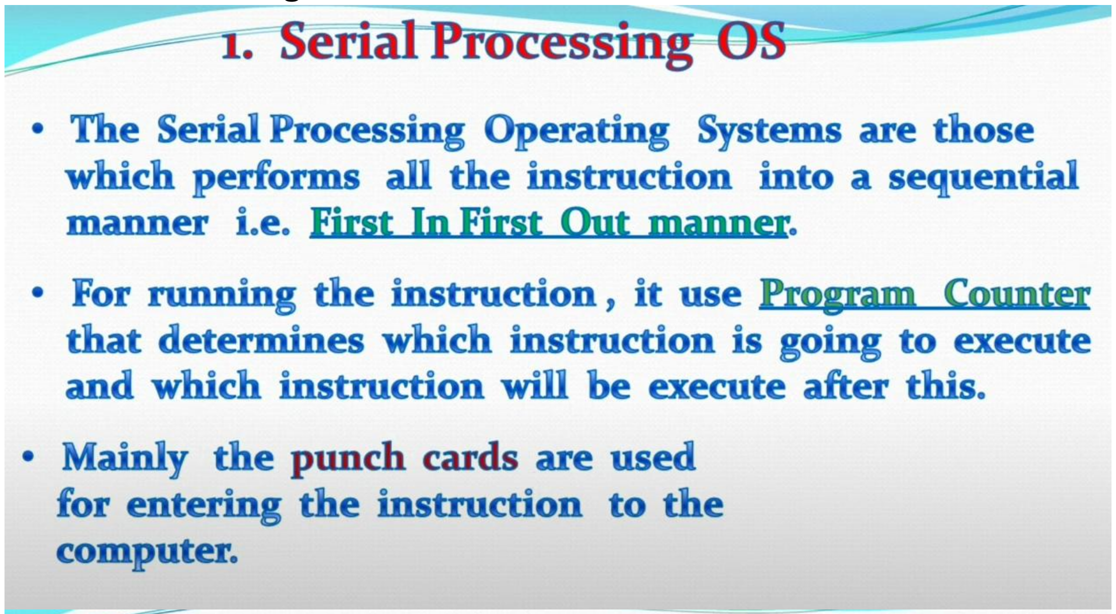
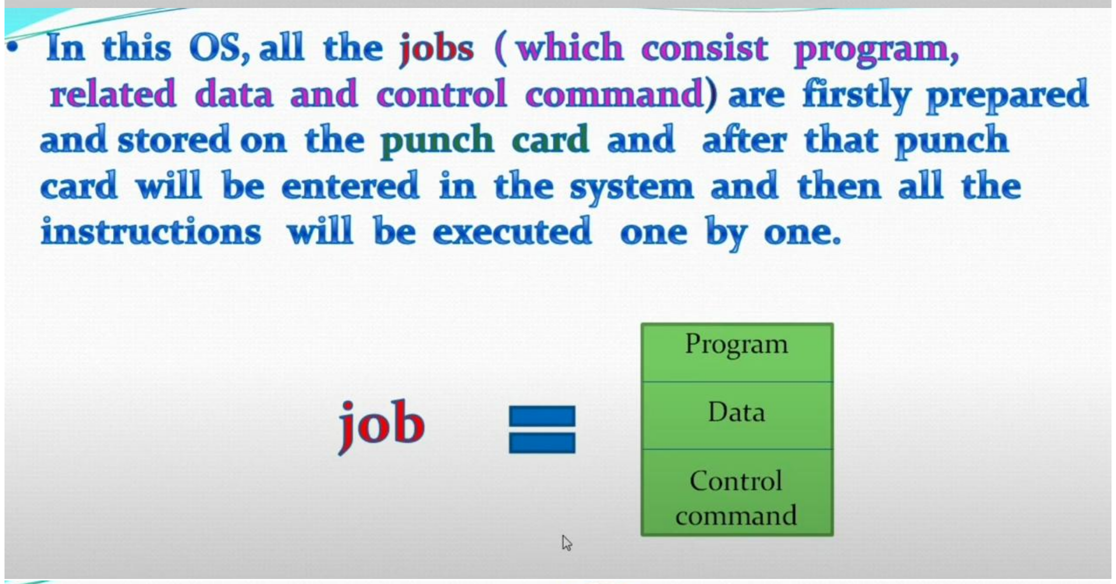
Operating System

1. Serial Processing OS:



1. Serial Processing OS

- The Serial Processing Operating Systems are those which performs all the instruction into a sequential manner i.e. First In First Out manner.
- For running the instruction , it use Program Counter that determines which instruction is going to execute and which instruction will be execute after this.
- Mainly the punch cards are used for entering the instruction to the computer.



- In this OS, all the jobs (which consist program, related data and control command) are firstly prepared and stored on the punch card and after that punch card will be entered in the system and then all the instructions will be executed one by one.

