



**Department of Physics,
Computer Science & Engineering**

CPSC 410 – Operating Systems I

Chapter 0: Introduction

Keith Perkins

Topics

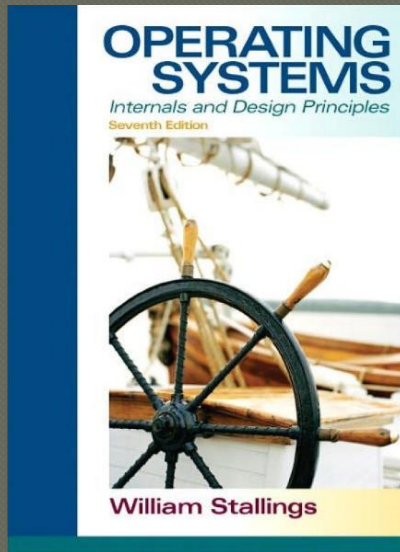
- Admin
 - prereqs, text, eval, etc...
- What is an Operating System (OS)?
- Where are OS coming from?
 - History by functionality
- Closing remarks
 - OS & Software Engineering

Admin: Your Background

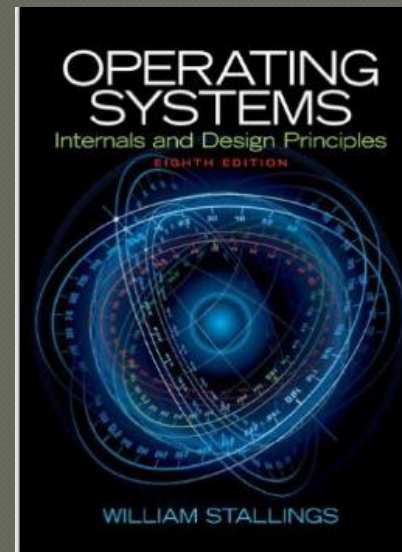
- Some high level programming language
- CPSC 270 AND CPEN 214
- Pre or coreq CPSC 330 or CPEN 315

Admin: Text



- Operating Systems Internals and Design Principles, William Stallings, 7th Edition, Pearson, 2011.



or



Admin: Evaluation

- Multiple projects  Probably 3 but may be as many as 8
- 1 midterm  Lateish in the semester
- 1 final

Admin: What you get from this class

- Some C++ experience
- How an OS works (multitasking)
- Scheduling
- Memory Management
- I/O management and File Management
- Threads and concurrency

Wish

Most Useful

Admin: Language

● C++

- I'll give a brief intro (2-3 weeks)
- One C++ starter project
- The rest will be OS specific

● Why C++

Admin: Compiling and debugging

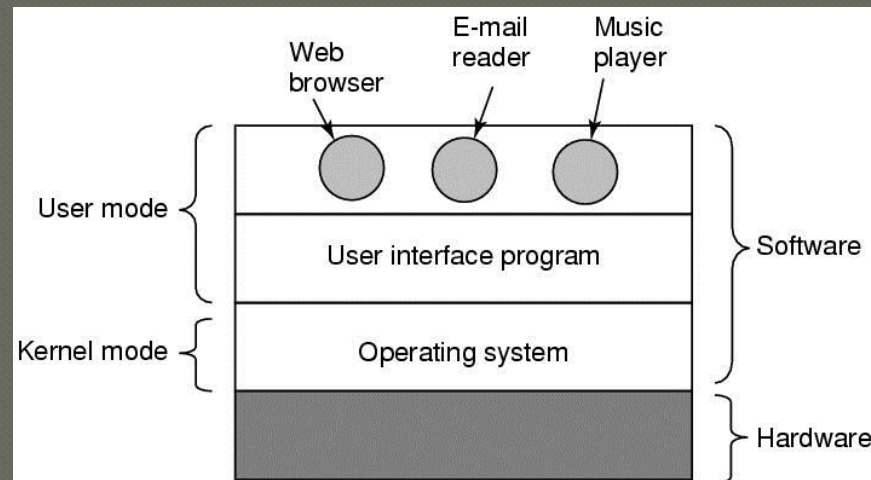
- ◉ C and C++ both compile to an executable
- ◉ Can use many compilers (clang, gcc, G++ ...)
- ◉ Show you how to use gcc and g++
- ◉ Show you how to use make files
- ◉ Show you how to debug
- ◉ Show you IDEs for C++
 - And I hope to demonstrate the advantage of an IDE
- ◉ All this starting next time

-
- But first a little OS

What is an OS?

● An Operating System is software that acts as:

- 1) an Extended Machine
 - To provide consistent, simpler, high-level interfaces (abstractions) between user programs and hardware
- 2) a Resource Manager
 - To provide an orderly and controlled access to hardware.



History of OS

1. Using I/O devices

- Problem

- Programs should not handle I/O devices directly
 - e.g., spinning disk, moving disk arm, find tracks to read/write

- Solution

- Build modules (device drivers) to:
 - manage the interaction with I/O devices.
 - provide a high-level & consistent abstraction across device types
 - e.g. the concept of a “file” accessed through read/write operations.

- Result

- Programs dealt with a well-defined & simple interface for a range of device types (e.g., disks/tapes, printers)

History of OS

2. Idle CPU Time (between job runs)

- Problem

- Because of how early computers were used (programmers booked time slots), the CPU was idle most of the time waiting for input (from one programmer to the next) and output (e.g., printer, disk).

- Solution

- Queue incoming programs, so that they can be executed one immediately after another (**batch processing**), and queue outgoing results for processing (**spooling**).

- Result

- CPU is freed sooner to take next job

History of OS

3. Idle CPU Time (waiting for I/O)

- Problem

- Since the CPU is faster than I/O devices (e.g., printer, disk), the CPU must sit idle waiting for I/O operations to finish.

- Solution

- Allow another job to take over the CPU while an I/O operation in the current job finishes (**multiprogramming**).
- Use **memory buffers** (to hold I/O data) and **interrupts** (to signal the CPU when an I/O operation is over).

- Result

- CPU performs **cooperative multitasking** (non-preemptive)

OS Lessons & Challenges

● Lessons for Software Engineering

- OS are some of the larger software projects ever attempted.
 - Hundreds of programmers
 - Millions of lines of code
- How to tackle such an effort while...
 - creating few bugs?
 - being adaptable to change?

Next Time

- C++
- Please have Visual Studio installed on your machine
- Compiling, linking, running, debugging
 - The hard way
 - The easy way