Introduction to Scientific Computing II

Lecture 3

Amir Farbin

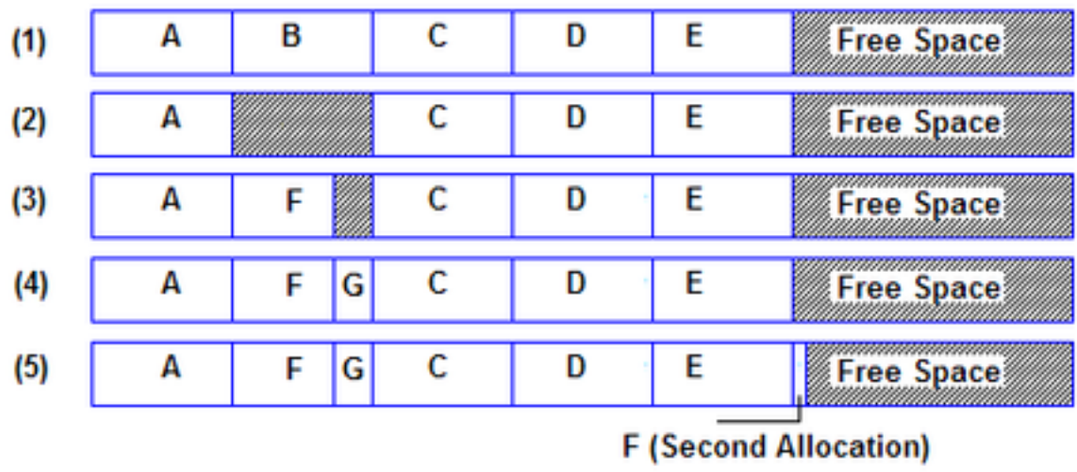
1. Storage, Filesystem
2. Firmware → Operating System → Apps
3. Machine Language → Python

Storage

- Storage devices provide an interface to read / write data into specific locations of some in non-violative media.
    - On traditional ***Hard Drive***, the data is written magnetically on a spinning metal disk.
        - The disk is divided up into sectors, the minimum storage unit.
        - Each sector has an address, which corresponds its physical location on the disk.
        - ***Solid State Drives*** store data on silicon… originally presented same interface as HDs, but new interfaces such as non-volatile memory express (NVMe) are designed for SSDs.
        - In Unix these are referred to as: /dev/hda, /dev/hdb, … /dev/sda, /dev/sdb
- ***Partition***: The disk sectors are partitioned into groups of sectors, each where a different file system can be created.
    - Partition table: keeps track of the locations of the partitions.
    - /dev/hda1, /dev/sdb2
    - Partitions can be further divided into logical partitions.

