CSC259: Operating System

Prepared By:
Ravi Maharjan
Samriddhi College
BSc CSIT 4th Semester

1

CSC259: Operating System

Unit 2: Process Management

2.1 Process vs Program, Multiprogramming, Process Model, Process States, Process Control Block

2.2 Threads, Thread vs Process, User and Kernel Space Threads

2.3 Inter Process Communication, Race Condition, Critical Selection

2.4 Implementing Mutual Exclusion: Mutual Exclusion with Busy Waiting (Disabling Interrupt, Lock Variables, Strict Alteration, Test and set Lock), Sleep and Wakeup, Semaphore, Monitors, Message Passing

2.5 Classical IPC problems: Producer Consumer, Sleeping Barber, Dining Philosopher Problem

2.6 Process Scheduling, Goals, Batch system scheduling (First-Come First-Served, Shortest Job First, Shortest Remanning Time Next), Interactive System Scheduling (Round Robin Scheduling, Priority Scheduling, Multiple Queues), Overview of Real Time System Scheduling

2

CSC259: Operating System

Process vs Program:

3

CSC259: Operating System

What is Process?

- A process is a program in execution. The process executes continuously one by one. A programmer uses a text editor or an **Integrated Development Environment (IDE)** to write a program in a programming language.
- When a program is run, it transforms into a process. It executes all of the tasks specified in the program. In addition to execution, processes can be created, deleted, and scheduled.
- The process is loaded into the main memory when the program is executed. In main memory, a process has a **stack, heap, data**, and **text**.
- It requires resources such as processing, memory, and input/output resources to complete management tasks. It may engage a processor or input/output during program execution, making a process different from a program.

4

CSC259: Operating System

Features of the Process:

1.Each process contains a definite set of data linked to it. For example, its parent's name, the address of the allotted memory space, and security properties such as ownership credentials and rights.

2.A process contains a limited lifetime, i.e., only when a process is being executed.

3.Processes are allotted system resources. Network ports and file descriptors are two examples.

4.It is an active entity.

5.It contains high resources.

6.It needs resources including memory address, CPU, I/O during its working.

7.Each process may produce child processes..

5

CSC259: Operating System

Program:

- A program is an executable file containing the set of instructions written to perform a specific task.
- Programs are read into the primary memory and executed by the kernel.
- In batch processing systems it is known as executing jobs and in the real-time operating system, it is known as a program. A user can run many programs simultaneously.
- The file will be inactive unless we execute it that's why a program is a passive entity.
- Edge.exe, notepad.exe, or any executable files are examples of programs.
- For example, chorme.exe is an executable file containing the set of instructions written so that we can view web pages.

6

CSC259: Operating System

Features of the Program:

- A program is a type of system activity that follows a set of instructions to complete a specific task.
- A program contains a set of instructions.
- A program is a passive entity i.e, a program will be inactive until it is executed.
- A program will be stored in secondary memory like a hard disk and have a longer life span.
- A single program can be responsible for more than one process.

7

CSC259: Operating System

Difference between:

Process	Program
An executing part of a program is called a process.	A program is a group of ordered operations to achieve a programming goal.
A process gets activated when a program is executed therefore it is an active entity.	A program will be inactive unless it is being used hence it is considered as a passive entity.
A process is active until the program is executed and has a short lifespan as compared to a program.	A program is stored in the secondary memory and has a higher lifespan when compared with the process.
A process consumes significant computational time to execute.	A program does not have computational time and cost.
A process needs high resources as compared to a program, it requires CPU, disk, Input/Output, memory address, etc.	A program does not have resource requirements. A program only needs memory space to store all the instructions.

CSC259: Operating System

Difference:

Process	Program
Processes can be interactive and may interact with users or other processes.	Programs, in themselves, are not interactive. They need to be executed as a process to interact with users or other processes.
Processes can communicate with each other using inter-process communication mechanisms.	Programs do not have the capability to communicate directly. This is done through processes.
It has its own control block, which is known as Process Control Block.	It doesn't have a control block.

9

CSC259: Operating System

Multiprogramming:

- A multiprogramming operating system is a type of operating system that allows multiple programs to run simultaneously on a single CPU. This is achieved by allocating CPU time to different programs in such a way that it appears that they are all running simultaneously.
- The operating system manages the resources, such as memory and input/output devices, for each program to ensure that they all run efficiently without interfering with each other.
- Multiprogramming operating systems are designed to make efficient use of system resources and to improve system throughput by allowing multiple programs to execute concurrently.

