CSCI 3310U
Operating Systems

Systems Programming Course

Today ‘s Learning Goals

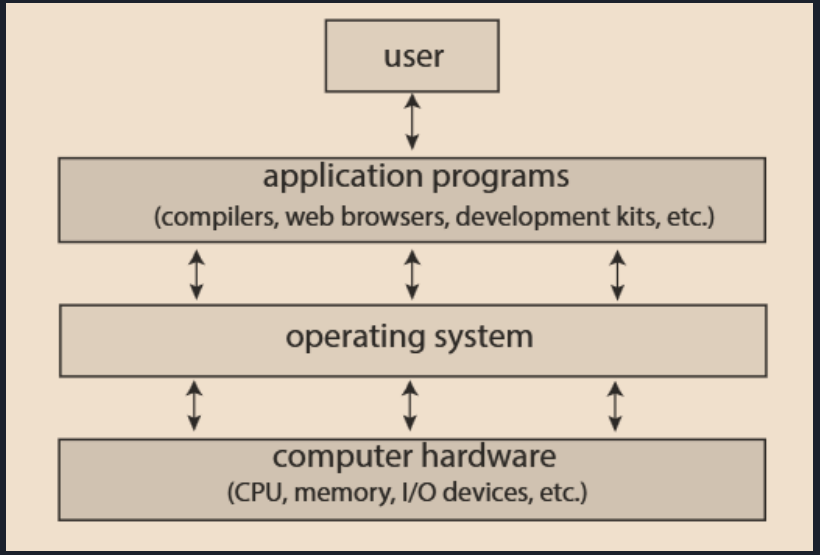
By the end of today’s lecture, students will be able to:

- Understanding what is Operating Systems?
- Operating Systems Functions
- Computer Hardware
- Additional Operating Systems Structures
- Systems Call
- Standard API Examples
- Systems Call Implementation

Operating Systems

- A computer system can be split up roughly into four main components.
- Four components include:
  - Hardware,
  - Operating System,
  - Application Programs
  - User.

Four components within the computer systems are interconnected to achieve the prime function of computer (executing programs).



High-level Abstraction overview of Computer Systems [1]

Operating Systems

- Operating Systems is a software program that perform several roles and Functions.

<u>These include:</u>

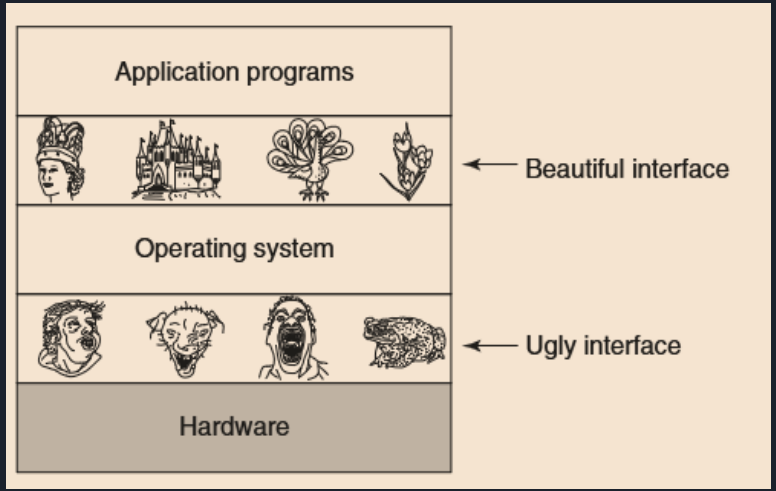
- Performs an intermediary role between the user and computer hardware. i.e exploiting the hardware resources to deliver a set of services to system users.

Operating Systems

- Operating Systems is a software program that perform several roles and Functions.

<u>These include:</u>

- Hide the hardware and present programs with clean and user-friendly interfaces (Abstracting hardware details) to the system users.

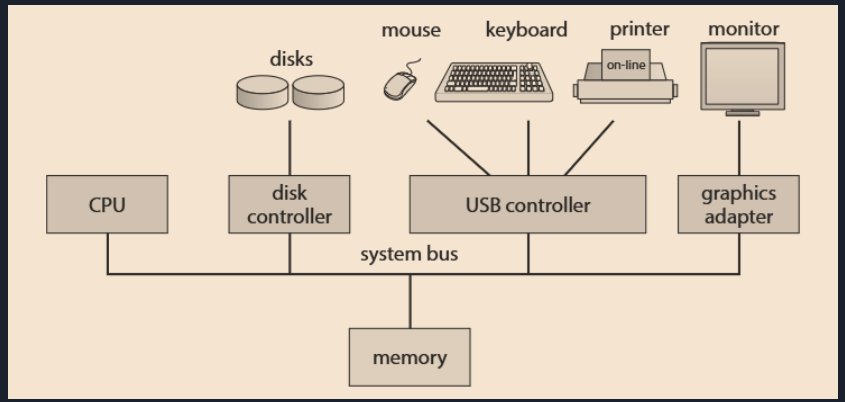


Operating systems abstracts Hardware details [2]

Operating Systems

- An operating system is closely tied to the computer hardware it runs on. Therefore, extending the set of computer's instructions and managing its resources.
- In a conceptual means, we can provide an abstraction of the personal computer into: CPU, I/O devices, memory (interconnected by a common system bus, that allows the communication between one and another over it.

