Storage Systems

Main Points

- File systems
  - Useful abstractions on top of physical devices
- Storage hardware characteristics
  - Disks and flash memory
- File system usage patterns

File Systems

- Abstraction on top of persistent storage
  - Magnetic disk
  - Flash memory (e.g., USB thumb drive)
- Devices provide
  - Storage that (usually) survives across machine crashes
  - Block level (random) access
  - Large capacity at low cost
  - Relatively slow performance
    - Magnetic disk read takes 10-20M processor instructions

File System as Illusionist: Hide Limitations of Physical Storage

- Persistence of data stored in file system:
  - Even if crash happens during an update
  - Even if disk block becomes corrupted
  - Even if flash memory wears out
- Naming:
  - Named data instead of disk block numbers
  - Directories instead of flat storage
  - Byte addressable data even though devices are block-oriented
- Performance:
  - Cached data
  - Data placement and data structure organization
- Controlled access to shared data

File System Abstraction

- File system
  - Persistent, named data
  - Hierarchical organization (directories, subdirectories)
  - Access control on data
- File: named collection of data
  - Linear sequence of bytes (or a set of sequences)
  - Read/write or memory mapped
- Crash and storage error tolerance
  - Operating system crashes (and disk errors) leave file system in a valid state
- Performance
  - Achieve close to the hardware limit in the average case

Storage Devices

- Magnetic disks
  - Storage that rarely becomes corrupted
  - Large capacity at low cost
  - Block level random access
  - Slow performance for random access
  - Better performance for streaming access
- Flash memory
  - Storage that rarely becomes corrupted
  - Capacity at intermediate cost (50x disk)
  - Block level random access
  - Good performance for reads; worse for random writes

File System Design

- For small files:
  - Small blocks for storage efficiency
  - Concurrent ops more efficient than sequential
  - Files used together should be stored together
- For large files:
  - Storage efficient (large blocks)
  - Contiguous allocation for sequential access
  - Efficient lookup for random access
- May not know at file creation
  - Whether file will become small or large
  - Whether file is persistent or temporary
  - Whether file will be used sequentially or randomly

File System Abstraction

- Directory
  - Group of named files or subdirectories
  - Mapping from file name to file metadata location
- Path
  - String that uniquely identifies file or directory
  - Ex: /cse/www/education/courses/cse451/12au
- Links
  - Hard link: link from name to metadata location
  - Soft link: link from name to alternate name
- Mount
  - Mapping from name in one file system to root of another

UNIX File System API

- create, link, unlink, createdir, rmdir
  - Create file, link to file, remove link
  - Create directory, remove directory
- open, close, read, write, seek
  - Open/close a file for reading/writing
  - Seek resets current position
- fsync
  - File modifications can be cached
  - fsync forces modifications to disk (like a memory barrier)

File System Interface

- UNIX file open is a Swiss Army knife:
  - Open the file, return file descriptor
  - Options:
    - if file doesn’t exist, return an error
    - If file doesn’t exist, create file and open it
    - If file does exist, return an error
    - If file does exist, open file
    - If file exists but isn’t empty, nix it then open
    - If file exists but isn’t empty, return an error
    - …

Interface Design Question

- Why not separate syscalls for open/create/exists?
  - Would be more modular!

```
if (!exists(name))
    create(name);   // can create fail?
fd = open(name);   // does the file exist?
```

File system coding exercise

- Chapter 11, question 3
  - Create a new file: open() / creat()
  - Write 100KB of data to it: write()
  - Flush the writes: fflush()
  - Delete the file: unlink()
  - Time these operations: gettimeofday()