UNIVERSITY of WISCONSIN-MADISON
Computer Sciences Department

CS 537
Introduction to Operating Systems

Andrea C. Arpaci-Dusseau
Remzi H. Arpaci-Dusseau

File System: User Perspective

Questions answered in this lecture:

What are files? What is file meta-data?

How are directories organized?

What operations can be performed on files?

How are files protected?

Motivation: I/O is Important

Applications have two essential components:

- Processing
- Input/Output (I/O)
  - What applications have no input? no output?

I/O performance predicts application performance

- Amdahl's Law: If continually improve only part of application (e.g., processing), then achieve diminishing returns in speedup
- `f`: portion of application that is improved (e.g., processing)
- $\texttt{speedup}_{\texttt{f}}$: speedup of portion of application
- $\texttt{Speedup}_{\texttt{Application}} = 1/ ((1-f) + (f/\texttt{speedup}_{\texttt{f}}))$
  - Example:
    - f = 1/2, $\text{speedup}_{\text{f}}$ = 2, $\text{speedup}_{\text{app}}$ = 1.33
    - f = 1/3, $\text{speedup}_{\text{f}}$ = 2, $\text{speedup}_{\text{app}}$ = 1.20

Role of OS for I/O

Standard library

- Provide abstractions, consistent interface
- Simplify access to hardware devices

Resource coordination

- Provide protection across users/processes
- Provide fair and efficient performance
  - Requires understanding of underlying device characteristics

User processes do not have direct access to devices

- Could crash entire system
- Could read/write data without appropriate permissions
- Could hog device unfairly

OS exports higher-level functions

- File system: Provides file and directory abstractions
- File system operations: mkdir, create, read, write

Abstraction: File

User view

- Named collection of bytes
  - Untyped or typed
  - Examples: text, source, object, executables, application-specific
- Permanently and conveniently available

Operating system view

- Map bytes as collection of blocks on physical non-volatile storage device
  - Magnetic disks, tapes, NVRAM, battery-backed RAM
  - Persistent across reboots and power failures

File Meta-Data

Meta-data: Additional system information associated with each file

- Name of file
- Type of file
- Pointer to data blocks on disk
- File size
- Times: Creation, access, modification
- Owner and group id
- Protection bits (read or write)
- Special file? (directory? symbolic link?)

Meta-data is stored on disk

- Conceptually: meta-data can be stored as array on disk

Abstraction: Directories

Organization technique: Map file name to blocks of file data on disk

- Actually, map file name to file meta-data (which enables one to find data on disk)

Simplest approach: Single-level directory

- Each file has unique name
- Special part of disk holds directory listing
  - Contains <file name, meta-data index> pairs
  - How should this data structure be organized???

Two-level directory

- Directory for each user
- Specify file with user name and file name

Directories: Tree-Structured

Directory listing contains <name, index>, but name can be directory

- Directory is stored and treated like a file
- Special bit set in meta-data for directories
  - User programs can read directories
  - Only system programs can write directories
- Specify full pathname by separating directories and files with special characters (e.g., \ or /)

Special directories

- Root: Fixed index for meta-data (e.g., 2)
- This directory: .
- Parent directory: ..

Example: `mkdir /a/b/c`

- Read meta-data 2, look for "a": find <"a", 5>
- Read 5, look for "b": find <"b", 9>
- Read 9, verify no "c" exists; allocate c and add "c" to directory

Acyclic-Graph Directories

More general than tree structure

- Add connections across the tree (no cycles)
- Create links from one file (or directory) to another

Hard link: "`ln a b`" ("a" must exist already)

- Idea: Can use name "a" or "b" to get to same file data
- Implementation: Multiple directory entries point to same meta-data
- What happens when you remove a? Does b still exist?
  - How is this feature implemented???
- Unix: Does not create hard links to directories. Why?

Acyclic-Graph Directories

Symbolic (soft) link: "`ln -s a b`"

- Can use name "a" or "b" to get to same file data, if "a" exists
- When reference "b", lookup soft link pathname
- b: Special file (designated by bit in meta-data)
  - Contents of b contain name of "a"
  - Optimization: In directory entry for "b", put soft link filename "a"

File Operations

Create file with given pathname /a/b/file
- Traverse pathname, allocate meta-data and directory entry

Read from (or write to) offset in file
- Find (or allocate) blocks of file on disk; update meta-data

Delete
- Remove directory entry, free disk space allocated to file

Truncate file (set size to 0, keep other attributes)
- Free disk space allocated to file

Rename file
- Change directory entry

Copy file
- Allocate new directory entry, find space on disk and copy

Change access permissions
- Change permissions in meta-data

Opening Files

Expensive to access files with full pathnames
- On every read/write operation:
  - Traverse directory structure
  - Check access permissions

Open() file before first access
- User specifies mode: read and/or write
- Search directories for filename and check permissions
- Copy relevant meta-data to open file table in memory
- Return index in open file table to process (file descriptor)
- Process uses file descriptor to read/write to file

Per-process open file table
- Current position in file (offset for reads and writes)
- Open mode

Enables redirection from `stdout` to particular file