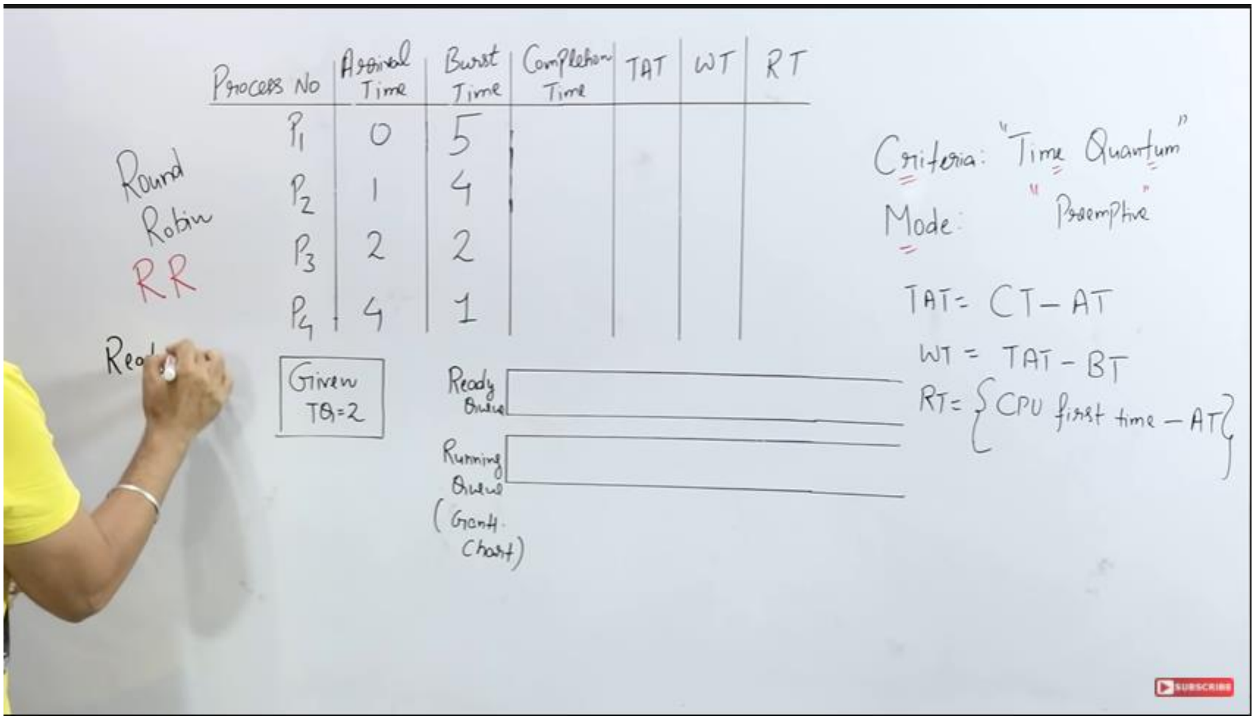
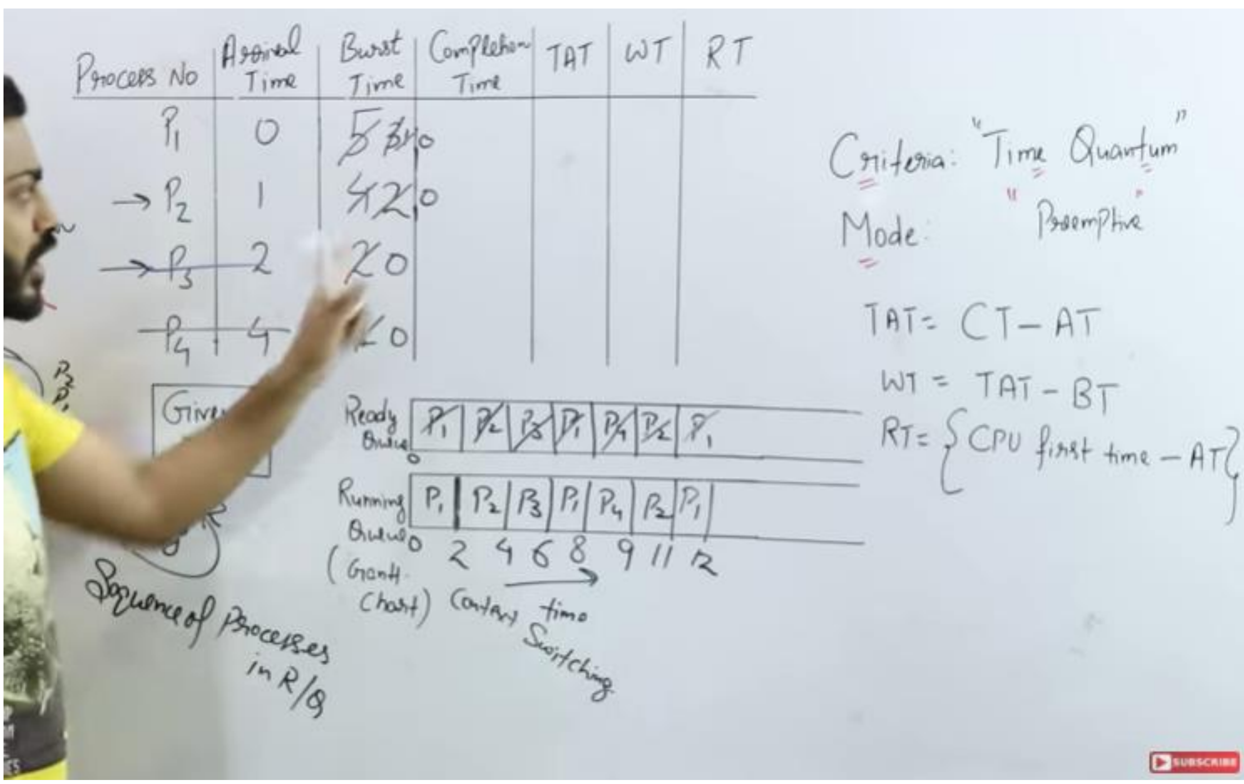
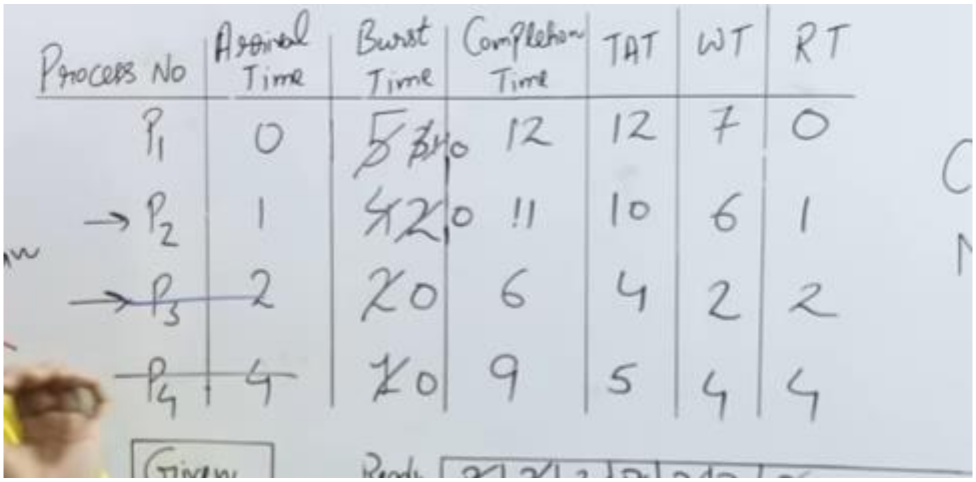
ROUND ROBIN