Note: Modern Computer systems adopt a more complex structure that may involve, for example: multiple buses.



Note: Common System bus allow access between different components and shared memory.

[1]

Operating Systems

- Every device controller is responsible for a certain categories of devices. Based on the controller functionality, more than one device can be attached. Example: In order to attach several USB devices, the USB port connects to a USB hub to allow for this functionality.
- A device controller is in-charge of data transfer between peripheral devices that it manages and their local buffer storage.
- Commonly, Operating systems possess a device driver for every device controller.
- Device driver interprets the device controller, providing a uniform interface of the device to the rest of the operating.

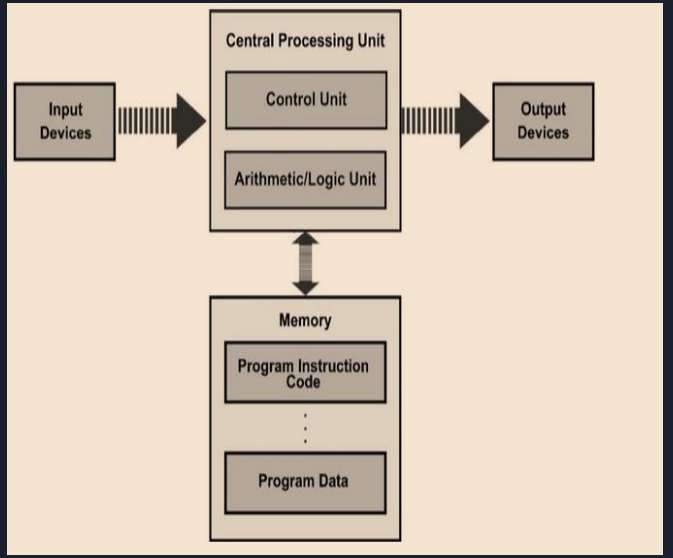
CPU and device controller perform parallel execution, where they compete for memory cycles. To efficiently access the shared memory, memory controller manages their access.

Operating Systems

- Given an example of a typical computer program I/O operation.
- The device driver loads the required registers in the device controller.
- Device controller inspects the content of these registers to decide what the appropriate actions need to be taken.Example, reading a character from the Keyboard).
- Device controller starts the data transfer from device to the local buffer.
- Upon completion, device controller notifies the device driver that the operation is completed.
- The device driver then provides control to other parts of the operating systems, perhaps returning data or pointer to the data in the case of read operation.

Operating Systems

- Central Processing Unit: This is considered the brain of the computer.
- Arithmetic and Logic unit: This is where the arithmetic (addition, subtraction, …etc) and logic (AND,OR,..etc) operations are performed.
- Control Unit: This is where instructions are provided to respond to the operations of ALU, memory and I/O devices. (ie, execute micro-operations). Also gives the timing and control signals needed by other computer components.



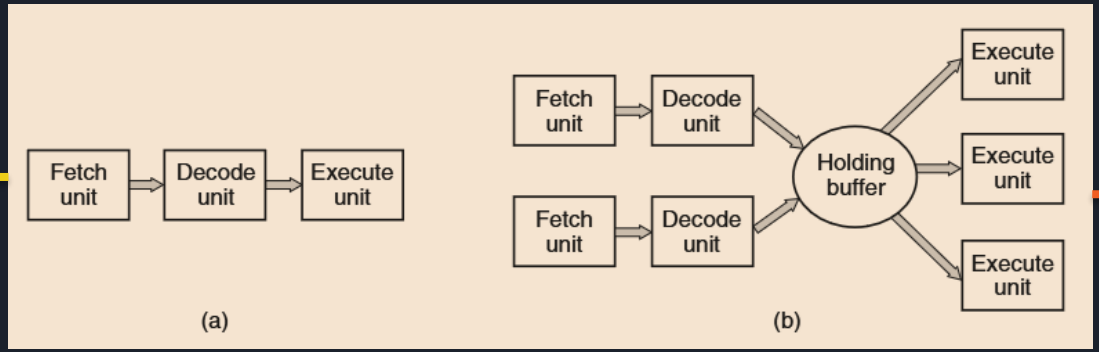
Computer Architecture by Von Neumann [3]

Operating Systems

- Get the first instruction from the memory.
- Decode the first instruction (identify what is its type and operands).
- Execute the first instruction.
- Then get, decode and execute succeeding instructions.
- Repeat until the program is completed.

