Chapter 1 – Introduction to Architecture

1.1 Definition of OS
1.2 Computer System Operation
1.4 Operating System Structure

www.os.book.com

slides (don't use)
virtual machine that we will download in the lab.

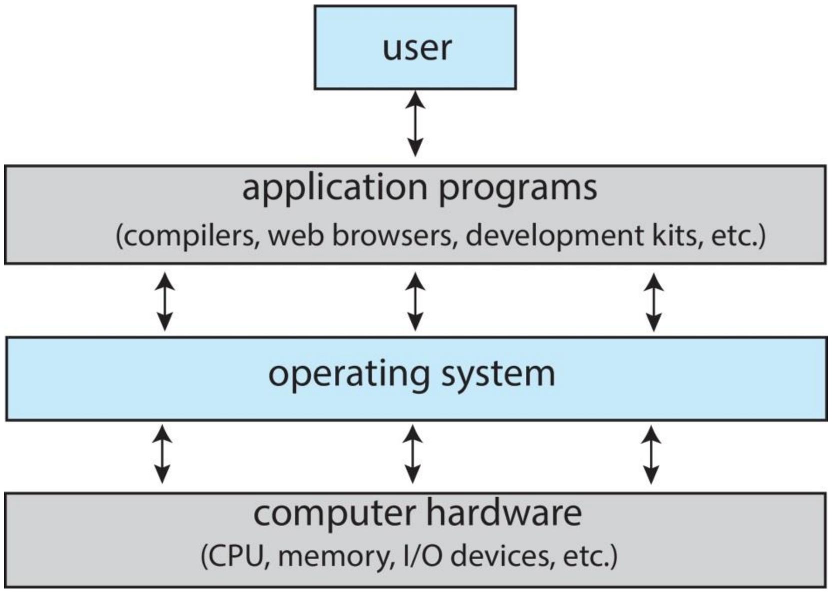
Chapter Objectives

- Describe the general organization of a computer system and the role of interrupts.
- Describe the components in a modern multiprocessor computer system.
- Illustrate the transition from user mode to kernel mode.
- Discuss how operating systems are used in various computing environments.
- Provide examples of free and open-source operating systems.

What is an OS? (What is its purpose)

- An operating system is a program that makes it
  convenient to use the hardware
  Efficient to use the hardware
  Acts as an intermediary between user and hardware (like a government.. no useful work of its own help users use the hardware)

Where does it fit in the Computer System


user
application programs
(compilers, web browsers, development kits, etc.)
operating system
computer hardware
(CPU, memory, I/O devices, etc.)

Hardware the central processing unit (CPU), the memory, and the input/output (I/O) devices—provides the basic computing resources for the system

Application programs such as word processors, spreadsheets, compilers, and web browsers—define the ways in which these resources are used to solve users' computing problems.

User could be another machine

We can also view a computer system as consisting of hardware, software, and data.

User's view of OS

Types of users:

Single PC user – wants ease of use not great resource utilization, touch screen usage, siri (voice recognition system)

Mainframe – resource utilization is very important since sharing is going on.

server – now individual usability is not as important as serving all clients.

Embedded Systems – have no user interaction… they can turn light on/off.

System's View

Resource allocator – RAM I/O devices, files etc. etc.
Controller – control the operations.

What is an Operating System?

No universally accepted definition

Some systems take up less than a megabyte of space and lack even a full-screen editor, whereas others require gigabytes of space and are based entirely on graphical windowing systems

Operating System includes for sure the kernel.
"The one program running at all times on the computer" - **kernel**.

Possibly some **system program** (ships with the operating system) associated with OS but not part of the kernel