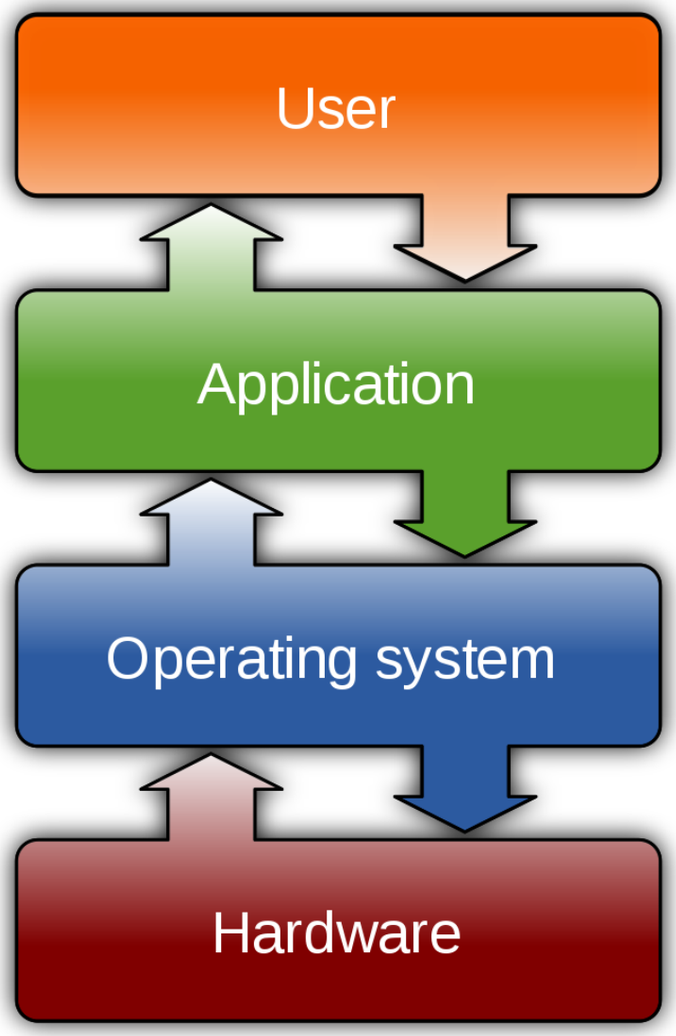
File system

- File system is a scheme for controlling how data is stored and retrieved from a partition.
    - File system organizes the sectors into blocks.
    - Organization:
        - Data is grouped into files.
            - Files have meta-data: e.g. when created, access permissions, …
        - Files are organized into directories.
    - FS holds a map Files names → Blocks.
        - Different file systems have different restrictions file names (e.g. allowed characters or case sensitivity).
        - General convention, “FOO.BAR”, FOO is filename, BAR is the extension which helps indicate the format of the file contents.
- FS enforce Access Control or Permissions… who can read what file.
- Different File system types:
    - Windows: FAT (FAT16, FAT32), exFAT NTFS, …
    - Mac: HFS, HFS+, APFS
    - Unix: ext2, ext3, ext4, ZFS, …
- SWAP: In some Operating Systems, the system can use storage as RAM when out of memory.



Software- Firmware

- There are several layers of software that interact with the computer at different levels.
    - During system boot, system starts with the Firmware, and hands control to next level, which then hands control on to then next level, and so on …
- ***Firmware***: Associated with the motherboard. Read Only Memory (ROM).
    - When you start a computer, it needs some instructions on what to do.
    - Test that everything is OK… e.g. memory.
    - Load user configuration.
    - Allow users to change configuration via menu system.
    - Start and configure peripherals, storage, …
    - Load the boot loader from storage into memory and hand-off
    - In the x86 world:
        - Basic Input/Output System (BIOS): Old
        - Unified Extensible Firmware Interface (UEFI): New



Boot Loader …

- Small program that loads the operating system kernel.
- Loaded by the firmware.
- Usually sits on the first sectors of the storage boot drive.
- May give the user the option of loading different operating systems sitting on different partitions of the storage.
- Must mount the partition, find the kernel (operating system), load it into memory, pass control to the kernel with potentially some configuration options (e.g. the root partition).

Operating System

- Software system that manages the computer hardware and software and provides common services for programs.
    - Sharing of resource between programs: processor, memory, storage, …
    - Intermediary between programs and hardware.
        - Provides Application Program Interface (API) / Software Development Kit (SDK) for building programs and interfacing them with OS.
- Examples: Windows, MacOS, Linux, iOS, Android, …
- Modern OSs are Multi-tasking: allow multiple programs to simultaneously run.
    - Each program ~ a process.
    - Pre-emptive multitasking: the OS gives slice of CPU time to each process.

Unix

- Multitasking, Multi-user, OS originally developed in 1970 by AT&T Bell Labs to run on mainframes with many connected terminals.
    - Written in C programming language.
- Many modern operating systems, including MacOS and Linux, implement Unix standards.
- “Unix Philosophy”
    - Plain text data storage.
    - Hierarchical file system.
    - Devices and inter-process communication via files
    - Main program that runs is the *kernel*.
    - Primary user interface is a *command-line* interpreter, called a shell.
    - Modular:
        - lots of small programs serve as tools
        - strung together via the *command-line* interpreter
        - passing information between each other via pipes

Graphical User Interface

- Though most of you associate an OS with its GUI, it is generally a layer on top of the core OS.
    - Establishes graphical metaphors aimed at simplifying user interaction with the OS.
        - Mouse cursor, windows, menus, buttons, …
        - Provides API for applications to build GUIs for themselves.
- For Unix, the X Windows system implements a client-server model:
    - The server runs the application that makes the API GUI calls.
    - These calls are transmitted to the network to a client.
    - The client runs the graphical environment, collects input, and draws the graphical elements.

Programming

- Each CPUs understands its own instruction sets.
- Low level operations:
  - Copy in/out data from memory into registers.
  - Perform operations on registers.
  - Conditional (if/then) and Control flow (jump)
  - Manage a stack (where small sets of data can be shared between different blocks of code)
- Each instruction are each assigned a specific binary value and are not human readable…
- Assembly is a human-read language that most closely mirrors Machine Language.

