Mahatma Gandhi Mission's College of Engineering and Technology
Kamothe, Navi Mumbai

MAHATMA GANDHI MISSION

Assignment -1

Subject-OS Div- A&B Sem-IV Class –SE

Date of Issue: 14/2/2022 Date of Submission: 25/2/2022

Q.No	Question
Q1.Fill in the blanks	
1)	Whenever a process needs I/O to or from a disk it issues a system call to the operating system
2)	On systems where there are multiple operating system, the decision to load a particular one is done by boot loader.
3)	The priority of a process will does not change itself if the scheduler assigns it a static priority.
4)	The main memory accommodates operating system.
5)	Highest Response Ratio Next(HRNN) is the most optimal scheduling algorithm
Q2. Choose Correct Options	
1)	Using transient code, _____ the size of the operating system during program execution. a) maintains b) changes c) increases d) decreases
2)	The operating system and the other processes are protected from being modified by an already running process because _____ a) every address generated by the CPU is being checked against the relocation and limit registers b) they have a protection algorithm c) they are in different memory spaces d) they are in different logical addresses
3)	Swapping _____ be done when a process has pending I/O, or has to execute I/O operations only into operating system buffers. a) must never b) maybe c) can d) must
4)	The FCFS algorithm is particularly troublesome for _____ a) operating systems b) multiprocessor systems c) time sharing systems d) multiprogramming systems
5)	In operating system, each process has its own _____ a) open files b) pending alarms, signals, and signal handlers

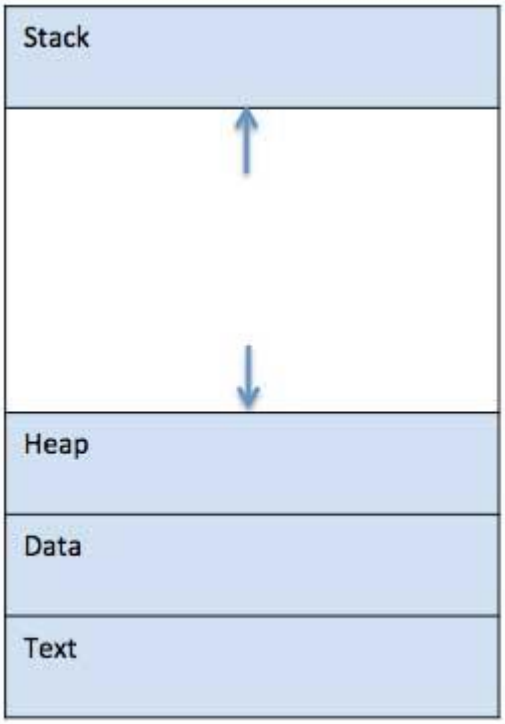
	c) address space and global variables d) all of the mentioned
6)	In a timeshare operating system, when the time slot assigned to a process is completed, the process switches from the current state to? a) Suspended state b) Terminated state c) Ready state d) Blocked state
7)	The portion of the process scheduler in an operating system that dispatches processes is concerned with _____ a) assigning ready processes to waiting queue b) assigning running processes to blocked queue c) assigning ready processes to CPU d) all of the mentioned
8)	Round robin scheduling falls under the category of _____ a) Non-preemptive scheduling b) Preemptive scheduling c) All of the mentioned d) None of the mentioned
9)	Which of the following statements are true? I. Shortest remaining time first scheduling may cause starvation II. Preemptive scheduling may cause starvation III. Round robin is better than FCFS in terms of response time a) I only b) I and III only c) II and III only d) I, II and III
10)	The real difficulty with SJF in short term scheduling is _____ a) it is too good an algorithm b) knowing the length of the next CPU request c) it is too complex to understand d) none of the mentioned
Q3. Answer the following questions in brief	
1) Ans:	Explain the concept of a process. A process is basically a program in execution. The execution of a process must progress in a

sequential fashion.

A process is defined as an entity which represents the basic unit of work to be implemented in the system.

To put it in simple terms, we write our computer programs in a text file and when we execute this program, it becomes a process which performs all the tasks mentioned in the program.

When a program is loaded into the memory and it becomes a process, it can be divided into four sections ─ stack, heap, text and data. The following image shows a simplified layout of a process inside main memory −



S.N.	Component & Description
1	**Stack** The process Stack contains the temporary data such as method/function parameters, return address and local variables.
2	**Heap** This is dynamically allocated memory to a process during its run time.
3	**Text** This includes the current activity represented by the value of Program Counter and the contents of the processor's registers.

4	**Data** This section contains the global and static variables.

Program

A program is a piece of code which may be a single line or millions of lines. A computer program is usually written by a computer programmer in a programming language. For example, here is a simple program written in C programming language −

```
#include <stdio.h>

int main() {
   printf("Hello, World! \n");
   return 0;
}
```

A computer program is a collection of instructions that performs a specific task when executed by a computer. When we compare a program with a process, we can conclude that a process is a dynamic instance of a computer program.

A part of a computer program that performs a well-defined task is known as an **algorithm**. A collection of computer programs, libraries and related data are referred to as a **software**.

