in-core version to ensure any previous writes to the file are stored on disk

Open, Close

Refer to file in use by a handle, rather than its name. Open the file to acquire handle

- open
 - check permissions, that the file exists
 - lock the file (if we don't want to permit other users at the same time)
 - Set up *file pointer* for access
- close
 - unlock the file (if locked when opened)
 - update meta data about time
 - free system resources (file descriptors, buffers)

File locking

- A **shared** lock is like a reader lock
 - Several processes can obtain it concurrently
- An **exclusive** lock like a writer lock
 - Only one process can hold it at a time
- May be *mandatory* or *advisory*
 - I.e., strictly enforced by the OS, or the application can override

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/XP
 - 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

Filesystems

- Provides a *namespace* for files via directories
 - Each file system part of a disk *partition*
- Can store files of variable size
- Provides protection by restricting access to files based on permissions

Directory

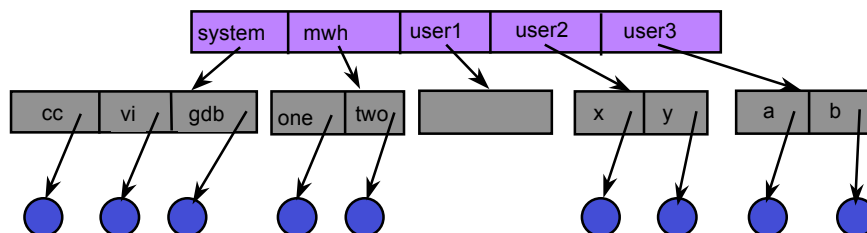
- Collection of files
- Operations
 - Search for a file
 - Create a file
 - Delete a file
 - List a directory
 - Rename a file
 - Traverse the file system

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



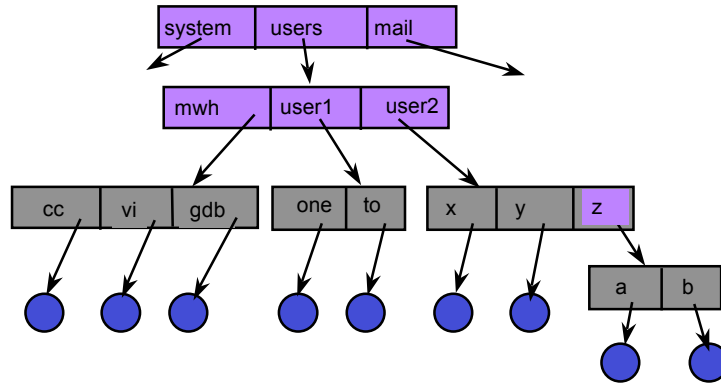
Search Path

- In a single level directory, the program to run can just be the name of that program
- With more than one directory, we must differentiate
 - But it's a pain to always prepend the program name with its directory name
 - Search path solves this problem

Tree Directories

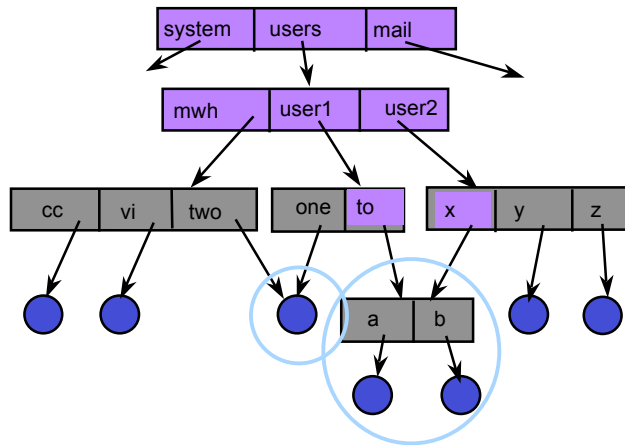
- Create a tree of files
 - Each directory can contain files or directory entries
 - Thus, each non-leaf in the tree is a directory
- Each process has a **current directory**
 - can name files *relative* to that directory
 - can change current directory as needed

