Unit 2

PART – A: (Multiple Choice Questions) **2x10=20 Marks**

1	Which module gives control of the CPU to the process selected by the short-term scheduler? a) dispatcher b) interrupt c) scheduler d) none of the mentioned
2	The interval from the time of submission/response of a process to the time of completion is termed as a) waiting time b) turnaround time c) response time d) throughput
3	In priority scheduling algorithm, when a process arrives at the ready queue, its priority is compared with the priority of: a) all process b) currently running process c) parent process d) init process
4	Which one of the following is a system call? a) process control b) process scheduling c) process d) none of the mentioned
5	The number of process completed in unit of time is known as : a) Response time. b) Turnaround time. c) Throughput. d) None of the above
6	Which module gives control of the CPU to the process selected by the short-term scheduler? A. dispatcher B. interrupt C. scheduler D. none of the mentioned
7	The processes that are residing in main memory and are ready and waiting to execute are kept on a list called: A. job queue B. ready queue C. execution queue D. process queue
8	The interval from the time of submission of a process to the time of completion is termed as: A. waiting time B. turnaround time C. response time D. throughput
9	In priority scheduling algorithm: A. CPU is allocated to the process with highest priority B. CPU is allocated to the process with lowest priority C. equal priority processes cannot be scheduled D. none of the mentioned

10	In multilevel feedback scheduling algorithm: A. a process can move to a different classified ready queue B. classification of ready queue is permanent C. processes are not classified into groups **D.** none of the mentioned
11	With multiprogramming, _____ is used productively. A. time B. space C. money **D.** All of these
12	An I/O bound program will typically have : A. a few very short CPU bursts B. many very short I/O bursts C. many very short CPU bursts **D.** a few very short I/O bursts
13	Round robin scheduling falls under the category of : A. Non preemptive scheduling B. Preemptive scheduling **C.** None of these
14	Which of the following algorithms tends to minimize the process flow time ? A. First come First served B. Shortest Job First C. Earliest Deadline First **D.** Longest Job First
15	Under multiprogramming, turnaround time for short jobs is usually _____ and that for long jobs is slightly _____. A. Lengthened; Shortened B. Shortened; Lengthened C. Shortened; Shortened **D.** Shortened; Unchanged

UNIT-2

PART – B: (Short Answer Questions)

1 **Differentiate between preemptive and non-preemptive scheduling.**

S.NO.	PREEMPTIVE SCHEDULING	NON-PREEMPTIVE SCHEDULING
1.	The CPU is allocated to the processes for a certain amount of time.	The CPU is allocated to the process till it ends it's execution or switches to waiting state.
2.	The executing process here is interrupted in the middle of execution.	The executing process here is not interrupted in the middle of execution.
3.	It usually switches the process from ready state to running state and vise-versa, and maintains the ready queue.	It does not switch the process from running state to ready state.
4.	Here, if a process with high priority frequently arrives in the ready queue then the process with low priority has to wait for long, and it may have to starve.	Here, if CPU is allocated to the process with larger burst time then the processes with small burst time may have to starve.
5.	It is quite flexible because the critical processes are allowed to access CPU as they arrive into the ready queue, no matter what process is executing currently.	It is rigid as even if a critical process enters the ready queue the process running CPU is not disturbed.

2 **What is starvation? How it is resolved.**

Priority scheduling can suffer from a major problem known as indefinite blocking, or starvation, in which a low-priority task can wait forever because there are always some other jobs around that have higher priority.

- If this problem is allowed to occur, then processes will either run eventually when the system load lightens or will eventually get lost when the system is shut down or crashes.

One common solution to this problem is aging, in which priorities of jobs increase the longer they wait. Under this scheme a low-priority job will eventually get its priority raised high enough that it gets run

3 **Define and differentiate between preemptive and non-preemptive scheduling.**

Pre-emptive Scheduling means once a process started its execution, the currently running

process can be paused for a short period of time to handle some other process of higher priority, it means we can pre-empt the control of CPU from one process to another if required.

Non-Pre-emptive Scheduling means once a process starts its execution or the CPU is processing a specific process, it cannot be halted or in other words we cannot preempt (take control) the CPU to some other process.