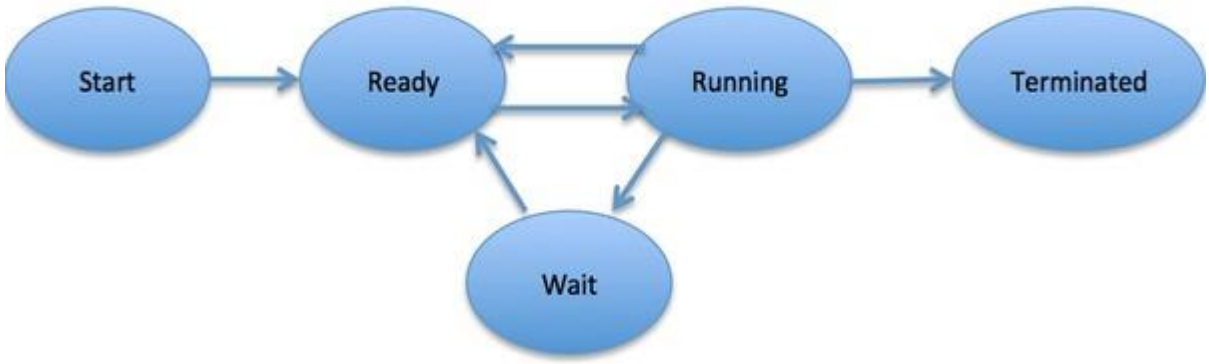
Process Life Cycle

When a process executes, it passes through different states. These stages may differ in different operating systems, and the names of these states are also not standardized.

In general, a process can have one of the following five states at a time.

S.N.	State & Description
1	**Start** This is the initial state when a process is first started/created.
2	**Ready** The process is waiting to be assigned to a processor. Ready processes are waiting to have the processor allocated to them by the operating system so that they can run. Process may come into this state after **Start** state or while running it by but interrupted by the scheduler to assign CPU to some other process.
3	**Running** Once the process has been assigned to a processor by the OS scheduler, the process

	state is set to running and the processor executes its instructions.
4	**Waiting** Process moves into the waiting state if it needs to wait for a resource, such as waiting for user input, or waiting for a file to become available.
5	**Terminated or Exit** Once the process finishes its execution, or it is terminated by the operating system, it is moved to the terminated state where it waits to be removed from main memory.



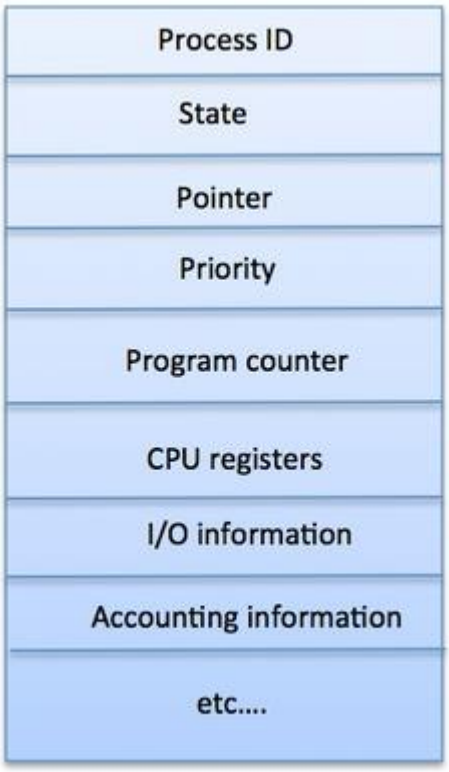
Process Control Block (PCB)

A Process Control Block is a data structure maintained by the Operating System for every process. The PCB is identified by an integer process ID (PID). A PCB keeps all the information needed to keep track of a process as listed below in the table −

S.N.	Information & Description
1	**Process State** The current state of the process i.e., whether it is ready, running, waiting, or whatever.
2	**Process privileges** This is required to allow/disallow access to system resources.
3	**Process ID** Unique identification for each of the process in the operating system.

4	**Pointer** A pointer to parent process.
5	**Program Counter** Program Counter is a pointer to the address of the next instruction to be executed for this process.
6	**CPU registers** Various CPU registers where process need to be stored for execution for running state.
7	**CPU Scheduling Information** Process priority and other scheduling information which is required to schedule the process.
8	**Memory management information** This includes the information of page table, memory limits, Segment table depending on memory used by the operating system.
9	**Accounting information** This includes the amount of CPU used for process execution, time limits, execution ID etc.
10	**IO status information** This includes a list of I/O devices allocated to the process.

The architecture of a PCB is completely dependent on Operating System and may contain different information in different operating systems. Here is a simplified diagram of a PCB −



The PCB is maintained for a process throughout its lifetime, and is deleted once the process terminates.

2) Draw and Explain the process state transition diagram.

Ans:

A transition diagram or state transition diagram is a directed graph which can be constructed as follows:

- There is a node for each state in Q, which is represented by the circle.
- There is a directed edge from node q to node p labeled a if $\delta(q, a) = p$.
- In the start state, there is an arrow with no source.
- Accepting states or final states are indicating by a double circle.

