INTRODUCTION 2
MEMORY, I/O, AND OPERATING ENVIRONMENTS

COP4600

Dr. Matthew Gerber

1/20/2016

THE MODERN OPERATING SYSTEM, PART II

Memory Management (1.7)

- Main memory is the largest storage device the CPU can directly address
  - There are exceptions here, but all of them are special cases
- Programs have to be in main memory to run
- The operating system must…
  - Keep track of what memory is being used, and by which processes
  - Decide what to move into and out of memory
  - Allocate and de-allocate memory as necessary
  - Make sure processes don’t “color outside the lines” of their memory (more on this later)

Input and Output: Files and Disks (1.8)

- A file is *a named, sequential array of bytes.*
- Almost all data on disks are files
  - There *are* exceptions
  - They're usually esoteric
- The operating system has to manage…
  - Creating, writing, reading and deleting files
  - Creating, writing, reading and deleting directories to organize files
  - Providing functions for processes to work with files
  - Mapping files onto disks
    - (really, partitions, but we'll call them disks for now)

Input and Output: Files and Disks (1.8)

- The operating system *also* has to manage disks themselves:
  - Organizing files on disks
  - Managing free space on disks
  - Allocating space on disks to files
  - Scheduling disks for processes
- By the way, technically a disk is just another I/O device
- They're so foundational we tend to think of them separately

Input and Output: Everything Else

- *Almost* all other devices in a modern operating system are viewed in one of two ways:
  - As a memory space
  - As a file
- ***Can you think of the biggest exception?***
  - (and even it's not totally immune…)

Input and Output: Everything Else

- Each device has a *device driver* that either:
  - Maps it to a file
  - Maps it to memory
  - Maps it to another core OS service (like **networking**)
  - Gives it a unique interface (very rare in a modern OS)

Security (1.9, sort of): Authentication

- *Most* modern operating systems allow more than one user
  - There are prominent exceptions to this
  - Allowing more than one user *at a time* is *not* ubiquitous yet
- Even operating systems that only allow one user usually have the concept of a user identity
- *Authentication* is the process of determining that *you are who you say you are.*
- Allowing users to misidentify themselves leads to all sorts of calamity, whether accidental or malicious

Security (1.9 – sort of): Authentication

- The classic authentication method is the password – something you *know*
  - Less useful than it sounds
  - Good passwords are hard to remember
    - (And by the contrapositive…)
  - Passphrases are (much) better but are not a panacea

Security (1.9 – sort of): Authentication

- Token-based, or classic *two-factor* authentication, adds something you *have*
  - When properly implemented, makes authentication much harder to defeat
  - *Much harder is not the same thing as impossible*
  - “Tokens” that are really computers can be compromised themselves
  - Be smart about what you allow to authenticate what

Security (1.9 – sort of): Authentication

- Biometric authentication adds something you *are*
  - Very difficult to bypass without personalized effort
  - Also very difficult to reliably implement

Security (1.9 – sort of): Authorization

- Once you know who somebody is you still need to know what that means
- *Authorization* is the process of determining, given who you are, *what you should be able to do*
- Certain users are able to perform certain actions, and not others
  - Classic example: “You can only access your own files”
- This needs to be correctly determined *and* successfully enforced for the system to remain secure

Security (1.9 – sort of): Authorization

- Certain users may have the right to perform other actions if they *escalate privileges* first
  - Classic example: UNIX-like systems' "Root access"
    - One method is to actually log in to the root account
    - Another is to explicitly escalate to root privileges temporarily, such as with **sudo**
  - Windows version: "Administrator access"
    - Older versions of Windows allow administrator actions immediately
    - User Account Control (UAC) in newer versions tries to put some brakes on
  - Other systems handle in different ways
  - The text uses **setuid** as an example…

Security (1.9 – sort of): Authorization

- Modern authorization makes heavy use of access control lists and grouping
- Authorization *seems* less dangerous than authentication…
- …but a lot of system-level attacks take place against badly designed authorization systems
  - Ever heard of a “privilege-escalation attack”?

Security (1.9 – sort of): Cryptography

- Practical cryptography is intended to do two things:
  - Allow you to send messages that are very hard for anybody other than your recipient to read
  - Allow you to sign messages in a way that is very hard for anybody else to impersonate
- Operating systems use cryptography for, among other things:
  - Secure storage in the face of potential theft
  - Secure communication over potentially hostile networks
- **Be realistic about the limits of cryptography!**

Putting It Together

- Modern operating systems need:
  - Process Management
  - Memory Management
  - I/O Management
  - Security
- …and how many places does all this show up?

OPERATING ENVIRONMENTS (1.11)

Physical Computers

- Large Systems
  - Initially required to centralize very scarce computing resources
  - Built around time-sharing; job-oriented more than interactivity-oriented
  - Decreasingly common, but not dead yet
- Desktop/Laptop Computing
  - What most people think of as “operating systems” at this point
  - Interrupt-driven, multitasking, etc.
  - Usually supports multiple users, may or may not support multiple at once

Physical Computers

- Mobile Computing
  - Laptops and tablets
  - Have a lot of the characteristics of desktop operating systems
  - Tend to be optimized for information consumption
- Real-Time Operating Systems
  - When things have to be done *now*
  - Popular for control systems
    - You really don’t want your X-Ray machine to decide to garbage collect at random times
  - Can (and do) show up as subsets of other systems

Virtual Computers

- *Emulation* is running code for another kind of computer
- *Interpretation* is running code for a theoretical computer
  - To note, calling Java “interpreted” is a stretch
- *Virtualization* is running a computer on a computer
  - Virtualization generally requires specialized hardware support…
  - …but almost all modern desktop CPUs have it

Networked Computers

- Client-Server Systems
  - Networked systems that allow a *client* computer to ask a *server* to do something on its behalf
  - Easy example: Web servers
  - More complicated example: Database servers
  - Still more complicated example: Game servers

Networked Computers

- Peer-to-Peer Systems
  - Actually a lot like client-server computing, except at any time, any computer can be a client, a server or both
  - Computers have to be able to discover each other, either via a central directory or exchanging information over the network
  - Famous for piracy, but there are plenty of other applications
  - There *are* reasons other than piracy that network administrators don't like it

Networked Computers

- Cloud Computing
  - Broadly constructed client-server systems running over the commodity Internet or private networks
  - Can include file servers, computation servers, virtual machines, etc.
  - Not a new technology so much as a radically scaled application of existing technologies
  - The innovation is mostly in…
    - …the scale itself, and
    - …the ability to usefully control client-server computing *at* that scale

NEXT TIME: PROCESSES AND THREADS