Process No
Arrival Time
Burst Time
Completion Time
TAT
WT
RT
P1 0 5
P2 1 4
P3 2 2
P4 4 1
Round Robin
RR
Given TQ=2
Ready Queue
Running Queue (Gantt Chart)
Criteria: "Time Quantum"
Mode: "Preemptive"
TAT= CT-AT
WT = TAT-BT
RT= {CPU first time - AT}


Process No
Arrival Time
Burst Time
Completion Time
TAT
WT
RT
P1 0
P2 1
P3 2
P4 4
Ready Queue P1 P2 P3 P1 P4 P2 P1
Running Queue P1 P2 P3 P1 P4 P2 P1
0 2 4 6 8 9 11 12
(Gantt Chart)
Context Switching
time
Sequence of Processes in R/Q
Criteria: "Time Quantum"
Mode: "Preemptive"
TAT= CT-AT
WT = TAT-BT
RT= {CPU first time - AT}



MULTILEVEL QUEUE

Another class of scheduling algorithms has been created for situations in which processes are easily classified into different groups.

For example, A common division is made between foreground(or interactive) processes and background (or batch) processes. These two types of processes have different response-time requirements, and so might have different scheduling needs. In addition, foreground processes may have priority over background processes.

A multi-level queue scheduling algorithm partitions the ready queue into several separate queues. The processes are permanently assigned to one queue, generally based on some property of the process, such as memory size, process priority, or process type. Each queue has its own scheduling algorithm.

Let us consider an example of a multilevel queue-scheduling algorithm with five queues:

1. System Processes
2. Interactive Processes
3. Interactive Editing Processes
4. Batch Processes
5. Student Processes

ADVANTAGES OF MULTILEVEL QUEUE SCHEDULING

With the help of this scheduling we can apply various kind of scheduling for different kind of processes:

For System Processes: First Come First Serve(FCFS) Scheduling.

For Interactive Processes: Shortest Job First (SJF) Scheduling.

For Batch Processes: Round Robin(RR) Scheduling

For Student Processes: Priority Scheduling

DISADVANTAGES OF MULTILEVEL QUEUE SCHEDULING

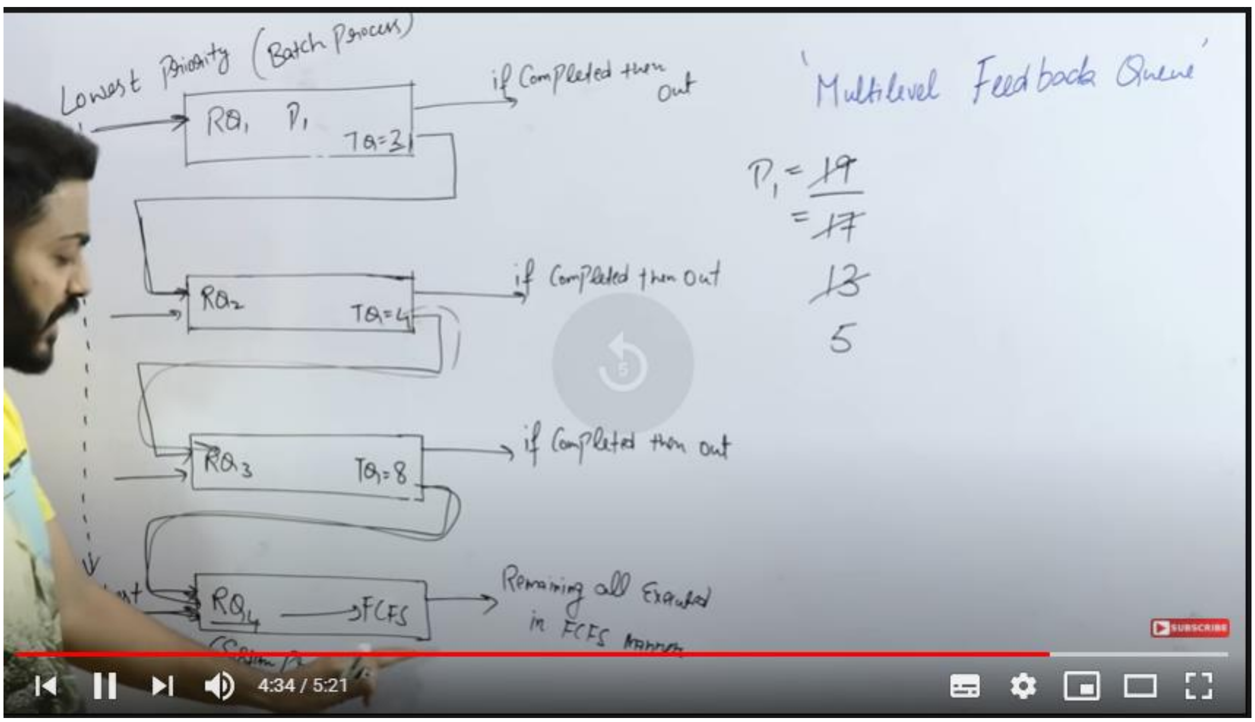
The main disadvantage of Multilevel Queue Scheduling is the problem of starvation for lower-level processes.

