CMPSCI 377: Operating Systems

Prashant Shenoy

Department of Computer Science,
University of Massachusetts
Amherst, MA 01003

E-mail: shenoy@cs.umass.edu, Phone: (413) 577-0850, Fax: (413) 545-1249
URL: http://www.cs.umass.edu/~shenoy/courses/377

Today: Introduction to Operating Systems

- Course Organization & Outline (handout)
- Prerequisite & Course Sign Up (handout)
- Introduction and History of Operating Systems
  - What is an Operating System (OS)?
  - Why are Operating Systems interesting and important?
  - A little historical perspective on OS.

Course Organization

- Course is over capacity!
- Enrollment policy

Prerequisites and Syllabus

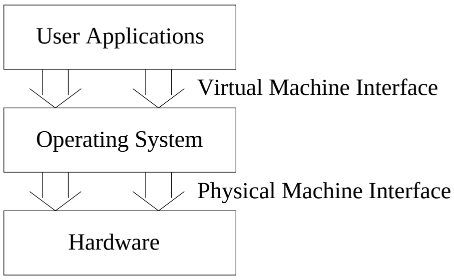
- CMPSCI 187 (Data structures) and CMPSCI 201 (Architecture)
- **Textbook: Applied Operating System Concepts 1ed** (Silberschatz,Galvin,Gagne)
- Course requirements:
  - 6 homeworks plus few in-class assignments (20%)
  - 3 programming assignments (40%)
  - 3 exams (40%)
  - Grading on the curve
- Programming assignments: will use Java
- Strict late policies and policies on cheating

Course Organization: Misc

- Accounts in the Ed-lab: 20 Linux-based PCs, 20 Dec Alphas
- Discussion section to help you with Lab assignments
- **Office hours:**
  - Instructor: TuThu: 4:00-5:00, CS 336 or by appt
  - TA: Wen Yi: wyi@ecs.umass.edu
  - Off hrs: Mon 3:00-4:00, Fri 2:00-3:00 Location: TBA

Introduction to Operating Systems

What is an Operating System?



- Operating system (OS)
  - Interface between the user and the architecture
  - Implements a virtual machine that is (hopefully) easier to program than raw hardware.

Operating System: Salient Features

- **Services:** The OS provides standard services (the interface) which the hardware implements.
  - Examples: the file system, virtual memory, networking, CPU scheduling, and time-sharing
- **Coordination:** The OS coordinates multiple applications and users to achieve fairness and efficiency (throughput).
  - Examples: concurrency, memory protection, networking, and security.
- **Goal:** Design an OS so that the machine is **convenient** to use (a software engineering problem) and **efficient** (a system and engineering problem).

Why Study Operating Systems?

- **Abstraction:** How to get the OS to give users an illusion of infinite memory, CPUs, resources, world wide computing, etc.
- **System Design:** How to make tradeoffs between
  - performance and the convenience of OS abstractions,
  - performance and the simplicity of OS design, and
  - putting functionality in hardware or software.

  As systems change the OS must adapt (e.g., new hardware, software).
- **Basic Understanding:** The OS provides the services that allow application programs to work at all.
- *System Intersection Point:* The OS is the point where hardware and software meet.

Why Study Operating Systems?

Not many operating systems are under development, so you are unlikely to get a job building an OS. However, understanding operating systems will enable you to use your computer more effectively. They also serve as an excellent example of system design issues whose results and ideas you will apply elsewhere.

Background: To understand this course you must have a solid basic understanding of hardware (CPU instruction sets, memory hierarchies, I/O systems, etc.) and solid programming skills (complex data structures, classes as an encapsulation mechanism, etc.)

$\Rightarrow$ Obviously, you cannot understand the implications of how components intersect without understanding the components.

History of Operating Systems

Phase 1: Hardware is very expensive, humans are cheap

1. One user at a time on the console
   - One function at a time (no overlap of computation and I/O)
   - User must be on the console to debug
2. Batch processing: load program, run, print results, dump, repeat
   - Users give their program (on cards or tape) to a human who then schedules the jobs
   - OS loads, runs, and dumps user jobs
   - More efficient use of the hardware, but debugging is more difficult
3. Data Channels, interrupts, overlap of I/O and computation
   - Buffering and interrupt handling in OS
   - Spool jobs on drum
   - No protection → One job at a time
   - Performance improves because I/O and processing happen concurrently

History of Operating Systems

Phase 1: Hardware is very expensive, humans are cheap

4. Multiprogramming: several programs run at the same time, sharing the machine, i.e., I/O and CPU processing overlap.

- One job runs until it performs I/O, then another job gets the CPU
- OS manages interactions between concurrent programs
  - Decides which spooled jobs to start
  - Protects one program's memory from other programs
  - Decides which process to resume when one gives up the CPU
- First OS failures
  - Multics announced in 1963, released in 1969
  - OS/360 released with 1000 known bugs

⇒ OS design and development is a science

History of Operating Systems

Phase 2: Hardware is cheap, humans are expensive

5. Interactive timesharing

- Terminals are cheap
  - Many users can interact with the system at once, debugging is easy
  - Process switching occurs much more frequently
- Memory is cheap - programs and data go on-line
  - 1 punch card = 100 bytes, 1MB = 10K cards
  - OS/360 was a stack of cards several feet high
- UNIX simplifies Multics so it can be built
  - Shell to accept interactive commands
  - File system to hold programs and data on disk
  - Rapid process switching to provide users with ability to interact with programs
  - Virtual memory holds lots of programs and data $\Rightarrow$ many processes can run simultaneously
- New problems - response time & thrashing
  - No control over number of simultaneous users

History of Operating Systems

Phase 3: Hardware is very cheap, humans are expensive

6. Personal computing: computers are cheap, so put one in each terminal

- Idea was to make the OS simple (again) by getting rid of support for multiprogramming, concurrency, and protection.
- Did not really work... Microsoft is putting all this functionality back into its OS
- Why? Distributed computing & networking - we still want to share resources, but now we want to share across machines

History of Operating Systems

Phase 4: Hardware is very cheap, processing demands are increasing

7. Parallel and distributed computing: allow multiple processors to share resources

- In parallel systems, multiple processors are in the same machine, sharing memory, I/O devices, clock, ...
- In distributed systems, multiple processor communicate via a network
- Advantages: increased performance, increased reliability, sharing of specialized resources

8. Real-time systems allow computers to control physical machines or provide high-quality interaction as in virtual reality

- Timing requirements provide deadlines by when tasks must be accomplished.
- Hard real-time OS must meet timing requirements. Omit features with unpredictable timing: user shell, virtual memory, disks.
- Soft real-time OS allow deadlines to be missed.

History Lesson

Batch processing was right for its time, but not anymore.

⇒**Change is one of the defining forces in computer science.**

Example

	1983	**1999**
MIPS	0.5	500
price/MIP	$100,000	$500
memory	1 MByte	1 GByte
network	10 Mbit/s	1 GB/s
store	1 GByte	1 TByte
addressable bits	32	64

- From 1953 to now (the 40 year history of computing), 9 orders of magnitude change in almost every computer system component.

History Lesson

This degree of change has no counterpart in any other area of business.

Examples:

- Transportation – over the last 200 years, we have gone from horseback (10 miles/hour) to the Concorde (1000 miles/hour) - 2 orders of magnitude.
- Communication – at the invention of the telephone (voice), TV (video) and fax (text & pictures), communication went from the speed of transportation to nearly the speed of light - 7 orders of magnitude.