- **Two Types of multiprogramming**:
  1. Multitasking OS
  2. Multiuser OS

10

CSC259: Operating System

1. Multitasking Operating System – A Multitasking operating system is an operating system that allows multiple programs or tasks to run concurrently on a single CPU (Central Processing Unit). This means that multiple programs can run at the same time, with the operating system dividing the CPU time between them in a way that gives the appearance of simultaneous execution

2. Multiuser Operating System – A multiuser operating system is an operating system that allows multiple users to access the same system simultaneously, with each user having their own set of resources and permissions. This means that multiple users can log in to the same computer or server and use it at the same time, each running its own applications and programs.

11

CSC259: Operating System

Process Model:

- In the process model, all the runnable software on the computer is organized into a number of sequential processes. Each process has its own virtual Central Processing Unit (CPU).
- The real Central Processing Unit (CPU) switches back and forth from process to process. This work of switching back and forth is called multiprogramming.
- A process is basically an activity. It has a program, input, output, and a state.

The operating system must have a way to ensure that all necessary processes are present.

The processes are created as a result of the four major events listed below:

- •Initialization of the system
- •Execution of a process creation system call by a running process
- •a user request for the creation of a new process
- •Initiation of batch work

12

CSC259: Operating System

- In general, when an operating system is booted, some processes are launched. Some of these processes are in the foreground, while others are in the background.
- The foreground process communicates with computer users or computer programmers. Background processes serve specific purposes.
- The **ps** program in a Unix system can be used to list all the running processes, and the **task manager** in Windows can be used to see what programs are currently running in the system.

13

CSC259: Operating System

Process State transition diagram/ Process Life Cycle:

In an operating system, a process is a program that is being executed. During its execution, a process goes through different states. Understanding these states helps us see how the operating system manages processes, ensuring that the computer runs efficiently.

There must be a minimum of five states. Even though the process could be in one of these states during execution, the names of the states are not standardized. Each process goes through several stages throughout its life cycle.

14

CSC259: Operating System

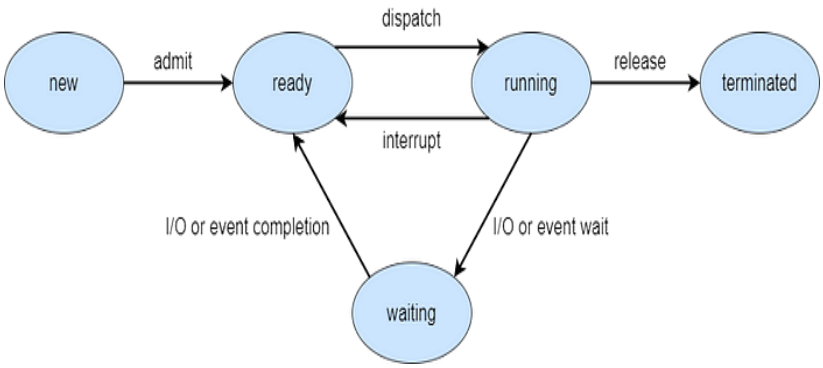


Figure: Process state diagram

15

CSC259: Operating System

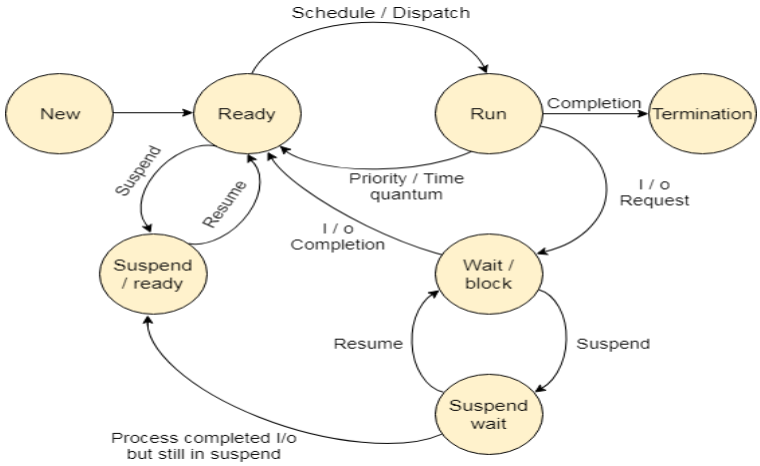


Figure: Process State Diagram

16

CSC259: Operating System

1. **New**

This is the state when the process is just created. It is the first state of a process.

