CS 140 Final Review Session

Administrative Details

- Final exam: 12:15-3:15pm, Thursday March 18, Skilling Aud (here)
- Questions about course material or the exam?
  - Post to the newsgroup with “Exam Question” in subject line

Plan For Today

- Lightning-quick review of pre-midterm course topics
- Slightly more in depth review of post-midterm topics
- Time for questions at the end

Pre-Midterm

- Concurrency + Synchronization primitives
  - What hardware support do you need?
- Scheduling
  - Fairness, response time, throughput, etc.
  - Many approaches
    - FCFS, Priority, BSD sched., balanced virtual time, etc. etc. etc

Pre-Midterm, cont.

- Linking/loading
  - Static vs. dynamic
- Virtual memory
  - HW support: page table management, TLB?
  - Handling page faults, eviction strategies
  - Lots of cool tricks you can play
    - Use mprotect to trigger faults on write to page, etc.

Disks + I/O

- How slow are they?
  - Many orders of magnitude slower than CPU registers, cache, RAM
- How do you know when I/O completes?
  - Polling vs. Interrupts
    - Polling ties up CPU
    - Context switches to handle interrupts very expensive
    - Depends on expected wait time

Optimizing Disk Accesses

- Seek time + rotational delay >> time to read a block once head is positioned
  - Sequential accesses amortize cost
- Stay in same cylinder group (no head seek necessary)
- Organize accesses intelligently
  - E.g. elevator algorithm
- Don't tie up CPU during I/O
  - Special hardware (DMA) can handle data transfer

File Systems

- Goals
  - Space utilization, optimal disk access patterns, robustness (crash recovery)
  - Flexibility: huge files, huge numbers of files
  - Organize files in coherent fashion
- Evaluating performance
  - Sequential access? Random access? Metadata overhead?

Simple FSs

- Extent-based
  - Pros: simple, files are sequential
  - Cons: fragmentation
- Linked
  - Pros: disk usage, low overhead
  - Cons: really bad random access, and sequential access can be bad
- Indexed (multilevel?)
  - Pros: disk usage, better random access
  - Cons: fixed max file size/disk size?

A better FS: FFS

- Uses bitmap for fast allocation
- Space allocated in large blocks for improved locality
  - Can be split into fragments
- Prefer allocations in same cylinder group for related data
- Downsides?
  - Needs synchronous metadata writes (more on this later)

Crash Recovery

- Run program (fsck) to repair file system after crash and ensure consistency
- Goal: design FS so that fsck can always recover system, and can do so efficiently

Crash Recovery, cont.

- Problem: crash during update to metadata can leave filesystem in unrecoverable state (if we're not careful)
- Ordered updates
  - Make sure writes go out to disk in particular order
  - Needs to be synchronous (slow)

Crash Recovery, cont.

- A better way: soft updates
  - Store information about dependencies
  - Can solve cyclic dependency problems
    - May require rollbacks
- Another way: journaling
  - Write operations out to log before actually doing them
  - Crash recovery: replay log
  - Periodically do checkpoints

A better on-disk data structure

- So far we've seen:
  - Linked list, index (single, multilevel)
- Now… B+ trees!
  - Generalization of binary trees
  - Insert, delete operations can be very complicated
    - In fact many implementations cut corners here
  - Goals: don't chase many pointers, get lots of useful data with each disk block read

Networking

- Layered model
  - Link (Network (Transport (Application)))
  - Link layer: MAC
    - Ethernet, or wireless
  - Network layer: IP
  - Transport: TCP or UDP
    - Differences?
- Sockets: abstractions for communication

Networked Filesystems

- Saw a few different approaches
  - Mostly talked about NFS and AFS
- Stateless vs. Stateful
- Caching strategies
  - Polling, callbacks, leases
- When are writes visible to other clients?

NFS

- Goals: Unix semantics, crash recovery, performance
- Uses RPCs (remote procedure calls) to manipulate 'vnode' objects on server
- Caching
  - How do we know if a cached file is stale?
  - Mtime and ctime attributes, before and after (in v3)
    - Client may not have to flush cache
- In v3, fine grain controls like expected write stability (do we want to wait until data is guaranteed not to disappear?)

AFS: a different approach

- Any AFS volume available under /afs/ directory
- All writes cached locally until file closed
  - Can cause some issues
  - Originally cached entire file (expensive to open large file)
- Actual location of file should be transparent to client
  - Volume migration: move to different server with minimal disruption

Protection

- Think of a matrix of users and objects
  - Cell(i,j) specifies rights of user i to access object j
- Unix slices on columns
  - Permissions stored with object
- Root user is the kitchen sink in Unix
  - Can do anything

Root, setuid, TOCTOU bugs

- Why do we need setuid programs?
  - Login as different user, change password, etc.
- Setuid programs very tricky to get right!
  - Time of check to time of use bugs (TOCTOU)
    - Malicious user tricks process with elevated privilege by racing to change file to symlink (as an example)
  - Ptrace bugs
    - Not allowed to ptrace a setuid program, but what if they drop privileges? Memory could reveal secret key, e.g.
  - Confused deputy
- Study examples carefully from lecture and make sure you understand the problems

Capabilities

- Slice access control matrix along rows
  - "Keys" for access maintained with user
- Unfortunately, little momentum to switch to radically different model

Security

- Two concepts
  - Secrecy: Untrusted users cannot read secret data
    - A process should not simultaneously read classified file and write to unclassified file
  - Integrity: Low-integrity users cannot pollute high-integrity data
- Modeled as a lattice
  - Consult lecture notes/Wikipedia for more details on lattices
- Flask architecture: separate security policy from implementation
  - Specify policy, then system enforces it

Virtualization

- Abstract the hardware
- Tons of applications
  - Fault isolation
  - Legacy code support
  - CPU utilization/sharing
  - Debugging
  - Security?? (maybe not)

Implementing Memory Accesses

- Add another level of indirection
  - Physical page -> Machine Page
  - Terminology can get really confusing here
    - Virtual address/page: virtual memory managed by guest OS
    - Physical address: address that guest OS uses to access memory
    - Machine address/page: address of memory on the host machine
  - This mapping is the "shadow page table"
  - Tricky problems: want to avoid as many faults as possible, but need to keep page tables in sync

Binary Translation

- Interpreting and emulating each instruction is too slow
- Some code can actually just be run on the host CPU
  - Most user code
  - Kernel code is trickier
    - Can use privileged operations
    - Once translated, might be a different size
      - Branches and jumps might not be safe

Memory Management

- Double LRU paging: bad! (Why?)
  - Random works well in VMM
- Operating Systems are greedy
  - Will use all the memory available
- Different VMs can share memory (what if somebody writes to it?)
- How can the VMM reclaim memory?
  - Balloon driver in guest hogs memory
  - VMM can now use this space
- Which VM to reclaim from?
  - Idle memory tax

Questions?