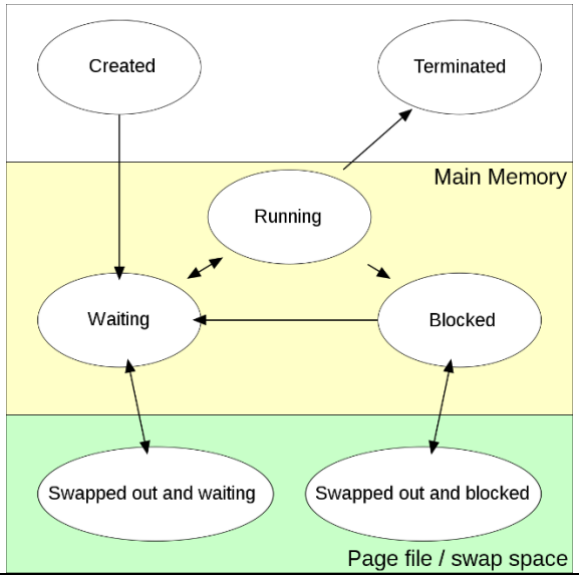
4 **What do you mean by processor affinity? How it is helpful in multiprocessing scheduling?**

Processor affinity, or CPU pinning, enables the binding and unbinding of a process or a thread to a central processing unit (CPU) or a range of CPUs, so that the process or thread will execute only on the designated CPU or CPUs rather than any CPU.

5 **Differentiate between turnaround time and waiting time.**

TURN AROUND TIME	WAITING TIME
The time since the process entered into ready queue for execution till the process completed it's execution.	The time process spent in the ready queue and for I/O completion.
Different CPU Scheduling algorithms produce different TAT for the same set of processes.	CPU Scheduling Algorithm doesn't affect the amount of time during which a process executes or does I/O but only the amount of time that a process spends waiting in the ready queue.
The turnaround time is generally limited by the speed of the output device.	Waiting time has no such major effect.

6 **Draw a neat process state diagram.**



7 **What do you mean by dispatch latency?**

The **dispatcher** is the module that gives control of the CPU to the

	process selected by the scheduler. The dispatcher needs to be as fast as possible, as it is run on every context switch. The time consumed by the dispatcher is known as **dispatch latency**
8	**Differentiate between a *process* and a *program*.**

PROGRAM	PROCESS
Program contains a set of instructions designed to complete a specific task.	Process is an instance of an executing program.
Program is a passive entity as it resides in the secondary memory.	Process is a active entity as it is created during execution and loaded into the main memory.
Program exists at a single place and continues to exist until it is deleted.	Process exists for a limited span of time as it gets terminated after the completion of task.
Program is a static entity.	Process is a dynamic entity.
Program does not have any resource requirement, it only requires memory space for storing the instructions.	Process has a high resource requirement, it needs resources like CPU, memory address, I/O during its lifetime.
Program does not have any control block.	Process has its own control block called Process Control Block.

9	**What do you mean by critical section?** Critical section is a code segment that can be accessed by only one process at a time. Critical section contains shared variables which need to be synchronized to maintain consistency of data variables.
10	**What do you mean by mutual exclusion?** If a process is executing in its critical section, then no other process is allowed to execute in the critical section.
11	**What are the different *scheduling criteria* for CPU scheduling?** • CPU Utilization • Throughput • Turnaround Time • Waiting time • Response time
12	**What is semaphore? What operations can be performed on a semaphore?** **semaphore** is a variable which can hold only a non-negative Integer value, shared between all the threads. **wait** and **signal** operations can be performed on a semaphore.
13	**What requirement is to be satisfied for a solution of a critical section problem?** • **Mutual Exclusion** : If a process is executing in its critical section, then no other process is allowed to execute in the critical section. • **Progress** : If no process is in the critical section, then no other process from outside can block it from entering the critical section.

	• **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.
14	**What is the use of cooperating processes?** • **Modularity:** Modularity involves dividing complicated tasks into smaller subtasks • **Information Sharing:** Sharing of information between multiple processes can be accomplished using cooperating processes • **Convenience:** There are many tasks that a user needs to do such as compiling, printing, editing etc. It is convenient if these tasks can be managed by cooperating processes. • **Computation Speedup:** Subtasks of a single task can be performed parallely using cooperating processes
15	**What are the requirements that a solution to the critical section problem must satisfy?** (Answer same as NO.13)
16	**What are the benefits of multithreaded programming?** Responsiveness: Program responsiveness allows a program to run even if part of it is blocked using multithreading. Resource sharing: hence allowing better utilization of resources. Economy: Creating and managing threads becomes easier. Scalability: One thread runs on one CPU. In Multithreaded processes, threads can be distributed over a series of processors to scale.

