Lecture 1: Overview and Introduction

Prof. Alan Mislove  (amislove@ccs.neu.edu)
What is an Operating System?
What is an Operating System?

- A program that acts as an intermediary between a user of a computer and the computer hardware
What is an Operating System?

• A program that acts as an intermediary between a user of a computer and the computer hardware
What is an Operating System?

- A program that acts as an intermediary between a user of a computer and the computer hardware

- Operating system goals:
  - Execute user programs and make solving user problems easier
  - Make the computer system convenient to use
  - Use the computer hardware in an efficient manner
What is an Operating System?

• A program that acts as an intermediary between a user of a computer and the computer hardware

• Operating system goals:
  • Execute user programs and make solving user problems easier
  • Make the computer system convenient to use
  • Use the computer hardware in an efficient manner

• Rest of this lecture:
  • Background
  • Overview of topics we’ll cover in this half of the course
Four Components of a Computer System

- User 1
- User 2
- User 3
- ... User n

- Compiler
- Assembler
- Text Editor
- ... Database System

- System and Application Programs

- Operating System

- Computer Hardware
Why study operating systems?

· Maturing field
  · Most people only use one OS

· But, lots of new technology
  · High performance computing
  · Distributed computing
  · Cloud computing
  · Fault-tolerant computing

· OS issues faced in many places
  · Security, protection, resource management

· New devices need OSes
  · iPhones, iPads, (Galaxy Tabs)
A brief history

- **Beginning:** One user and one program per computer
  - No overlap of computation or I/O
  - OS was just a set of libraries

- But, want computers to do work for more than one user
  - Statistical multiplexing
  - OS loaded jobs, ran jobs, cleaned up

- Multiprogramming was next big step forward
  - Run multiple jobs at once, but much more complex

- Then, timesharing systems took off
  - Each user gets “slices” of time, appears like their own machine

- Now, distributed/cloud operating systems
Outline

• OS Definition

• OS interface with subsystems
  • I/O systems
  • Storage systems

• Basic functionality
  • Protecting resources
  • Managing resources
    • Processes
    • Memory
    • Storage
What do Operating Systems do?
What do Operating Systems do?

• Depends on the point of view
• Users want convenience, ease of use
  • Don’t care about resource utilization
• But shared computer such as mainframe or minicomputer must keep all users happy
• Users of dedicated systems such as workstations have dedicated resources but frequently use shared resources from servers
• Handheld computers are resource poor, optimized for usability and battery life
• Some computers have little or no user interface, such as embedded computers in devices and automobiles
Operating System Definition

• OS is a **resource allocator**
  • Manages all resources
  • Decides between conflicting requests for efficient and fair resource use

• OS is a **control program**
  • Controls execution of programs to prevent errors and improper use of the computer
Operating System Definition (Cont.)

• But, no universally accepted definition

• “Everything a vendor ships when you order an operating system” is good approximation
  • But varies wildly (IE, anyone?)

• “The one program running at all times on the computer” is the **kernel**. Everything else is either a system program (ships with the operating system) or an application program.
Computer Startup

• **bootstrap program** is loaded at power-up or reboot
  - Typically stored in ROM or EPROM, generally known as **firmware**
  - Initializes all aspects of system
  - Loads operating system kernel and starts execution

• Will not focus on bootstrapping in this course (take CS5600)
Outline

- OS Definition

- OS interface with subsystems
  - I/O systems
  - Storage systems

- Basic functionality
  - Protecting resources
  - Managing resources
    - Processes
    - Memory
    - Storage
Computer System Organization

- Computer-system operation
  - One or more CPUs, device controllers connect through common bus providing access to shared memory
  - Concurrent execution of CPUs and devices competing for memory cycles

![Diagram of computer system organization]

- Diagram includes components such as CPU, disk controller, USB controller, memory, and peripherals like disks, mouse, keyboard, printer, and monitor.

Based on slides by Silbershatz, Galvin, and Gagne
How do the devices communicate?

- 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
Common Functions of Interrupts

- Interrupt transfers control to the interrupt service routine generally, through the **interrupt vector**, which contains the addresses of all the service routines.

- Interrupt architecture must save the address of the interrupted instruction.

- Incoming interrupts are *disabled* while another interrupt is being processed to prevent a *lost interrupt*.

- A *trap* is a software-generated interrupt caused either by an error or a user request.