job
=
Program
Data
Control command

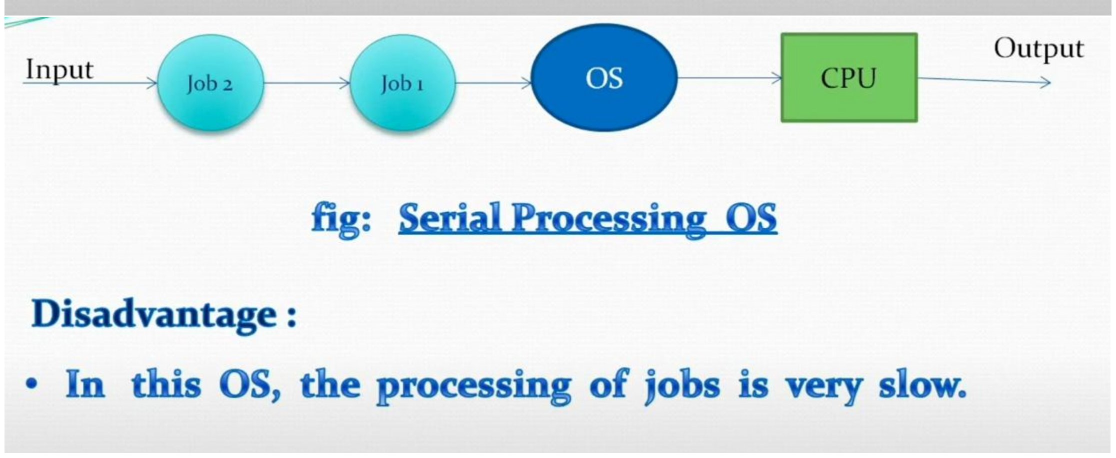


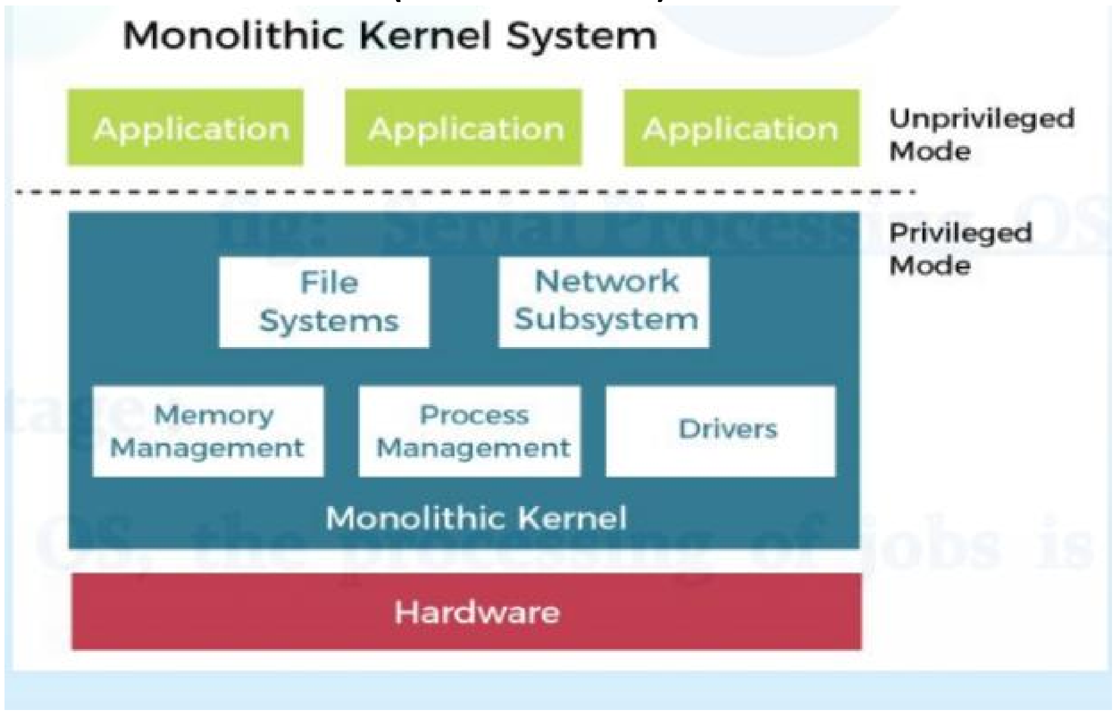
fig: Serial Processing OS

Disadvantage :

- In this OS, the processing of jobs is very slow.

OS Strcutures

1. Monolithic Structure : (unix follows this)



Advantages:

1.**Fast Execution** – Since all services (like memory management, file system, and device drivers) run in kernel mode, there is minimal context switching, making execution faster.

1.**Direct Hardware Access** – Monolithic kernels provide direct access to system hardware, leading to improved performance.

Disadvatage:

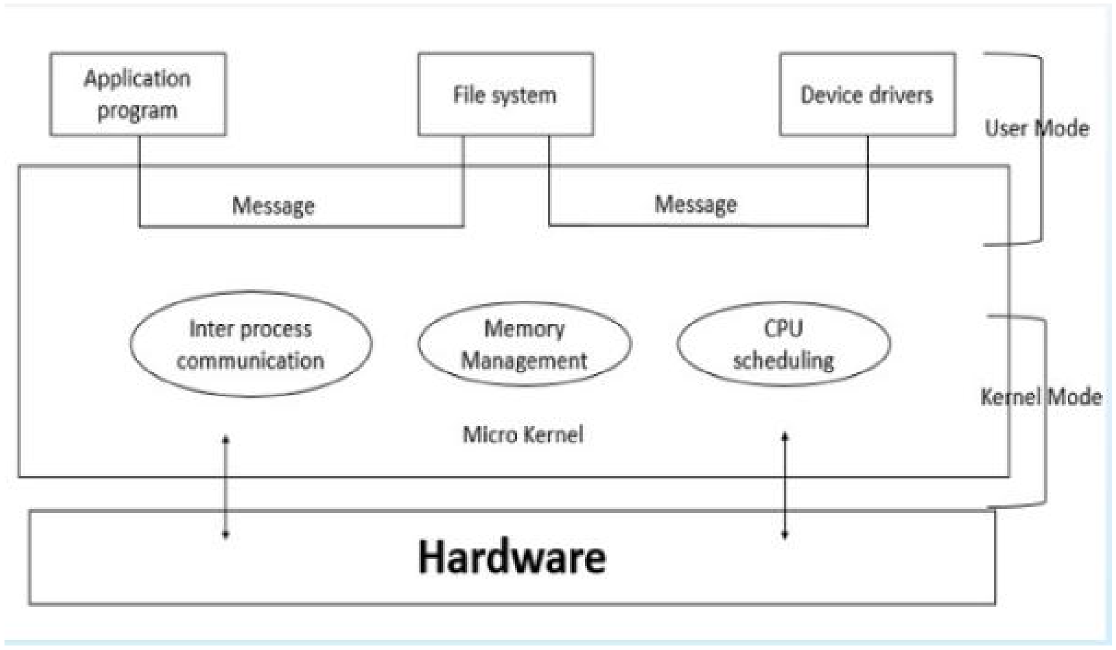
1.Difficult to manage

2. All functionalities are inclued into one kernel level incase of failure of one funcitonality lead to failure of entire kernel

3. No reliability

4. Security issues

2. MicroKernel



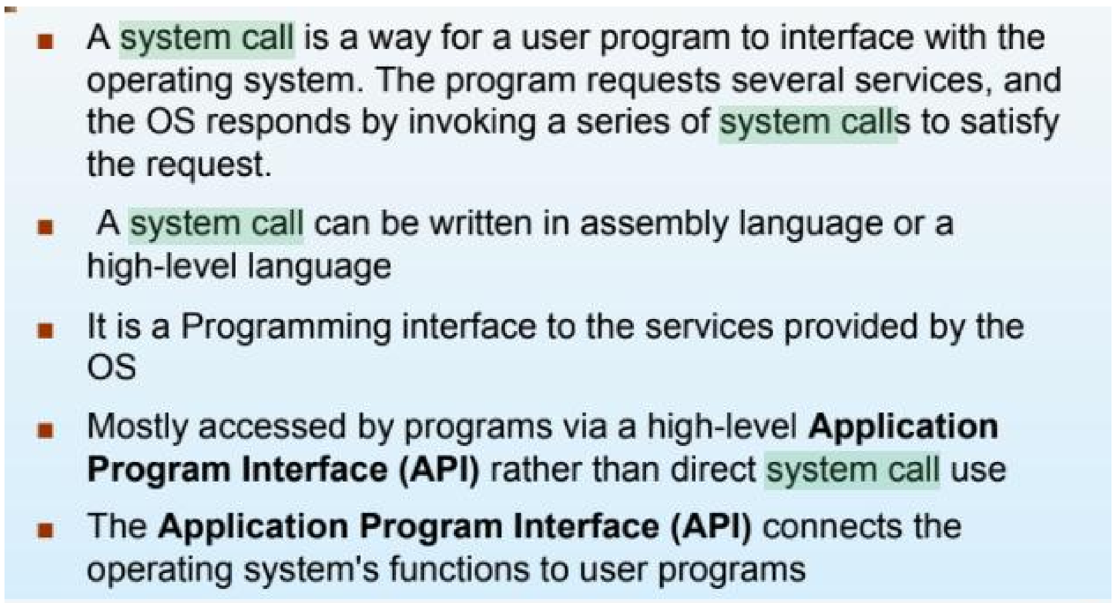
- Only core functionalities are included in the kernel level
- All other functinalities moved to the system programs
- Communication between programs use passing message