UNIT-2

PART – C: (Long Answer Questions)

A	**Differentiate between Preemptive and Non-Preemptive scheduling.** **Preemtive Scheduling** **Non-Preemtive Scheduling**	**[5]**

	In this resources(CPU Cycle) are allocated to a process for a limited time.	Once resources(CPU Cycle) are allocated to a process, the process holds it till it completes its burst time or switches to waiting state.
	Process can be interrupted in between.	Process can not be interrupted until it terminates itself or its time is up.
	If a process having high priority frequently arrives in the ready queue, low priority process may starve.	If a process with long burst time is running CPU, then later coming process with less CPU burst time may starve.
	It has overheads of scheduling the processes.	It does not have overheads.
	flexible	rigid
	cost associated	no cost associated
	In preemptive scheduling, CPU utilization is high.	It is low in non preemptive scheduling.
	Examples of preemptive scheduling are Round Robin and Shortest Remaining Time First.	Examples of non-preemptive scheduling are First Come First Serve and Shortest Job First.

B

Consider the following set of processes with their CPU Burst time, arrival time given in milliseconds and priority.

Process	CPU Burst time	Arrival time	Priority
P1	3	0	1
P2	2	1	0
P3	4	3	2
P4	5	4	0
P5	3	5	1

Draw three Gantt charts for execution of the processes using SRTF, RR (Time quantum=2) and preemptive priority scheduling. Separately compute average waiting time and average turnaround time of the processes on execution of the three algorithms. **[10]**

A

Explain Peterson's solution on critical section problem.

Peterson's solution is a software based solution to the critical section problem. Consider two processes P0 and P1. For convenience, when presenting Pi, we use Pi to denote the other process; that is, j == 1 - i.

The processes share two variables:

boolean flag [2] ;

int turn;

Initially flag [0] = flag [1] = false, and the value of turn is immaterial (but is either 0 or 1). The structure of process Pi is shown below.

