Announcements

- Project 0 is online
- This project is done individually
- Deadline is Feb 1, 11:59pm
- Curator instructions will be posted on website

What does an OS do?

- Software layer that sits between applications and hardware
- Performs services
  - Abstracts hardware
  - Provides protection
  - Manages resources

Core OS Functions

- Hardware abstraction through interfaces
- Protection:
  - Preemption
  - Interposition
  - Privilege (user/kemel mode)
- Resource Management
  - Virtualizing of resources
  - Scheduling of resources

Goals for Resource Management

- Fairness
  - Assign resources equitably
- Differential Responsiveness
  - Cater to individual applications’ needs
- Efficiency
  - Maximize throughput, minimize response time, support as many apps as you can
- These goals are often conflicting.
  - All about trade-offs

Evolution of OS (III)

- Recent (last 15 years or so) trends
- Multiprocessing
  - SMP: symmetric multiprocessors
    - OS now must manage multiple CPUs with equal access to shared memory
- Network Operating Systems
  - Most current OS are NOS.
  - Users are using systems that span multiple machines; OS must provide services necessary to achieve that
- Distributed Operating Systems
  - Multiple machines appear to user as single image.
  - Maybe future? Difficult to do.
OS and Performance

- Time spent inside OS code is wasted, from user’s point of view
  - In particular, applications don’t like it if OS does B in addition to A when they’re asking for A, only
  - Must minimize time spend in OS – how?
- Provide minimal abstractions
- Efficient data structures & algorithms
  - Example: $O(1)$ schedulers
- Exploit application behavior
  - Caching, Replacement, Prefetching

Common Performance Tricks

- Caching
  - Pareto-Principle: 80% of time spent in 20% of the code; 20% of memory accessed 80% of the time.
  - Keep close what you predict you’ll need
  - Requires replacement policy to get rid of stuff you don’t
- Use information from past to predict future
  - Decide what to evict from cache: monitor uses, use least-recently-used policies (or better)
- Prefetch: Think ahead/speculate:
  - Application asks for A now, will it ask for A+1 next?

Final thought: OS aren’t perfect

- Still way too easy to crash an OS
- Example 1: “fork bomb”
  - main() { for(;;) fork(); } stills brings down most Unixes
- Example 2: livelock
  - Can be result of denial-of-service attack
  - OS spends 100% of time servicing (bogus) network requests
- Example 3: buffer overflows
  - Either inside OS, or in critical system components – read most recent Microsoft bulletin.

Things to get out of this class

- (Hopefully) deep understanding of OS
- Understanding of how OS interacts with hardware
- Understanding of how OS kernel interacts with applications
- Kernel Programming Experience
  - Applies to Linux, Windows, Mac OS-X
  - Debugging skills
- Experience with concurrent programming
  - Useful in many other contexts (Java, C#, …)

Project 0

- Implement User-level Memory Allocator
  - Use address-ordered first-fit