Tree Directories



Acyclic Graph Directories

- Users can share directories and files



Acyclic 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: one user deletes a shared file
 - Then deleting for all users implies dangling links
 - Should stay until the last referent deletes it
 - Maintains a reference count to determine this
- Programs may need to be aware of sharing
 - disk usage utilities, backup utilities
- What if the graph was not acyclic?
 - Need garbage collection for dead files

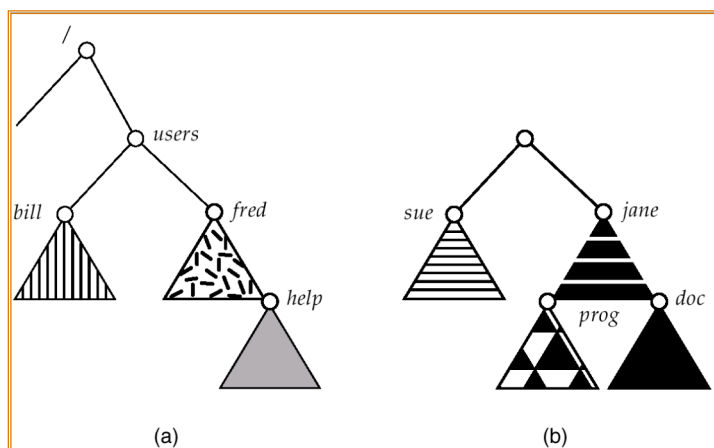
Symbolic vs. Hard links

- Hard link
 - Metadata in the file system “points” to the file or directory entry directly
 - Dangling links not permitted
- Symbolic link
 - Use a level of indirection: encoded as a path in the file system
 - Dangling links permitted: if the file is removed, then opening the link will fail
 - Makes writing directory traversal programs easier

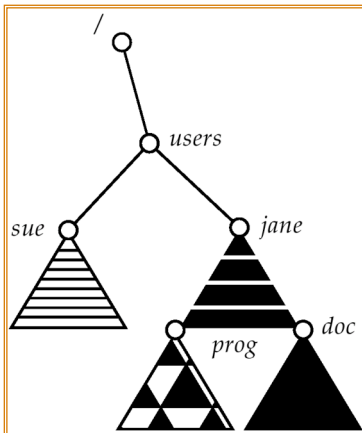
File System Mounting

- 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



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

UNIX File Protection Example

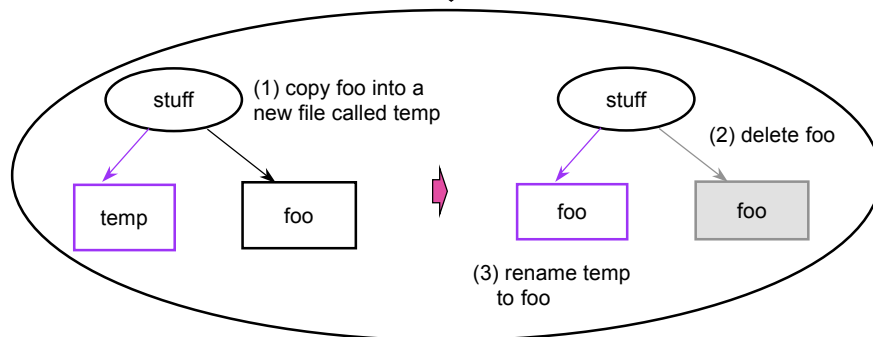
Stuff is a directory:
user mwh has r/w/x on the dir

stuff

foo is a file:
user mwh has r, but
not write on this file

foo

mwh can still write the file!



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 manage their own groups
- Cells: collections of computers
 - csic, wam, ... creates networked namespace

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
 - session semantics
 - “copy” the file on open
 - write-back on close
- Immutable files
 - once “published”, the file never changes
 - “updates” by using a version # on the filename
 - new versions of the file are given a new name

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)