Operating Systems: What did we learn?

1. Programming abstractions:
2. How they are implemented
   - Implementation constrained by hardware
   - Hardware implements things that are usable by software

- Programming abstractions "learnt":
  - Processes – needed by OS, but applications can use them via fork() and other calls
  - Threads – needed to use a hardware capability (CPU)
    - lots of complications: need support for locks, semaphores. Be aware of deadlocks etc.
  - Memory – you learnt them in programming class
  - Files – you learnt them in programming class
  - Protection and security – you knew they were there…

How are they implemented?

- Processes: Table for maintaining info (PCB), how to create, destroy and schedule processes
- Memory: Logical vs physical memory (you didn't have to worry about this because the compiler does that for you), paging and virtual memory (you didn't worry about this unless you cared about performance)
- Files: Directories, partitions, files, blocks, disk scheduling etc. (you didn't have to worry about this unless you worried about performance)
  - RAID: probably new, especially the details
- Protection: Access matrix, ACL, capabilities
- So, what drives these implementations?

Operating System is all about tradeoffs

- There are no magic bullets. Everything is a compromise.
- The compromise is made by the OS manufacturer, by analyzing “typical scenarios” and optimizing the OS to work for those scenarios.
- “According to the Office performance benchmarks, Windows XP SP3 is also considerably faster than Vista SP1” - PC World
  - What does this sentence mean?
- Take away message: you can either learn what those tradeoffs are and make sure that your code works well with them
  - Row-major ordering means certain types of memory accesses are good

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)?

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

Designing an Operating Systems

- Goal is to understand how the technologies that we studied so far apply to typical machines
- First we focus on PDAs and Laptops
  - Both are mobile, inexpensive
  - Battery is a big concern
    - Quick startup
    - Quick shutdown
    - Frequent suspends

PDA

- Small mobile devices
- Important design elements:
  - Inexpensive
  - Mobile (small, rugged, good battery life)
  - Constrained CPU, memory, storage, screen
  - CPU: 200 MHz
  - Memory: 64 MB
  - Storage: Flash or Microdrive
- OS: Symbian, PalmOS, MS Windows Mobile, QNX, Linux, MacOS?

PDA and Process

- Usually: only one user, process at any one time
  - Palm context switches by “freezing” process state and unthawing old process
- Process Synchronization: Little system support.
- Many multimedia applications (video, audio, cellular calls)