Some Notations that are used in the transition diagram:

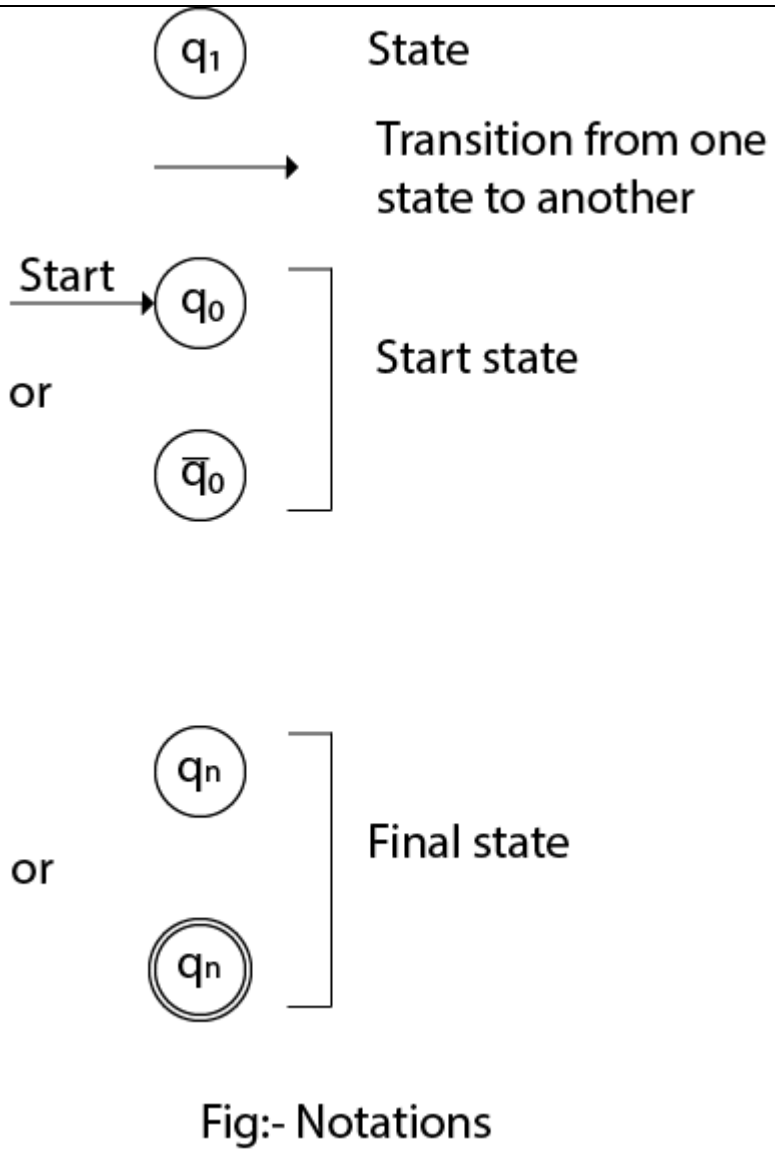


Fig:- Notations

3)

Ans:

Explain different types of schedulers.

There are three types of schedulers available :

1. **Long Term Scheduler** :

   Long term scheduler runs less frequently. Long Term Schedulers decide which program must get into the job queue. From the job queue, the Job Processor, selects processes and loads them into the memory for execution. Primary aim of the Job Scheduler is to maintain a good degree of Multiprogramming. An optimal degree of Multiprogramming means the average rate of process creation is equal to the average departure rate of processes from the execution memory.

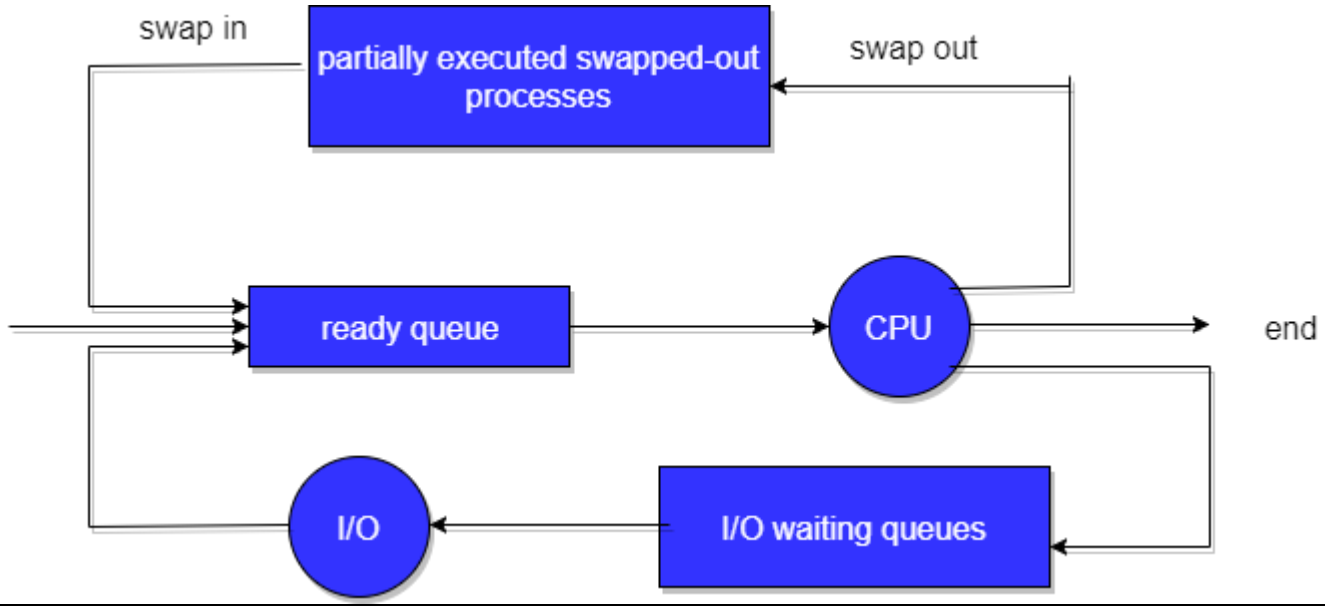
2. **Short Term Scheduler** :

   This is also known as CPU Scheduler and runs very frequently. The primary aim of this scheduler is to enhance CPU performance and increase process execution rate.

