CS 162 Operating Systems and Systems Programming

Professor: Anthony D. Joseph

Spring 2003

Lecture 25: Course Review

25.0 Course Goals

1. Provide you with the knowledge you need to make informed decisions.
   - Is it better to buy a computer with more memory or a faster processor (or faster memory versus faster processor)?
   - Why is my company's Web server slow? Is it the network, the server, the application?
2. Experience with different design tradeoffs, choices, and decisions.
   - What is the cost of using a software modem instead of a software modem? Everything that's done in hardware can be done in software, but when does it make sense?
   - How do I enable my company's users to share information with collaborators at other companies? With good performance. Without compromising security.
3. Design abstractions: separating policy from mechanism
   - What abstractions should the operating system provide?
   - How should I implement privacy controls?

25.1 OS as Illusionist

We used the Operating System as starting point for understanding/analyzing the issues.

Operating systems have two functions:

1. Coordinator and traffic cop
2. Standard services

Physical Reality	Abstraction
Single CPU	Infinite # of CPUs (multiprogramming)
Interrupts	Cooperating sequential threads
Limited memory	Unlimited virtual memory
No protection	Each address space has its own machine
Unreliable, fixed size messages	Reliable, arbitrary messages and network services

25.2 Concepts

We've abstracted out three key concepts. They apply to more than just operating systems

1. Locality/Caching – basis for TLB's, paging, file systems, distributed systems, etc.
   - Spatial versus temporal locality
   - Thrashing
   - Multi-level hierarchies
   - Same issue in HW and SW
2. Scheduling – adaptive management of resources
   - Constrained resources require careful management
   - Multi-level adaptive feedback
   - Countermeasures for misbehaving users and applications
3. Layering – Abstraction on top of abstraction
   - Use divide and conquer to simplify a hard problem.
   - Makes it easier to design, debug, extend
   - Performance penalty

25.3 Major topics

1. Threads: state, creation, dispatching
   - Why – Abstraction for *concurrency*: overlap I/O and computation, share HW resources (and information) across multiple users and programs. Modularity makes system easier to extend.
   - How – Context switching, and thread dispatching (mechanism).and scheduling. Decompose task into smaller units/functions.
   - But – performance overhead for context switching.
2. Synchronization: races, inconsistency, semaphores, monitors, and condition variables.
   - The cost of concurrency! Without sharing concurrency is useless, but remember the "Too Much Milk Lecture"
   - Non-reproducibility – Hard to debug!
   - Use atomic operations as a start, but complicated to use and OS interactions (load/store, interrupt disable, test&set).
   - Create higher level abstractions to ease the burden:
     - Critical sections and mutual exclusion – policy.
     - Locks and semaphores – mechanism.
     - Monitors: separate mutex (locks) and scheduling constraints (condition variables) – mechanisms.
   - Language-level interactions with primitives. Be careful!
   - Biggest caveats: Deadlock and starvation
     - Starvation: Indefinite waiting for a resource by a thread (can end, but doesn't have to).
     - Deadlock: Circular chain of waiting (doesn't end without external intervention). Requires: limited resource, no resource preemption, multiple independent requests, circular chain of requests. Break the chain – detect/fix or prevent
3. Scheduling: shortest (remaining) time to completion first, round robin, FIFO
   - Policy: minimize response time, maximize throughput, fair.
   - Lots of choices: algorithm, time slice, dynamic adaptation (multi-level feedback), etc. – most choices don't really matter unless resources are constrained.
4. Memory management & address spaces:
   - Isolate processes/programs from all others and OS– protection.

     - Dual mode operation: kernel versus user mode – operations themselves must be protected: How do you enter/leave kernel mode?
     - This can be done without hardware support:
       - Strong typing
       - Software fault isolation
     - But inter-process communication breaks this (bugs can leak).
   - Illusion of infinite memory:
     - Build a hierarchy out of fast, small -> large, small technologies
   - Transparent (can't tell if physical memory is shared)
     - Address translation
     - Base & bounds, paging, segmentation, multi-level translation, TLB's for caching/performance (replacement policy and write-back/write-through are considerations – thrashing).
     - Complexity versus functionality tradeoffs
5. Virtual memory: demand paging, thrashing
   - Exploit spatial and temporal locality
   - Caching misses: compulsory, capacity, conflict, policy
   - Lots of page replacement policies: Again, most important when resources are limited! Approximations work well.
   - Application working set size is important
6. File systems:
   - I/O system performance: overhead, latency, bandwidth
     - Disk seeks, rotational delay, sector sizes
     - Scheduling is important: FIFO, elevator (SCAN)
   - File headers and directories: abstraction of bytes, named files, protection, durability
     - Management policies based upon file usage patterns
     - Caching for performance
     - Protection and access control are important
     - Transactions: Implement atomic, persistent operations (durability) for unreliable components.
       - Two-phase locking for coordinating multiple threads

7. Distributed computing
   - Cheaper, more reliable, incremental scalability

- In reality, not more reliable
- Coordination is more difficult than in single machine case.

8. Networks: protocol layers, windowing, RPC
   - Build protocols layer-by-layer
   - Lots of different network technologies
   - Goals: arbitrary message size, ordered, reliable, process-to-process, routed anywhere, secure
   - Goals are hard (lots can go wrong)
   - Remote Procedure Call is key abstraction for 2-way communication:
     - Cross-domain communication
     - Location-transparency
     - Microkernel is ultimate in RPC usage
9. Network file systems: cache coherence
   - Transparent access to files on a remote disk: NFS, AFS
   - Caching, consistency, and false sharing issues
   - Multiprocessors: shared-bus, switched, Network of Workstations (Similar problems to filesystems)
10. Security: access control, encryption, Trojan horses
    - Why you should never trust a computer!
    - Intentional and accidental misuse
    - Three parts:
      - Authentication – who user is
        - Passwords, encryption (private and public key encryption)
      - Authorization – who is allowed to do what
        - Access control lists
      - Enforcement – make sure people do what they're supposed to do
        - Kernel does this for OS

25.4 Problem Areas

1. Performance
   Abstractions like threads, RPC aren't free
   Remember threads in OS/2
   Caching doesn't work when there's little locality

2. Failures – how do we build systems that continue to work even when parts of the system break?
   Still a problem today!
3. Security – basic tradeoff between making computer systems easy to use vs. hard to misuse