CS 140 – Summer 2008 – Handout #27: Course Review & Wrap-up

Operating Systems

- What does an OS do:
  - **Manage and protect hardware resources.**
  - **Export useful, *abstractions* to access resources:**
    - **E.g. Process, Signals, Virtual Memory, Files, Sockets**
- Typical "kernel" organization:
  - **OS kernel runs in most privileged processor mode.**
    - **Full access to all hardware resources**
  - **Runs everything else in "user" mode.**
    - **Restricted access to resource**

Processes and Threads

- Implementation:
  - **state, creation, dispatching, context switch.**
- Synchronization:
  - **Race conditions, inconsistencies.**
  - **Mutual exclusion, Critical sections.**
  - **Semaphores: P() and V().**
    - **Producer & Consumer problems.**
    - **Scheduling problems**
  - **Semaphore implementations.**
    - **Atomic operations: interrupt disable, test-and-set.**
  - **Monitors & Condition Variables**
  - **Deadlock.**

State of the Art: Processes

- Pretty much all OS environments have:
  - **Address space + one or more threads of control**
  - **Synchronization primitives – Semaphores**
  - **Language environments with monitors (Java, C#)**
- Modern architectures support for non-blocking ops
  - **Compare and swap**
  - **Research: Transactional memory**
- Problems still exist
  - **Deadlock – Careful programming,orderings.**
  - **Race conditions – Source of bugs; automatic checkers? (e.g. Eraser [Savage97])**

CPU Scheduling, Memory management

- Allocation -- Non-preemptible resources.
- Scheduling -- Preemptible resources.
  - **FIFO, Round-robin, STCF**
  - **Adaptive: Exponential Queue, Fair Share**
  - **Lottery**
- Linking, static relocation.
  - **Segments: Code, heap, stack.**
  - **Linker passes, relocation info, cross-references.**
- Dynamic memory management:
  - **First fit, Best fit.**
  - **Implementation: Bitmaps, Arenas/pools, Garbage collection.**

State of the Art: CPU/Memory

- CPU scheduling – still a research area
  - **Quality of Service**
  - **Real-time**
  - **Guarantees (fairness, latency, deadlines, etc.)**
- Growth of code has forced dynamic linking
  - **Static linking only at module level**
- Garbage collection is still controversial for OS work
  - **Norm: Implicit memory management (alloc/free)**
  - **Reference counts**

Virtual memory management

- Base and bounds.
- Paging.
- Segmentation.
- Page & segmentation.
- TLBs.
- Demand paging.
- Page replacement: LRU, clock.
- Thrashing, Working sets.

State of the art: Virtual Memory

- All OS virtual memory systems look the same:
  - **Software segments with demand paged memory**
- Paging algorithms less important:
  - **Just buy enough memory.**
  - **Programs with huge data handle own memory (e.g. DB)**
- All architecture support paged virtual memory
  - **OSes assume address space abstraction.**
  - **Ease fragmentation of main memory.**
- Research:
  - **Big pages for dealing with TLB overheads**

File systems, Disks

- Addressing:
  - **Sequential, random, keyed.**
- Storage management:
  - **Contiguous allocation.**
  - **Linked files.**
  - **Indexed files. (Multi-level Unix style index).**
  - **Crash Recovery (Logging, shadowing)**
- Block cache
- Freelist, bitmaps.
- Naming:
  - **Descriptor organization, directories.**
- Disk scheduling: FIFO, SSTF, Scan/Elevator.

State of the art: File Systems

- Nearly every OS supports a hierarchical FS
  - **Write-ahead logging for performance and crash recovery.**
- Trade off buffer cache and virtual memory
  - **Memory mapped files example.**
- Disk scheduling implemented but not used too much
  - **Modern disks use caching, hide disk geometry**
- Research: Content based retrieval (FS like DB)
  - **Fast search – FS needs to interpret file contents.**
- More FS research & development:
  - Distributed, replicated, reliable file systems; huge FS (e.g. ZFS)
  - File system/storage/disk virtualization
  - Virtual FS name spaces (e.g. FUSE, Portal, etc.)