3. **Medium Term Scheduler** :

This scheduler removes the processes from memory (and from active contention for the CPU), and thus reduces the degree of multiprogramming. At some later time, the process can be reintroduced into memory and its execution van be continued where it left off. This scheme is called **swapping**. The process is swapped out, and is later swapped in, by the medium term scheduler.

Swapping may be necessary to improve the process mix, or because a change in memory requirements has overcommitted available memory, requiring memory to be freed up. This complete process is descripted in the below diagram:



4) Explain multithreading models.

Ans:

Multithreading Models in Operating System exhibit the ways of mapping the user threads to the kernel threads. Here, we will learn about the three multithreading models, **Many to One model, One to One model and Many to Many model**. Many to One multithreading model maps **many** user threads to only **one** kernel thread. One to One multithreading model maps **a single** user thread to **a single** kernel thread. Many to Many multithreading models maps **many** user threads to a **lesser** or **equal** number of kernel threads.

There is much more to learn about these models. Proceeding further we will discuss each of these models briefly and we will also discuss the relating terms multithreading, user thread and kernel thread. So let's start.

Multithreading Models in Operating System

31st July 2019 by Neha T Leave a Comment

Multithreading Models in Operating System exhibit the ways of mapping the user threads to

the kernel threads. Here, we will learn about the three multithreading models, **Many to One model, One to One model and Many to Many model**.

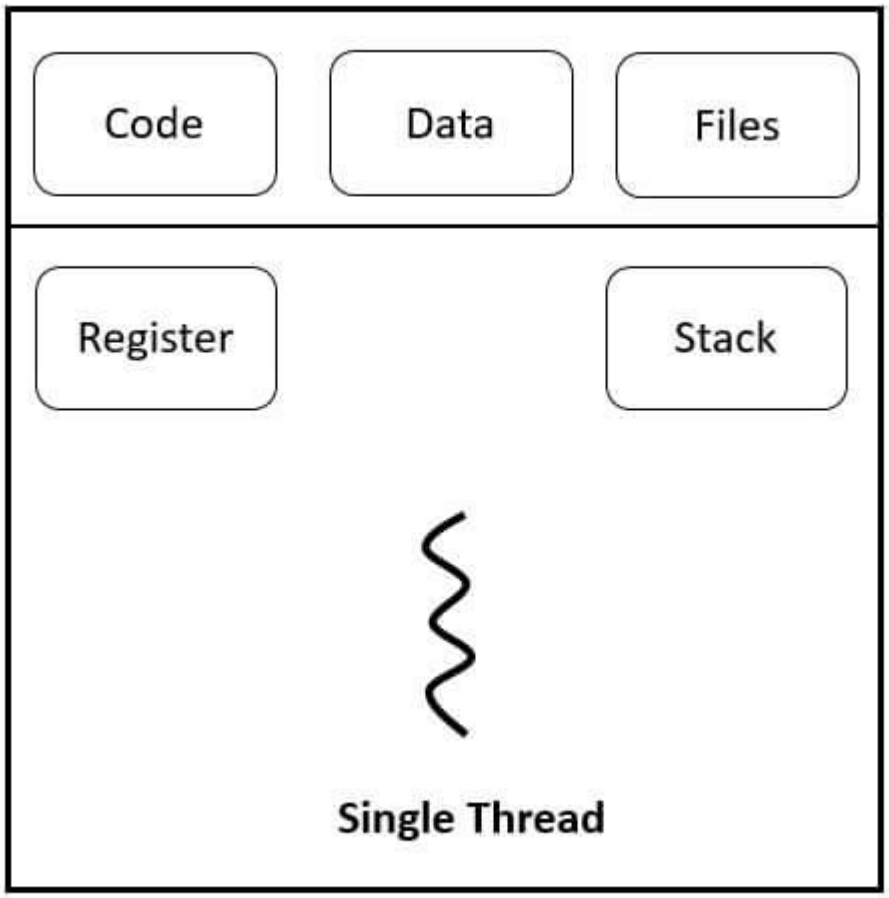
Many to One multithreading model maps **many** user threads to only **one** kernel thread. One to One multithreading model maps **a single** user thread to **a single** kernel thread. Many to Many multithreading models maps **many** user threads to a **lesser** or **equal** number of kernel threads.

There is much more to learn about these models. Proceeding further we will discuss each of these models briefly and we will also discuss the relating terms multithreading, user thread and kernel thread. So let's start.