Let us understand what is starvation?

Starvation:

Due to starvation lower-level processes either never execute or have to wait for a long amount of time because of lower priority or higher priority process taking a large amount of time.

MULTILEVEL FEEDBACK QUEUE (OVERCOMING STARVATION)



MULTI-PROCESSOR SCHEDULING

In multiple-processor scheduling multiple CPU's are available and hence Load Sharing becomes possible. However multiple processor scheduling is more complex as compared to single processor scheduling. In multiple processor scheduling there are cases when the processors are identical i.e. HOMOGENEOUS, in terms of their functionality, we can use any processor available to run any process in the queue.

Approaches to Multiple-Processor Scheduling

One approach is when all the scheduling decisions and I/O processing are handled by a single processor which is called the Master Server and the other processors executes only the user code. This is simple and reduces the need of data sharing. This entire scenario is called Asymmetric Multiprocessing.

A second approach uses Symmetric Multiprocessing where each processor is self scheduling. All processes may be in a common ready queue or each processor may have its own private queue for ready processes. The scheduling proceeds further by having the scheduler for each processor examine the ready queue and select a process to execute.

Processor Affinity –

Processor Affinity means a processes has an affinity for the processor on which it is currently running. When a process runs on a specific processor there are certain effects on the cache memory. The data most recently accessed by the process populate the cache for the processor and as a result successive memory access by the process are often satisfied in the cache memory. Now if the process migrates to another processor, the contents of the cache memory must be invalidated for the first processor and the cache for the second processor must be repopulated. Because of the high cost of invalidating and repopulating caches, most of the SMP(symmetric multiprocessing) systems try to avoid migration of processes from one processor to another and try to keep a process running on the same processor. This is known as PROCESSOR AFFINITY.

There are two types of processor affinity:

1. Soft Affinity – When an operating system has a policy of attempting to keep a process running on the same processor but not guaranteeing it will do so, this situation is called soft affinity.

2. Hard Affinity – Hard Affinity allows a process to specify a subset of processors on which it may run. Some systems such as Linux implements soft affinity but also provide some system calls like *sched_setaffinity()* that supports hard affinity.

Load Balancing –

Load Balancing is the phenomena which keeps the workload evenly distributed across all processors in an SMP system. Load balancing is necessary only on systems where each processor has its own private queue of process which are eligible to execute. Load balancing is unnecessary because once a processor becomes idle it immediately extracts a runnable process from the common run queue. On SMP(symmetric multiprocessing), it is important to keep the workload balanced among all processors to fully utilize the benefits of having more than one processor else one or more processor will sit idle while other processors have high workloads along with lists of processors awaiting the CPU.

There are two general approaches to load balancing :

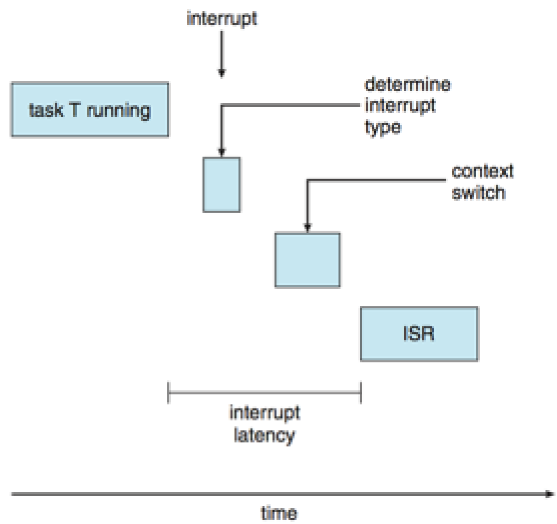
1. Push Migration – In push migration a task routinely checks the load on each processor and if it finds an imbalance then it evenly distributes load on each processors by moving the processes from overloaded to idle or less busy processors.

2. Pull Migration – Pull Migration occurs when an idle processor pulls a waiting task from a busy processor for its execution.

REAL TIME CPU SCHEDULING

Real-Time CPU Scheduling

- **Soft real-time systems** – no guarantee as to when critical real-time process will be scheduled
- **Hard real-time systems** – task must be serviced by its deadline
- Two types of latencies affect performance
  1. Interrupt latency – time from arrival of interrupt to start of routine that services interrupt
  2. Dispatch latency – time for schedule to take current process off CPU and switch to another



THREADS

Thread is a basic unit of CPU utilization.

It comprises of

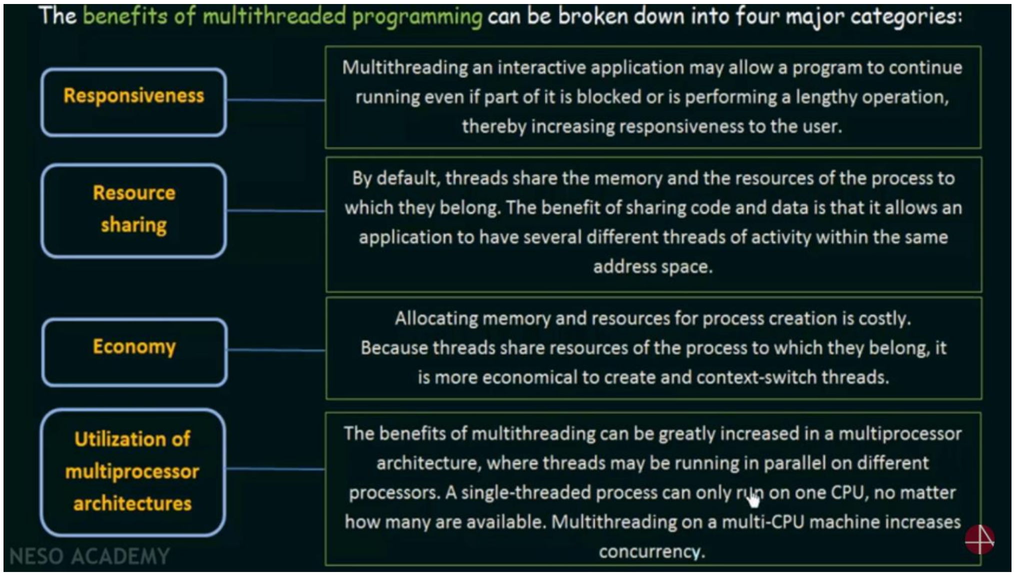
1. Thread ID
2. A Program Counter
3. Register Set
4. Stack

It shares with other threads belonging to the same process its code section, data section, and other operating-system resources, such as open files and signals.