do{

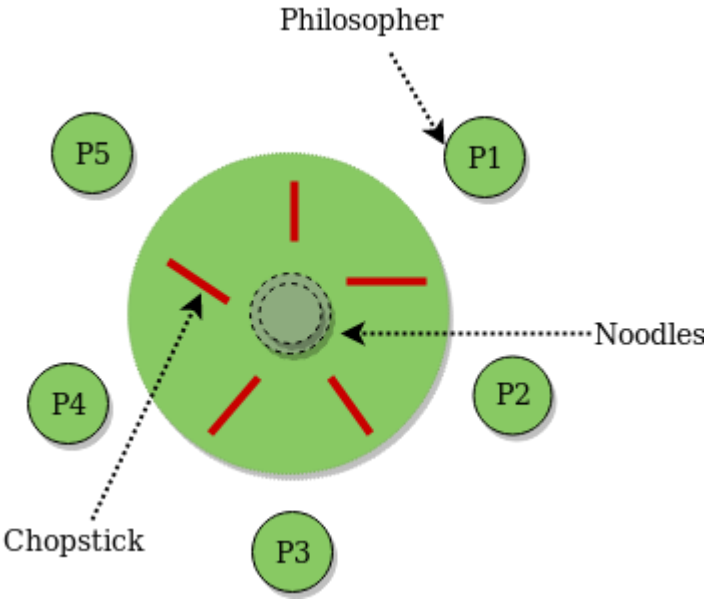
flag[i]=true

	turn=j while(flag[j] && turn==j); critical section flag[i]=false Remainder section }while(1); To enter the critical section, process Pi first sets flag [il to be true and then sets turn to the value j, thereby asserting that if the other process wishes to enter the critical section it can do so. If both processes try to enter at the same time, turn will be set to both i and j at roughly the same time. Only one of these assignments will last; the other will occur, but will be overwritten immediately. The eventual value of turn decides which of the two processes is allowed to enter its critical section first. We now prove that this solution is correct. We need to show that: 1. Mutual exclusion is preserved, 2. The progress requirement is satisfied, 3. The bounded-waiting requirement is met. To prove property 1, we note that each Pi enters its critical section only if either flag [jl == false or turn == i. Also note that, if both processes can be executing in their critical sections at the same time, then flag [i] ==flag [jl == true. These two observations imply that P0 and P1 could not have successfully executed their while statements at about the same time, since the value of turn can be either 0 or 1, but cannot be both. Hence, one of the processes say Pj-must have successfully executed the while statement, whereas Pi had to execute at least one additional statement ("turn == j"). However, since, at that time, flag [j] == true, and turn == j, and this condition will persist as long as Pi is in its critical section, the result follows: To prove properties 2 and 3, we note that a process Pi can be prevented from entering the critical section only if it is stuck in the while loop with the condition flag [j] == true and turn == j; this loop is the only one. If Pi is not ready to enter the critical section, then flag [j] == false and Pi can enter its critical section. If Pi has set flag[j] to true and is also executing in its while statement, then either turn == I or turn == j. If turn == i, then Pi will enter the critical section. If turn == j, then Pi will enter the critical section. However, once Pi exits its critical section, it will reset flag [jl to false, allowing Pi to enter its critical section. If Pi resets flag [j 1 to true, it must also set turn to i. Thus, since Pi does not change the value of the variable turn while executing the while statement, Pi will enter the critical section (progress) after at most one entry by Pi (bounded waiting).

B	**What do you mean by Semaphore? Discuss the Dining Philosophers problem using semaphore.**
	A semaphore S is an integer variable that, apart from initialization, is accessed only through two standard atomic operations: wait and signal. These operations were originally termed P (for wait; from the Dutch proberen, to test) and V (for signal; from verhogen, to increment). The classical definition of wait in pseudocode is
	wait(S) {
	while (S <= 0)
	; // no-op
	S --;
	}
	The classical definitions of signal in pseudocode is
	Signal(S){
	S++;

}

The Dining Philosopher Problem – The Dining Philosopher Problem states that K philosophers seated around a circular table with one chopstick between each pair of philosophers. There is one chopstick between each philosopher. A philosopher may eat if he can pickup the two chopsticks adjacent to him. One chopstick may be picked up by any one of its adjacent followers but not both.



Dining-Philosophers Problem

• The structure of Philosopher i:

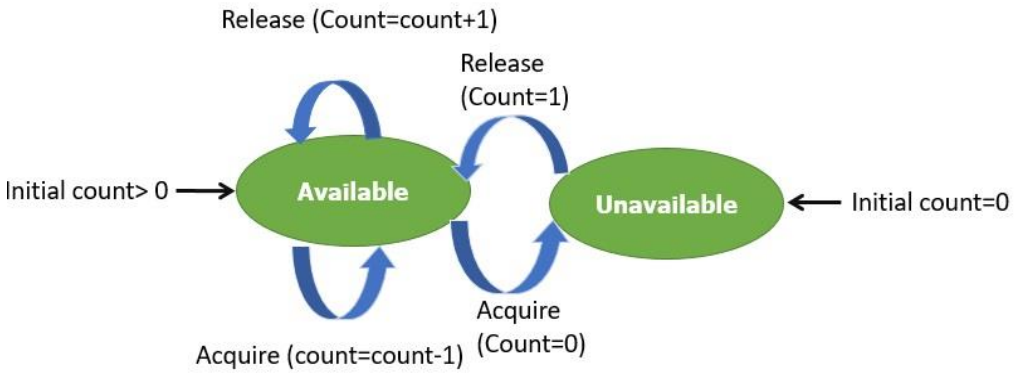
```
process P[i]
 while true do
  {  THINK;
    PICKUP(CHOPSTICK[i], CHOPSTICK[(i+1)%5]);
    EAT;
    PUTDOWN(CHOPSTICK[i], CHOPSTICK[(i+1)%5])
  }
```

There are three states of philosopher : **THINKING, HUNGRY and EATING**. Here there are two semaphores : Mutex and a semaphore array for the philosophers.

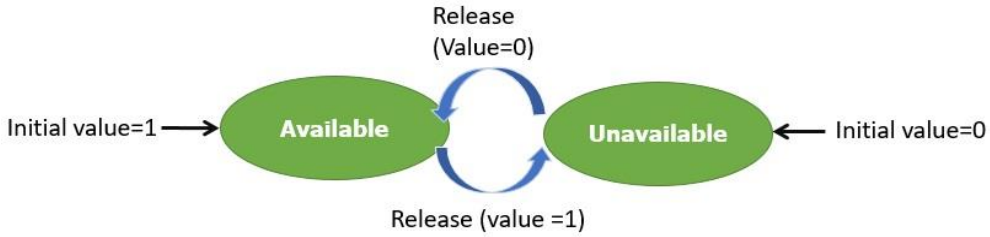
A

What do you mean by binary semaphore and counting semaphore? Explain implementation of wait () and signal.