Contents: Multithreading Models in Operating System

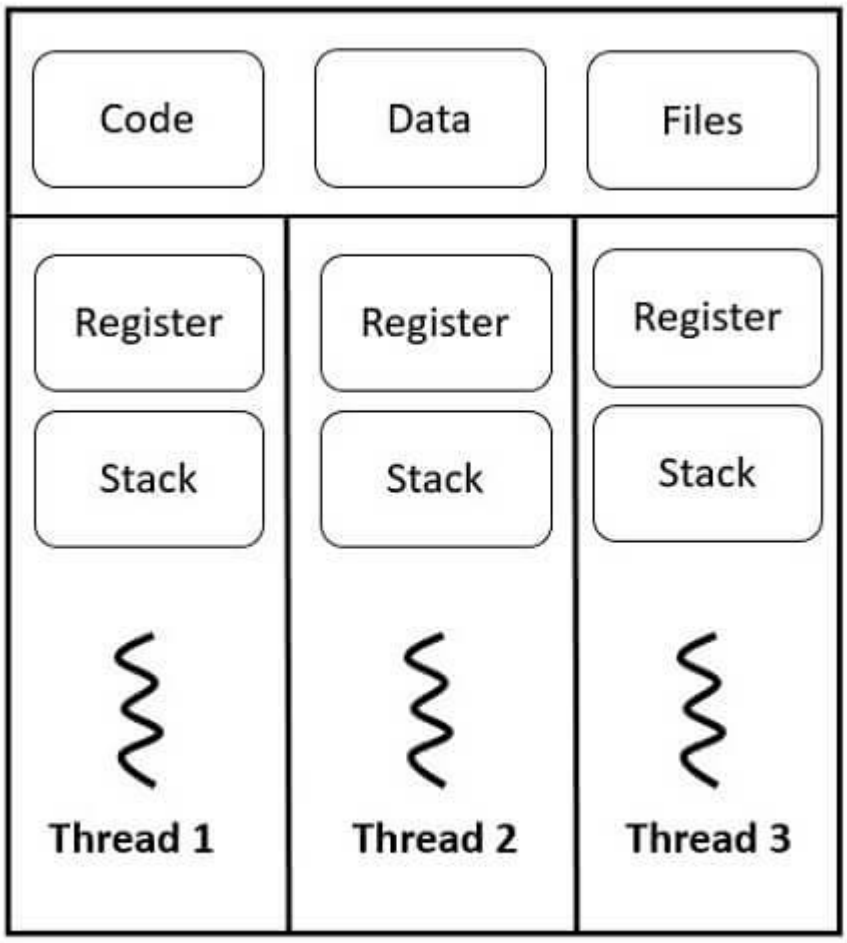
Multithreading

As we know, each process has its own address space which contains **code**, **data**, **files**, **registers**, **stack** and a **single thread** of control. The process with a single thread of control is a traditional or classical process. Its drawback is that if the thread of the process gets blocked, the whole process will get blocked.



Traditional Process with single thread

So the solution is to divide the single thread of control into multiple threads. The process with multiple threads can execute several tasks at a time. The multiple threads of the same process share the code, data and files of the process and each thread maintains its own register and stack. This is called **multithreading** and the process with multiple threads is called **multithreaded process**.



Process with Multiple threads

In the multithreaded process, even if one of the thread gets blocked it does not block the execution of the whole process as the processor can schedule another thread for execution. The simplest example of Multithreading is the word processor. The word process runs multiple threads at a time like a thread for displaying the graphics, the other for responding to the keystroke, say one for spell check, one for navigating a word or phrase on the page and so on.

User Thread

Threads that are implemented at the user level are termed as user-level thread. The thread management of user threads is done by the thread library present at the userspace. The kernel does not have any information about the user-level threads.

It's up to the developer to program the application to be multithreaded using the routines present in the thread library. The thread library at the userspace has the code of creating

and destroying the threads, or for passing the data message between the threads. The thread library is also responsible for scheduling the threads or for saving and restoring the context of the user threads.

Kernel Thread

The threads implemented at the kernel level are termed as kernel-level threads. The thread management is at the kernel level and the kernel threads cannot be managed by the code in the application level.

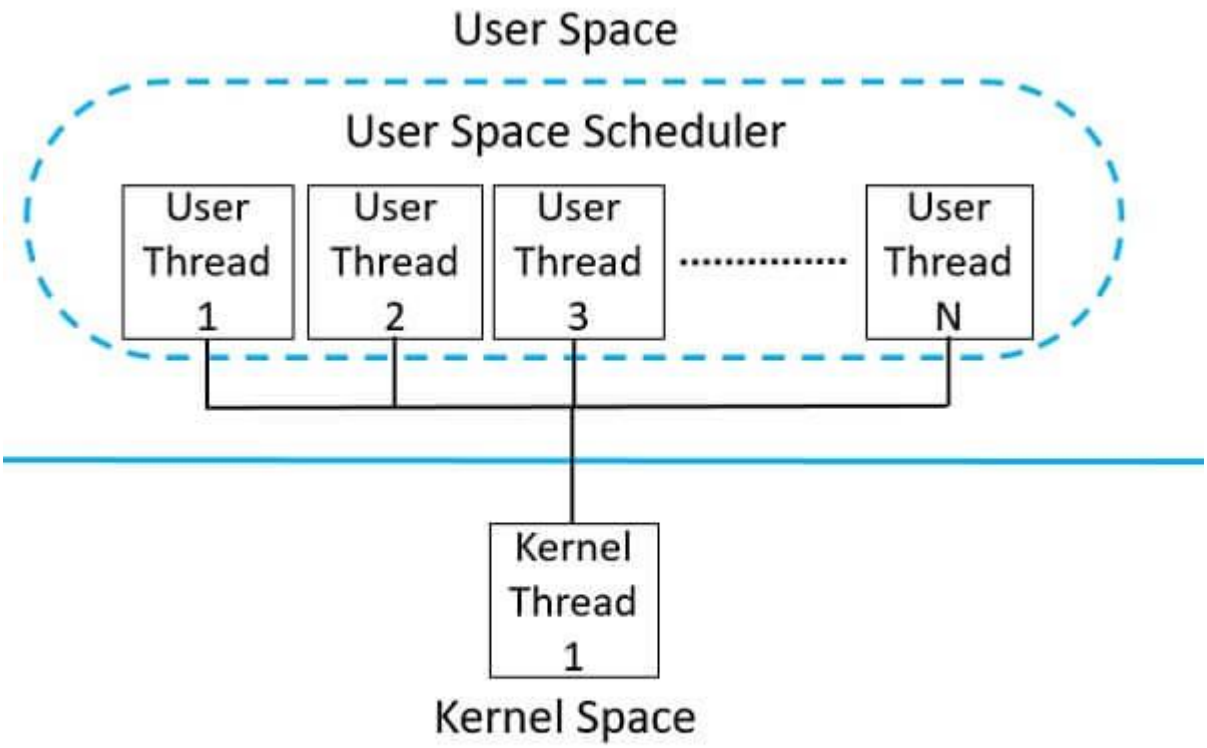
It is the job of the kernel to maintain complete information of the entire process. The kernel also maintains the information about every single thread that is present inside the process.