A traditional process has a single thread of control.

If it has multiple threads of control, it can perform more than one task at a time.

BENEFITS OF MULTITHREADING



TYPES OF THREADS

USER THREADS

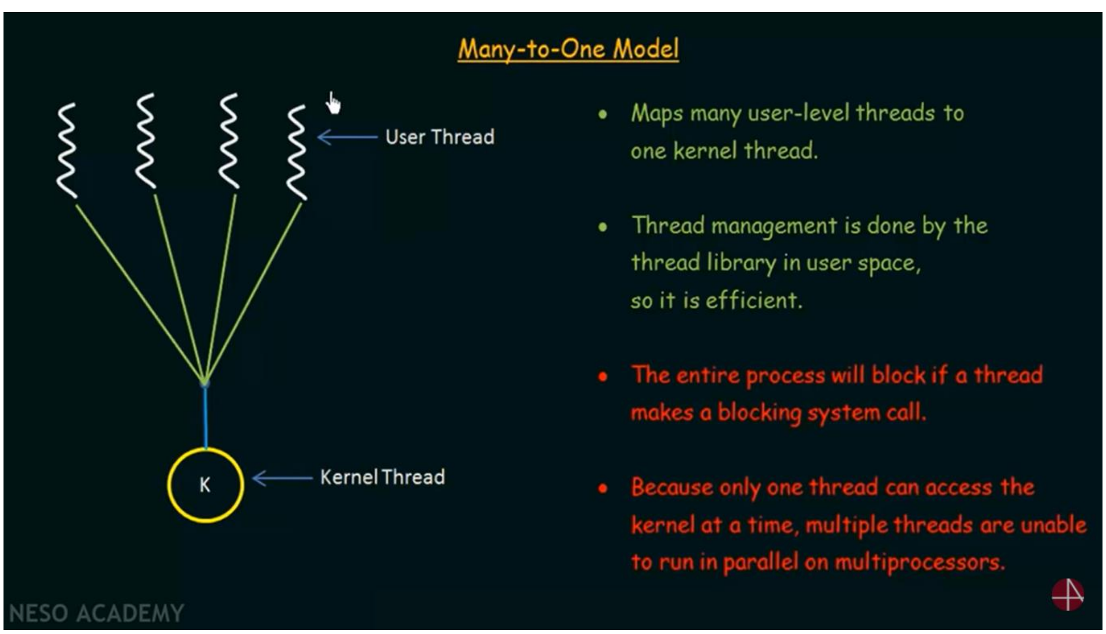
Supported above the kernel and are managed without the support of the Kernel.

KERNEL THREADS

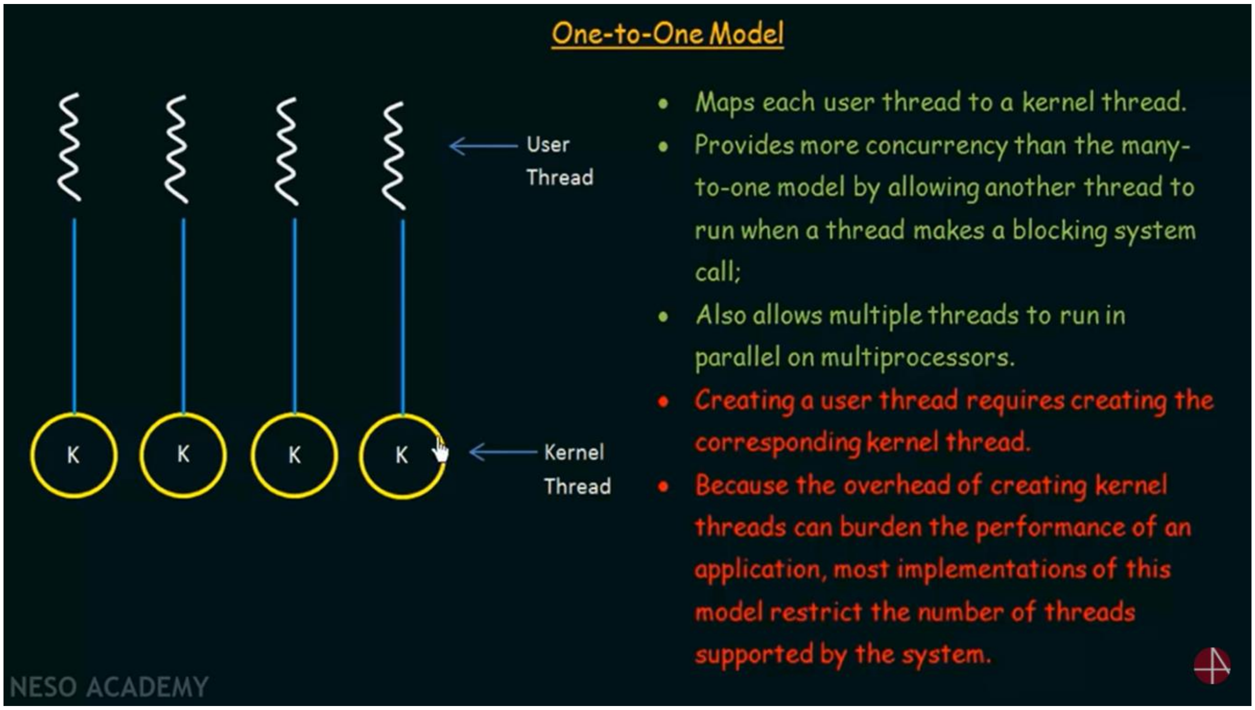
Supported and Managed directly by the Operating System.