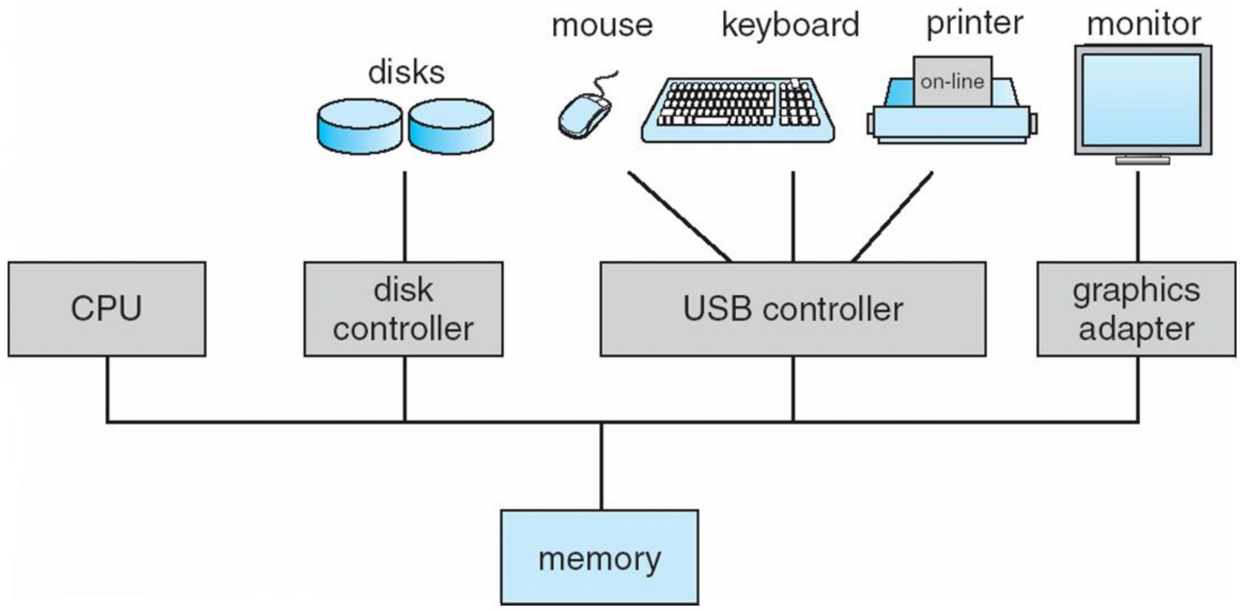
Possibly **application programs** very few..

Middleware – A set of software frameworks to connect to databases, web etc and networking etc.. Example Mobile OS, etc offers middleware

Section 1.2 Computer System Operation

Figure

- **Computer-system operation**



- Each device controller is in charge of a particular device type
- Each device controller has a local buffer and some registers. I/O is from the device to local buffer of controller
- One or more CPUs, device controllers connect through common bus providing access to shared memory
- CPUs and devices competing for memory cycles
- I/O devices and the CPU execute concurrently
- Operating Systems contain **device driver** for interfacing with the code of device controller

CPU: Goes through a fetch-decode –execute cycle. It is master all other controllers are slaves. No controller can do anything unless directed by CPU's instruction.

RAM : Holds all data and programs in it that is currently running
Disk : Holds all other programs and data.
Devices : Interact with controller

There is a memory controller too and CPU and device controllers compete for memory cycles. CPU directs the device controllers The entire architecture is interrupt driven.

Interrupts

CPU remains in fetch-decode-execute-decode cycle fetching instruction from memory and executing an ALU instruction or may schedule an I/O from the disk.

The interrupt controller signals the CPU when an interrupt occurs.
For instance , CPU might schedule the disk controller to read a file from the disk. Device controller informs CPU that it has finished its operation by causing an interrupt

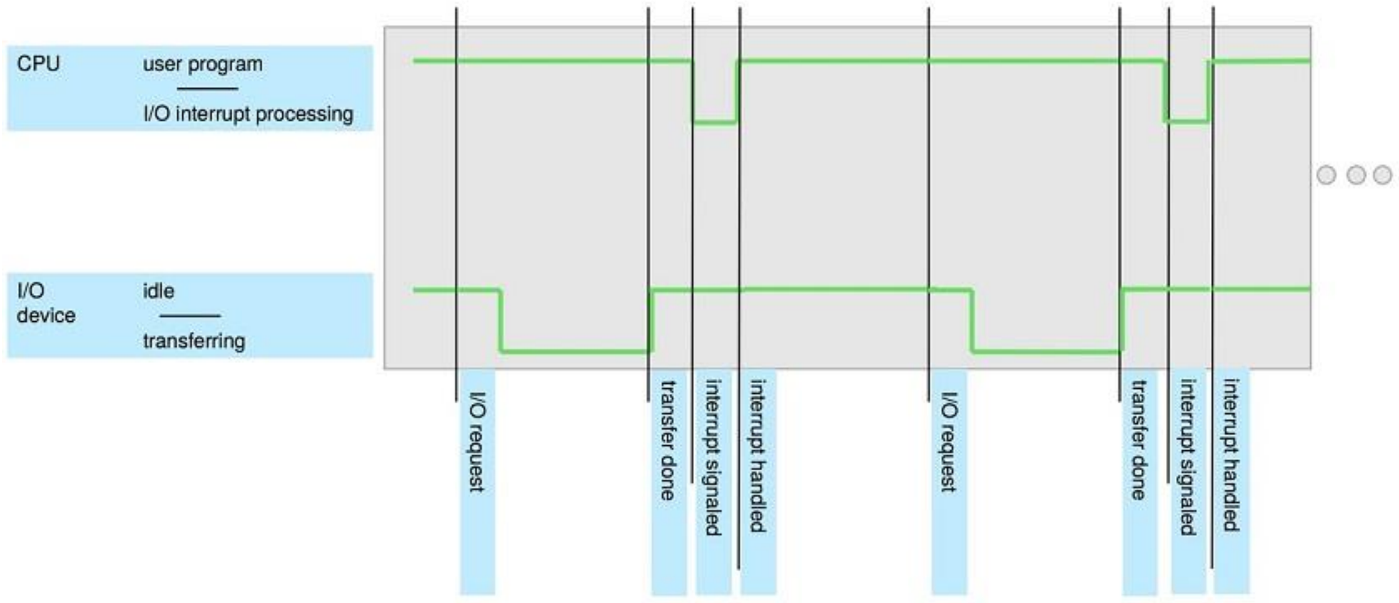
Thus, An operating system is **interrupt driven**