2. **Ready**

Whenever a process is created, it directly enters in the ready state, in which, it waits for the CPU to be assigned. The OS picks the new processes from the secondary memory and put all of them in the main memory.

The processes which are ready for the execution and reside in the main memory are called ready state processes. There can be many processes present in the ready state.

3. **Running**

One of the processes from the ready state will be chosen by the OS depending upon the scheduling algorithm. Hence, if we have only one CPU in our system, the number of running processes for a particular time will always be one. If we have n processors in the system then we can have n processes running simultaneously.

17

CSC259: Operating System

4. **Block or wait**

From the Running state, a process can make the transition to the block or wait state depending upon the scheduling algorithm or the intrinsic behavior of the process.

During the execution of process, the process might require some I/O operation like writing on file or some more priority process might come. In these situation, the running process will have to go into the waiting or blocked state and the other process will come for its execution.

5. **Completion or termination**

When a process finishes its execution, it comes in the termination state. All the context of the process (Process Control Block) will also be deleted the process will be terminated by the Operating system.

18

CSC259: Operating System

Drawbacks of the five-state model

- There is one major drawback of the five-state model. As we know, the processor works much faster than I/O devices. Therefore, a situation may occur where every process might go to waiting/blocked state.
- The CPU stays idle until atleast one process leaves the waiting state. This degrades performance.
- We can overcome the problem by adding another state in the model, which is Suspend State

CSC259: Operating System

6. **Suspend ready**

A process in the ready state, which is moved to secondary memory from the main memory due to lack of the resources (mainly primary memory) is called in the suspend ready state.

If the main memory is full and a higher priority process comes for the execution then the OS have to make the room for the process in the main memory by throwing the lower priority process out into the secondary memory. The suspend ready processes remain in the secondary memory until the main memory gets available.

7. **Suspend wait**

Instead of removing the process from the ready queue, it's better to remove the blocked process which is waiting for some resources in the main memory. Since it is already waiting for some resource to get available hence it is better if it waits in the secondary memory and make room for the higher priority process. These processes complete their execution once the main memory gets available and their wait is finished.

CSC259: Operating System

Types of Schedulers

•**Long-Term Scheduler**:

• Long term scheduler is also known as job scheduler. It choose the process from the pool (secondary memory) and keeps them in the ready queue maintained in the primary memory. It selects and loads the processes into the memory for execution with the help of CPU scheduling. This decides the degree of multiprogramming. Once a decision is taken it lasts for a long time which also indicates that it runs infrequently. Hence it is called a long-term scheduler.

•**Short-Term Scheduler:**

•Short term scheduler is also known as a CPU scheduler that increases system performance as per the chosen set of criteria. This is the change of ready state to running state of the process.

•Short-term scheduler will decide which process is to be executed next and then it will call the dispatcher. A dispatcher is a software that moves the process from ready to run and vice versa. In other words, it is context switching. It runs frequently.

21

CSC259: Operating System

Types of Schedulers

•**Medium Scheduler:** Suspension decision is taken by the medium-term scheduler. The medium-term scheduler is used for swapping which is moving the process from main memory to secondary and vice versa. The swapping is done to reduce degree of multiprogramming.

22

CSC259: Operating System

Multiprogramming

We have many processes ready to run. There are two types of multiprogramming:

•**Preemption –** Process is forcefully removed from CPU. Pre-emption is also called time sharing or multitasking.

•**Non-Preemption –** Processes are not removed until they complete the execution. Once control is given to the CPU for a process execution, till the CPU releases the control by itself, control cannot be taken back forcibly from the CPU.

Degree of Multiprogramming

The number of processes that can reside in the ready state at maximum decides the degree of multiprogramming, e.g., if the degree of programming = 100, this means 100 processes can reside in the ready state at maximum.

23

CSC259: Operating System

Operation on The Process

•**Creation:** The process will be ready once it has been created, enter the ready queue (main memory), and be prepared for execution.

•**Planning:** The operating system picks one process to begin executing from among the numerous processes that are currently in the ready queue. Scheduling is the process of choosing the next process to run.

