CS503A — Lecture # 12 — 2019-05-08

Administrivia

- midterm solution
- lab 2 discussion (?)
- lab 3 – delayed → { will publish as soon as practical / extension

- concurrency:
  - if you are reading a shared variable, you must protect the read unless you can prove that no other {thread / process} will write to that variable
    - read may not be atomic
  - if you are writing a shared variable, you must protect the write unless you can prove that no other {thread / process} will write or read that variable
    - write & read may not be atomic

- race conditions are hard to test for because they usually happen rarely (usually, long after the system has gone into production)
  - need analytic tools
  - careful inspection
  - another pair of eyes

{ can't prove absence of race / deadlock because you need to show true at all possible execution paths → can show presence (brought it into observance; can't prove a negative) / can prove: "no deadlock here"

- message passing: easier to reason about
  - study Erlang

Previously on CSS 503A...

- userland vs kernel
  - system calls from userland processes to request kernel services

- concurrency: processes & threads

- filesystems:
  - abstraction/data structure on top of mass storage devices

- filesystem API: uniform interface
  - regular files
  - devices

- filesystem layers
  - application
    open/close/read/write/lseek
  - logical filesystem
    filename → file#/file handle/protection
  - file organization
    knows (file control block)
    NTFS: master file table entry
  - basic filesystem
    block # → drive/cylinder/track
    buffer/cache
  - I/O control
    device access
  - device
    - electromagnetic signals

File System Implementation

- disk: data structure
  - boot sector
  - partition table
  - superblock
    - magic #
    - mount count / max mount count
      - run disk check utility
    - block size / blocks/ group
    - # free blocks
    - # free inodes
    - 1st inode (/ or mount point)
      - ↳ root file system

* loopback device
  - kernel driver:
    - lets you mount a file as if it's a separate filesystem

file control block - aka inode

- multiple filesystem types
  - ISO 9660
  - Ext 2 - 3 - 4
  - NTFS
  - FAT 16 / FAT 32
  - FUSE
    - userland filesystem
      " Filesystem in User Space
    - Virtual Filesystem

- directory → "special file" : map name → inode #

- inode : data structure
  - file metadata
    - size
    - time stamps
    - owner/group
    - permission bits
    - # links
    - ptr to data blocks

- name is not an intrinsic property of a file
  - hard link : 2 file names → single inode
  - Symbolic (soft) link : directory entry that points to inode that points to directory entry
    - cannot do hard links to directories
      - early unix : directory is file
      - now : not so much
    - need inode # + device # to uniquely identify file

Disk Structure

Superblock
- where are the inodes?
- free/allocated block list

inode: file metadata

directory
- "special" file
  - set set in inode
  - used to be able to open/read/close
    → now requires separate setuid system calls

Disk Allocation & Indexing

- Memory allocation problem
  - like malloc / new
- ⊕ Latency
  - disk rotation
  - read/write head seek

- Clever placement can make a world of difference

- 2 basic strategies
  - Contiguous (Array)
  - Linked list (sequential)
  - or: some combination

Disk Allocation & Indexing (cont.)

- contiguous allocation
  - OS-360 / VM/CMS
    - Allocate "cylinders"
    - + minimize disk seeks
    - - requires pre-allocation (inflexible)
    - - fragmentation
      - "external" fragmentation

Disk Allocation & Indexing (cont.)

- linked list
  - + inode (FCB Control Block) only requires pointer to 1st block
    - Smaller inode
  - + Fragmentation is not an issue (file is pre-fragmented)
  - - only suitable for sequential access
  - - link pointers reduce available block data
  - - broken link → unaccessible data
    (e.g. write failure due to OS bug or H/W fault)
    → {really bad things when it happens}

- allocation by large blocks:
  best of both worlds performance
  - as disk fragments, performance degrades
  Windows: defragmentation utility