Lesson 06: Basics of Software Development (W02D2)

Balboa High School

Michael Ferraro

<mferraro@balstaff.org>

Do Now

1. What is the main reason why flash drives are able to retrieve and return data more quickly than traditional hard drives?

2. From memory, write the three commands we used when writing the `HelloWorld` program. The three commands are related to the process of
   (a) edit,
   (b) compile, &
   (c) run.

Aim

Students will finish the overview of computer basics and learn about software development and its history. Students will receive textbooks and Problem Set #1.

Do Now Solutions

1. What is the main reason why flash drives are able to retrieve and return data more quickly than traditional hard drives?

Do Now Solutions

1. What is the main reason why flash drives are able to retrieve and return data more quickly than traditional hard drives?

   $\rightarrow$ RAM (NVRAM, in this case) is *solid state* — no moving parts!

Do Now Solutions

1. What is the main reason why flash drives are able to retrieve and return data more quickly than traditional hard drives?

   $\rightarrow$ RAM (NVRAM, in this case) is *solid state* — no moving parts!

2. From memory, write the three commands we used when writing the `HelloWorld` program. The three commands are related to the process of (a) edit, (b) compile, and (c) run.

Do Now Solutions

1. What is the main reason why flash drives are able to retrieve and return data more quickly than traditional hard drives?

   $\rightarrow$ RAM (NVRAM, in this case) is *solid state* — no moving parts!

2. From memory, write the three commands we used when writing the `HelloWorld` program. The three commands are related to the process of (a) edit, (b) compile, and (c) run.

   (a) edit: `gedit HelloWorld.java`

Do Now Solutions

1. What is the main reason why flash drives are able to retrieve and return data more quickly than traditional hard drives?

   $\rightarrow$ RAM (NVRAM, in this case) is *solid state* — no moving parts!

2. From memory, write the three commands we used when writing the `HelloWorld` program. The three commands are related to the process of (a) edit, (b) compile, and (c) run.

   (a) edit: `gedit HelloWorld.java`

   (b) compile: `javac HelloWorld.java`

Do Now Solutions

1. What is the main reason why flash drives are able to retrieve and return data more quickly than traditional hard drives?

   $\rightarrow$ RAM (NVRAM, in this case) is *solid state* — no moving parts!

2. From memory, write the three commands we used when writing the `HelloWorld` program. The three commands are related to the process of (a) edit, (b) compile, and (c) run.

   (a) edit: `gedit HelloWorld.java`
   (b) compile: `javac HelloWorld.java`
   (c) run: `java HelloWorld`

Act I

Computer Basics, Continued

Peripheral Devices

- Peripherals are the non-core parts of a computer, not necessary for the computer to run (but might be necessary for usability!)
- Examples: Keyboard, Printer, Scanner, CD-ROM reader

ROM $\neq$ RAM!

- Unlike RAM, ROM [Read-Only Memory] contents can't be changed (well, you can "flash" some ROM chips to change their contents, so-called firmware updates)
- Useful for storing instructions that need always be present, consistent
- Present on motherboards as BIOS chips, on MP3 players to hold their very slim operating systems.

The Boot Process: ROM → HD → RAM

- System BIOS [Basic Input/Output System] has knowledge of how to find and communicate with hard disks; can find the **boot sector** of a hard disk

The Boot Process: ROM → HD → RAM

- System BIOS [Basic Input/Output System] has knowledge of how to find and communicate with hard disks; can find the **boot sector** of a hard disk
- Boot sector of hard disk contains information about how to load the operating system, whose files are written on the hard disk; referred to as the "boot loader," examples include GRUB, LILO, and the anonymous Windows boot loader

The Boot Process: ROM → HD → RAM

- System BIOS [Basic Input/Output System] has knowledge of how to find and communicate with hard disks; can find the **boot sector** of a hard disk
- Boot sector of hard disk contains information about how to load the operating system, whose files are written on the hard disk; referred to as the "boot loader," examples include GRUB, LILO, and the anonymous Windows boot loader
- Operating system (OS) starts

The Boot Process: ROM → HD → RAM

- System BIOS [Basic Input/Output System] has knowledge of how to find and communicate with hard disks; can find the **boot sector** of a hard disk
- Boot sector of hard disk contains information about how to load the operating system, whose files are written on the hard disk; referred to as the "boot loader," examples include GRUB, LILO, and the anonymous Windows boot loader
- Operating system (OS) starts
- OS loads instructions from the various programs it starts into RAM, where the CPU has fast access to them

OS Examples

- Examples:
  - Modern OSes: MS Windows { 10, 8, 7, Vista, & XP}, Ubuntu Linux [a], Mac OS/X, Android, Solaris[b]
  - Older OSes: MS-DOS, SCO Unix, MS Windows 3.1[c]

---

[a] The `apcs02` server runs the Ubuntu flavor of Linux

[b] Was *Sun* Solaris, now owned by Oracle — just like Java

[c] Early versions of Windows still required DOS "beneath" them!

Two Key Roles of an OS

A full[a] OS serves two key functions:

1. Act as a "**program**" **capable of running other programs**, making sure they play nice together

2. **Hide the details of the hardware it runs on** from the programs; i.e., hide the differences between communicating with different brands' network cards so the programs using the network connection don't have to worry about dealing with every kind of network card in existence — this task is left to the writers of drivers; this idea is called *abstraction* (more later!)

---

[a] Non-full OSes would be very slim OSes that run on specialized hardware to perform a very limited number of functions, like a simple MP3 player.