•**Application:** The processor begins running the process as soon as it is scheduled to run. During execution, a process may become blocked or wait, at which point the processor switches to executing the other processes.

24

CSC259: Operating System

•**Killing or Deletion:** The OS will terminate the process once its purpose has been fulfilled. The process's context will be over there.

•**Blocking:** When a process is waiting for an event or resource, it is blocked. The operating system will place it in a blocked state, and it will not be able to execute until the event or resource becomes available.

•**Resumption:** When the event or resource that caused a process to block becomes available, the process is removed from the blocked state and added back to the ready queue.

•**Context Switching:** When the operating system switches from executing one process to another, it must save the current process's context and load the context of the next process to execute. This is known as context switching.

25

CSC259: Operating System

•**Inter-Process Communication:** Processes may need to communicate with each other to share data or coordinate actions. The operating system provides mechanisms for inter-process communication, such as shared memory, message passing, and synchronization primitives.

•**Process Synchronization:** Multiple processes may need to access a shared resource or critical section of code simultaneously. The operating system provides synchronization mechanisms to ensure that only one process can access the resource or critical section at a time.

•**Process States:** Processes may be in one of several states, including ready, running, waiting, and terminated. The operating system manages the process states and transitions between them.

26

CSC259: Operating System

Process Control Block (PCB):

The Process Control Block (PCB) is a data structure used in operating system that stores essential information about an individual process or task. It is also known as the Task Control Block (TCB) or Control Block in some operating systems. The PCB is a fundamental concept in process management and plays a crucial role in enabling multitasking and coordinating the execution of multiple processes in a system.

Each process in an operating system is represented by its unique PCB. When a process is created, the operating system allocates memory for its PCB, populates it with relevant information, and keeps it in the system's process table. The PCB remains associated with the process throughout its lifetime, even if the process is temporarily suspended or preempted.

27

CSC259: Operating System

Process Control Block
Process ID (PID)
Process State
Program Counter(PC)
CPU Registers
Memory Management Information
Priority
Parent Process ID (PID)
I/O Status Information
Accounting Information
Inter-process Communication Information (IPC)

Figure: Process Control Block (PCB) diagram

28

CSC259: Operating System

Attributes of PCB:

1.Process ID(PID): A distinct Process ID (PID) on the PCB serves as the process's identifier within the operating system. The operating system uses this ID to keep track of, manage, and differentiate among processes.

2.Process State: The state of the process, such as running, waiting, ready, or terminated, is indicated. The operating system makes use of this data to schedule and manage operations.

3.Program Counter(PC): The program counter value, which indicates the address of the following instruction to be performed in the process, is stored on the PCB. The program counter is saved in the PCB of the running process during context switches and then restored to let execution continue where it left off. It is also called as a pointer.

4.CPU registers: Looks at how the process's associated CPU registers are now working. Examples include ***stack pointers***, general-purpose registers, and ***program status flags***. Processes can continue operating uninterrupted during context changes by saving and restoring register values.

29

CSC259: Operating System

5.Memory Management Information: Includes the process's memory allocation information, such as the ***base and limit registers or page tables***. This information allows the operating system to manage the process's memory requirements appropriately.

6.Priority: Some operating systems provide a priority value to each process to decide the order in which processes receive CPU time. The PCB may have a priority field that determines the process's priority level, allowing the scheduler to distribute CPU resources appropriately.

7.Parent Process ID(PPID): The PID of the parent process that gave rise to the present process. This data is important for process management and tracking process linkages, particularly in scenarios requiring process hierarchy or process tree architectures.

30

CSC259: Operating System

8.I/O status: The PCB maintains information about I/O devices and data related to the process. Open ***file descriptors, I/O buffers***, and pending I/O requests are all included. Storing this information enables the operating system to manage I/O operations and efficiently handle input/output requests.

9.Accounting information: Keeps track of the process's resource utilization data, such as ***CPU time, memory usage, and I/O activities***. This data aids in performance evaluation and resource allocation choices.

10.Inter-Process Communication (IPC) information: If a process communicates with other processes, the PCB may contain fields or pointers to communication channels, message queues, shared memory regions, or synchronization primitives. This allows processes to communicate and share data successfully.