adv :
reliable and secure
less overhead
small size

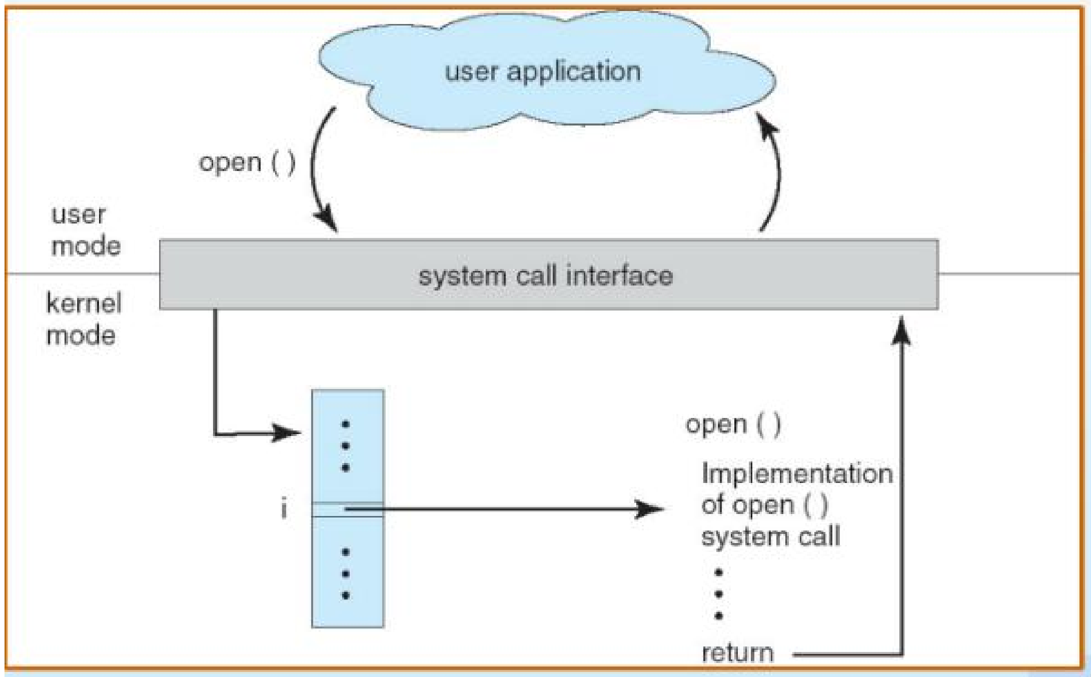
Dis:
Slower execution
Complex communication

System Call

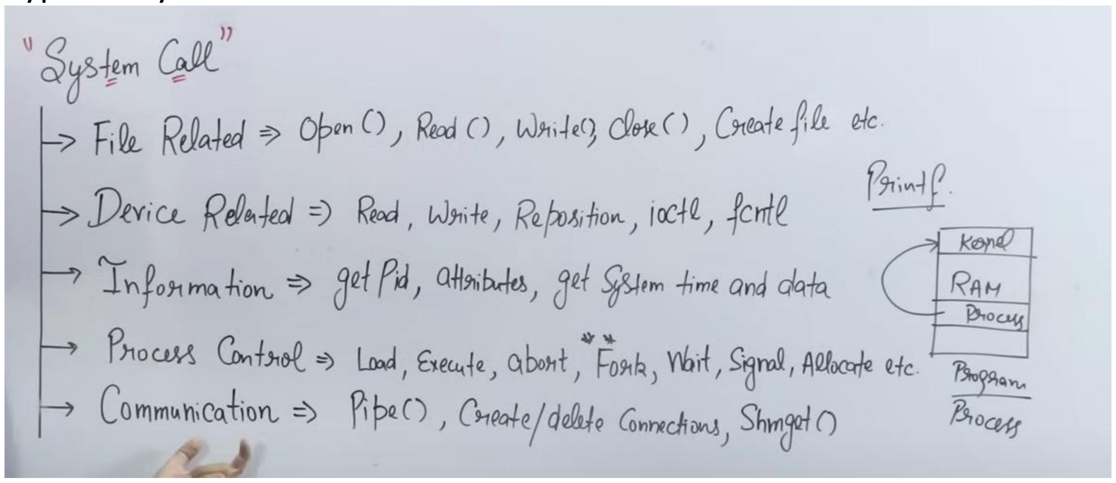
A system call is a mechanism that allows a user-space program to request services from the operating system by switching from user mode to kernel mode.



- A system call is a way for a user program to interface with the operating system. The program requests several services, and the OS responds by invoking a series of system calls to satisfy the request.
- A system call can be written in assembly language or a high-level language
- It is a Programming interface to the services provided by the OS
- Mostly accessed by programs via a high-level **Application Program Interface (API)** rather than direct system call use
- The **Application Program Interface (API)** connects the operating system's functions to user programs



Types of system calls:



"System Call"

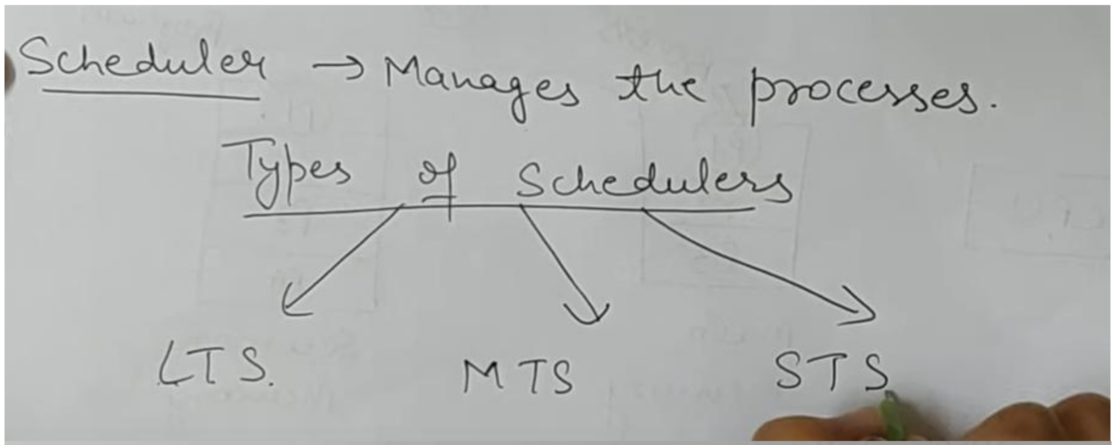
- File Related ⇒ Open(), Read(), Write(), Close(), Create file etc.
- Device Related ⇒ Read, Write, Reposition, ioctl, fcntl
- Information ⇒ get Pid, attributes, get System time and data
- Process Control ⇒ Load, Execute, abort, **Fork, Wait, Signal, Allocate etc.
- Communication ⇒ Pipe(), Create/delete Connections, Shmget()

Printf.
Kernel
RAM
Process
Program
Process

Multithreading

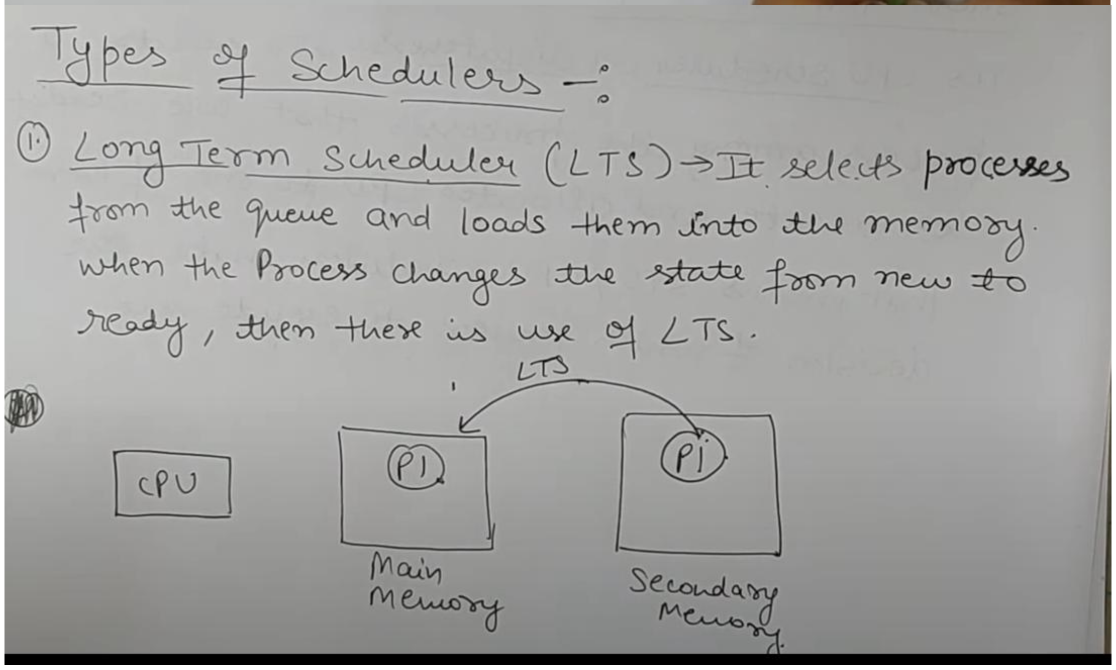
Multithreading is the ability of a process to execute multiple threads concurrently, sharing the same memory space, to improve efficiency and responsiveness.