Counting Semaphore: This type of Semaphore uses a count that helps task to be acquired or released numerous times. If the initial count = 0, the counting semaphore should be created in the unavailable state. The value of the Counting Semaphore can ranges over an unrestricted domain.



Binary Semaphore: The binary semaphores are quite similar to counting semaphores, but their value is restricted to 0 and 1. In this type of semaphore, the wait operation works only if semaphore = 1, and the signal operation succeeds when semaphore= 0. It is easy to implement than counting semaphores.



Wait( ) Operation: This type of semaphore operation helps you to control the entry of a task into the critical section. However, If the value of wait is positive, then the value of the wait argument X is decremented. In the case of negative or zero value, no operation is executed. It is also called P(S) operation.After the semaphore value is decreased, which becomes negative, the command is held up until the required conditions are satisfied.

Implementation of wait:

```
wait (S){
value--;
if (value < 0) {
add this process to waiting queue
block(); }
}
```

Signal operation: This type of Semaphore operation is used to control the exit of a task from a critical section. It helps to increase the value of the argument by 1, which is denoted as V(S).

Implementation of Signal:

```
Signal (S){
value++;
if (value <= 0) {
remove a process P from the waiting queue
wakeup(P); }
```

B	**Consider the following five processes, with the length of the CPU burst time given in milliseconds.** **Process Burst time P1 - 10, P2 - 29, P3 - 3, P4 – 7, P5 - 12 Consider the First come First serve (FCFS), Non Preemptive Shortest Job First (SJF), Round Robin (RR) with (quantum=10ms) scheduling algorithms. Illustrate the scheduling using Gantt chart.** • **Which algorithm will give the minimum average waiting time?** **Ans:**The Gantt-chart for FCFS scheduling is P1 P2 P3 P4 P5 0 10 39 42 49 61 Turnaround time = Finished Time – Arrival Time Turnaround time for process P1 = 10 – 0 = 10 Turnaround time for process P2 = 39 – 0 = 39 Turnaround time for process P3 = 42 – 0 = 42 Turnaround time for process P4 = 49 – 0 = 49 Turnaround time for process P5 = 61 – 0 = 61 Average Turnaround time = (10+39+42+49+61)/5 = 40.2 The Gantt-chart for SJF scheduling is P3 P4 P1 P5 P2 0 3 10 20 32 61 Turnaround time for process P1 = 3 – 0 = 3 Turnaround time for process P2 = 10 – 0 = 10 Turnaround time for process P3 = 20 – 0 = 20 Turnaround time for process P4 = 32 – 0 = 32 Turnaround time for process P5 = 61 – 0 = 61 Average Turnaround time = (3+10+20+32+61)/5 = 25.2 The Gantt-chart for RR scheduling is P1 P2 P3 P4 P5 P2 P5 P2 0 10 20 23 30 40 50 52 61 Turnaround time for process P1 = 10 – 0 = 10 Turnaround time for process P2 = 61 – 0 = 61 Turnaround time for process P3 = 23 – 0 = 23 Turnaround time for process P4 = 30 – 0 = 30 Turnaround time for process P5 = 52 – 0 = 52 Average Turnaround time = (10+61+23+30+52)/5 = 44.2 So SJF gives minimum turnaround time.
A	**Explain the different criteria used in operating system during scheduling and what importance they are having in choosing the optimal algorithm for a given snapshot**? **Scheduling Criteria**

1. **CPU utilisation –**
   The main objective of any CPU scheduling algorithm is to keep the CPU as busy as possible. Theoretically, CPU utilisation can range from 0 to 100 but in a real-time system, it varies from 40 to 90 percent depending on the load upon the system.

2. **Throughput –**
   A measure of the work done by CPU is the number of processes being executed and completed per unit time. This is called throughput. The throughput may vary depending upon the length or duration of processes.

3. **Turnaround time –**
   For a particular process, an important criteria is how long it takes to execute that process. The time elapsed from the time of submission of a process to the time of completion is known as the turnaround time. Turn-around time is the sum of times spent waiting to get into memory, waiting in ready queue, executing in CPU, and waiting for I/O.