Multithreading Models in Operating System

1. Many to One Multithreading Model

Many to one multithreading model maps many user threads to a single kernel thread. The thread library present at the user space is responsible for thread management at the user level. The user threads do not require any kernel support.

In many to one model, a single user thread can access the kernel at a time. So, the multiple threads can’t run in parallel. In this scenario, if a thread makes a system blocking call the entire process can get blocked. As only one thread has the access to the kernel at a time.



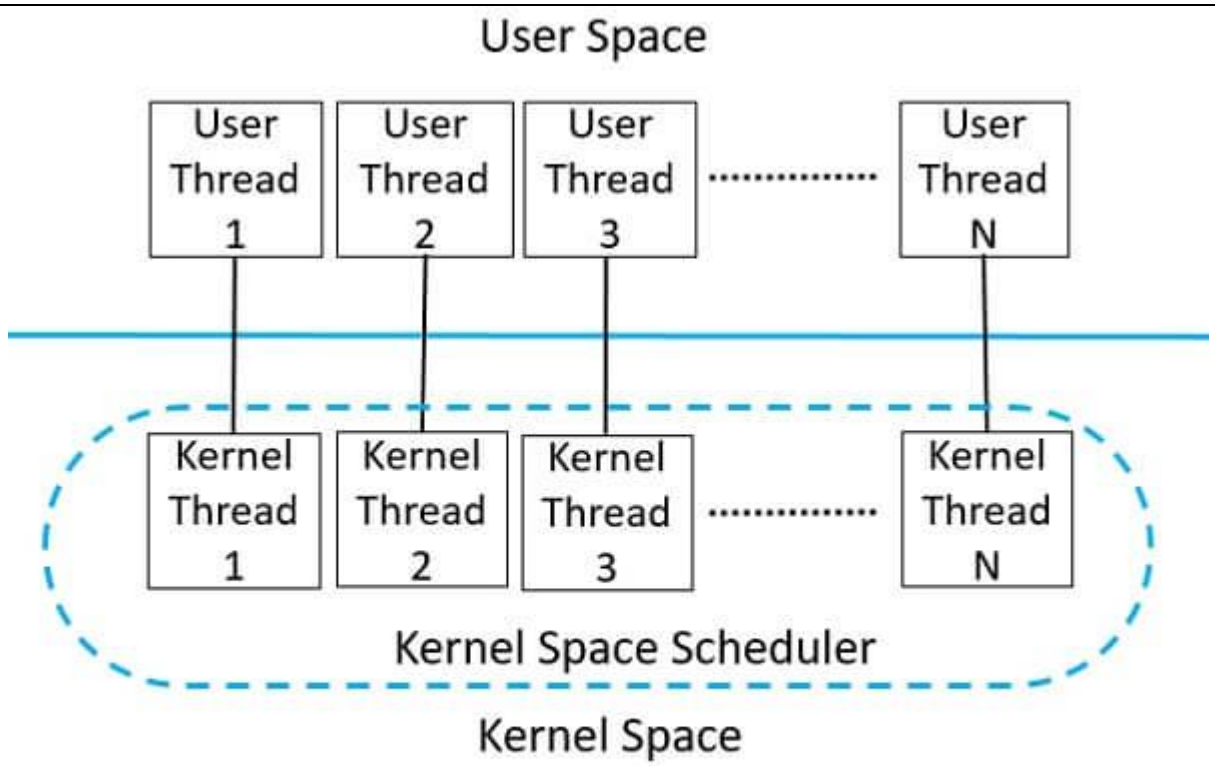
Many to one (M:1) Multithreading Model

This model is implemented by the **Green** Threads library of **Solaris** Operating system. This type of model is implemented by the operating systems that do not create kernel threads.

2. One to One Multithreading Model

One to one multithreading model maps one user thread to one kernel thread. So, for each user thread, there is a corresponding kernel thread present in the kernel. In this model, thread management is done at the kernel level and the user threads are fully supported by the kernel.

Here, if one thread makes the blocking system call, the other threads still run. So, when compared with many to one model, the one to one model provides more **concurrency**.