Schedulers



Scheduler → Manages the processes.

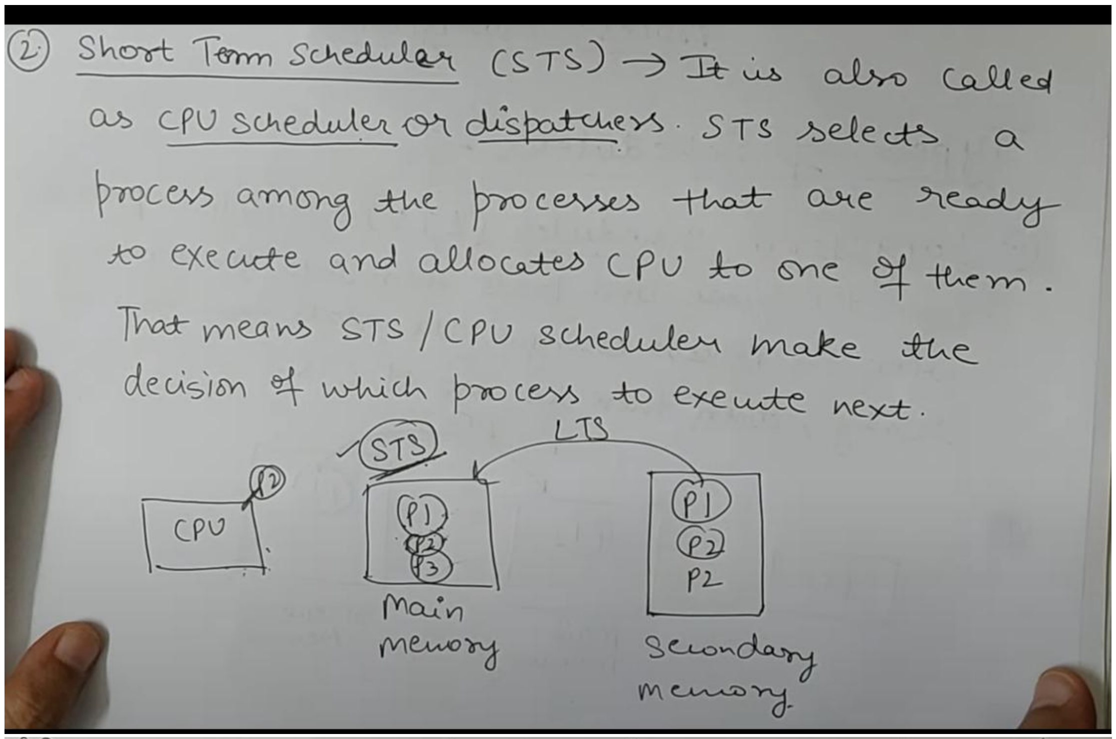
$$\begin{tikzcd}
 & \text{Types of Schedulers} \arrow[dl] \arrow[d] \arrow[dr] & \\
\text{LTS.} & \text{MTS} & \text{STS}
\end{tikzcd}$$



Types of Schedulers :-

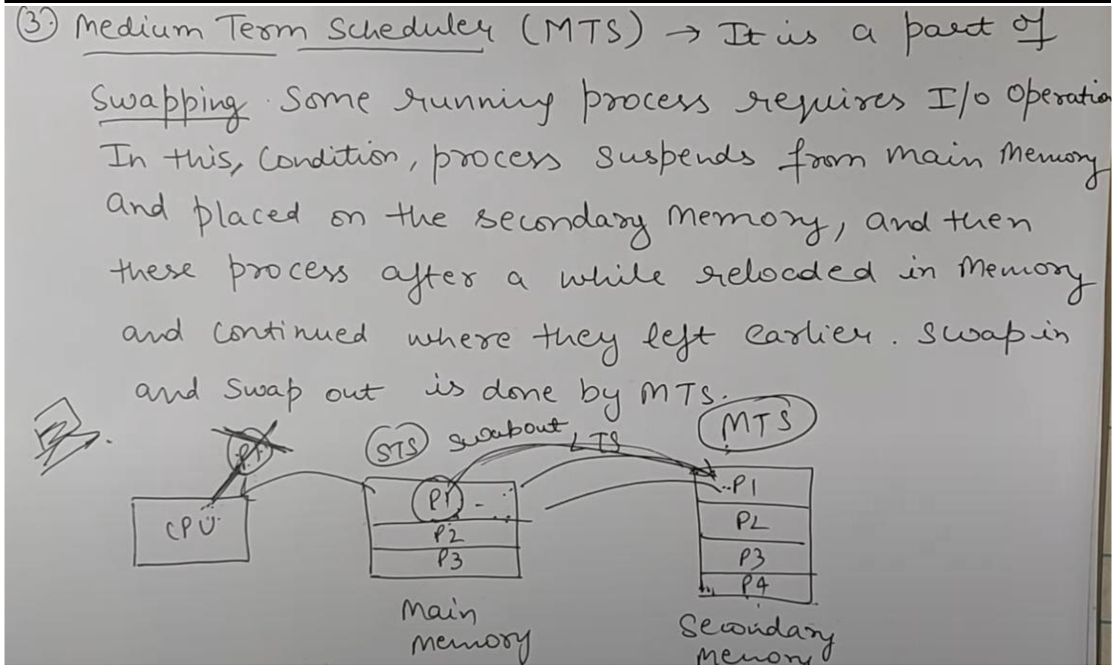
(1) Long Term Scheduler (LTS) → It selects processes from the queue and loads them into the memory. when the Process changes the state from new to ready, then there is use of LTS.

LTS
CPU
P1
Main Memory
P1
Secondary Memory.



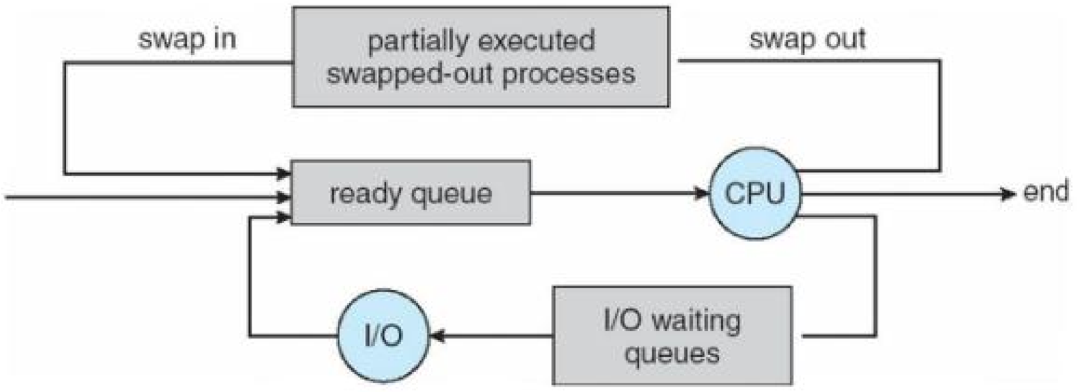
② <u>Short Term Scheduler</u> (STS) → It is also called as <u>CPU scheduler</u> or <u>dispatchers</u>. STS selects a process among the processes that are ready to execute and allocates CPU to one of them. That means STS / CPU scheduler make the decision of which process to execute next.