31

CSC259: Operating System

Role of Process Control Block (PCB):

The role or work of PCB in process management is that it can access or modified by most OS utilities including those are involved with memory, scheduling, input/output resource access. It can be said that the set of the process control block give information of the current state for the operating system. Data structuring for processes is often done in terms of process control blocks. For example, pointers to other process control blocks inside any process control block allows the creation of those queues of processes in various scheduling states. The various information that is contained by process control block are:

- Naming the process
- State of the process
- Resource allocated to the process
- Memory allocated to the process
- Scheduling information
- Input/output devices associated with process

32

CSC259: Operating System

Thread in Operating System:

- A **thread** is a lightweight subprocess, a basic unit of CPU utilization. It shares resources like memory and file handles with other threads in the same process, but it runs independently.
- A thread is a path of execution within a process (part of process/segment). Each process may have multiple threads.
- Threads are also called lightweight processes as they possess some of the properties of processes.
- In an operating system that supports multithreading, the process can consist of many threads.
- The idea to achieve parallelism by dividing a process into multiple threads.

33

CSC259: Operating System

For example:

Process: opening web browser/ATM machine

Threads: opening tabs in a browser/services provided by ATM.

Example:

MS word uses multiple threads. One thread to format the text, another thread to process inputs etc.

Main objective of thread:

- Threads improve CPU utilization and increased the performance of applications.
- Efficient CPU utilization in multi-core systems.
- Faster context switching compared to processes.
- Simplified inter-thread communication.
- Useful in tasks like I/O handling, parallelism, and real-time systems.

34

CSC259: Operating System

Why Do We Need Thread?

•Threads run in parallel improving the application performance. Each such thread has its own CPU state and stack, but they share the address space of the process and the environment.

•Threads can share common data so they do not need to use inter-process communication. Like the processes, threads also have states like ready, executing, blocked, etc.

•Priority can be assigned to the threads just like the process, and the highest priority thread is scheduled first.

•Each thread has its own Thread Control Block (TCB). Like the process, a context switch occurs for the thread, and register contents are saved in (TCB). As threads share the same address space and resources, synchronization is also required for the various activities of the thread.

35

CSC259: Operating System

- Running of threads concurrently is called multithreading.
- Running of multiple task at a time is called as multitasking.

If a process have a single thread, it is called single threaded process..
If a process having multiple threads, it is called multithreaded process.

Different cases:

1. **Multiple process having single thread per each process.**
2. **Multiple process having multiple threads per each process..**

Components of Threads

These are the basic components of the Operating System.

- •Stack Space
- •Register Set
- •Program Counter

36

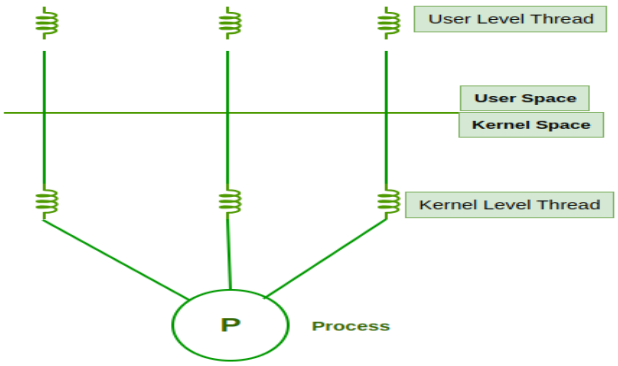
CSC259: Operating System

Types of Thread in Operating System

Threads are of two types. These are described below.

1.User Level Thread

2.Kernel Level Thread



37

CSC259: Operating System

1.User-level thread:

- User level thread is created by the user and the operating system does not recognize the user level thread. Threads that are managed entirely by the **user-level libraries** without kernel involvement.
- If a user performs a user-level thread blocking operation, the whole process is blocked. The kernel level thread does not know nothing about the user level thread.
- The kernel-level thread manages user-level threads as if they are single-threaded processes? : Java thread, POSIX threads, etc.
- User Level Thread is a type of thread that is not created using system calls. The kernel has no work in the management of user-level threads.
- In case when user-level threads are single-handed processes, kernel-level thread manages them..

38

CSC259: Operating System

Advantages of User-Level Threads

- •Implementation of the User-Level Thread is easier than Kernel Level Thread.
- •Context Switch Time is less in User Level Thread.
- •User-Level Thread is more efficient than Kernel-Level Thread.
- •Because of the presence of only Program Counter, Register Set, and Stack Space, it has a simple representation.

