CS350 Operating Systems

CS 350 Introduction to Operating Systems
Instructor: Dr. Nael Abu-Ghazaleh
CS Dept., SUNY–Binghamton Spring 2008

• Agenda for today:
  – Go over syllabus: Policies, Grading, Odds and Ends
  – Introduction/Motivation
Projects

• Project are challenging – Please take them seriously
  – Need to pick up or brush up on C++
    * First project will help you do that
  – Need to work in a unix environment
    * unix help sessions next week
  – Big piece of software, many files, can be intimidating
    * Help sessions with each project
    * Each project includes a walk-through, where you read the code and answer questions
    * Some support material on class webpage

• What’s in it for you?
  – “Example isn’t a way of teaching, it is the only way of teaching” – Einstein
  – “I hear and I forget, I see and I remember, I do and I understand” – Chinese proverb
A couple of other points of emphasis

• I will expect you to understand
  – Some midterm and homework questions will push you. Examples:
    ∗ Apply what you learned in a new situation
    ∗ Relate different concepts to each other
    ∗ Evaluate new ideas, compare alternatives under specific conditions
  – Ask Questions, come see me if you dont understand. Stop me in class if I am going too fast. Keep up with the reading; cramming will not work

• Class participation (5%) can make a difference in your grade
What is an Operating System?

• An OS is **not**:
  – An OS is not a command interpreter
  – An OS is not a library of commands
  – An OS is not a set of utilities
  – An OS is not a language or compiler

• What is an OS?
What is an operating system?

“A wizard that gives the illusion of infinite CPUs, infinite memory, single worldwide computing, etc...”

“Government for the computer: controls the operation of the user programs and the I/O devices”

“A skilled beautician: hides the ugly low-level system details. Makes it easy for the average user to get more out of his/her computer”
What is an OS?

- **(A User-Computer Interface):**
  - Abstracts away the complexities of the hardware, making it convenient to use it

- **(A Resource Manager):**
  - Manages the computer resources so that they can be shared, safely, fairly and efficiently

- **A Control Program:**
  - Controls execution of programs to prevent errors and abuse
What is an OS?

- What belongs in the OS?
- Anyone install Windows or Linux lately? How big?
- What is the difference between a user program and the OS; they are both software, right?
What makes up an OS?

- No universally accepted definition
  - Everything your vendor ships when you order an OS is a reasonable approximation
    ∗ varies widely
  - “The one program always running on the computer” is the OS kernel: everything else is either a system program or application program
Why Study Operating Systems?

- Bring together hardware (e.g., logic, assembly and computer organization) and software (e.g., data structures, algorithms, programming languages, etc.)
  - OS is Critical to designing and implementing “good” programs (efficient, maintainable, etc.)
  - Important for designing and implementing “good” hardware

- Understand how computers work: this course is an expansive “look under the hood”

- Get exposed to some fundamental computer science problems (e.g., synchronization, mutual exclusion, scheduling, deadlock) and practical solutions to them

- Understand System Design Tradeoffs (simplicity vs. performance; putting functionality in hw vs. sw)
The Evolution of Operating Systems

- We look at this to provide insight to why we need an OS and what an OS does
- Staggering changes in the underlying system:
  - fast expanding capabilities (processor speed/organization, memory and storage capacities, etc.)
  - Much lower cost
  - Personal computing, Networking, WWW, mobile computing
- Techniques have to vary over time and adapt to changing tradeoffs
- what makes sense now may not make sense later
Hardware Expensive; People Cheap

• First Generation (1949–1956)
  – user-at-console. Pre-OS: user does everything
  – a common set of routines that were repeatedly used emerged (for example, to read the program from the card reader and to manage the printer for output). A seed of an OS?

• Computer costs millions of $’s, optimize for more efficient use of computer

• User-at-console not efficient:
  – Time has to be scheduled statically for users (say in 1 hour slots) – what if user finishes early or needs more time?
  – Long set-up time to place cards (for the program, compiler and OS routines) in the card reader

• Can we do better?
Second Gen.: Batch Processing (1956–1963)

- Operator batch queues jobs (e.g., in card reader)
- A *resident monitor* program stays always in memory
  - Control starts in monitor
  - Monitor loads a new program and transfers control to it
  - User program branches back to monitor when its done
- How does this solve the two problems with user-at-console? (scheduling and set-up time)
- Special cards (control cards) tell the monitor how to run the program
Off-Line Processing

- Problem: I/O very slow. CPU forced to wait a long time

- Off-line processing use faster magnetic tapes to buffer input and output and free the CPU

- Spooling: Overlap the I/O of one job with computation of another
  - One Job Executes
  - OS Reads the cards for the next job onto the disk
  - Prints out the results of the previous job from disk
Other Problems? Of course :-) 

- Small jobs delayed by large jobs

- What if program is buggy and:
  - enters an infinite loop
  - overwrites the monitor
  - misuses the printer causing it to lock-up

- More capabilities are needed from the OS (can you think of any?)