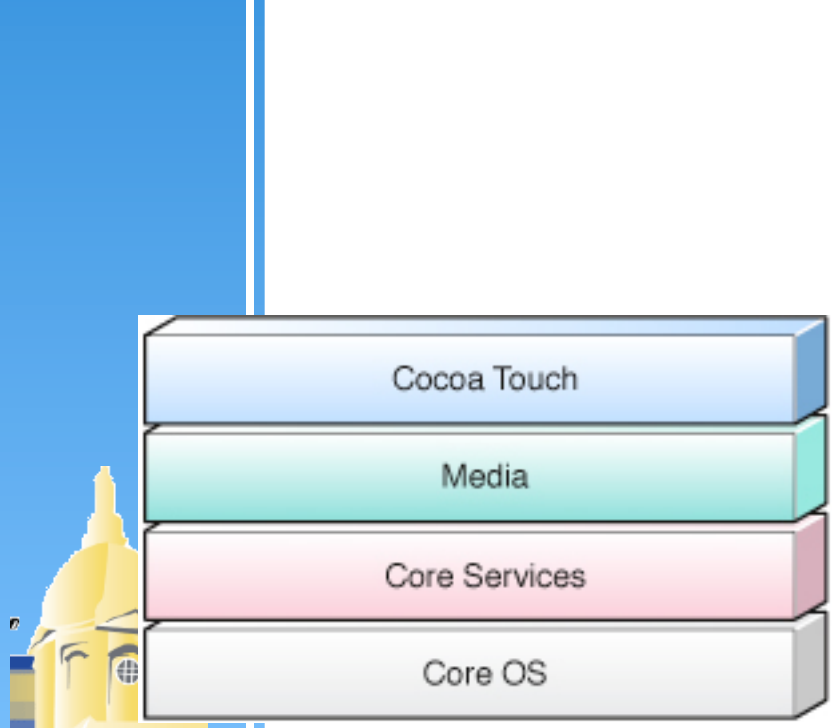
PDA and memory/storage

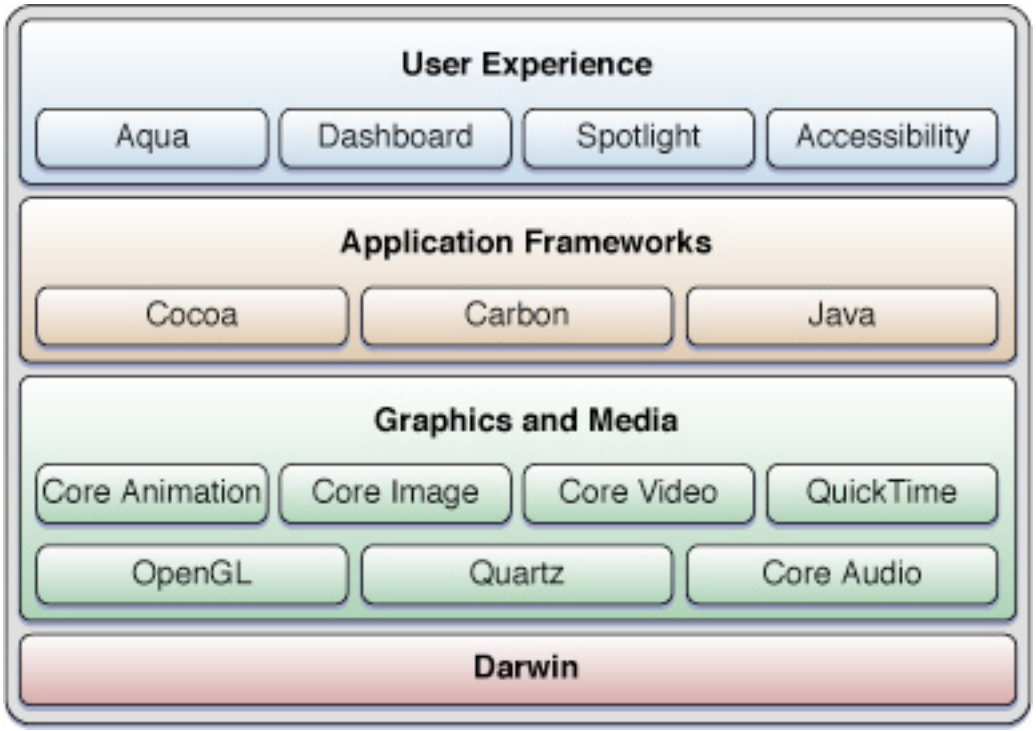
- Usually no MMU

- Storage: Flash or Microdrive
  - Flash has no moving components, however can only be rewritten a finite number of times
  - Mobile device and so storage should be consistent

PDA and security

- Heavily uses physical security feature
- Overall: What is the roll of PDA and whatever we learnt?
  - Why do we even discuss PDA class machines?





Laptop class

- Important design factors:
  - Cost, weight
  - CPU: as fast as your lap can tolerate
  - Memory: up to 4 GB
  - Disk: up to 320 GB
    - Sandisk 64 GB flash
  - Energy consumed depends on amount of resource
- OS: MS Windows, Mac OSX, Linux, FreeBSD, …

Laptops and Processes

- Modern laptops are multi-core
  - Mostly interactive tasks and hence prefer interactive applications
  - Frequent suspend - does that affect scheduling?
- Process synchronization
- Users use productivity apps, multimedia apps and solitaire

Laptops - memory and storage

- What do you do with 4 GB on a laptop?
  - Leave memory of exited programs to quicken startup?
    - Energy cost
  - Use massive buffered IO?
    - Reliability when memory runs out
- Disks and Flash
  - Disks support fully operational, spin-down, park modes

Laptops and protection

- Physical security still possible
- Rarely multiuser

Desktop

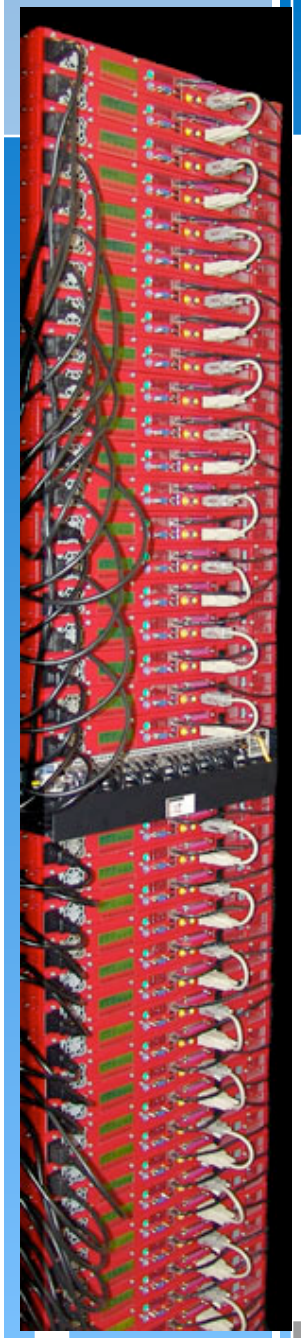
- Dual processor/quad core
  - 3+ GHz dual core x2 and 64 bit processor
- GBs of memory
- Multiple hard disks
  - Hard disk can be up to 1 TB per disk!!

Desktop and Process scheduling…

- What do you do with these beasts?
  - Web browse
  - Emails
  - Word
  - Multimedia encoding/creation
- Scheduling a balance of interactive and batch processing

Memory and File system

- RAID becomes increasingly necessary for most machines, given that 500 GB hard drive is ~$60
- Desktops, if they knew that they would be on UPS, can afford to really use a lots of caching and buffering
- Security wise, desktops are similar to workstations in that they are single user at a time



Data center server

- One of the specification is the size that server will take in a rack. 1U is the smallest size and blade servers, which fit one unit are all the rage
- Dual (Quad Core Xeon, 2x4MB Cache, 2.66 GHz, 1333 MHz FSB), 16 GB memory, 2x73GB 15k rpm hard disk - $10000
- 1 rack - 60 racks
  - ($ 0.6 m)

Servers

- Mission critical systems
  - Three tier systems - production, backup and test
  - Virtual hosting to protect against interference with other processes
  - Data center support service level agreements (SLA) - OS should be aware of these
  - On demand computing
  - Autonomic management
- Each rack can consume 10 Kw
  - Additional 10 Kw in cooling
  - Data center can be powered exclusively by a 300 MW power station.

Hot topics

- 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.