

# CSE-380/381

# Operating Systems

Instructor: Matt Blaze

# Important Administrative Stuff

- This is CSE-380/381 - Operating Systems
  - Meets Tuesday & Thursday, 1330-1500
  - We should be in Moore 216
- This is really two courses!
  - The “lecture course” (CSE 380)
    - lectures, homework, exams
  - The “lab” course (CSE 381)
    - weekly (optional) recitation session, Wednesday 7-8
    - no exams, grade based on individual and group projects
- Most students required to register for both courses

# Meet your staff...

- Instructor for CSE-380: Matt Blaze
  - Office: Levine 611
  - Office Hours: TBA & by appointment (email me!)
- Co-Instructors for CSE-381:
  - Eric Cronin, Gaurav Shah, and Micah Sherr
  - Office: GRW 461
  - Office Hours:
    - Cronin: Friday 1100-1200
    - Shah: Tuesday 1600-1700
    - Sherr: Thursday 1100-1200

# Meet more of your staff... TAs

- Sandy Clark
  - (room TBA)
  - Friday 1330-1430
- Joey Schorr
  - (room TBA)
  - (time TBA)
- Luke Zarko
  - (room TBA)
  - Wednesday 1430-1530
- EVERYONE is best reached by email (see course web page for addresses); make appointments for office visits when possible

# Textbook, course web page

- A.S. Tanenbaum. *Modern Operating Systems (2nd Edition)*. Prentice-Hall 2001.
  - it's in the book store
- Course web page is:  
[www.crypto.com/courses/fall107/cse380/](http://www.crypto.com/courses/fall107/cse380/)
  - Check frequently for updates and news
- There's also a course mailing list and news group

# Prerequisites

- Basic understanding of computer architecture and organization (CSE-240)
- Working knowledge of C language and Unix shell and programming environment
- Shell login on ENIAC Linux cluster
  - Unix shell account
  - Check to make sure you have a working account. (Do this TODAY)

# Grading

- For 380:
  - Two Midterms (20% + 20%)
  - One Final (40%)
  - About four or five homeworks (20%)
    - fascist, inflexible late homework policy strictly enforced!
  - Some extra credit opportunities
- For 381:
  - Individual project (50%)
  - Group project (50%)

# A quick word from CSE-381...

- Weekly meetings scheduled for
  - Wednesday 1900-2000
    - Towne 313
- You need to spend time over the next weeks meeting other students and figuring out who you might want to work with on the group project
- 381 textbook:
  - W. Richard Stevens and Stephen A. Rago. *Advanced Programming in the UNIX Environment (2/e)*. Addison-Wesley Professional. 2005

# What's an Operating System?

- Good question!
- You might think you know now
- You probably won't think you know in the middle of the course
- You may or may not know at the end of the course

# Why would I want an Operating System?

- Let's take a simple program
  - print “this is not a printout” on the printer
  - terminate
- This should be very simple

# No OS

- Get printer manual
  - find out how to send messages to it
- Write program:
  - Put the character string “this is not a printout” in a memory buffer
  - do the stuff that printer requires to send buffer to it.
  - go into endless loop
    - (wait for someone to turn the computer off)
- Get hold of a computer
  - it’s got to be all yours
- Translate your program into machine code
- Figure out a way to get the program into memory
  - font panel switches
  - burn a ROM
  - punch cards
- Start the program (somehow)
- Turn computer off when done.

# With OS

- Put program in file:
  - Put character string “this is not a printout” in a memory buffer
  - Issue *system call* to send buffer to printer
- Compile the program and tell the OS to run the program file
- That’s it.
- (but what’s a system call?)

# An OS provides an abstract interface to programs

- Processes
  - hides programs from one another
- Traffic cop
  - resource management
  - who gets to run, when?
- Memory management
  - protection from other programs' mistakes
- Security
  - protection from other programs' malice
- System call interface
  - abstract, simplified interface to services
  - like a function library, but communicates with OS
- Portability
  - programs don't have to take into account details of their environment
- Device management
- Communication
  - between processes
  - to devices & networks

# Things related to the OS (but not really part of it)

- GUIs and user interfaces
  - Applications (e.g., web browser)
  - Compilers
  - Libraries (sort of)
- 
- These things are usually implemented as “user” programs

# Processes / Time Sharing

- Allows program to act as if it “owns” the machine
  - not worry about other running programs
  - not use more than its fair share of resources
  - get services from the OS
- Exploits two hardware / CPU mechanisms:
  - relocatable memory
  - “kernel mode” and “user mode” CPU state
    - “traps”

# Relocatable Memory

- CPU feature that allows programs to not know in advance where they will be in memory
  - and also avoid stepping on each other's memory by mistake
- Many ways to this is implemented:
  - offsets, prefixes, virtual memory
  - details don't matter yet
- Every running program can have its own private *address space*

# “Supervisor mode” / “User mode” and “traps”

- When the CPU is in “supervisor” mode, it can do anything, and can address any part of memory.
  - the OS kernel runs in supervisor mode
  - supervisor mode can switch to user mode at will
- When the CPU is in “user” mode, it has access only to its own address space, and can’t talk directly to devices
  - User process run in user mode
  - But after a trap we can switch to supervisor mode...

# “Traps”

- A user mode program can switch to supervisor mode by issuing a “trap”
  - but there’s a catch... when it issues a trap it starts running OS code in the supervisor address space
  - user program can’t overwrite this code
- Traps are used to implement system calls to provide services to user processes that require:
  - communication with a hardware device
  - communication with another process or with the OS
- Traps also happen when devices issue *interrupts*
  - real time clock ticks, message arrives on network, etc.

# How an OS runs processes...

## (an oversimplification)

- OS must keep track of which process are assigned which sections of memory plus other stuff
- To run a new process, assign it some memory and put its code there
  - switch to user mode and start running at the first address of the program
- The OS keeps a record of every process
  - assigned memory is, current program counter, etc.
  - this is called the process' *context*
  - enough information to restart process where it left off

# Eventually, a trap happens

- Either because the running program issued it or because a device did (e.g., the clock ticked)
- First the OS records the state of the running process' context in its record
- Next it will do whatever servicing the trap requires
  - e.g., send something to the printer
- finally, it will pick a user process to restart
  - maybe the one that was running, maybe not
  - restarts based on the new process' context record
  - back in user mode now

# So now what?

- If you understand Unix-like processes, you're in good shape!
- We'll study in detail:
  - Process and memory management
  - OS interfaces and services
  - Interprocess communication
  - Algorithms for making sure these things work correctly and efficiently
  - File systems
  - and more...