CS 537: Operating Systems
Fall 2007

Course Introduction

Mike Swift

Today's agenda

- Administrivia
  - course overview
    - course staff
    - general structure
    - your to-do list
- OS overview
  - functional
    - resource mgmt, major issues
  - historical
    - batch systems, multiprogramming, time shared OS's
    - PCs, networked computers

Course overview

- Everything you need to know will be on the course web page:

  http://www.cs.wisc.edu/~cs537-2

  - Schedule
  - Readings
  - Writings
  - Projects

- But to tide you over for the next hour …
  - course staff
    - Mike Swift
    - Sriram Subramanian
  - general structure
    - read the text after to class
    - class will supplement rather than regurgitate the text
    - sections will focus on the project, quizzes, writing
    - we really want to encourage *discussion*, both in class and in section

- your to-do list …
  - please read the entire course web thoroughly, *today*
  - project 1 is posted on the web now and will be discussed in section next week; due two weeks from Thursday

Registration Stuff

- This class has a significant amount of work
  - 4 programming projects
  - 8 Quizzes
  - Writing assignments
  - Dates are not flexible
- If you're going to drop this course
  - please do it soon!

Readings

- Textbook: Operating Systems Concepts
  - Readings are assigned to be done after lecture

Grades

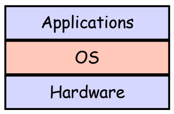
- Exams: 22-47%
  - No midterm
  - Final is optional
  - 8 quizzes throughout the semester
    - I will drop your lowest score
- Projects: 38-50%
  - 4 projects, roughly every 4 weeks
  - Programming will be in C
- Quizzes: 22-30%
  - During section
  - 1 question per quiz will come from the textbook problems
- Writing: 15-20%
  - There will be two short (3-5 page) research papers

Schedule

1. Overview of operating systems
2. System calls and OS structure
3. Processes/threads/synchronization
4. Memory management
5. Disks
6. File systems
7. Security
8. Advanced topics

What is an Operating System?

- An operating system (OS) is:
  - a software layer to abstract away and manage details of hardware resources
  - a set of utilities to simplify application development



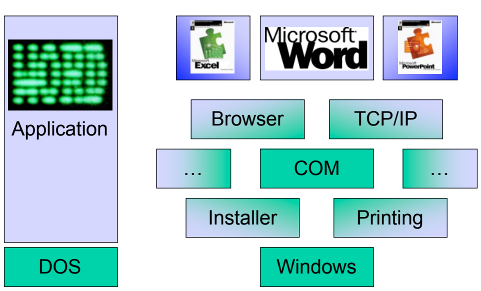
  - “all the code you didn’t write” in order to implement your application

9/3/07 10

What is Windows?

9/3/07 11

What is Windows?



9/3/07 12

The OS and hardware

- An OS mediates programs' access to hardware resources
  - Computation (CPU)
  - Volatile storage (memory) and persistent storage (disk, etc.)
  - Network communications (TCP/IP stacks, ethernet cards, etc.)
  - Input/output devices (keyboard, display, sound card, etc.)
- The OS abstracts hardware into logical resources and well-defined interfaces to those resources
  - processes (CPU, memory)
  - files (disk)
    - programs (sequences of instructions)
  - sockets (network)

Why bother with an OS?

- Application benefits
  - programming simplicity
    - see high-level abstractions (files) instead of low-level hardware details (device registers)
    - abstractions are reusable across many programs
  - portability (across machine configurations or architectures)
    - device independence: 3Com card or Intel card?
- User benefits
  - safety
    - program "sees" own virtual machine, thinks it owns computer
    - OS protects programs from each other
    - OS fairly multiplexes resources across programs
  - efficiency (cost and speed)
    - share one computer across many users
    - concurrent execution of multiple programs

What Functionality belongs in OS?

- No single right answer
  - Desired functionality depends on outside factors
  - OS must adapt to both user expectations and technology changes
    - Change abstractions provided to users
    - Change algorithms to implement those abstractions
    - Change low-level implementation to deal with hardware
- Current operating systems driven by evolution

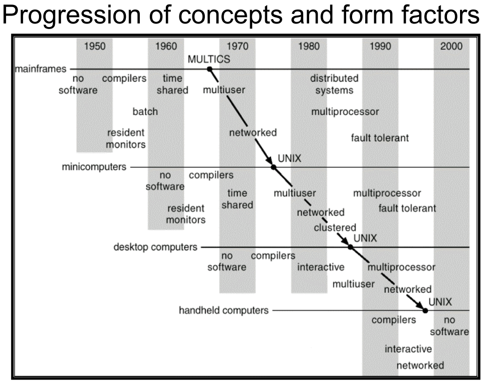
The major OS issues

- **structure**: how is the OS organized?
- **sharing**: how are resources shared across users?
- **naming**: how are resources named (by users or programs)?
- **security**: how is the integrity of the OS and its resources ensured?
  - **protection**: how is one user/program protected from another?
- **performance**: how do we make it all go fast?
- **reliability**: what happens if something goes wrong (either with hardware or with a program)?
- **extensibility**: can we add new features?
- **communication**: how do programs exchange information, including across a network?