STS
LTS
CPU
P2
P1
P2
P3
P1
P2
P2
Main memory
Secondary memory

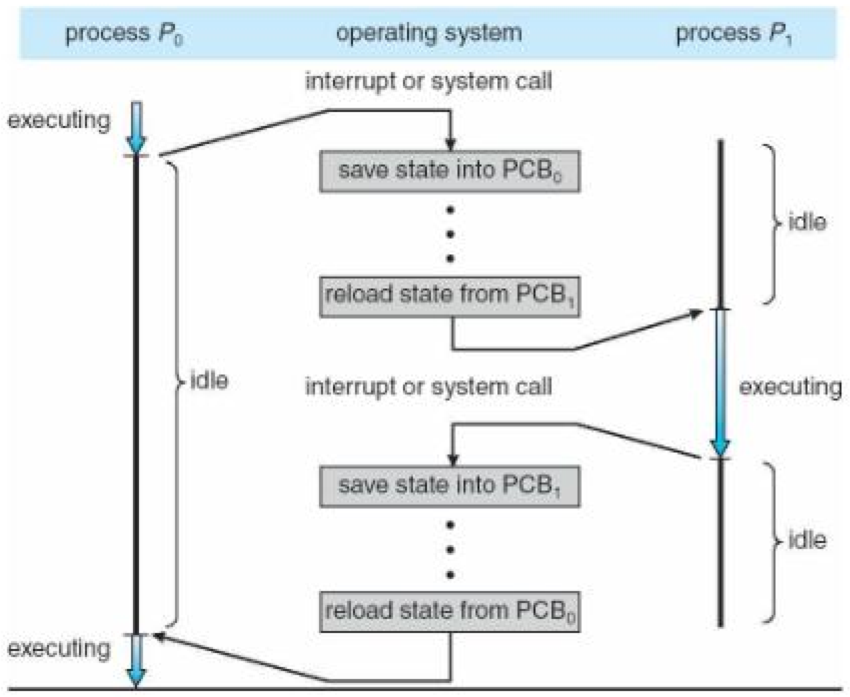


③ <u>Medium Term Scheduler</u> (MTS) → It is a part of <u>Swapping</u>. Some running process requires I/o Operatio
In this, Condition, process suspends from main Memory and placed on the secondary memory, and then these process after a while reloaded in Memory and continued where they left earlier. Swap in and Swap out is done by MTS.

Eg.
P1
STS
swapout
LTS
MTS
CPU
P1
P2
P3
P1
P2
P3
P4
Main memory
Secondary memory



Context Switching -



When the CPU switches from one process to another, it needs to save the current process's state and load the state of the new process. This is called a **context switch.**

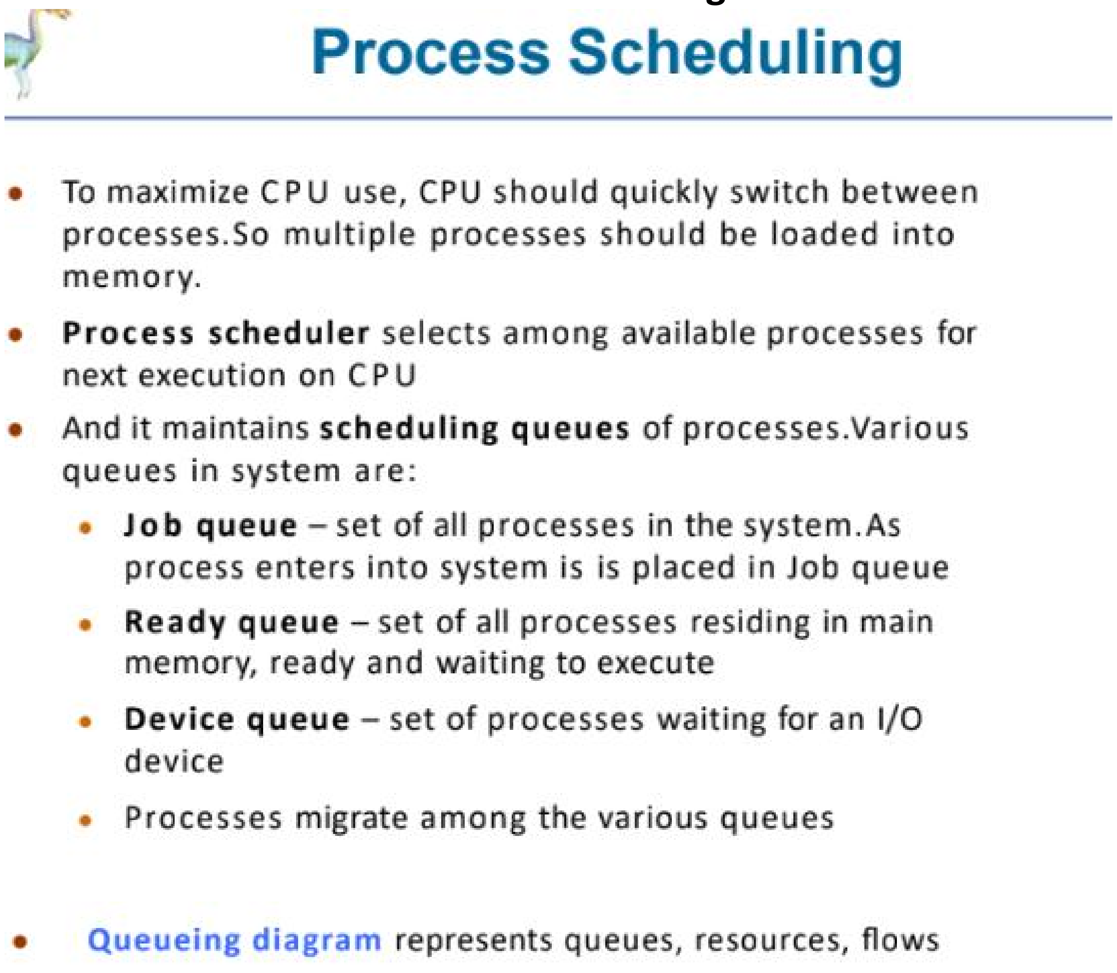
The process's state is stored in a Process Control Block (PCB).

Context switching takes time, and during this time, the system isn't doing any useful work — it's just switching between processes.

If the operating system or PCB is more complex, the switch takes longer.

The switching speed also depends on the hardware. Some CPUs have multiple sets of registers, allowing them to store multiple process states and switch faster.

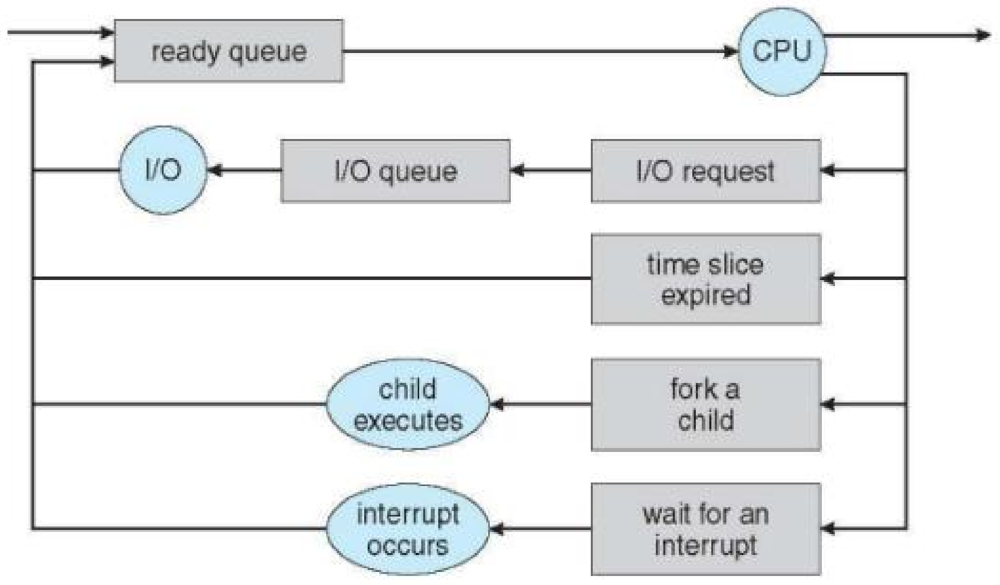
Process Scheduling



Process Scheduling

- To maximize CPU use, CPU should quickly switch between processes.So multiple processes should be loaded into memory.
- **Process scheduler** selects among available processes for next execution on CPU
- And it maintains **scheduling queues** of processes.Various queues in system are:
  - **Job queue** – set of all processes in the system.As process enters into system is is placed in Job queue
  - **Ready queue** – set of all processes residing in main memory, ready and waiting to execute
  - **Device queue** – set of processes waiting for an I/O device
  - Processes migrate among the various queues