Disadvantages of User-Level Threads

- •There is a lack of coordination between Thread and Kernel.
- •In case of a page fault, the whole process can be blocked.

39

CSC259: Operating System

2. Kernel Level Threads

A kernel Level Thread is created by Operating system and it is easily recognized by the OS.

Kernel Level Threads has its own thread table where it keeps track of the system. The operating System Kernel helps in managing threads. Kernel Threads have somehow longer context switching time. Kernel helps in the management of threads.

Advantages of Kernel-Level Threads

- •If one thread blocks, other threads **can continue** to execute.
- •Applications that block services are to be handled by the Kernel-Level Threads.
- •Whenever any process requires more time to process, Kernel-Level Thread provides more time to it.

Disadvantages of Kernel-Level threads

- •Kernel-Level Thread is slower than User-Level Thread.
- •Implementation of this type of thread is a little more complex than a user-level thread.

40

CSC259: Operating System

Difference between user level thread and kernel level thread

S.No	User level thread	Kernel level Thread
1	It is implemented by user.	It is implemented by kernel.
2	Context switching is easy	Context switching is difficult.
3	Multithreading is not compatible	Multithreading is compatible.
4	Implementation is easy and takes less time.	Implementation is difficult and takes more time.
5.	User-level thread is generic and can run on any operating system.	Kernel-level thread is specific to the operating system.
6	Blocking operation: if one thread is in blocking state all the remaining will be in blocking state.	If any thread is in blocking state, then other threads will not be in blocking state.

CSC259: Operating System

Difference between Process and Thread

S.No	User level threads	Kernel level threads
7	Does not recognized by OS.	Recognized by OS
8.	Thread management is easy as there is thread library where code is available for thread creation, destroy, data transfer etc	Thread management is difficult as there will also be code that involved with OS and need system calls for creation.

CSC259: Operating System

Benefits of Threads

•**Enhanced throughput of the system:** When the process is split into many threads, and each thread is treated as a job, the number of jobs done in the unit time increases. That is why the throughput of the system also increases.

•**Effective Utilization of Multiprocessor system:** When you have more than one thread in one process, you can schedule more than one thread in more than one processor.

•**Faster context switch:** The context switching period between threads is less than the process context switching. The process context switch means more overhead for the CPU.

•**Responsiveness:** When the process is split into several threads, and when a thread completes its execution, that process can be responded to as soon as possible.

CSC259: Operating System

•**Responsiveness:** When the process is split into several threads, and when a thread completes its execution, that process can be responded to as soon as possible.

•**Communication:** Multiple-thread communication is simple because the threads share the same address space, while in process, we adopt just a few exclusive communication strategies for communication between two processes.

•**Resource sharing:** Resources can be shared between all threads within a process, such as code, data, and files. Note: The stack and register cannot be shared between threads. There is a stack and register for each thread.

CSC259: Operating System

Difference between Process and Thread

S.No	Process	Thread
1	Process is task under execution by CPU.	It is a light weight process (part of process) Taking lesser resources than process.
2	Process itself is an alone	Group of threads make process.
3	Creation of process takes more time.	Creation of thread is simple and take less time.
4	Termination of process takes more time.	Termination of thread is simple and take less time.
5.	Communication between process takes more time as memory does not share.	Communication between thread takes less time as memory is shared.
6	Does not share data among processes.	Data is shared among threads.

CSC259: Operating System

Difference between Process and Thread

S.No	Process	Threads
7	If one process is blocked, then no other process can execute until the first process is unblocked.	While one thread is blocked and waiting, a second thread in the same task can run.
8.	Context switching will take more time	Context switching is simple and take less time.

CSC259: Operating System

Thread models in OS:

- A **thread model** in an operating system refers to the **strategy used to map user-level threads to kernel-level threads**. It determines how thread management (like creation, scheduling, and synchronization) is handled by the OS and user libraries.. We need to go for mapping because it need to be recognized by OS.
- Kernel which are mapped to user level are called as virtual processors.

Multithreading Models

Some operating system provide a combined user level thread and Kernel level thread facility. **Solaris** is a good example of this combined approach. In a combined system, multiple threads within the same application can run in parallel on multiple processors and a blocking system call need not block the entire process.

47

CSC259: Operating System

Multithreading models are three types

1. Many to one relationship(M:1)
2. One to one relationship.(1:1)
3. Many to many relationship(M:N)

Why is Thread Modeling Important?