More OS issues…

- **concurrency**: how are parallel activities (computation and I/O) created and controlled?
- **scale**: what happens as demands or resources increase?
- **persistence**: how do you make data last longer than program executions?
- **distribution**: how do multiple computers interact with each other?
- **accounting**: how do we keep track of resource usage, and perhaps charge for it?

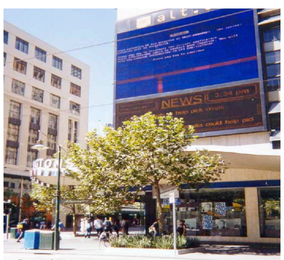
Progression of concepts and form factors



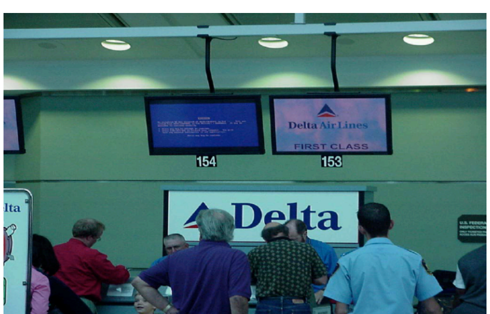
Why is this material critical?

- Concurrency
  - Therac-25, Ariane 5 rocket (June 96)
- Communication
  - Air Traffic Control System
- Virtual Memory
  - Blue Screens of Death
- Security
  - Credit card data

Where's the OS? Melbourne



Where's the OS? Mesquite, TX



Multiple trends at work

- "Ontogeny recapitulates phylogeny"
  - Ernst Haeckel (1834-1919)
    - ("always quotable, even when wrong")
- "Those who cannot remember the past are condemned to repeat it"
  - George Santayana (1863-1952)
- But new problems arise, and old problems re-define themselves
  - The evolution of PCs recapitulated the evolution of minicomputers, which had recapitulated the evolution of mainframes
  - But the ubiquity of PCs re-defined the issues in protection and security

Protection and security as an example

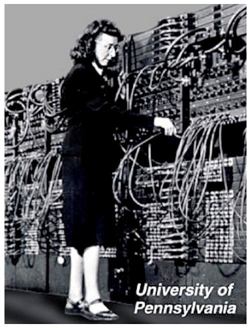
- none
- OS from my program
- your program from my program
- my program from my program
- access by intruding individuals
- access by intruding programs
- denial of service
- distributed denial of service
- spoofing
- spam
- worms
- viruses
- stuff you download and run knowingly (bugs, trojan horses)
- stuff you download and run unknowingly (cookies, spyware)

History of the OS

- Two distinct phases of history
  - Phase 1: Computers are expensive
    - Goal: Use computer's time efficiently
    - Maximize throughput (I.e., jobs per second)
    - Maximize utilization (I.e., percentage busy)
  - Phase 2: Computers are inexpensive
    - Goal: Use people's time efficiently
    - Minimize response time

OS history

- In the very beginning…
  - OS was just a library of code that you linked into your program; programs were loaded in their entirety into memory, and executed
  - interfaces were literally switches and blinking lights
  - Programming done by connecting wires to plugs
- Not much need for an OS

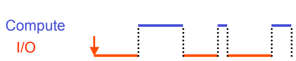


First commercial systems

- 1950s Hardware
  - Enormous, expensive, and slow
  - Input/Output: Punch cards and line printers
- Goal of OS
  - Get the hardware working
  - Single operator/programmer/user runs and debugs interactively
- OS Functionality
  - Standard library only (no sharing or coordination of resources)
  - Monitor that is always resident; transfer control to programs
- Advantages
  - Worked and allowed interactive debugging
- Problems
  - Inefficient use of hardware (throughput and utilization)

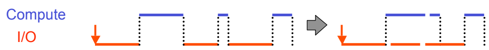
Batch Processing

- Goal of OS: Better throughput and utilization
- Batch: Group of jobs submitted together
  - Operator collects jobs; orders efficiently; runs one at a time
- Advantages
  - Amortize setup costs over many jobs
  - Operator more skilled at loading tapes
  - Keep machine busy while programmer thinks
  - Improves throughput and utilization
- Problems
  - User must wait until batch is done for results
  - Machine idle when job is reading from cards and writing to printers