- Queueing diagram represents queues, resources, flows



- A common representation of process scheduling is shown in queueing diagram.As soon as any process enters into the system it is kept into job queue i.e. all processes in the system are kept in job queue or secondary storage.
- Long term scheduler or job scheduler select processes from this pool and loads them into memory for execution.
- Now processes got memory and are ready to execute.
- Short term scheduler or CPU scheduler ,select process from memory and allocates CPU to it.
- Process executes on CPU for some time and it needs to wait for following events:

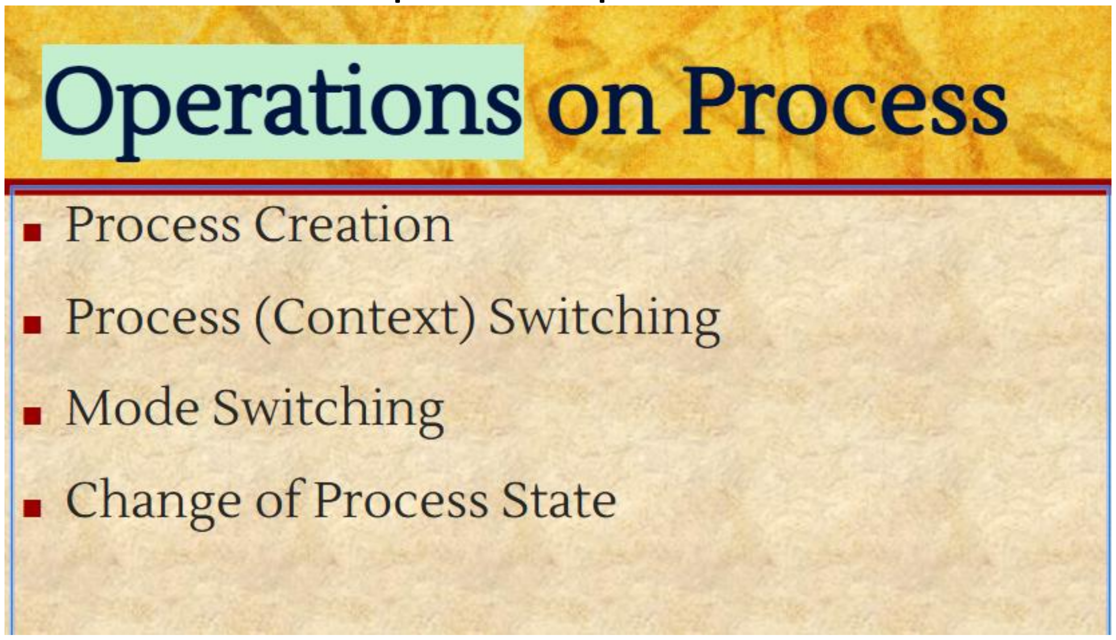
1. Process makes I/O request and waits in I/O or device queue
2. Time slice of process expires and it need more cpu time so again submitted to ready queue
3. Interrupt occurred in the system ,process waits for interrupt to complete
4. Process is forked/divided into child processes and waiting for them to complete

- After finishing all above events process is again submitted to ready queue , and cycle continues and finally when process completes its execution, it is terminated .

Pre-emptive vs Non Pre-emptive

Feature	Preemptive Scheduling	Non-Preemptive Scheduling
Interruption	Process can be interrupted	Process runs till completion
CPU Allocation	Can switch between processes	Once assigned, runs fully
Response Time	Faster response time	Slower response time
Overhead	Higher (due to context switching)	Lower
Example	Round Robin, Priority Scheduling	FCFS, SJF (Non-Preemptive)

Operations on processes



Operations on Process

- Process Creation
- Process (Context) Switching
- Mode Switching
- Change of Process State

Atomic Operations:

Operations that cannot be interrupted or divided during execution. They either complete fully or do not execute at all.

LiveLock

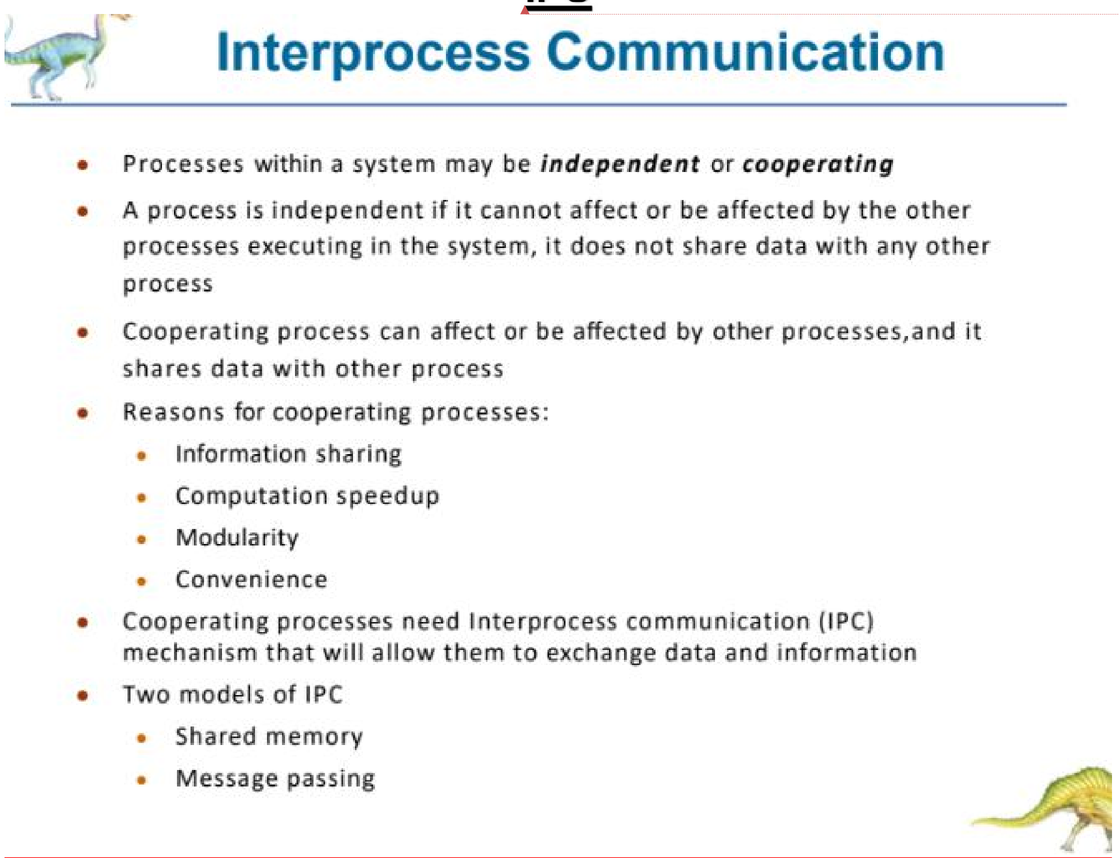
Livelock in OS occurs when two or more processes continuously change their state in response to each other but never make actual progress. It's similar to deadlock, but instead of being stuck, processes keep running without achieving anything useful.