THREAD MODELS

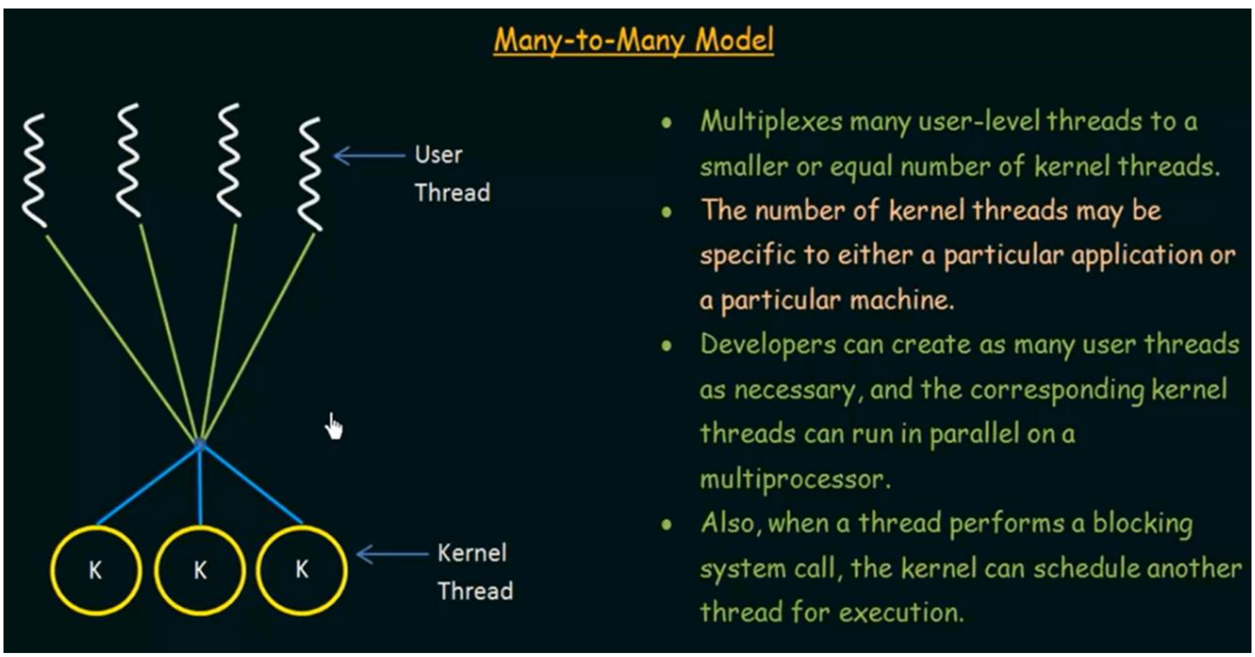
MANY-TO-ONE MODEL



ONE-TO-ONE MODEL



MANY-TO-MANY MODEL



PROCESS VS. THREAD

S.N.	Process	Thread
1.	Process is heavy weight or resource intensive.	Thread is light weight taking lesser resources than a process.
2.	Process switching needs interaction with operating system.	Thread switching does not need to interact with operating system.
3.	In multiple processing environments each process executes the same code but has its own memory and file resources.	All threads can share same set of open files, child processes.
4.	If one process is blocked then no other process can execute until the first process is unblocked.	While one thread is blocked and waiting, second thread in the same task can run.
5.	Multiple processes without using threads use more resources.	Multiple threaded processes use fewer resources.
6.	In multiple processes each process operates independently of the others.	One thread can read, write or change another thread's data.

THREAD STATES

1. Spawn → One Threads spawn another thread within the process.
2. Blocked → Threads is waiting for an event.
3. Unblocked → The event had occurred.
4. Finished → The thread is done doing the task it was assigned to do.

OPENMP

What is OpenMP?

- An Application Program Interface (API) that may be used to explicitly direct multithreaded, shared memory parallelism

- Three main API components
  - Compiler directives
  - Runtime library routines
  - Environment variables

- Portable & Standardized
  - API exist both C/C++ and Fortan 90/77
  - Multi platform Support (Unix, Linux etc.)

TYPES OF PARALLELISM

DATA PARALLELISM

Focus on distributing data across different parallel computing nodes.

TASK PARALLELISM

Focus on distributing threads across different parallel computing nodes.

Data Parallelism	Task Parallelism
Same operations are performed on different subsets of same data.	Different operations are performed on the same or different data.
Synchronous computation	Asynchronous computation
Speedup is more as there is only one execution thread operating on all sets of data.	Speedup is less as each processor will execute a different thread or process on the same or different set of data.
Amount of parallelization is proportional to the input data size.	Amount of parallelization is proportional to the number of independent tasks to be performed
Designed for optimum load balance on multi processor system.	Load balancing depends on the availability of the hardware and scheduling algorithms like static and dynamic scheduling.

AMDAHL'S LAW

$$speedup \leq \frac{1}{S + \frac{(1-S)}{N}}$$

S → Portion of program executed serially.

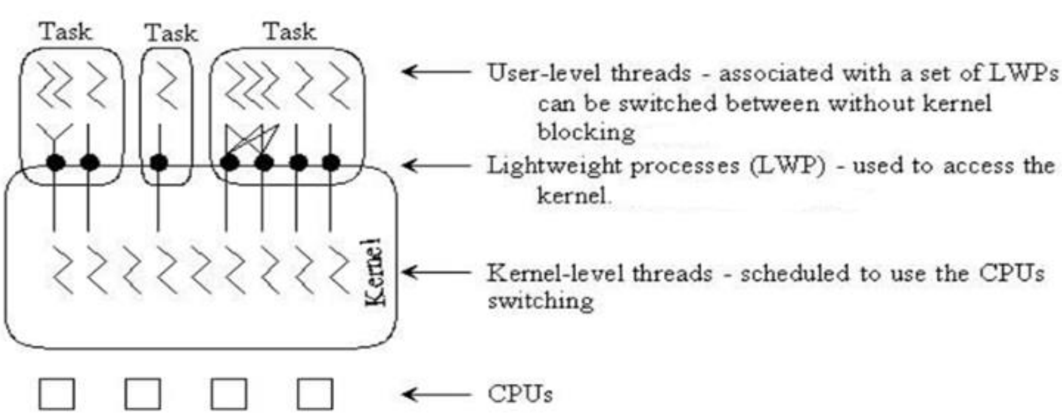
N → No of cores.