4. **Waiting time –**
   A scheduling algorithm does not affect the time required to complete the process once it starts execution. It only affects the waiting time of a process i.e. time spent by a process waiting in the ready queue.

5. **Response time –**
   In an interactive system, turn-around time is not the best criteria. A process may produce some output fairly early and continue computing new results while previous results are being output to the user. Thus another criteria is the time taken from submission of the process of request until the first response is produced. This measure is called response time.

B

Explore the Reader's Writer's problem and Producer Consumer problem by using Semaphore.

Reader's Writer's Problem:

→A data set is shared among a number of concurrent processes

– Readers – only read the data set, do not perform any updates

– Writers – can both read and write the data set (perform the updates).

• If two readers read the shared data simultaneously, there will be no problem. If both a reader(s) and writer share the same data simultaneously then there will be a problem.

• In the solution of reader-writer problem, the reader process share the following data structures:

Semaphore Mutex, wrt;

int readcount;

• Where →Semaphore mutex is initialized to 1.

→ Semaphore wrt is initialized to 1.

→ Integer readcount is initialized to 0.

The structure of a writer process:

while (true) {

wait (wrt) ;
// writing is performed
signal (wrt) ;
}

The structure of a reader process:

while (true) {
wait (mutex) ;
readcount ++ ;
if (readcount == 1) wait (wrt) ;
signal (mutex)
// reading is performed
wait (mutex) ;
readcount - - ;
if (readcount == 0) signal (wrt) ;
signal (mutex) ;
}

Producer Consumer Problem:

The producer consumer problem is a synchronization problem. There is a fixed size buffer and the producer produces items and enters them into the buffer. The consumer removes the items from the buffer and consumes them.

A producer should not produce items into the buffer when the consumer is consuming an item from the buffer and vice versa. So the buffer should only be accessed by the producer or consumer at a time.The producer consumer problem can be resolved using semaphores.

Producer Process:

```
do {

  .

  . PRODUCE ITEM

  .

  wait(empty);

  wait(mutex);

  .

  . PUT ITEM IN BUFFER

  .

  signal(mutex);

  signal(full);
```

```
} while(1);
```

In the above code, mutex, empty and full are semaphores. Here mutex is initialized to 1, empty is initialized to n (maximum size of the buffer) and full is initialized to 0.

The mutex semaphore ensures mutual exclusion. The empty and full semaphores count the number of empty and full spaces in the buffer.

After the item is produced, wait operation is carried out on empty. This indicates that the empty space in the buffer has decreased by 1. Then wait operation is carried out on mutex so that consumer process cannot interfere.

After the item is put in the buffer, signal operation is carried out on mutex and full. The former indicates that consumer process can now act and the latter shows that the buffer is full by 1.

Consumer Process:

```
do {


  wait(full);

  wait(mutex);

  . .

  . REMOVE ITEM FROM BUFFER

  .

  signal(mutex);

  signal(empty);

  .

  . CONSUME ITEM

  .

} while(1);
```

The wait operation is carried out on full. This indicates that items in the buffer have decreased by 1. Then wait operation is carried out on mutex so that producer process cannot interfere.

Then the item is removed from buffer. After that, signal operation is carried out on mutex and empty. The former indicates that consumer process can now act and the latter shows that the empty space in the buffer has increased by 1.