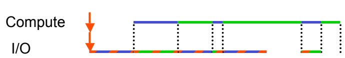
Spooling

- Hardware
  - Mechanical I/O devices much slower than CPU
  - Read 17 cards/sec vs. execute 1000s instructions/sec
  - Disks were much faster than card readers and printers
- Problem
  - Machine idle when job waits for I/O to/from disk
- Goal of OS
  - Improve performance by overlapping I/O with CPU execution
- Spooling: Simultaneous Peripheral Operations On-Line
  1. Read card punches to disk
  2. Compute (while reading and writing to disk)
  3. Write output from disk to printer
- OS Functionality
  - Buffering and interrupt handling
  - Choose which job to run next



Multiprogrammed Batch Systems

- Observation: Spooling provides pool of ready jobs
- Goal of OS
  - Improve performance by always running a job
  - Keep multiple jobs resident in memory
  - When job waits for disk I/O, OS switches to another job
- OS Functionality
  - Job scheduling policies
  - Memory management and protection
- Hardware: asynchronous I/O devices
  - Need some way to know when devices are done
    - interrupts
    - polling
- Advantage: Improves throughput and utilization
- Disadvantage: Machine not interactive



Inexpensive Peripherals

- 1960s Hardware
  - Expensive mainframes, but inexpensive keyboards and monitors
    - multiple terminals into one machine
  - Enables text editors and interactive debuggers
- Problems
  - Programmer productivity
- Goal of OS
  - Improve user's response time
- OS Functionality
  - Time-sharing: switch between jobs to give appearance of dedicated machine each user has illusion of entire machine to him/herself
    - divide CPU equally among the users
    - if job is truly interactive (e.g. editor), then can jump between programs and users faster than users can generate load
  - Concurrency control and synchronization
- Advantage
  - Users easily submit jobs and get immediate feedback

Inexpensive Personal Computers

- 1980s Hardware
  - Entire machine is inexpensive
  - One dedicated machine per user
- Goal of OS
  - Give user control over machine
- OS Functionality
  - Abstract the hardware
  - Remove: time-sharing of jobs, protection, and virtual memory
- Advantages
  - Simplicity
  - Works with little main memory
  - Machine is all your own (performance is predictable)
- Disadvantages
  - No time-sharing or protection between jobs

9/3/07 © 2004-2007 Ed Lazowska, Hank Levy, Andrea and Remzi Arpaci-Dussea, Michael Swift 31

Inexpensive, Powerful Computers

- 1990s+ Hardware
  - PCs with increasing computation and storage
  - Users connected to the web
- Goal of OS
  - Allow single user to run several applications simultaneously
  - Provide security from malicious attacks
  - Efficiently support web servers
- OS Functionality
  - Add back time-sharing, protection, and virtual memory
  - New security problems:
    - Protecting people from code

9/3/07 © 2004-2007 Ed Lazowska, Hank Levy, Andrea and Remzi Arpaci-Dussea, Michael Swift 32

Current Systems

- Conclusion: OS changes due to both hardware and users
- Current trends
  - Multiprocessors
  - Networked systems
  - Virtual machines
- OS code base is large
  - Millions of lines of code (118 million for Vista)
  - 1000 person-years of work (5000 programmers for Vista)
- Code is complex and poorly understood
  - System outlives any of its builders
  - System will always contain bugs
  - Behavior is hard to predict, tuning is done by guessing

9/3/07 © 2004-2007 Ed Lazowska, Hank Levy, Andrea and Remzi Arpaci-Dussea, Michael Swift 33

Other Types of OS

- Distributed OS
  - distributed systems to facilitate use of geographically distributed resources
    - workstations on a LAN
    - servers across the Internet
  - supports communications between jobs
- Parallel OS
  - Some applications can be written as multiple parallel threads or processes
  - can speed up the execution by running multiple threads/processes simultaneously on multiple CPUs
  - need OS and language primitives for dividing program into multiple parallel activities
  - need OS primitives for fast communication between activities
    - degree of speedup dictated by communication/computation ratio

Other types of OS

- Embedded OS
  - Pervasive computing
    - cheap processors embedded everywhere
      - cell phones, PDAs, games, iPod, network computers, …
  - Typically very constrained hardware resources
    - slow processors, little memory (8 KB - 1 MB)
- Real-time OS
  - Device control
    - Cars, planes, space shuttles
  - Must be dependable
    - A crash can cost lives
  - Must hit deadlines
    - Airplane must respond to pilot

CS 537

- In this class we will learn:
  - what are the major components of most OS's?
  - how are the components structured?
  - what are the most important (common?) interfaces?
  - what policies are typically used in an OS?
  - what algorithms are used to implement policies?
- Philosophy
  - you may not ever build an OS
  - but as a computer scientist or computer engineer you need to understand the foundations
  - most importantly, operating systems exemplify the sorts of engineering design tradeoffs that you'll need to make throughout your careers – compromises among and within cost, performance, functionality, complexity, schedule …