THREAD SCHEDULING

CONTENTION SCOPE

- **Contention Scope:**
- On systems implementing the many-to-one and many-to-many models, the thread library schedules user-level threads to run on an available LWP. This scheme is known as **process contention scope (PCS),**
- (When we say the thread library *schedules* user threads onto available LWPs, we do not mean that the threads are actually running on a CPU. That would require the operating system to schedule the kernel thread onto a physical CPU.) To decide which kernel-level thread to schedule onto a CPU, the kernel uses **system-contention scope (SCS).**

User vs. Kernel Thread



PROCESS SYNCHRONIZATION

CO-OPERATIVE PROCESS

In this process the execution of one process affects the execution of other.

Two or more process share a common variable.

They can share some memory or buffer.

They can share some common code.

INDEPENDENT PROCESS

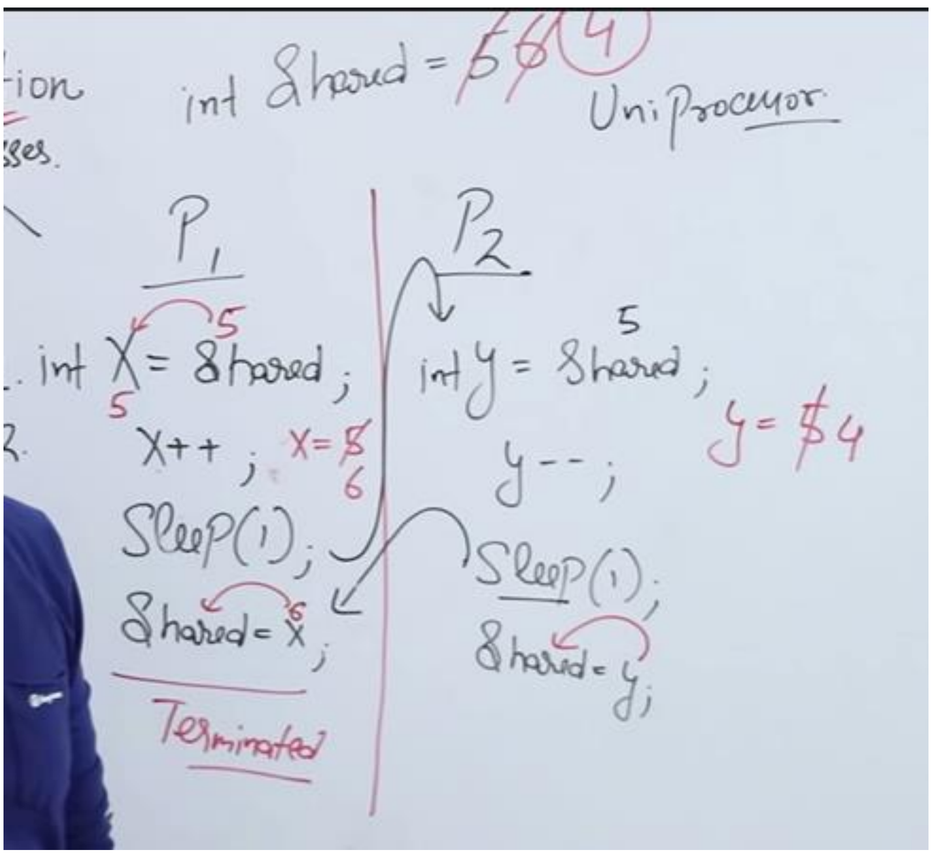
Execution of one process doesn't effect the other.

PROCESS SYNCHRONIZATION

It is essential that the co-operative process are synched properly else they might create problems like dreadlocks.

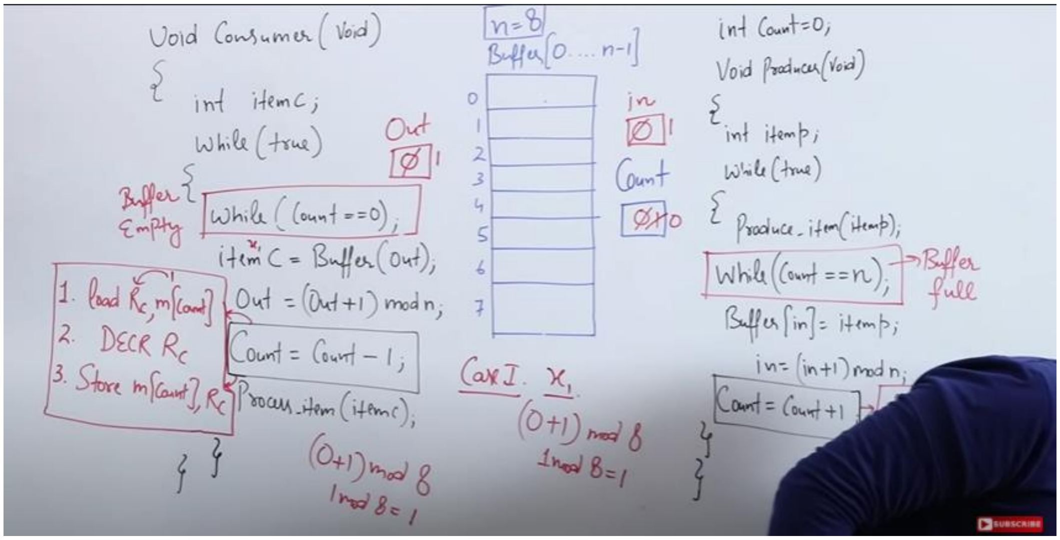
RACE CONDITION

A situation where several processes access and manipulate the same data concurrently and the outcome of the execution depends on the particular order in which the access takes place.