A	**What is a thread? Discuss and differentiate between user level and Kernel level thread with their advantages and disadvantages. What are the different thread models we are having? Explain them in detail.** . **[8]** **Ans:** A thread is a single sequence stream within in a process. It is also called lightweight processes. In a process, threads allow multiple executions of streams. The CPU switches rapidly back and forth among the threads giving illusion that the threads are running in parallel. A thread can be in any of several states (Running, Blocked, Ready or Terminated). An operating system that has thread facility, the basic unit of CPU utilization is a thread. A thread has or consists of a program counter (PC), a register set, and a stack space. Threads are not independent of one other like processes as a result threads shares with other threads their code section, data section, OS resources such as open files and signals. User - Level Threads The user-level threads are implemented by users and the kernel is not aware of the existence of these threads. It handles them as if they were single-threaded processes. User-level threads are small and much faster than kernel level threads. They are represented by a program counter(PC), stack, registers and a small process control block. Also, there is no kernel involvement in synchronization for user-level threads. **Advantages of User-Level Threads** Some of the advantages of user-level threads are as follows − • User-level threads are easier and faster to create than kernel-level threads. They can also be more easily managed. • User-level threads can be run on any operating system. • There are no kernel mode privileges required for thread switching in user-level threads. **Disadvantages of User-Level Threads** Some of the disadvantages of user-level threads are as follows − • Multithreaded applications in user-level threads cannot use multiprocessing to their advantage. • The entire process is blocked if one user-level thread performs blocking operation. Kernel-Level Threads Kernel-level threads are handled by the operating system directly and the thread management is done by the kernel. The context information for the process as well as the process threads is all managed by the kernel. Because of this, kernel-level threads are slower than user-level threads. **Advantages of Kernel-Level Threads** Some of the advantages of kernel-level threads are as follows − • Multiple threads of the same process can be scheduled on different processors in kernel-level threads. • The kernel routines can also be multithreaded. • If a kernel-level thread is blocked, another thread of the same process can be scheduled by the kernel. **Disadvantages of Kernel-Level Threads**

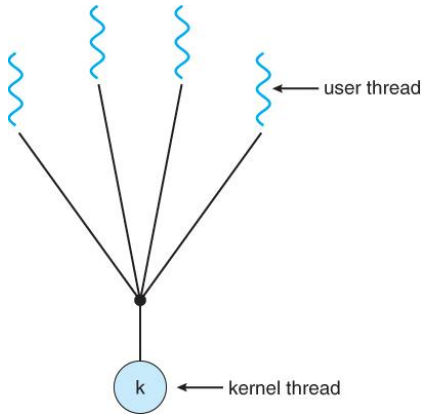
Some of the disadvantages of kernel-level threads are as follows −

- A mode switch to kernel mode is required to transfer control from one thread to another in a process.
- Kernel-level threads are slower to create as well as manage as compared to user-level threads.

Thread Models:

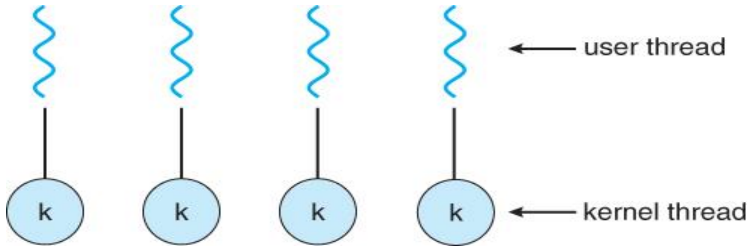
Many-To-One Model

- In the many-to-one model, many user-level threads are all mapped onto a single kernel thread.
- Thread management is handled by the thread library in user space, which is very efficient.
- However, if a blocking system call is made, then the entire process blocks, even if the other user threads would otherwise be able to continue.
- Because a single kernel thread can operate only on a single CPU, the many-to-one model does not allow individual processes to be split across multiple CPUs.
- Green threads for Solaris and GNU Portable Threads implement the many-to-one model in the past, but few systems continue to do so today.



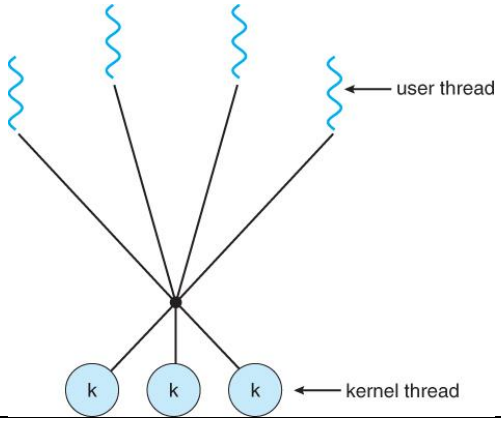
One-To-One Model:

- The one-to-one model creates a separate kernel thread to handle each user thread.
- One-to-one model overcomes the problems listed above involving blocking system calls and the splitting of processes across multiple CPUs.
- However the overhead of managing the one-to-one model is more significant, involving more overhead and slowing down the system.
- Most implementations of this model place a limit on how many threads can be created.
- Linux and Windows from 95 to XP implement the one-to-one model for threads.