Starvation

Starvation in OS is a problem that occurs when low-priority processes are indefinitely blocked from executing due to high-priority processes.

Formatted[SS21CO002 Faisal Khan]: Left

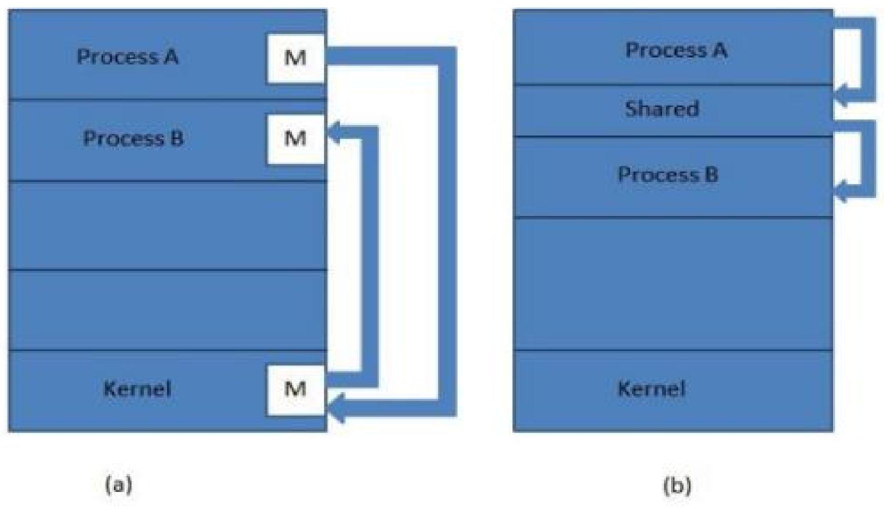
IPC



Interprocess Communication

- Processes within a system may be ***independent*** or ***cooperating***
- A process is independent if it cannot affect or be affected by the other processes executing in the system, it does not share data with any other process
- Cooperating process can affect or be affected by other processes,and it shares data with other process
- Reasons for cooperating processes:
  - Information sharing
  - Computation speedup
  - Modularity
  - Convenience
- Cooperating processes need Interprocess communication (IPC) mechanism that will allow them to exchange data and information
- Two models of IPC
  - Shared memory
  - Message passing

(a) Message passing.   (b) shared memory.



Formatted[SS21CO002 Faisal Khan]: Font: Bold, Font color: Text1, Text Outline, Glow, Shadow, Reflection, Bevel

Formatted[SS21CO002 Faisal Khan]: Centered

Interprocess Communication – Shared Memory

- It requires an area of memory shared among the processes that wish to communicate
- Shared memory region resides in the address space of process creating shared memory segment, other process that wish to communicate must attach it to their address space
- Two processes can exchange information by reading and writing data in the shared area
- Example : Producer Consumer Problem, which is common paradigm for cooperating processes , Producer process produces information that is consumed by consumer process
- Like we think server as a producer and client as a consumer
- Two processes must be sychronised and we must have a buffer that is filled by producer and emptied by consumer. Two types of buffers can be used
- Unbounded buffer
- Bounded buffer

Interprocess Communication – Message Passing

- Second model of IPC is Message Passing. In this method, processes communicate with each other without using any kind of shared memory.
- If two processes p1 and p2 want to communicate with each other, they proceed as follows:

- Establish a communication link
- Start exchanging messages using basic primitives.
  We need at least two primitives:
  – **send**(message, destination) or **send**(message)
  – **receive**(message, host) or **receive**(message)
- Implementation of the link depends on the situation, it can be either a direct communication link or an in-direct communication link.
  Direct Communication links are implemented when the processes use a specific process identifier for the communication
  Indirect Communication is done via a shared mailbox (port), which consists of a queue of messages. The sender keeps the message in mailbox and the receiver picks them up.

Principle of Concurrency

Principles of Concurrency

Concurrency refers to executing multiple tasks **simultaneously** to improve efficiency. The key principles are:

1. **Interleaving & Overlapping** – Multiple processes or threads run in a way that they appear to execute simultaneously.
2. **Resource Sharing** – Multiple processes share CPU, memory, and I/O devices.
3. **Non-Determinism** – The order of execution may vary due to scheduling.
4. **Synchronization** – Mechanisms like locks and semaphores ensure orderly execution.
5. **Mutual Exclusion** – Prevents multiple processes from accessing shared resources simultaneously.
6. **Deadlock Avoidance** – Ensures that processes don't indefinitely wait for resources.
7. **Fairness** – Every process gets a chance to execute, preventing starvation.

These principles help manage multiple processes efficiently while avoiding conflicts.

Producer & Consumer Problem:

Producer-Consumer Problem

It is a **synchronization problem** where:

- The **producer** generates items and places them in a **shared buffer**.
- The **consumer** takes items from the buffer and processes them.

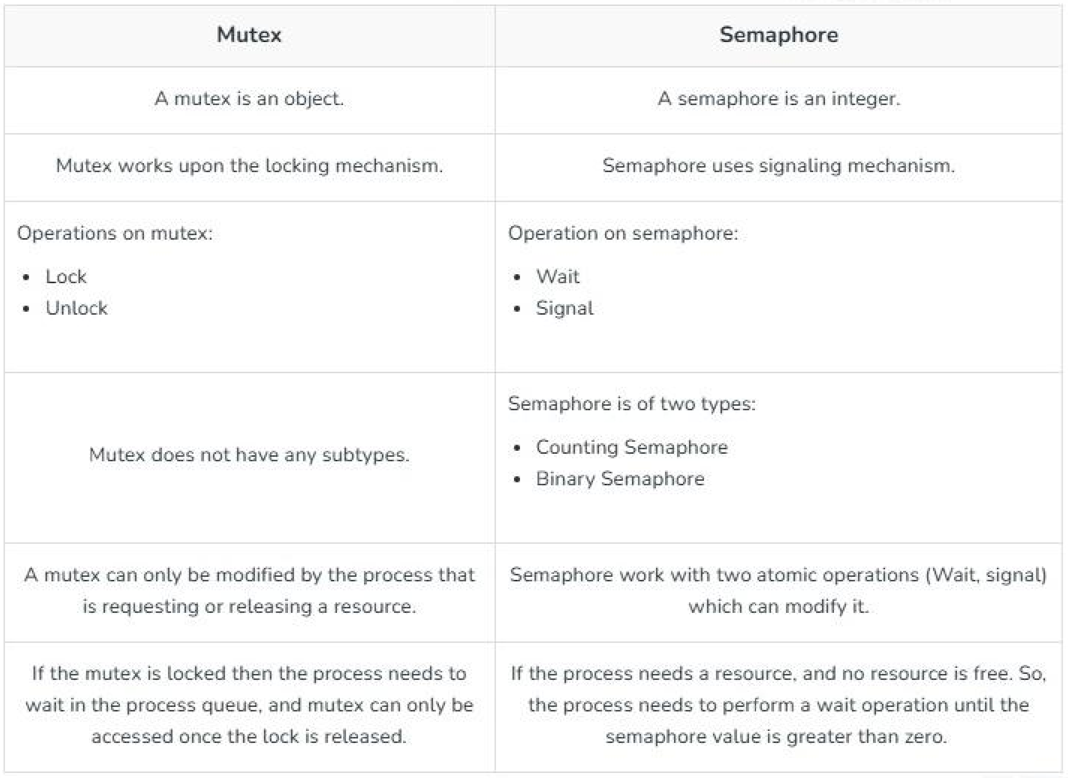
Key Points:

- The buffer has a **bounded size** (fixed capacity).
- **Synchronization** is required to avoid **race conditions**.