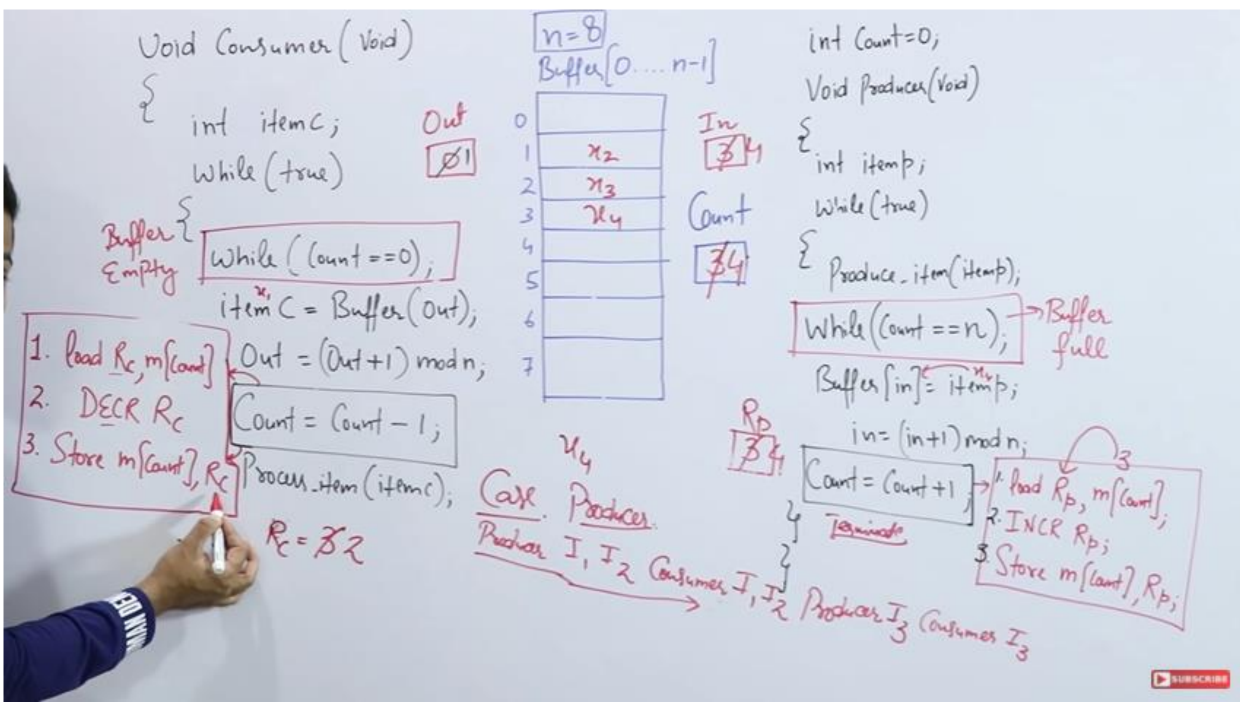


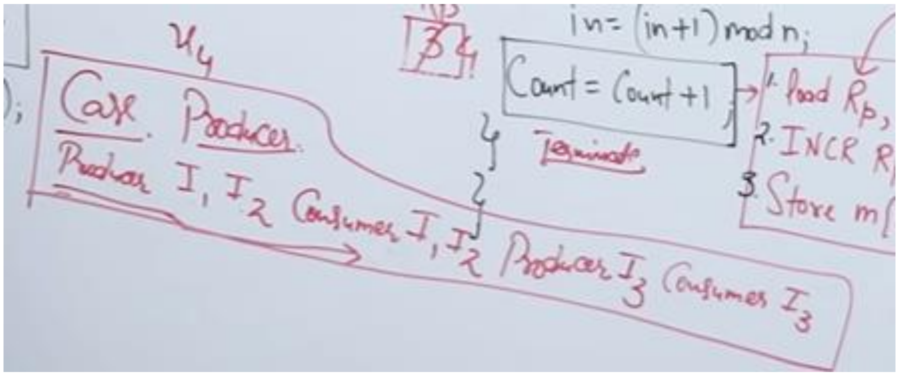
PRODUCER – CONSUMER PROBLEM

BEST CASE – PERFECT SYNC



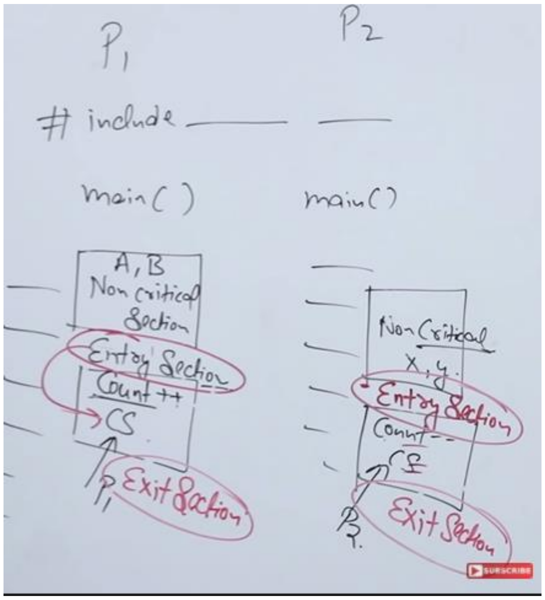
SECOND CASE – FAILED SYNC





CRITICAL SECTION

It is a part of a program where shared resouces are accessed by varoius programs.



CONDITIONS REQUIRED TO AVOID RACE CONDITION:

No two processes may be simultaneously inside their critical regions.

No assumptions may be made about speeds or the number of CPUs.

No process running outside its critical region may block other processes.