- It affects **performance**, **scalability**, and **resource usage**.
- It decides how efficiently an application can utilize **multiple CPU cores**.
- It also impacts **responsiveness** and **blocking behavior** of threads.

48

CSC259: Operating System

1. Many to One Model (M:1)

- Many-to-one model maps many user level threads to one Kernel-level thread. Thread management is done in user space by the thread library.

- When thread makes a blocking system call, the entire process will be blocked. Only one thread can access the Kernel at a time, so multiple threads are unable to run in parallel on multiprocessors.

- If the user-level thread libraries are implemented in the operating system in such a way that the system does not support them, then the Kernel threads use the many-to-one relationship modes.

49

CSC259: Operating System

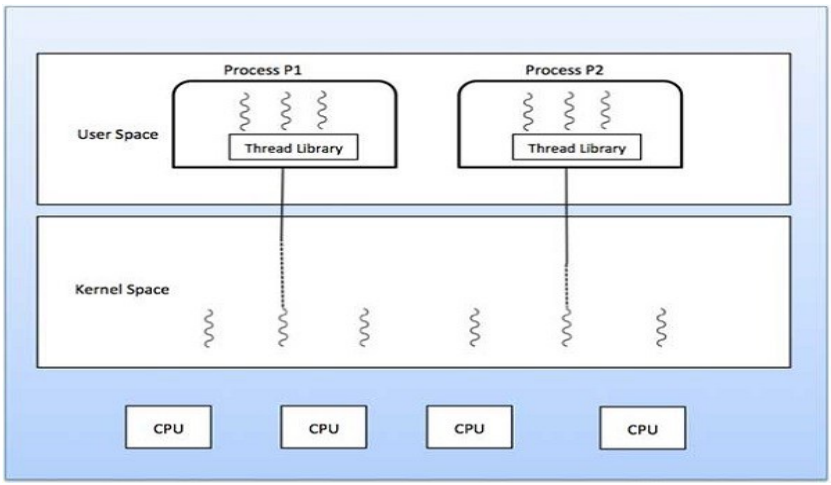


Figure: Many to One Model

50

CSC259: Operating System

2. One to One Model (1:1)

- There is one-to-one relationship of user-level thread to the kernel-level thread. This model provides more concurrency than the many-to-one model.
- It also allows another thread to run when a thread makes a blocking system call.
- It supports multiple threads to execute in parallel on microprocessors.
- In this model, we have create more kernels which degrades the performance of CPU. Disadvantage of this model is that creating user thread requires the corresponding Kernel thread.
- OS/2, windows NT and windows 2000 use one to one relationship model

51

CSC259: Operating System

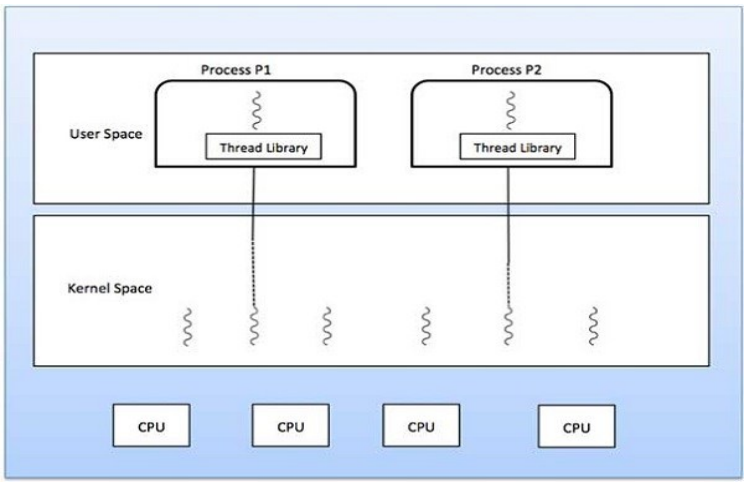


Figure: One to One Model

52

CSC259: Operating System

3. Many to Many Model(M:N)

- The many-to-many model multiplexes any number of user threads onto an equal or smaller number of kernel threads.
- The following diagram shows the many-to-many threading model where 6 user level threads are multiplexing with 6 kernel level threads.
- In this model, developers can create as many user threads as necessary and the corresponding Kernel threads can run in parallel on a multiprocessor machine.
- This model provides the best accuracy on concurrency and when a thread performs a blocking system call, the kernel can schedule another thread for execution.