- An operating system is **interrupt driven**.
Interrupt Handling

• The operating system preserves the state of the CPU by storing registers and the program counter

• Determines which type of interrupt has occurred:

• Separate segments of code determine what action should be taken for each type of interrupt
Interrupt Timeline

- **CPU**
  - user process executing
  - I/O interrupt processing

- **I/O Device**
  - idle
  - transferring

- I/O request
- Transfer done
- I/O request
- Transfer done
I/O Structure: Two options
I/O Structure: Two options

• After I/O starts, control returns to user program only upon I/O completion
  • Wait instruction idles the CPU until the next interrupt
  • Wait loop (contention for memory access)
  • At most one I/O request is outstanding at a time, no simultaneous I/O processing
I/O Structure: Two options

• After I/O starts, control returns to user program only upon I/O completion
  • Wait instruction idles the CPU until the next interrupt
  • Wait loop (contention for memory access)
  • At most one I/O request is outstanding at a time, no simultaneous I/O processing

• After I/O starts, control returns to user program without waiting for I/O completion
  • **System call** – request to the operating system to allow user to wait for I/O completion
  • **Device-status table** contains entry for each I/O device indicating its type, address, and state
  • Operating system indexes into I/O device table to determine device status and to modify table entry to include interrupt
Storage Structure: Many levels

- Main memory – only large storage media that the CPU can access directly
  - Random access
  - Typically volatile
- Secondary storage – extension of main memory that provides large nonvolatile storage capacity
- Magnetic disks – rigid metal or glass platters covered with magnetic recording material
  - Disk surface is logically divided into tracks, which are subdivided into sectors
- SSDs – solid-state memory disks with no moving parts
Storage Hierarchy

- Storage systems organized in hierarchy
  - Speed
  - Cost
  - Volatility

- **Caching** – copying information into faster storage system; main memory can be viewed as a cache for secondary storage
Storage-Device Hierarchy

- registers
- cache
- main memory
- electronic disk
- magnetic disk
- optical disk
- magnetic tapes
Caching

• Important principle, performed at many levels in a computer (in hardware, operating system, software)

• Information in use copied from slower to faster storage temporarily

• Faster storage (cache) checked first to determine if information is there
  • If it is, information used directly from the cache (fast)
  • If not, data copied to cache and used there

• Cache smaller than storage being cached
  • Cache management important design problem
  • Cache size and replacement policy
Outline

• OS Definition

• OS interface with subsystems
  • I/O systems
  • Storage systems

• Basic functionality
  • Protecting resources
  • Managing resources
    • Processes
    • Memory
    • Storage
How do we protect resources?

- Interrupt driven by hardware
- Software error or request creates **exception** or **trap**
  - Division by zero, request for operating system service
- Other process problems include infinite loop, processes modifying each other or the operating system
- **Dual-mode** operation allows OS to protect itself and other system components
  - **User mode** and **kernel mode**
  - **Mode bit** provided by hardware
    - Provides ability to distinguish when system is running user code or kernel code
    - Some instructions designated as **privileged**, only executable in kernel mode
    - System call changes mode to kernel, return from call resets it to user
How to prevent processes hogging CPU?

- Timer to prevent infinite loop / process hogging resources
  - Set interrupt after specific period
  - Operating system decrements counter
  - When counter zero generate an interrupt
  - Set up before scheduling process to regain control or terminate program that exceeds allotted time
How do we protect a process’s memory?

- Rogue processes could access any part of memory
  - Even parts that are not their’s

- Kernel memory, other processes’ memory, etc

- Many OSes has little protection against such processes
  - Mac OS 9, e.g.

How to protect against this?
- Need to only allow processes to mess with their own memory

Requires user/kernel privilege separation
- Why?
Dividing up memory

- Want many processes to use memory at once
  - But, global addresses -- why is this a problem?

- Many CPU instructions involve jumps
  - To other memory addresses

- Options?
  - Compile for separate locations
  - Patch on load

- Or, assume some hardware support
  - Use position-independent code (jumps relative to PC)
  - Give each process a private view of memory
Processes

- Definition: A process is a program in execution. It is a unit of work within the system. Program is a *passive entity*, process is an *active entity*.
- Process needs resources to accomplish its task
  - CPU, memory, I/O, files
  - Initialization data
- Process termination requires reclaim of any reusable resources
- Single-threaded process has one *program counter* specifying location of next instruction to execute
  - Process executes instructions sequentially, one at a time, until completion
- Multi-threaded process has one program counter per thread
- Typically system has many processes, some user, some operating system running concurrently on one or more CPUs
  - Concurrency by multiplexing the CPUs among the processes / threads