Outline

- **Introduction**
  - what is an operating system?
  - why you should care?
- **Course organization, policies and guidelines**
  - topics
  - workload and grading
  - collaboration policy
- **History of operating systems**

[Silberschatz/Galvin: Chapter 1]

What is an Operating System?

- An operating system is
  - a **government**: legislates/enforces proper use of system resources
  - a **resource allocator**
  - a **control program**: prevents errors and improper use

Reasons for an Operating System

- An operating system provides
  - **convenience** for the user
  - **efficiency**
    - particularly important for large, shared multiuser systems
    - less important these days with dedicated single-user systems
  - a simple, more powerful **virtual machine**
    - convenient abstractions for hardware resources (e.g., disks)
  - **sharing** of resources
  - **isolation/protection** among user programs
Why Study Operating Systems?

- Arguments against:
  - “very few OS designers/implementors needed”
  - “all I need to know is in the manual pages”
  - “everybody is going to run Windows anyway”

- Arguments for:
  - need to know about (large) system design in general
  - OSes include several important design/optimization problems
    - resource sharing and management
    - protection and security
    - flexibility, robustness, and performance
    - design of good interfaces
  - growing need for OSes
    - embedded systems
    - several large applications contain mini-OSes
  - crucial for understanding application-hardware interactions

What This Course is About

- Understanding the general principles of OS design
  - focus on general-purpose, multi-user, uniprocessor systems
  - emphasis on widely applicable concepts, rather than the features of any specific OS
    - protected kernels
    - processes and threads
    - concurrency and synchronization
    - memory management and virtual memory
    - file systems
  - Understanding problems, solutions, and design choices
  - Understanding implementations of these concepts in a non-trivial instructional OS (Nachos)

What This Course Does Not Cover

- Specific features of commercial OS products
  - “how do I do X in operating system Y?”

- Topics deferred to advanced courses
  - Networking, NFS
  - Analytical modeling
  - Transactions and Database OSes
  - Distributed OSes

Prerequisites

- Official
  - V22.0201: Computer Systems Organization I

- Would prefer
  - you to have taken the Honors section of V22.0201
  - you to have some basic familiarity with
    - programming in C/C++
    - UNIX tools and development environment
      - command familiarity
      - editors (vi, emacs), compilers (gcc), debuggers (gdb), makefiles (GNU make)

- Since this is an Honors section
  - no recitations scheduled
  - you will have to pick up necessary knowledge of the above on your own in the first few weeks
  - we will help (course web page has links to online tutorials)
Tentative Course Schedule

Lectures 1-2: Overview
Lectures 3-11: Process management
- processes and threads, scheduling, synchronization, deadlocks
Midterm Exam: March 05, 2001
Spring Break: March 12—18, 2001
Lectures 12-18: Storage management
- memory management, virtual memory, file systems
Lectures 19-21: I/O systems
Lectures 22-24: Protection and security
Final Exam: May 04, 2001

Assessment of Student Background

- Programming languages
  - Pascal
  - Java
  - C/C++
- UNIX environments
  - commands: ls, cat, mkdir, cd
  - editors: vi, emacs
  - program development environments:
    - compilers (gcc) and makefiles
    - debuggers (dbx, gdb)
- Computer systems organization
  - CPU, RAM, disk, cache
  - interrupts, DMA

Workload and Grading

- This is a challenging course: minimum 15 – 20 hours of effort/week
  - Classes and assigned readings
  - Six programming projects: 50%
    - each due approximately 2 weeks after it is handed out
    - you will do these in groups of 3 students
    - additional info on next few slides
  - Midterm exam 20%
  - Final exam (only material after the midterm) 30%
- Grading will be on the curve
  - relative to overall class performance
  - I will use subjective factors to decide whether to push those on the border to the next higher grade

Nachos Projects

- Nachos: An “instructional” operating system
  - developed by Thomas Anderson and others at U.C. Berkeley
  - has seen widespread use in undergraduate classes since 1995
  - what is it?
    - a user program that runs on a standard OS
    - all the features of a real OS, but much simpler
    - ~8000 lines in a restricted subset of C++
    - includes (simulated) hardware and hooks for standard OS components
- Course programming projects
  - flesh out the baseline implementation of various OS modules
    - protected kernels, threads and synchronization, multiprogramming support, I/O (files), and virtual memory
  - each project builds upon what you have already done
    - at the end, you will be able to see execution of multiple user programs on shared hardware in a protected fashion
    - effectively, you would have built a non-trivial OS
Nachos Projects (contd.)

- Project groups
  - each Nachos lab will be done in groups of 2-4 students
  - **form groups today: send me e-mail (one per group) latest by Friday**
    - look for complementary skills (additional info on next slide)
  - **Lab 1 is due two weeks from today (January 31, 2001)**

- Nachos project guide
  - contains all of the required information
    - what you have to do in each lab
    - how do you hand in for the lab
    - how will you be graded, etc.