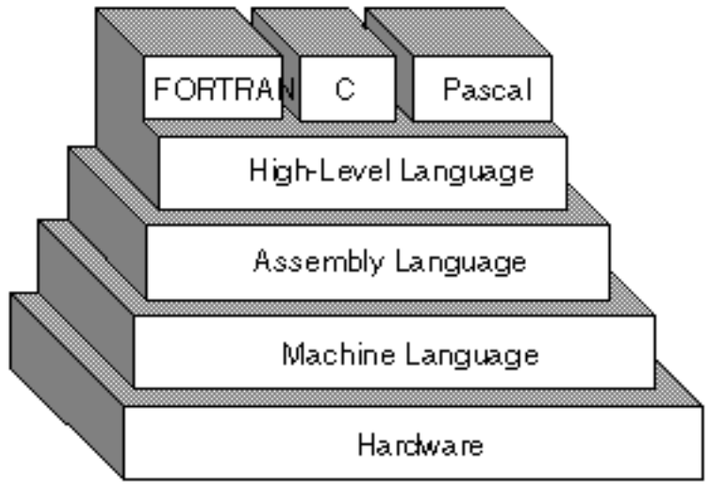
```
x3000 LD R1, x006          ; load data pointer
x3001 LDR R2, R1, #0       ; load data value
x3002 BRz x005             ; branch to end if zero
      ;
      ; repeating statements go here
      ;
x3003 ADD R1, R1, #1       ; increment data pointer
x3004 BRnzp x001           ; branch back to top
x3005 HALT
      ;
      ; data section
      ;
x3006 x4000 ; address of data

; The translation into LC-2 Machine Code
      0011 0000 0000 0000      ; load at x3000
X3000 0010 001 0 0000 0110     ; LD R1, x006
x3001 0110 010 001 000000      ; LDR R2, R1, #0
x3002 0000 010 0 0000 0101     ; BRz x005
      ;
      ; repeating statements go here
      ;
x3003 0001 001 001 1 00001     ; ADD R1, R1, #1
x3004 0000 111 0 0000 0001     ; BRnzp x001
x3005 1111 0000 0010 0101      ; HALT
      ;
      ; data section
      ;
x3006 0100 0000 0000 0000      ; x4000
```

from: http://www.eecs.umich.edu/courses/eecs284/example1.htm

Programming Languages

- Writing Assembly (Machine) code requires
    - knowledge of the specific CPU,
    - working at level of very small operations and
    - awareness of registers, memory, and hardware.



- High-level languages provide high level abstractions and human friendlier syntax for programming.
- Two types:
    - ***Compiled***: The text of the high level language is converted by a *compiler* into machine code. The machine code is run subsequently. Usually 2 steps:
        1. Compile to code into machine language → library
        2. Link the code against libraries → executable
    - ***Interpreted***: A program runs that reads the text of the high level language and performs the operations.

High-level Language

- 3 Fundamental Elements of any programing language:

    1. Primitives:

        - Numbers, Characters, …
        - Mathematical Operations: +, -, *, /, …
        - Logical Operations: and, or, …
        - Conditionals: if-then-else, …

    2. ***Means of Combination***: e.g. list

    3. ***Means of Abstraction***:

        - Assignment: x = 1
        - Definition: def x: 1
        - Function: def f(x): x

- Beyond these, there are universal programming concepts and patterns (e.g. object oriented programming) that enable or facilitate building sophisticated software.

- While we will learn these in python, look beyond the syntax and specifics of the programming language.

Why Python?

- Interpreted: no compilation time. Multi-platform.
    - Expense of speed, but the time consuming code are often in complied libraries.
- Large library: almost any package out there has a python API.
- Easy to read. Convenient syntax.
- Convenient data structures that are simple to build.
    - Advanced data structures: list, dictionaries, sets…
    - Dynamic typing: no need to declare the type of a variable.
    - Built-in memory management (reference counting + garbage collection): No need to worry about memory addresses or allocating/freeing memory.
    - Dynamic name resolution (late binding): same code can be reused for different data.
- Multi-paradigm:
    - structured programming: functions, sub-routines, etc…
    - functional programming: filter, map, reduce, lambda, generators, …
    - object-oriented programming: class, inheritance, …
    - …
- Language of choice for Data Science.