Nathan Tung

CS111 Sections of Chapter 2, 8 Outline #16

Chapter 2, section 2.5. (pages 91-112), Chapter 8, Sections 8.1.1 (pages 8-3 to 8-6), 8.2.1 (pages 8-8 to 8-13), and 8.5.4 (pages 8-40 to 8-51).

- Case Study: UNIX File System Layer/Naming
  - Files have user-chosen names
  - Users can group them by their names, creating a naming network, etc.
  - Block layer
    - Physical disk (magnetic disk, flash disk, etc.) that holds block units
    - Name mapping algorithm allows us to access contents of the block by block number
    - Name discovery: block layer keeps track of which blocks are in use or available for assignment
    - Super block holds info like the system's block size of the disks
      - List of free blocks is implemented differently across systems
      - Sometimes linked list of unused blocks
  - File Layer
    - Meant to hold files that are a linear array of bytes of arbitrary length
    - The first seven "indirect blocks" that contain no data, but block numbers
  - Inode Number Layer
    - Names inodes by inode number
    - Name discovery: inode numbers (integers) are tracked here like block numbers
  - File Name Layer
    - Proposed name is bound to inode
  - Path Name Layer
    - Assign structured names to files
  - Links
    - Create synonyms between files via LINK and UNLINK
    - Cycles are not allowed in naming graph
  - Renaming
    - UNLINK(to), LINK(from,to), UNLINK(from)
    - Shortcomings: it's not atomic
  - The Absolute Path Name Layer
    - Establish a well-known name
    - Root directory contains bindings for users, file names, directories, etc.
  - Symbolic Link Layer
    - MOUNT keeps inode in memory, UNMOUNT undoes this
    - Symbolic links are indirect names called soft links
    - Solutions to binding between inodes and directory entries across disks
      - Make inodes unique across all disks
      - Create synonyms for files
    - Direct bindings are hard links
  - Implementing File System API

- Faults, Failures, and Fault Tolerant Design (Faults, Failures, Modules)
  - Fault is an imperfection; failure is a catastrophic result of that fault
    - Example: tire blows out (failure) due to weak casing (fault)
  - Examples in different sources
    - Software: programming fault in using wrong equality sign causes failure in system crash
    - Hardware: gate is faultily stuck at 0, may produce a failure in a summation that needs a 1
    - Design: miscalculation fault in installing not enough memory; failure to accept calls later on
    - Implementation: fault in installing less memory than designed; failure same as above
    - Operations: operator faultily ran the check program twice, creating wrong tax calculations
    - Environment: fault lies in lightning or bacteria causing destructive/contaminative failure
  - Latent (fault has no effects at the moment) vs. active (errors appear in data values or control signals)
  - Errors can be masked, but the module itself has probably failed

        - Subsystem failure is a fault from the larger subsystem's point of view
    - Fault tolerance: notice and help in active faults and component subsystem failures
        - Error containment: it's hard to confine error effects to just a subsystem portion
            - Mask the error
            - Detect and report error at interface (fail-fast design)
                - The module has found the error, so it actually meets specifications
            - Immediately stop dead (fail-stop)
                - Larger system has to detect this error (like with a timer)
            - Do nothing or fail without warning (crash, fail-thud)
    - Transient fault, or single-event upset, is temporarily triggered by passing external event like thunder
        - Fix: try the operation again
            - If it works, we had a soft error
            - Persistent faults cause a hard error
    - Intermittent fault is a persistent fault that is only sometimes active like increased noise levels
    - Latency is also important; too much latency may allow a second error to come up and mask the first one
- Measures of Reliability and Failure Tolerance: Availability and Mean Time to Failure
    - Model of system says it operates correctly for some time, then fails (TTF: time to failure)
    - Availability = (time system runs)/(time system should run)
        - Downtime = 1 – Availability
    - Ergodic: processes with ensemble average the same as time average
    - MTTF (mean time to failure), MTTR (mean time to repair), MTBF (mean time between failures)
    - Backward looking: evaluate how system is doing; predict how system performs in the future
    - Components like magnetic disks have a "bathtub curve"
        - Components usually fail early due to defects or fail late due to wear-and-tear
        - Infant mortality vs. burn out
        - Products get "burned in" to weed out early defects; these probably form expected lifetime
            - Surviving products form the "yield"
    - Design for iteration: keep doing what produces good results, reducing the infant mortality rate
    - Preventive maintenance is a means to increasing mean time to failure and availability
    - We measure very low MTTF by "accelerated aging" by finding failure cause, testing, and extrapolating
    - Nines are used to capture availability, but says nothing about MTTF
    - Fail soft: systems exhibit this when they provide partial service in the face of failure
- Magnetic Disk Fault Tolerance
    - Durable storage: best to use non-volatile (magnetic disk: low cost, large capacity, non-volatile)
    - Magnetic disks have three layers
        - Raw storage (spinning disk)
        - Fail-fast storage (hardware-firmware for detecting failures in raw storage)
        - Careful storage (uses earlier error detecting to create more reliable system)
        - Durable storage (only in highly-available systems; develops disk failure model/error masking)
    - Durable Storage: RAID
        - Errors can still happen if a sector decays along with all its copies before being noticed
        - Solution 1: implement a clerk to check all replicas for decay periodically
        - Solution 2: not as good, but replace disks when they reach the end of the bathtub curve
    - Detecting errors by system crashes
        - CAREFUL_PUT can create a hard error by corrupting data before seen by disk subsystem
            - Use an end-to-end checksum to monitor integrity as it goes through OS buffer
            - End-to-end argument: the checksum is always needed, but checksum inside the disk system may not be essential
            - However, checksum actually can correct burst errors and handle erasure code like RAID
    - Checksum only detects the error; masking via recovery is another technique