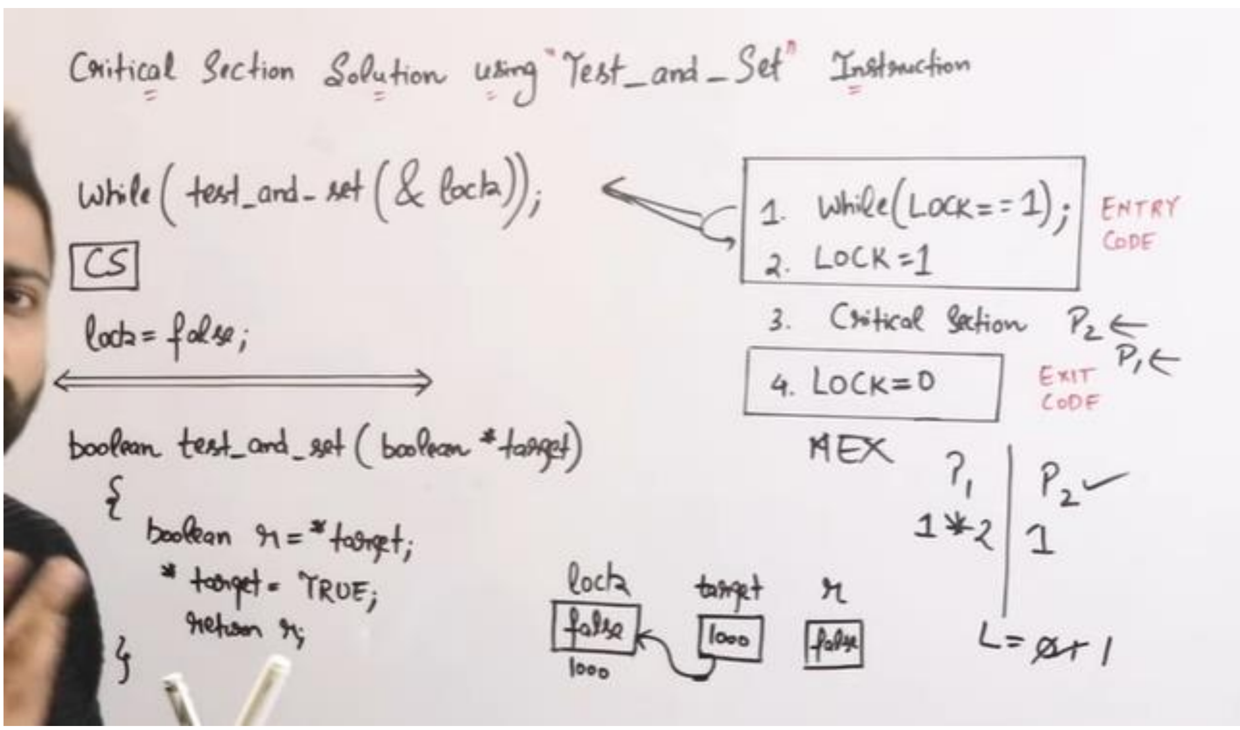
No process should have to wait forever to enter its critical region.

SYNC MECHANISM – 4 RULES

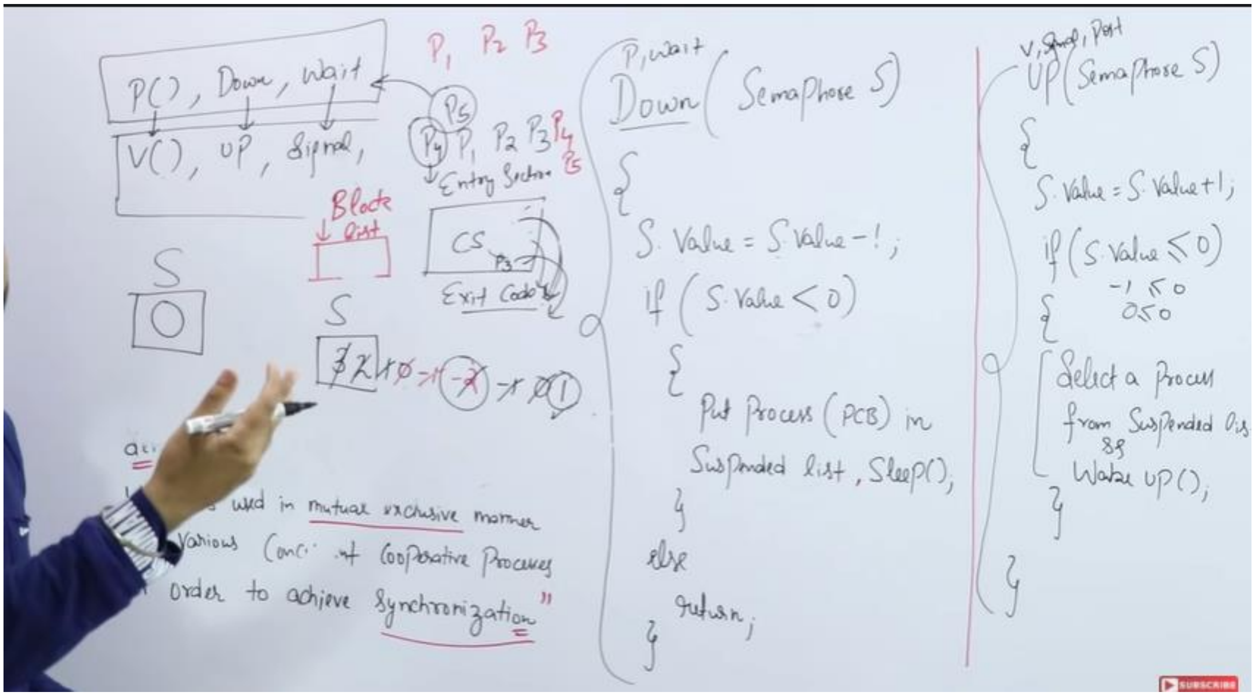
1. Mutual Exclusion.
2. Progress.
3. Bounded Wait.
4. No assumptions related to Hardware speed.

1. **Mutual Exclusion** - If process P_i is executing in its critical section, then no other processes can be executing in their critical sections
2. **Progress** - If no process is executing in its critical section and there exist some processes that wish to enter their critical section, then the selection of the processes that will enter the critical section next cannot be postponed indefinitely
3. **Bounded Waiting** - A bound must exist on the number of times that other processes are allowed to enter their critical sections after a process has made a request to enter its critical section and before that request is granted
   - Assume that each process executes at a nonzero speed
   - No assumption concerning **relative speed** of the ***n*** processes

TEST AND SET



SEMAPHORES



S = 0

Means there are 0 processes in the suspended/waiting list

S = -1

There is 1 processes in the suspenede/waiting list.

S = 10

It means, 10 process can enter the Critical section.

Deadlock Vs. Starvation

- Deadlock refers to the situation when processes are stuck in circular waiting for the resources.

- On the other hand, starvation occurs when a process waits for a resource indefinitely.