Cases:

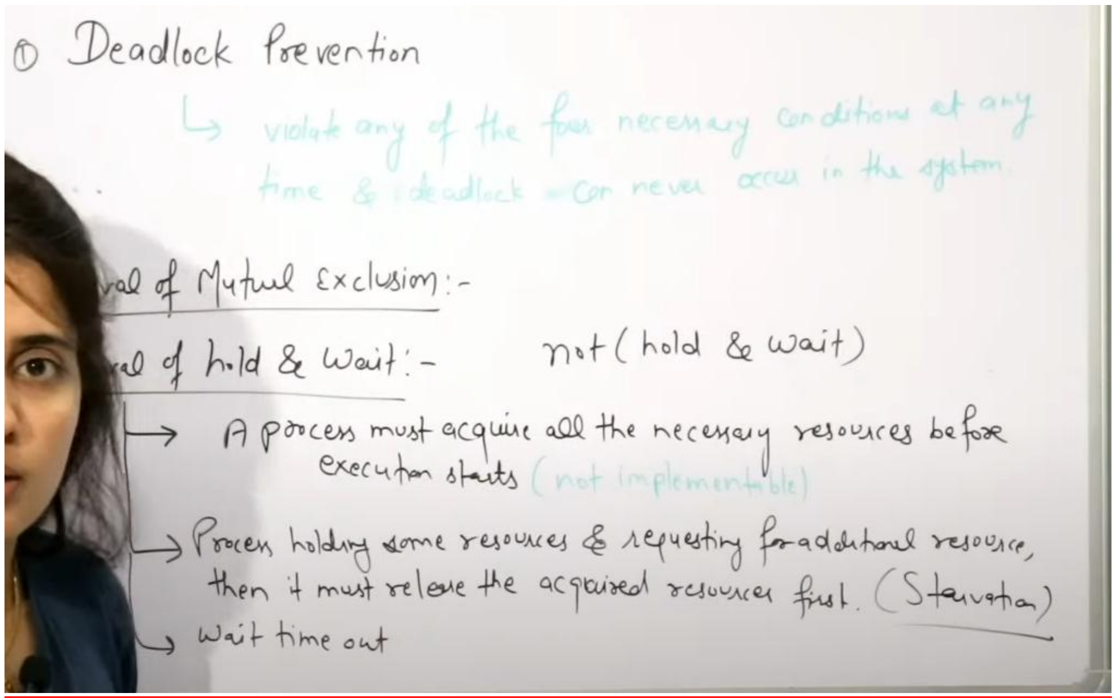
1. **Sequential Execution:**
    - Producer and Consumer run one after another without interference.
2. **Concurrent Execution (Leads to Inconsistency):**
    - Without synchronization, both may access the buffer simultaneously, causing **data corruption** or **inconsistencies**.

Mutex VS Semaphore



Mutex	Semaphore
A mutex is an object.	A semaphore is an integer.
Mutex works upon the locking mechanism.	Semaphore uses signaling mechanism.
Operations on mutex: • Lock • Unlock	Operation on semaphore: • Wait • Signal
Mutex does not have any subtypes.	Semaphore is of two types: • Counting Semaphore • Binary Semaphore
A mutex can only be modified by the process that is requesting or releasing a resource.	Semaphore work with two atomic operations (Wait, signal) which can modify it.
If the mutex is locked then the process needs to wait in the process queue, and mutex can only be accessed once the lock is released.	If the process needs a resource, and no resource is free. So, the process needs to perform a wait operation until the semaphore value is greater than zero.

Deadlock Prevention



① Deadlock Prevention

↳ violate any of the four necessary conditions at any time & deadlock can never occur in the system.

val of Mutual Exclusion :-

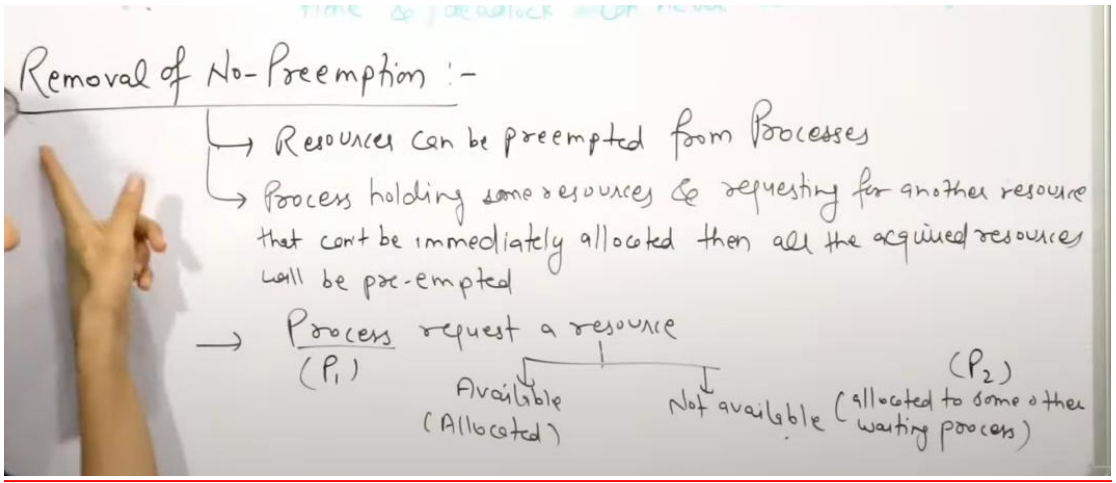
al of hold & Wait :- not (hold & wait)

→ A process must acquire all the necessary resources before execution starts (not implementable)

→ Process holding some resources & requesting for additional resource, then it must release the acquired resources first. (Starvation)

→ wait time out

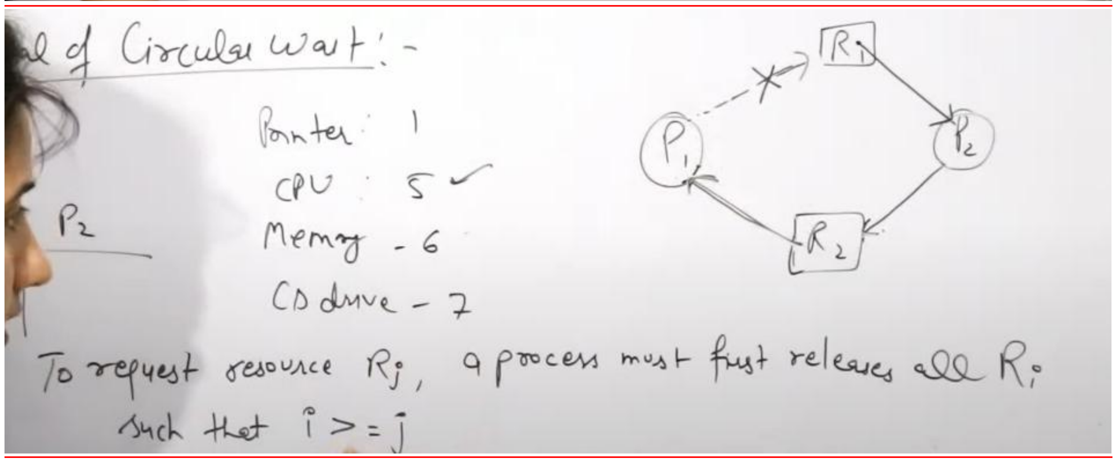
Formatted[SS21CO002 Faisal Khan]: Left



Removal of No-Preemption :-

- Resources can be preempted from Processes
- Process holding some resources & requesting for another resource that can't be immediately allocated then all the acquired resources will be pre-empted

→ Process (P₁) request a resource
- Available (Allocated)
- Not available (P₂) (allocated to some other waiting process)



l of Circular wait :-

Pointer	1
CPU	5 ✓
Memory	6
CD drive	7

P2

To request resource R_j, a process must first releases all R_i such that $i >= j$