INTRODUCTION 1
COMPUTER ORGANIZATION AND THE TIME-SHARING OS

COP4600

Dr. Matthew Gerber

1/13/2016

COMPUTER ORGANIZATION

The Von Neumann Machine

- Stored Programs
- CPU with Control Unit and Arithmetic/Logic Unit
- Memory
- Input and Output

Architectural Variants

- Single-Processor Systems (1.3.1)
  - Systems with one general-purpose CPU
- Multiprocessor Systems (1.3.2)
  - Most *smartphones* are now multiprocessor systems
  - Higher throughput, better economy, (better reliability?)
  - Practically you will probably only encounter symmetric multiprocessing
- Clustered systems (1.3.3)
  - Multiprocessor systems over a network
  - Introduce all sorts of synchronization issues

Working with Stored Programs (1.2.1)

- An initial “boot” program is run as soon as the computer is turned on
  - *The word “boot” is a contraction of “bootstrap” – referring to the idea of “pulling yourself up by your own bootstraps”*
- In a modern context, the boot program’s sole job is to load and start the kernel
- The kernel then does everything else – including loading and starting other programs
- Programs then become processes (more on this later), and execute from (and only from) main memory

Working with Memory (1.2.2)

THE IRON LAWS OF THE MEMORY HIERARCHY:

- Fast memory is expensive
- Cheap memory is slow
- (Volatility doesn't help here either)

Working with Memory (1.2.2)

- As a consequence we use small fast memory and large cheap memory
  - Registers
  - Cache (in multiple levels)
  - Main Memory
  - SSDs
  - Magnetic Disks
- Optical disks and magnetic tapes still have a role here… maybe?

Working with I/O (1.2.3)

- CPUs perform I/O via device controllers connected to a bus
- Various methods of communication
- We will get to these in detail later, but for now a quick overview…
  - Polled
  - Interrupt-driven
  - Direct Memory Access

Interrupts (throughout Chapter 1)

- All modern systems are organized around *interrupts*
- *Traps* used to be a distinct concept from interrupts, but now mean essentially the same thing
- Interrupts immediately stop execution of the running program (i.e., interrupt it) and transfer control to an *interrupt handler*
- The interrupt handler executes, then (usually) returns control to the program that was running

Interrupts (throughout Chapter 1)

- Interrupts can be generated by the computer itself
  - Hardware events
  - Timer events
- Interrupts can also be generated by programs
  - Requests to use system services
- (Basically) everything the operating system does is in response to a interrupt

THE TIME-SHARING OPERATING SYSTEM

The Time-Sharing Operating System (1.4)

- *Multiprogramming* – scheduling more than one program at once – was originally simply to keep computers busy
- The rise of interactive computers made this insufficient
- *Multitasking* (Interactive Time-Sharing) is multiprogramming with switches frequent enough that users can interact with running programs
- Multitasking requires a time-shared operating system
- Originally these were intended to serve multiple users; all modern single-user operating systems now time-share between applications (smartphones were the last holdouts, and Apple gave up in 2010)

So…

- We’ve got more than one program running on the computer now.
- Those programs are written by humans.
- How do we avoid total chaos?

Dual-Mode Operation (1.5)

- We don’t want normal processes to be able to access the hardware or mess with other processes
- *But…* they have to be allowed to run and do normal things
  - You really don’t want to ask the operating system every time you want to do math
- (At least) two modes of operation
  - User Mode
    - Only allowed to access specific instructions and regions of memory
  - Kernel Mode
    - Also known as Privileged Mode, System Mode, Monitor Mode…
    - Allowed to do anything

Dual-Mode Operation (1.5)

- Only the OS is allowed to use kernel mode
- The computer itself needs to support this
  - Otherwise you're relying on good behavior by programs
  - This won't do
- Privileged instructions can only be executed in kernel mode
- Privileged instructions include all hardware access, so…
- The operating system must provide system function calls
  - User mode programs make requests of the operating system
  - The OS, running in kernel mode, fulfills the requests

Dual-Mode Operation (1.5)

- How does this happen?
  - Interrupts!
    - (Properly, these are really *traps*, but again, we stopped caring a while ago)
  - Programs execute arithmetic/logic/allocated-memory instructions at will
  - When they need system services, they raise interrupts
  - OS receives the interrupt, services it, and returns control

Process Management (1.6)

- A program becomes a process when it's loaded into memory and started
- Nothing happens without processes
- Processes need resources to do anything
  - Memory
  - I/O capabilities
  - Take this thought far enough and the CPU is a resource!
    - (We don't usually actually set it up that way.)

Process Management (1.6)

- The operating system must…
  - Load programs into memory to turn them into processes
  - Create, delete, suspend and resume processes
  - Schedule processes and threads to CPUs
  - Assign resources to processes
  - Provide synchronization between processes
  - Provide communication between processes

NEXT TIME: MEMORY, I/O AND EVERYTHING ELSE