A few more details. . .

Functions of an OS (not all-inclusive!)

- load **drivers**, or software modules enabling the OS to communicate with peripherals so that **applications** can take advantage of devices like printers and USB drives; can be written by different people after OS is written

A few more details...

Functions of an OS (not all-inclusive!)

- load **drivers**, or software modules enabling the OS to communicate with peripherals so that **applications** can take advantage of devices like printers and USB drives; can be written by different people after OS is written
- provide services like file system creation and maintenance, user management (accounts and security privileges), process management (time-sharing between concurrently running programs)

A few more details...

Functions of an OS (not all-inclusive!)

- load **drivers**, or software modules enabling the OS to communicate with peripherals so that **applications** can take advantage of devices like printers and USB drives; can be written by different people after OS is written
- provide services like file system creation and maintenance, user management (accounts and security privileges), process management (time-sharing between concurrently running programs)
- provide environment within which applications can run
  $\rightarrow$ **application are that which you will write, programs that perform specific functions**

BIOS-OS-Apps Hierarchy

- Systems are typically designed in layers; e.g., computer networks (like ethernet adapters and switches), transport mechanisms (software, like TCP), networked applications (like a web browser)
- Each layer interacts with the ones directly above and below
- Layers can sometimes be replaced with similar so long as the replacement interacts with the other layers in ways the other layers expect; e.g., replace Windows XP (OS layer) with Debian Linux and system can still work!

Files and Directories

- OS can communicate with a hard disk to read and write 0s and 1s
- OS creates strings of binary values that represent folders (directories) and files that "live" within those containing structures
- Files and Directories are, at a logical level, trees; physically, they're stored as clusters of adjacent bytes pointing to other clusters, forming chains of clusters we call files — see here

Files and Directories

- OS can communicate with a hard disk to read and write 0s and 1s
- OS creates strings of binary values that represent folders (directories) and files that "live" within those containing structures
- Files and Directories are, at a logical level, trees; physically, they're stored as clusters of adjacent bytes pointing to other clusters, forming chains of clusters we call files — see here
- *What would happen if there was some low-level corruption on a hard disk, and some 0s and 1s were flipped? What if damage is where. . .*
  - *a pointer to another cluster is?*
  - *elsewhere?*

Other Topics (in the reading)

- Shells (GUI vs traditional)
- Console apps (e.g.: programs that run at the "command prompt")
- Algorithm (covered in class later)

Act II

Software Development

Early Computers (1940s - 1960s)

- Physically large – size of entire rooms
- Size largely due to
  - lack of transistors – vacuum tubes used instead; very HOT and couldn't be miniaturized
  - lack of microfabrication techniques (parts hand-built!)
  - "software" didn't exist as we know it; a function was hard-coded (wired)

Moving Past Hard-Wired Programs

- John Von Neumann (ca. 1946) conceived of programs stored in memory with machine-specific codes
- Programmers used punch cards that were fed to card readers
- Punch cards held software, just as a hard drive holds your source code files
- Software development on cards must have been incredibly slow... (Mr. Binkowski used these!)

Stages of Software Development

- Discussed in §2.1 of the Litvin text
- You are responsible for understanding the process at a high level for now
- You'll gain first-hand experience with these roles later during projects

History of Programming Languages

- Punch cards that specified numerical CPU instructions
- Assembly language replaced numerical instructions with statements like `JMP` (jump), `CALL`, & `INC` (increment); An *assembler* program converts assembly language to machine code that a CPU can run
- High-level programming languages like Java: No longer would you have to manipulate data at a very low level (*this byte goes here, this byte goes there. . .*) — you could write high-level procedures[a] and use library functions[b] that language designers make available to you

[a] Or *methods* in Java

[b] E.g., `System.out.println()` has been provided for you to print lines on the screen.

Enter the High-Level Languages

- See http://www.levenez.com/lang/ for a great "family tree" of programming languages. BTW, is JavaScript derived from Java?
- Notable languages
  - FORTRAN (1954)
  - COBOL (1959)
  - BASIC (1964)
  - PASCAL (1970)
  - C (1970s)
  - C++ (early 1980s)
  - **Java (1995)** ← our focus
  - C#, Ruby, PHP, PERL

Edit-Compile-(Link)-Run Cycle

- First step in writing software: Create source code in a text **edit**or
- Run source code through a **compile**r
- Use a **link**er to "connect" compiled source with system libraries (*static* vs *dynamic* binaries)[a]
- Compiled/linked product can be **run** on the host machine's CPU

This process is slightly different across different languages!

[a] We'll rarely refer to this step in our work with Java.

Interpreted vs. Compiled S/W

- C — write source code and compile/link into executable file that runs on specific system
- PERL and JavaScript — write source and run with interpreter anywhere (since code isn't compiled behorehand, it doesn't have the same performace as compiled code)
- Java — a hybrid approach whereby compiled *bytecode* can be run by a *JVM* (Java Virtual Machine) on different kinds of systems; `java.exe` is the JVM for PCs

Important Textbook Reading

Make sure that you pay special attention to the following as you read the textbook while completing PS #1:

- §2.2: Read the short blurb about *syntax and logic errors*
- §2.3: What is *reusable code*?

HW

Get started on PS #1: §§1-2 are due by next class. (That's §1 & §2, inclusive!)

Note: For the required reading, you'll need to find the Chapter 1 PDF file using the link in the problem set.