In this manner, programs are carried out.

Simple Three stage pipeline

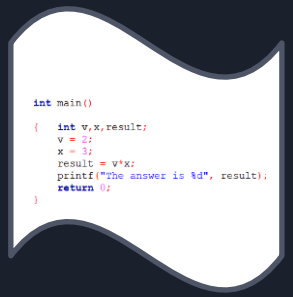


Superscalar CPU

[2]

Operating Systems

- Example: Given the following example



Macro-operations
“High-Level”

Compiler

.exe

-Load Value v into …
-Load Value x into ….
-Multiply and store result into memory

Micro-Instructions

Passed to the OS to load it into the memory

Operating System

Loaded by OS directly from Hard disk to Memory

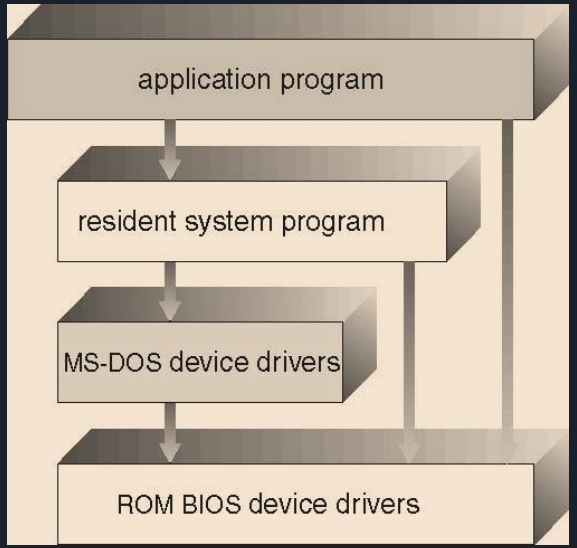
Main Memory

Accessed by CPU
Execute the Program

CPU is unable to access the program directly from Hard disk

Additional Operating Structures

- Simple Structure (MS-DOS) written to provide the most functionality in the least space.
- Not divided into modules.
- Interfaces and levels of functionality are not well separated.



[4]

Additional Operating Structures

- Most modern operating systems are considered a Hybrid model.
- Hybrid combines multiple approaches to address performance, security, usability.
- Linux kernels in kernel address space, so monolithic, plus modular for dynamic loading of functionality.
- Windows mostly monolithic, plus microkernel for different subsystem personalities.

Additional Operating Structures (Mobile)

- Popular operating systems include Android Operating System.
- Based on Linux kernel but modified: Provides process, memory, device-driver management, Adds power management
- Runtime environment includes core set of libraries and Dalvik virtual machine.
- Apps developed in Java plus Android API.

System Calls

- Process Control
  - Examples: (end a process upon completion , load a process, execute, abort…etc).
- File Manipulation
  - Examples: (Create file, delete file, write to a file,..…etc).
- Device Management
  - Example : Logically detach devices ie (safely remove hard drive even though it is still physically plugged in… , etc).
- Information Maintainance
  - Example: Get time or date,…… etc
- Communication : send and receive messages between different processes, … etc

System Calls

- *System call is the programmatic manner in which a computer program requests a service or task from the Kernel of the operating system.*

- *These calls are usually coded in C, C++, Rust.*

- *Three most common APIs are :*
  - *Win32 API for Windows*
  - *POSIX API for POSIX-based systems*
  - *Java API for the Java virtual machine (JVM)*

Standard API Example

- Below is an example of a standard API for a read() function in UNIX and Linux
- On a successful read, the number of bytes read is returned.
- A return value of 0 indicates end of file
- If an error occurs, read() returns −1

```
#include <unistd.h>

ssize_t     read(int fd, void *buf, size_t count)
```

return value — function name — parameters

System Call Implementation

- The system call interface invokes the intended system call in OS kernel and returns status of the system call and any return values.
- The caller needs know nothing about how the system call is implemented.
- Just needs to obey API and understand what OS will do as a result call.
- Most details of OS interface are hidden from programmer by API.

Summary of Learning Goals

The Goal of Today:

- Understanding what is Operating Systems?
- Operating Systems Functions
- Computer Hardware
- Additional Operating Systems Structures
- Systems Call
- Standard API Examples
- Systems Call Implementation