Operating System
COMS W4118
Lecture 2

Alexander Roth

2015 – 01 – 22

1 High-level Overview

1.1 Four Componenets of a Computer System

- Layer system in computer architecture.
- User programs do not need to know about the low level hardware systems of a host.
- The operating system will provide an interface from the hardware to the user programs.
- Provides an API for user programs, so they don't need to know about hardware.

1.2 Operating System Definition

- OS manages all resources.
- Controls execution of programs to prevent errors and improper use of the computer.
- Decides between conflicting requests for efficient and fair resource use.
- The operating system has to juggle tasks.
- Operating system is a master, it controls and dictates.
- It is also a server because it takes commands from user programs and does things to the hardware according to the user program.
- No universally accepted definition.
- "Everything a vendor ships when you order an operating system" is a good approximation.

- "The one program running at all times on the computer" is the **kernel**.
- Everything else is either a system program or an application program.
- We will focus on the behavior of the kernel.
- The kernel is just one particular program that gets run first.
- The kernel runs as long as the computer is active.
- The kernel is a block of code that is sitting in memory that will only operate when the user applications request it to do something.
- The kernel goes through states of inactivity and activity.

1.3 Computer System Organization

- Computer-system operation
  - One or more CPUs, device controllers connect through common bus providing access to shared memory
  - Concurrent execution of CPU's and devices competing for memory cycles.
- Registers are very small pieces of memory that are embedded into CPU's or pseudo-CPU's as typically a few bytes. Very fast memory embedded in processors.
- **Simplified:** CPU can only do operations on registers (not entirely true today).
- Each subsystem in the machine now has their own processing unit, sometimes even more powerful than the actual CPU.

1.4 Common Functions of Interrupts

- **von Neumann Architecture:** We have a memory that contains both program and data. The CPU will go through each instruction in memory (encoding in bytes) and will fetch other memory in order to handle operations.
- The program counter register handles which instruction is fetched by the machine.
- All the CPU does is run the program in memory.
- The CPU will keep executing instructions, when will it stop?
- The CPU runs on a timer, and when the timer runs down, control of the system is transferred to the operating system.

- Happens hundreds of times a second.
- When an timer interrupt comes in, the CPU jumps to a piece of known code within the operating system.
- This creates the illusion of multiple programs running at the same time, when in actuality one program is running at once, while the others are waiting in the background.
- CPU can only execute one thread of byte-code at a time. Then a time interrupt occurs.
- CPU will save the state to somewhere else and execute another program elsewhere.
- This is beneficial to multi-user systems, such as Unix.
- Operating systems have scheduling algorithms that handle which processes are being executed when and with what kind of priority they are given.
- The operating system will preempt programs in order to allow another process to run in the meantime.
- 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.

1.4.1 Interrupts

- Three ways interrupts happen:
  1. Timing interrupt: as described above
  2. All the controllers (I/O devices) have the ability to be interrupted by the disk. Example: The disk interrupt will tell the operating system that the information being read in from the disk is available. Typically scheduling algorithms give priority to I/O events as that is what users interact with. Used to signal completion of tasks or I/O errors.
  3. When a program fails, a software interrupt will occur. Error handling or errors will occur. The programmer will set up memory locations that are invalid, these areas will through interrupts if they are accessed.
- Switching from one program to another is very expensive. **Context switching**, must save state and data from program and then switch to another program and bring up the old state there.

- Exceptions occur from running a piece of code in the kernel that throws an error.
- System call mechanisms also piggy-back on interrupt mechanisms.
- Segmentation fault is a software interrupt.

1.5 Interrupt Timeline

- When a program requests an I/O, it will take quite some time for the CPU.
- You do not want to waste CPU time, but you want to know exactly when it is done, so we use interrupts.

1.6 I/O Structure

Two ways to handle I/O:

1. Call I/O and have the CPU wait. (Old implementation)
2. Operating system will juggle processes so you don't waste CPU.

1.7 Storage-Device Hierarchy

- There's a memory hierarchy that everyone will learn if they take a database course.
    1. Registers
    2. Cache
    3. Main Memory
    4. Solid-state disk
    5. Hard Disk
    6. Optical Disk
    7. Magnetic Tape
- Higher up; smaller, faster memory.
- Lower down; bigger, slower memory.
- *Vector* is the best memory structure ever because the cache makes *linked list* slow as fuck.
- A structure has to be aligned so that it fits into 1024 bytes.
- Making sure that code is easily accessed and easily cacheable is very important to the operating system.

1.8 How a Modern Computer Works

1.8.1 Old Method

- The CPU issues an I/O request.
- The device it was issued to fulfills the request and sends back an interrupt.
- The CPU will transfer the data to a local buffer from the I/O device.
- Takes a long time.

1.8.2 Modern Method

- Direct Memory Access.
- CPU sends request to device.
- Device learns where to send the information and where to get it.
- Device doesn't need to interrupt the CPU.

1.9 Transition from User to Kernel Mode

- Timer to prevent infinite loop/process hogging resources
  - Timer is set to interrupt the computer after some time period.
  - Keep a counter that is decremented by the physical clock.
  - Operating system set the counter (privileged instruction)
  - When counter zero generates an interrupt
  - Set up before scheduling process to regain control or terminate program that exceeds allotted time.
- Dual modes of operation. Privileged operations can only occur when the CPU is in privileged mode (kernel mode).
- What is available to the user are C functions that act as system calls.
- Library code activates these smaller C functions.
- Bottom-level C functions are just wrappers around the assembly code.