- Computing resources
  - On an ITS Sparc/Solaris machine
    - remote access using telnet, ssh
  - your accounts will be set up by *early next week*
    - Download Nachos code and look through it in the meantime

Some Suggestions about the Nachos Projects

- Allocate time for reading the Nachos code
  - Read the overview documents and the project guide before starting
  - The project guide tells you which directories/files you need to look at

- Carefully design your solution before you start coding
  - Getting the logic right is as hard (if not harder) than coding/debugging

- Start early
  - If you plan to work only the last 2-3 days, you **will not** have enough time to complete the project

- Resolve any group dynamics issues before they become problems
  - The reason there are groups is because there is that much work
  - Working in a group is difficult, but it will serve you well

Course Resources

- Text book(s)
  -(recommended) Tanenbaum, *Modern Operating Systems*

- Nachos project guide

- Course web page:
  - [http://www.cs.nyu.edu/courses/spring01/V22.0202-001/index.htm](http://www.cs.nyu.edu/courses/spring01/V22.0202-001/index.htm)

- Class mailing list: v22_0202_001_sp01@cs.nyu.edu
  - send questions of general interest here

- E-mail to the TA or me (vijayk@cs.nyu.edu)
  - lab-specific questions should be directed to the TA

A Note about the Course Materials

- Lecture slides
  - originally prepared by Profs. Krishna Palem and Malcolm Harrison
  - have taken on a very “Vijay Karamcheti” flavor

- Nachos projects, documentation, and resources
  - borrow heavily from resources developed by Jeff Chase and others at Duke University
Outline

- Introduction
  - what is an operating system?
  - why you should care?
- Administrivia
  - course organization
  - workload and grading
- History of operating systems

[Lectures Summary: History of OSes]

No OS (1950s)

- Bare machine, single user
  - a button which executed a bootstrap loader
  - programming in octal, no assembler or compiler
  - input via paper tape, or punched cards
- Operator
  - user
  - machine operator
- Main perceived problems
  - human actions were slow
  - inefficient use of expensive hardware

Batch Systems (early 1960s)

- Reduce set-up time by batching jobs with similar requirements
- One user at a time
- I/O devices
  - punched cards, magnetic tape, text on paper
- Operation
  - load jobs (punched cards) onto magnetic tape
  - process jobs on tape serially
  - output to tape
  - print output tape
- Main perceived problem
  - turn-around time: up to two days
  - CPU often underutilized
    - most of the time spent reading and writing from tape
Innovation: Resident Monitor

- Automatic job sequencing
  - use of control cards
- Job Control Language
  - commands
    - mount this tape
    - compile
    - run
- OSes begin to be important
  - IBM: Fortran Monitor System
- Main perceived problems:
  - turn-around time
  - inefficient use of expensive hardware
  - CPU still mostly idle

Innovation: Spooling

- Use of disks to buffer input/output to tapes
  - disks are random-access I/O devices
- Overlapped I/O and computation
  - one job’s I/O can be overlapped with another’s computation
  - need for interrupts
- Independent I/O controllers
  - CPU: starts I/O operation; continues computation
  - Controller: does I/O; interrupts CPU
- Initially off-line spooling, later on-line

Multiprogramming Systems (1960s)

- Many programs simultaneously in memory
  - objective: to keep CPU busy
  - OS switches between user processes
- How to ensure that these programs do not interfere with each other?
- Hardware innovations
  - memory protection
  - privileged instructions

Time Sharing and Interactive Systems

- Originally proposed by Strachey ~1960
  - programs could interact with user
- Programs
  - could wait for I/O for an arbitrary time
    - CPU switched to another job
  - however, resident jobs took up valuable memory
    - needed to be swapped out to disk
    - virtual memory
- OS research in 60s
  - CTSS, MULTICS at MIT
  - Atlas (spooling, demand paging) at Manchester U
OS Requirements (late 1960s)

- Multiprogramming
  - memory allocation & protection
  - I/O operations were responsibility of OS
- Interactive systems
  - scheduling issues
  - swapping, or virtual memory
- Users wanted permanent files
  - hierarchical directory systems

OSes in the 1960s

- Increased in size and complexity
- Were not well understood
  - IBM: OS/360
  - CDC: Sipros, Chippewa, NOS
  - OS structure was specialized to the hardware
- Much money was spent and wasted
- WHY ??

UNIX (early 1970s)

- Originally developed at Bell Labs for the PDP-7
  - Ken Thompson
  - Dennis Ritchie
- Smaller & simpler
  - process spawn and control
    - each command creates a new process
  - simple inter-process communication
  - command interpreter not built in: runs as a user process
  - files were streams of bytes
  - hierarchical file system
- Advantages
  - written in a high-level language
  - distributed in source form
  - powerful OS primitives on an inexpensive platform

Personal Computers (1980s)

- Originally
  - single user
  - simplified OSs
    - no memory protection
    - MS-DOS
- Now run sophisticated OSes
  - Windows NT, Linux
Windowing Systems (1980s)

- Originally based on work at Xerox Parc
- Popularized by the Macintosh
- Characterized by
  - graphical interface
  - mouse control

Networks of Workstations (1990s)

- High-speed network connections
- Local & world-wide
- Client-server systems
  - file systems
  - remote windowing systems
- Support a variety of node OSes
  - Unix, Windows NT

The Future

- Distributed systems
  - network is invisible
- Micro-kernel and extensible OSes
  - support multiple OS flavors
    - e.g., Mach, Amoeba, Windows NT
  - allow insertion of application-specific functionality
- Embedded devices and network computers
  - computer runs a very thin OS (Java Virtual Machine)

- Unfortunately, we will not talk about these in this course
  - but, opportunities in my research group

Next Lecture

- Review of computer-system structures
  - I/O structure and storage hierarchy
  - hardware protection
- Operating-system structures
  - system components and services, system calls
  - virtual machines
  - system design and implementation

Reading

- Silberschatz/Galvin: Chapters 2-3