- Monitor
  - job1
  - job2
  - job3
  - job4

- Problem: I/O much slower than CPU. CPU idles
- Idea: have a pool of ready jobs in memory, switch to one when another needs I/O
- I/O in background (using polling, interrupts, or Direct Memory Access (DMA))
- Problems/capabilities needed/other features to be supported?
OS Features needed for Batch Multiprogramming

- I/O routine supplied by system (since it is shared by multiple jobs now)
- Memory Management: system must allocate the memory among the different jobs
- CPU scheduling: system must decide which job to run next
- Device allocation (what devices to allocate to what job)
- OS is getting more complex (OS 360 was released with 1000 bugs)
Time Sharing Systems (1960’s –)

- It is nice to know whether your program is buggy immediately (not come back next day to see the printout)
- Computers are no longer exclusively number crunchers – can you imagine running a word-processor in a batch system?
- Time sharing, allow multiple users to interactively use a computer – commands to the operating system are entered through the keyboard, and responses can be immediate
- User returns to own machine
Time Sharing Operating Systems

- The CPU is multiplexed among several jobs in memory (and on disk)
- timer wakes OS up and allows it to decide who gets next time quota
- A job is swapped into memory from disk or out of memory onto disk
- When a job is swapped in, the resources it needs must be available (enough memory, required I/O devices, etc)
- Interactive communication is provided to user (keyboard/monitor)
- How is this different that Batch Multiprocessing?
- Batch multiprogramming – maximize throughput; Time Sharing – minimize response time
- What new functions must the OS be able to support?
Modern Systems

- Most modern OS’ are time sharing
  - Personal computers generally have a single user, do they need time-sharing?
- We’ve come full circle: Computers Cheap, Humans Expensive
  - Anyone here not have a PC? more than one PC?
  - We already made a sacrifice in favor of humans
- New uses for computers: Networking, Multiple CPUs, WWW, distributed computing, mobile computing, pervasive computing
- Parallel Computing
- Small footprint OS
- OS’ are complex (small OS – 100k lines) requiring 10s to 100s of people years to develop
- Understand OS’ to modularize and simplify them
What to take from this overview

- What type of requirements from the OS in a time-shared system
- Necessity to overlap I/O and computation for efficiency
- Maximizing user-level objectives rather than absolute efficiency
Where We are Heading

- Computer System and OS Review
- Processes and Process Management
- Threads
- CPU scheduling
- Concurrency, Mutual Exclusion and Synchronization
- Deadlock and Starvation
- Memory Management and Virtual Memory
- File Management
- I/O Management
- Advanced Topics: Distributed Operating Systems; Security; Networking; other
A tour of Computer Organization

- A Computer System consists of:
  - One or more Processors: the workhorse(s)
  - Memory: holds instructions and data
  - I/O Devices: interface with the world

- Bootstrap program loaded at powerup (from ROM or EEPROM)
  - Initializes the system
  - Loads the OS and starts execution
Computer System Operation

- I/O devices and the CPU can execute concurrently.
- Each device controller is in charge of a particular device type.
- Each device controller has a local buffer.
- CPU moves data from/to main memory to/from local buffers.
- I/O is from the device to local buffer of controller.
- Device controller informs CPU that it has finished its operation by causing an interrupt.
Processor Overview

• The processor consists of:
  – Arithematic Logic Unit (the number crunccher)
  – General Purpose Registers (data/address)
  – Control Registers (PC, IR, etc.)
  – Control unit

• Your program is compiled into a machine language program

• The processor executes a sequence of machine instructions

• Instructions are encoded in the machine specific instruction format
Processor Overview (contd)

- Fetch – Decode – Execute cycle:
  - fetch an instruction from memory (at location pointed to by program counter) into the instruction register
  - Decode the instruction
  - execute the instruction
  - Repeat until the program halts

- The program counter is automatically incremented to point to the next instruction

- Types of instructions:
  - Data processing: perform an arithmetic or logical operation
  - Loads and stores
  - Control instructions: modify the value of the program counter (branches and jumps)
  - I/O operations (these are usually privileged to the OS; sometimes memory mapped I/O)

- What happens on a procedure call? (the context of a process)