Interrupt Handling

- The operating system preserves the state of the CPU by storing registers and the program counter when an interrupt occurs before switching to the Interrupt Service Routine.
- interrupt occurs, **interrupt vector**, of addresses is then indexed by a unique device number, given with the interrupt request, to provide the address of the interrupt service routine.
- Separate segments of code determine what action should be taken for each type of interrupt

Interrupt Chaining : When you have more devices than interrupt vector each element in the vector can point to another one.

Interrupt Timeline



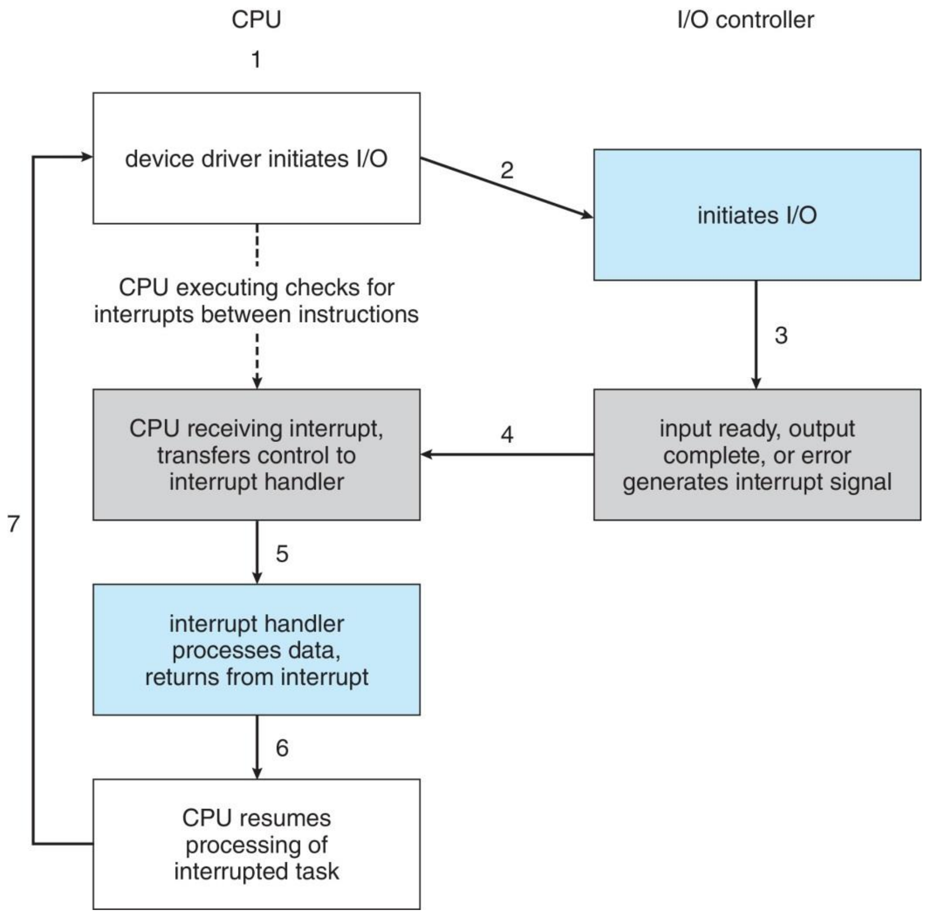
Note that when a program requests I/O there are not interrupts. It occurs only upon I/O Completion.
Interrupts are initiated by hardware and they are asynchronous and external to the running program.

TRAPS

- Traps are also called “software interrupts”..
- These include system calls, division by zero, seg faults etc.
- Some traps may lead to hardware interrupts.
- Separate segments of code determine what action should be taken for each type of trap.
- Traps are synchronous and internal.

Implementation of Interrupts:

We say that the device controller **raises an interrupt** by asserting a signal on the **interrupt request line,** the CPU **catches** the interrupt and **dispatches** it to the interrupt handler, and the handler **clears the interrupt** by servicing the device. Figure 1.4 summarizes the interrupt-driven I/O cycle.