One to one (1:1) Multithreading Model

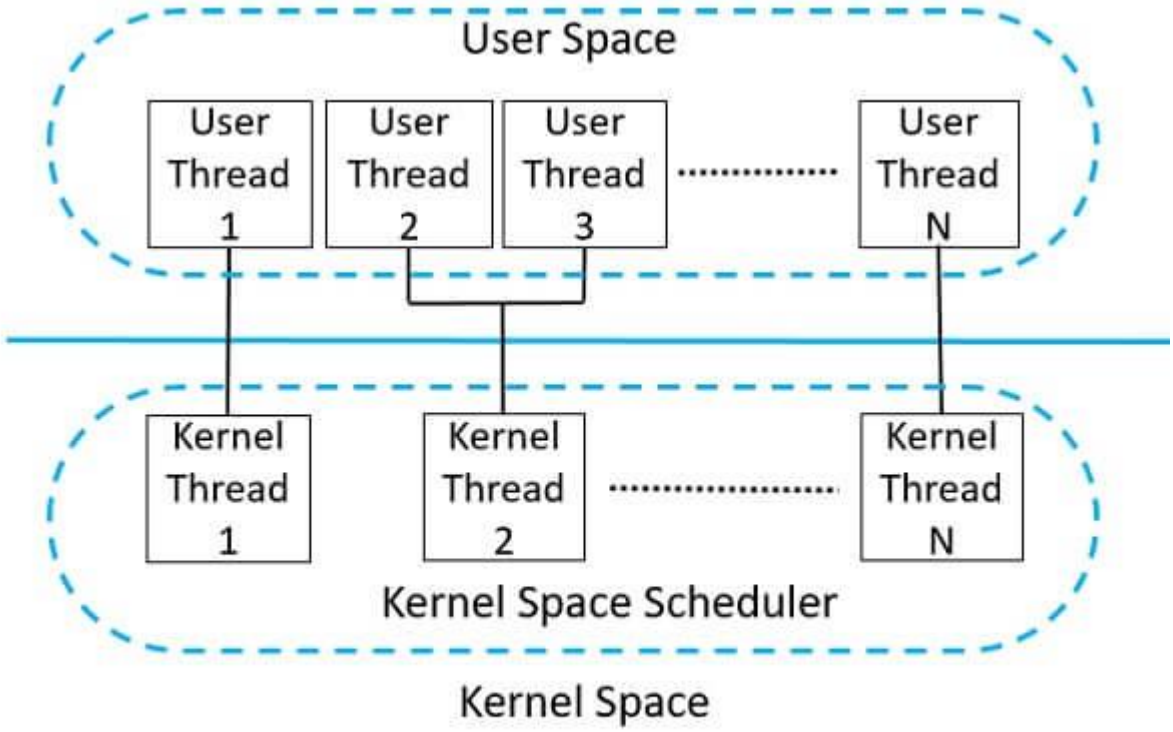
Implementing a kernel thread corresponding to each user thread increases the overhead of the kernel, resulting in slow thread management. So, the operating system implementing this model restricts the number of threads that can be supported by the system.

The operating system, **Linux** and **Windows** family implement one to one multithreading model.

3. Many to Many Multithreading Model

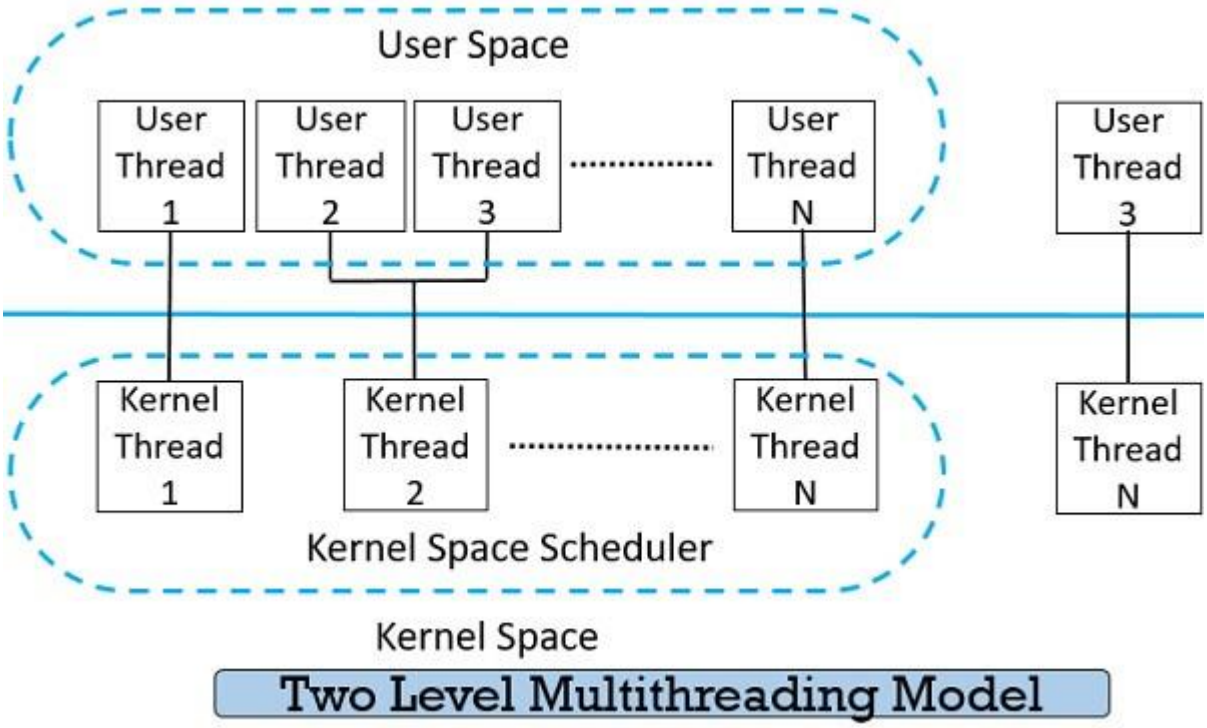
Many to many multithreading models in operating system maps many user threads to a lesser or equal number of kernel threads. This model does not suffer from any of the drawbacks of many to one model and one to one model.

