Operating Systems: Hollistic view

- Operational States
  - Startup / Boot strap process
  - Manage resources in the steady state
    - Processes/threads/synchronization
    - Memory/VM/swap
    - File system/storage
- Functionality
  - Hardware support required to perform meaningful operations
    - TLB size, cache size, disk locations and maps etc.
    - If TLB is too small, then performance will suffer
  - Software required to use these abstractions
    - POSIX, Win32 etc.
    - If POSIX did not support threads, then we cannot use multiple CPUs for same process

Boot strap process

- Steps taken between powering on the machine and when the operating system has full control
- Firmware initializes/probes the hardware, locates bootable device and starts the master boot record
- Master boot record uses probed hardware info to locate bootable partitions and choose the boot sector on one of these partitions
- Boot program knows about file systems etc. and starts the OS kernel
  - Kernel uses hardware information to load drives
  - Kernel initializes data structures
    - Process control block with at least an idle task
    - File data structures etc..
- Question: How can you speed up this process?

Operating Systems

- Operating systems helps juggle resources and makes it appear to have more resources than we actually have
  - Use idle CPU to schedule another process
- OS overhead itself is useless work. Ideally, OS should achieve its goals with zero overhead. That means the OS policies are typically simple.
  - We rarely use complex policies that might give good performance in the long run unless we know for a fact that we will get better performance most of the time
  - Knowing the future would help. Frequently, we approximate by using the past to predict the future. Fails when changing between phases.
- Question: When resources become plentiful, what is the role of OS? Process scheduling in a 32 core laptop processor
  - Question: When resources are extremely scarce, what is the role of OS (100 MHz laptop processor)?

Processes/threads

- Process control block to represent process and its current state
  - Threads to describe multiple threads of execution
- Process/thread scheduling multiplexes multiple processes/threads in order to improve throughput
  - Round-Robin, FCFS, gang scheduling ….
- Thread synchronization primitives to allow applications to use multiple CPUs simultaneously
- Hardware support for context switching CPU, thread synchronization etc. helps
  - User level implementation is faster than kernel level
- Question: How important is process scheduling for desktops/laptops/PDAs/servers?

Memory management

- Ideally, memory is fast enough to keep up with CPU demand
  - Practically, fast memory is expensive while inexpensive memory is slow
- Manage memory hierarchy to achieve good performance by keeping data in fastest memory
  - Keep data in TLB, Cache
  - Predicting future helps. Use past to predict future
- Virtual memory makes memory appear larger
  - Internal or external fragmentation
  - Fixed size blocks vs variable sized blocks
- Can perform interesting tasks depending on the hardware support (MMU)

File system

- Manage persistent data
  - Develop data structures to name objects (files) and manage them (directories)
  - Achieve reliability using replication (RAID)
  - File system should be tuned for small files/large files, sequential/random access, …
  - Predictable future is good, use past to tune systems
- Achieve good performance by buffering objects in memory
  - Tradeoff reliability of storage system
- Disk IO is slow. Hence, we wire-down pages that are in the middle of IO
- Question: What happens to disk scheduling on 32 core processor?

Managing IO

- Hardware support is preferred
  - DMA vs programmed IO
    - When the DMA controller is running, we may have to wire-down pages
  - DMA controller, Graphics co-processor, Network processor, Disk controller, Bus controller etc. etc.
    - Require drivers to control each device
    - Drivers written by vendors
    - Reliability of OS is the sum total of OS + drivers
      - Assume that the graphics driver crashed. What can the OS do?

Lifecycle

- Suppose we have two processes that require the CPU. The first one had the CPU and you would like to let the second process run, ie context switch. Should you do it at this time?
  - Cost of context switch
  - Opportunity cost of flushing TLB/cache
  - Cost of losing IO locality for file system
  - Cost of flushing buffers to disks and bringing in new pages
    - Pages might be wired during transfer preventing new process from running (by making them wait for memory to be freed by previous process which was context switched and hence is not running anyways)
- A good scheduler would optimize across all these parameters: quickly

Next class

- We will ponder how one would build OS for PDA, laptop, desktop, server, etc.
- Hot research areas: Energy management for servers/laptops, Virtual machine support for isolation (Java, Xen, VMWare, Parallels, Wine etc.), Grid/cluster computing to harness lots of machines, autonomic OS/storage etc.