CPU
I/O controller
1
device driver initiates I/O
2
initiates I/O
CPU executing checks for interrupts between instructions
3
CPU receiving interrupt, transfers control to interrupt handler
4
input ready, output complete, or error generates interrupt signal
7
5
interrupt handler processes data, returns from interrupt
6
CPU resumes processing of interrupted task

Maskable and Non-maskable Interrupt Request Lines:

Most CPUs have two interrupt request lines. One is the **nonmaskable** interrupt, which is reserved for events such as unrecoverable memory errors. The second interrupt line is **maskable:** it can be turned off by the CPU before the execution of critical instruction sequences that must not be interrupted. The **maskable** interrupt is used by device controllers to request service.

Storage Managements

Memory – volatile memory – RAM
NVM non volatile memory – Hard Disk , tape etc.

I/O Structure

- All I/O requests come from System call
- Device-status table contains entry for each I/O device indicating its type, address, and state(busy/free)
- OS indexes into I/O device table to determine device status and schedules an I/O by modifying the table entry to include interrupt upon completion
- operating systems have a **device driver** for each device controller. This device driver understands the device controller and provides the rest of the operating system with a uniform interface to the device.
- To start an I/O operation the device driver loads the registers within the device controller
- The device controller, in turn, examines the contents of these registers to determine what action to take (such as "read a character from the keyboard"). The controller starts the transfer of data from the device to its local buffer.
- Once the transfer of data is complete, the device controller informs the device driver via an interrupt that it has finished it operation. The device driver then returns control to the operating system,.

DEVICE	STATUS	Process Waiting	Interrupt Schedule
Disk 0	Busy	P2	Upon Completion
Disk 1	IDLE		None

Disk 2	Busy	P3 P4 P1	Upon Completion
Disk 3	IDLE		None

END OF LECTURE

1.4   OS Structure.

Multiprogramming and Timesharing

■ **Multiprogramming** (**Batch system**) needed for efficiency

- Single user cannot keep CPU and I/O devices busy at all times
- Multiprogramming organizes jobs (code and data) so CPU always has one to execute
- A subset of total jobs in system is kept in memory
- One job selected and run via **job scheduling**
- When it has to wait (for I/O for example), OS switches to another job

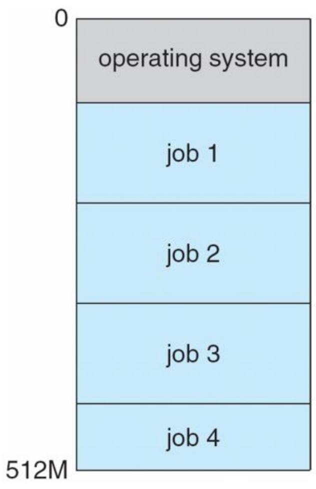
■ **Timesharing** (**multitasking**) is logical extension in which CPU switches jobs so frequently that users can interact with each job while it is running, creating **interactive** computing

  - **Response time** should be < 1 second
  - Each user has at least one program executing in memory ➩**process**
  - If several jobs ready to run at the same time **CPU scheduling**
  - If processes don't fit in memory, **swapping** moves them in and out to run
  - **Virtual memory** allows execution of processes not completely in

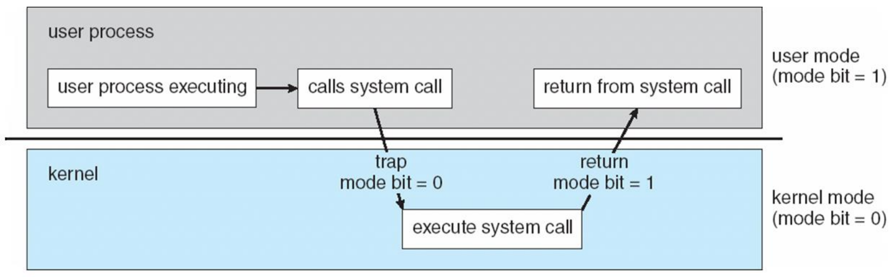
memory

1.3 Memory Layout for Multi-programmed System

-



Dual Mode and Privileged Instructions

- **Dual-mode** operation allows OS to protect itself and other system components
  - **User mode** and **kernel mode**
  - **Mode bit** provided by hardware
    - Provides ability to distinguish when system is running user code or kernel code
    - Some instructions designated as **privileged**, only executable in kernel mode
    - System call changes mode to kernel, return from call resets it user mode
    - 

The instruction to switch to kernel mode is an example of a privileged instruction. Some other examples include I/O control, timer management, and interrupt management. As we shall see throughout the text, there are many additional privileged instructions.

Some times multiple modes can be useful when you are working with Virtual machines

TIMER

- How to ensure that no program is taking CPU’s time forever.

- Timer to prevent infinite loop / process hogging resources
    - Timer is set to interrupt the computer after some time period
    - Keep a counter that is decremented by the physical clock.
    - When counter zero generate an interrupt
    - Set up before scheduling process to regain control or terminate program that exceeds allotted time