Many to many multithreading models does not restrict the number of user thread created. The developer can create as many threads as required. Neither the process block if a thread makes a blocking system call. As the kernel can schedule other thread for execution.



Many to Many (M:N) Multithreading Model

There exist a variation in many to many model, which depicts that there can be a **user thread that bounds to a kernel thread**. This kind of model in multithreading is referred as the **two-level model**. This model provides efficient and fast thread management. The version older than Solaris 9 support the two-level model.



Two Level Multithreading Model

5) Consider the following set of processes, with the length of the CPU burst given in milliseconds:

Process	Burst Time	Priority
*P*1	2	2
*P*2	1	1
*P*3	8	4
*P*4	4	2
*P*5	5	3

The processes are assumed to have arrived in the order *P*1, *P*2, *P*3, *P*4, *P*5, all at time 0.

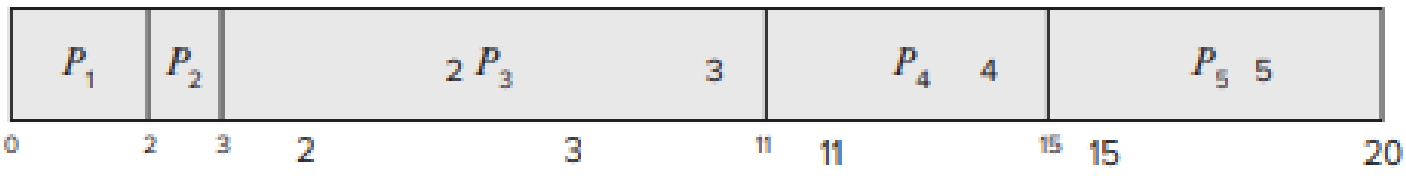
a. Draw four Gantt charts that illustrate the execution of these processes using the following scheduling algorithms: FCFS, SJF, non-preemptive priority (a larger priority number implies a higher priority), and RR (quantum = 2).

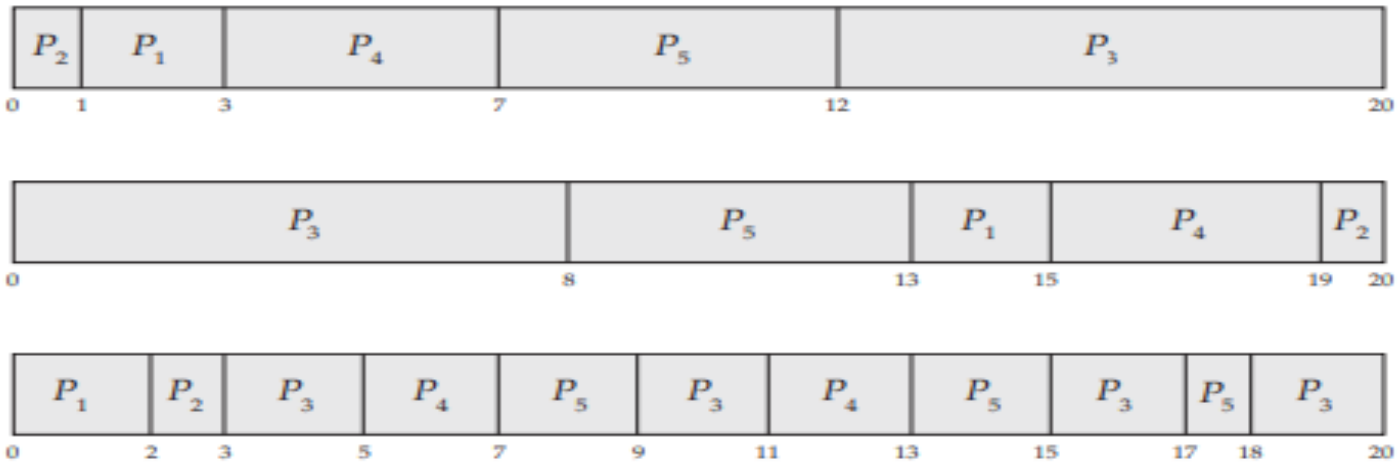
b. What is the turnaround time of each process for each of the scheduling algorithms in part a?

c. What is the waiting time of each process for each of these scheduling algorithms?

d. Which of the algorithms results in the minimum average waiting time (over all processes)?

Ans: a. The four Gantt charts:





b. Turnaround time:

	FCFS	SJF	Priority	RR
P_1	2	3	15	2
P_2	3	1	20	3
P_3	11	20	8	20
P_4	15	7	19	13
P_5	20	12	13	18

c. Waiting time (turnaround time minus burst time):

	FCFS	SJF	Priority	RR
P_1	0	1	13	0
P_2	2	0	19	2
P_3	3	12	0	12
P_4	11	3	15	9
P_5	15	7	8	13

d. SJF has the shortest wait time.