Operating System:
A program that acts as an intermediary between a user application and the computer hardware.
Operating system goals:
Execute user programs – Make the computer system convenient to use – Use the computer hardware in an efficient manner
Computer System Structure

- Hardware (provides basic computing resources • CPU, memory, I/O devices)
- Operating System (Controls and coordinates use of hardware among various applications and users)
- Application Programs (define the ways in which the system resources are used to solve the computing problems of the users)
- Users (People, machines, other computers)

Kernal:
kernel is the most important program in the operating system that has complete control over everything in the system and running at all times on the computer.
System Program:
A program that controls some aspect of the operation of a computer.
Example: operating system, networking system, web site server.
System Call:
Programmatic way in which a computer program requests a service from the kernel of the operating system on which it is executed.
Shell:
a shell is a user interface for access to an operating system's services.
In general, operating system shells use either a command-line interface (CLI) or
graphical user interface (GUI)
Program:
A computer program is a collection of instructions that can be executed by a computer to perform a specific task.
What Happens When We Run a Program:
• Translates high level programs into an executable file.
• The exe contains instructions that the CPU can understand, and data of the program (all numbered with addresses)
• Instructions run on CPU: hardware implements an instruction set architecture (ISA)
• CPU also consists of a few registers, e.g.,
  – Pointer to current instruction (program counter or PC)
  – Operands of instructions, memory addresses
• To run an exe, CPU – fetches instruction pointed at by PC from memory
  – loads data required by the instructions into registers
  – decodes and executes the instruction
  – stores results to memory
• Most recently used instructions and data are in CPU caches for faster access

what does the OS do:

OS manages program memory, It Loads program,takes the w executable code and data from disk to memory.
manages CPU, Initializes program counter (PC) and other registers to begin execution.
manages external devices, Read/write files from disk.

virtualizes CPU:

Each process has the illusion of having the complete CPU.

Process abstraction:

A process not having the idea that the CPU runs other processes is called process abstraction.
OS provides the process abstraction by virtualizing the CPU.
Memory virtualization:
Each process thinks it has a dedicated memory space for itself, numbers code and data starting from 0 virtual address.
Policy: which process to run
Mechanism: how to “context switch” between processes

Interrupt:
An interrupt is a signal sent to the processor that interrupts the current process
1. polling: always in check
2. vectored interrupt system: check in interrupt vector.

Design goals of an operating system:
• Convenience, abstraction of hardware resources for user programs • Efficiency of usage of CPU, memory, etc. • Isolation between multiple processes

How does OS create a process:
OS gives a PID then,
Allocates memory and creates memory image
Loads code, data
Creates runtime stack, heap
PC points to first instruction

process control block (PCB):
Information about each process is stored in a process control block (PCB).

Multiprocessing is the use of two or more CPU within a single computer system.
Symmetric Multiprocessing(shared memory) Asymmetric Multiprocessing(master cpu)

Cluster Systems:
multiple systems working together

Protection: any mechanism for controlling access of processes or users to resources defined by the OS.
Security: defense of the system against internal and external attacks

Job queue: set of all processes in the system
Ready queue: set of all processes , ready and waiting to execute
Device queues: set of processes waiting for an I/O device

- **Long-term scheduler** (or job scheduler) – selects which processes should be into the ready queue
- **Short-term scheduler** (or CPU scheduler) – selects which process should be executed next and allocates CPU
- **Medium term scheduling :** many inturupts can happen in run a process, to handle those middle term scheduler plays role.

Processes can be described as either:
- **I/O-bound process** – spends more time doing I/O than computations, many short CPU bursts
- **CPU-bound process** – spends more time doing computations; few very long CPU bursts

Threads: A thread is the smallest sequence of programed instructions that can be managed independently by a os scheduler.

Translation lookaside buffer (TLB) : cache of recent VA->pa mapping

How does OS read/write to registers:
1. Explicit I/O instructions
2. Memory mapped I/O

A simple execution of I/O requests
1. Polling status to see if device ready – wastes CPU cycles
2. Programmed I/O – CPU explicitly copies data from device

Requirements for Solution for a Critical-Section Problem: • 1 **Mutual Exclusion**: only one process can be in its critical section at a given point of time. • 2 **Progress**: No process can execute its critical section again just after finishing the execution • **3 Bounded Waiting:** A bound must exist on the number of times that other processes are allowed to enter their critical sections. **Solution:** Peterson, semaphores	**Deadlock can arise if four conditions hold simultaneously.** **Mutual exclusion:** only one process at a time can use a resource **Hold and wait:** a process holding at least one resource is waiting to acquire additional resources held by other processes. **No preemption:** A resource can't be preempted from the process by another process, forcefully. **Circular wait:** Circular wait is a condition when the first process is waiting for the resource held by the second process, the second process is waiting for the resource held by the third process, and so on. Method of handeling deadlock: 1. Prevent or $\boxed{\text{avoid}}$ 2. Detect and recover **3.** Ignore (leave it to application softwares) **Dead lock, Avoidance algorithms:** • **Single instance** of a resource type – Use a **resource-allocation graph** (no circle pls) • **Multiple instances** of a resource type – Use the **banker's algorithm**	2.Detect and recover Resource Allocation Graph -> wait for graph **Recovery from Deadlock:** Resource Preemption • Selecting a victim – minimize cost • Rollback –restart process for that state • Starvation – same process may always be picked as victim