Many-To-Many Model:

- The many-to-many model multiplexes any number of user threads onto an equal or smaller number of kernel threads, combining the best features of the one-to-one and many-to-one models.
- Users have no restrictions on the number of threads created.
- Blocking kernel system calls do not block the entire process.
- Processes can be split across multiple processors.
- Individual processes may be allocated variable numbers of kernel threads, depending on the number of CPUs present and other factors.



B

What is the purpose of CPU Scheduling? Mention various scheduling criteria's. Explain in brief various CPU scheduling algorithm.

CPU Scheduler's main objective is to increase system performance in accordance with the chosen set of criteria. It is the change of ready state to running state of the process. CPU scheduler selects a process among the processes that are ready to execute and allocates CPU to one of them.

Various Scheduling Criteria:

1. CPU utilization- Keep the CPU as busy as possible.
2. Throughput – Number of processes that complete their execution per time unit
3. Turnaround Time-The interval from the time of submission of a process to the time of completion. Turnaround Time is the sum of the periods spent waiting to get into memory, waiting in the ready queue, executing on the CPU, and doing I/0 waiting time.
4. Waiting Time-sum of the periods spent waiting in the ready queue.
5. Response Time- – amount of time it takes from when a request was submitted until the first response is produced, not output (for time-sharing environment)

Various CPU Scheduling Algorithm:

First Come First Serve

First Come First Serve is the full form of FCFS. It is the easiest and most simple CPU scheduling algorithm. In this type of algorithm, the process which requests the CPU gets the CPU allocation first. This scheduling method can be managed with a FIFO queue.

As the process enters the ready queue, its PCB (Process Control Block) is linked with the tail of the queue. So, when CPU becomes free, it should be assigned to the process at the beginning of the queue.

Characteristics of FCFS method:

It offers non-preemptive and pre-emptive scheduling algorithm.

Jobs are always executed on a first-come, first-serve basis

It is easy to implement and use.

However, this method is poor in performance, and the general wait time is quite high.

Shortest Remaining Time

The full form of SRT is Shortest remaining time. It is also known as SJF preemptive scheduling. In this method, the process will be allocated to the task, which is closest to its completion. This method prevents a newer ready state process from holding the completion of an older process.

Characteristics of SRT scheduling method:

This method is mostly applied in batch environments where short jobs are required to be given preference.

This is not an ideal method to implement it in a shared system where the required CPU time is unknown.

Associate with each process as the length of its next CPU burst. So that operating system uses these lengths, which helps to schedule the process with the shortest possible time.

Priority Based Scheduling

Priority scheduling is a method of scheduling processes based on priority. In this method, the scheduler selects the tasks to work as per the priority.

Priority scheduling also helps OS to involve priority assignments. The processes with higher priority should be carried out first, whereas jobs with equal priorities are carried out on a round-robin or FCFS basis. Priority can be decided based on memory requirements, time requirements, etc.

Round-Robin Scheduling

Round robin is the oldest, simplest scheduling algorithm. The name of this algorithm comes from the round-robin principle, where each person gets an equal share of something in turn. It is mostly used for scheduling algorithms in multitasking. This algorithm method helps for starvation free execution of processes**.**

Characteristics of Round-Robin Scheduling

Round robin is a hybrid model which is clock-driven

Time slice should be minimum, which is assigned for a specific task to be processed. However, it may vary for different processes.

It is a real time system which responds to the event within a specific time limit.

Shortest Job First

SJF is a full form of (Shortest job first) is a scheduling algorithm in which the process with the shortest execution time should be selected for execution next. This scheduling method can be preemptive or non-preemptive. It significantly reduces the average waiting time for other processes awaiting execution.

Characteristics of SJF Scheduling

It is associated with each job as a unit of time to complete.

In this method, when the CPU is available, the next process or job with the shortest completion time will be executed first.

It is Implemented with non-preemptive policy.

This algorithm method is useful for batch-type processing, where waiting for jobs to complete is not critical.

It improves job output by offering shorter jobs, which should be executed first, which mostly have a shorter turnaround time.

Multiple-Level Queues Scheduling

	This algorithm separates the ready queue into various separate queues. In this method, processes are assigned to a queue based on a specific property of the process, like the process priority, size of the memory, etc. However, this is not an independent scheduling OS algorithm as it needs to use other types of algorithms in order to schedule the jobs. **Characteristic of Multiple-Level Queues Scheduling:** Multiple queues should be maintained for processes with some characteristics. Every queue may have its separate scheduling algorithms. Priorities are given for each queue.