Networks & communication

- Link-level
  - **Point-to-point, bus, encoding**
  - **Ethernet**
- Network Level.
  - **Datagrams**
  - **Virtual circuits**
  - **IP**
- End-to-End.
  - **TCP – Acks, sliding window**
  - **Distributed consensus**

State of the art: Networking

- Clearly the biggest impact on OS area
- Distributed system:
  - **New problems: Latency, bandwidth, failure, and trust.**
  - **Breaks everything.**
- Research: clean-slate networking
  - **If you were going to redesign the internet to be simpler, more flexible, and more secure, how would you do it?**
- Possibility: *centralized syste*m
  - **easier management/control**
  - **how to guarantee scalability and performance?**
- Research: mobility
  - **e.g. How do we implement cell phones over the internet?**

Protection & Security

- Authentication:
  - **Passwords, keys.**
- Authorization determination:
  - **Access matrix**
    - **Access lists. Capabilities.**
- Access enforcement:
  - **Security kernel.**
  - **DAC or MAC**
- Attack methods (myriad!):
  - **Abuse of privileges. Trojan Horse. Listener/spyware. Spoiler/DoS.**
  - **Remote attacks. Open doors/misconfiguration. Local privilege escalation. TOCTTOU. Ambient privilege (setuid, Confused Deputy problem, etc.)**
  - **Worms and viruses. Zombies/botnets. Rootkits/hiding.**

Security Defenses

- Logging
- Caller/user identification (avoid anonymity)
- Principle of minimum privilege
- Correctness proofs
- Information Flow Control & Taint Propagation
- Encryption (didn't really cover the following:)
  - **Private (single-) key: DES, crypt()**
  - **Public (two-) key systems: RSA**
  - **Digital signatures. (e.g. encrypt w/private key)**
  - **Message Digests/secure hashes (MD5, SHA-1)**

State of the art: Security and Protection

- Huge issue today:
  - **Authentication – passwords, too weak.**
  - **Authorization – ACL on files, some capabilities.**
  - **Access enforcement – Need no bugs!**
- Implications – Beware of complexity
  - **Most features != most secure.**
- Encryption being deployed
  - **Also digital signatures/integrity checks for system files**
- Research: "Secure operating systems"
  - **Implementing/deploying IFC in practical/usable systems**
  - **Fixing legacy systems (without breaking them)**
  - **Making it easier (or possible) to write secure software**

Major concepts

- Locality: spatial, temporal
- Scheduling - sharing resources
  - **Best algorithms know future, but we use past instead.**
- Layering - hiding complexity with abstraction
  - **Synchronization, network protocols, file systems, etc.**
- Hierarchy - achieving scalability without slowdown
  - File systems, networks/routing, DNS, storage
- Caching - using hierarchy to reduce latency
  - **MMU, memory, disk, file system, function results, shared data, etc.**

CS140: Congratulations!

- Congratulations on finishing the Pintos assignments!
  - **You are now an official OS expert/kernel hacker!**
- You should now be better able to:
  - Write applications that are more functional and secure, and make better use of OS capabilities.
  - Reduce/manage software complexity by using abstraction, hierarchy, layering, etc..
  - Reduce and hide latency with caching, multithreading
  - Design and debug multithreaded code (or avoid using it!)
  - Modify Linux/BSD/OSX, or your own custom OS
  - Read and understand OS papers and documentation.
  - Improve upon the design and implementation of computer software and systems in general.

CS 140: Next steps

- Take the final on Saturday, 12:15–3:15 p.m.
- Enjoy using, developing and modifying system software of all kinds
  - **Work on more OS and systems projects – at companies, in research labs, or on your own!**
- Take (and succeed in) more fun CS/systems courses!
  - Compilers (e.g. CS143/243), Networks (CS144/244A/344) Distributed Systems (CS244B), Security (CS155/244B), Comp. Arch. (EE 182/282!), **CS 240 (interesting OS papers!)**
- Read OS conference proceedings and attend conferences
  - **OSDI, SOSP, USENIX, ASPLOS, DefCon, etc.**