53

CSC259: Operating System

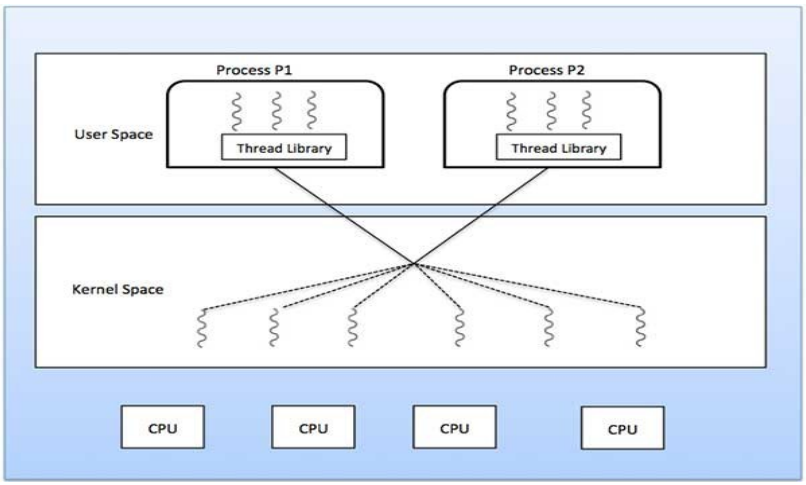


Figure: Many to Many Model

54

CSC259: Operating System

Inter process Communication (IPC):

In general, Inter Process Communication is a type of mechanism usually provided by the operating system (or OS). The main aim or goal of this mechanism is to provide communications in between several processes. In short, the intercommunication allows a process letting another process know that some event has occurred.

Working together with multiple processes require an inter process communication (IPC) method which will allow them to exchange data along with various information.

In **Interprocess Communication (IPC)**, two fundamental mechanisms are used to allow processes to exchange data and coordinate their actions: **Shared Memory** and **Message Passing**. There are generally two primary models of inter process communication

1. **Shared memory**
2. **Message passing**

55

CSC259: Operating System

1.Shared memory:

It can be referred to as a type of memory that can be used or accessed by multiple processes simultaneously. It is primarily used so that the processes can communicate with each other. Therefore, the shared memory is used by almost all POSIX and Windows operating systems as well.

2.Message passing:

It is a type of mechanism that allows processes to synchronize and communicate with each other. However, by using the message passing, the processes can communicate with each other without restoring the shared variables.

Usually, the inter-process communication mechanism provides two operations that are as follows:

•send (message)

•received (message)

56

CSC259: Operating System

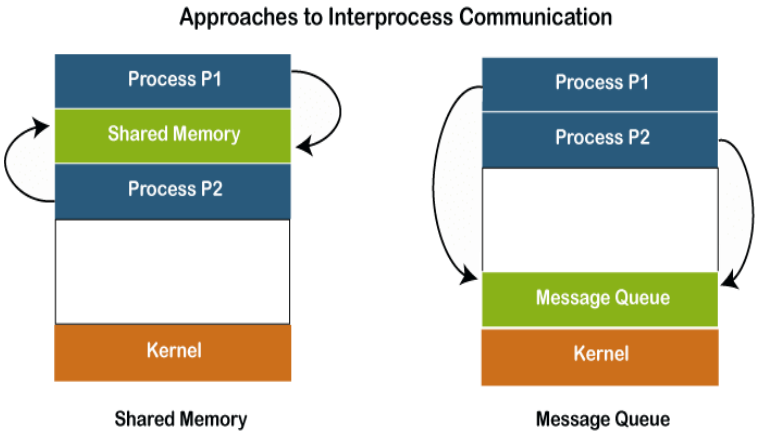


Figure: Interprocess communication

57

CSC259: Operating System

Process Synchronization:

Process synchronization in operating system is important when multiple processes run either in serial or parallel mode.

Mainly, there are two types of processes.

1. Independent process
2. Cooperative process

1. Independent process:

- If one process execution does not affect the execution of another process, then it is called as independent process.
- There will be no sharing of memory(buffer)
- No need of synchronization

58

CSC259: Operating System

2. Cooperative Process:

- One process execution affects the execution of another process, it is called as cooperative process.
- Memory will be shared with different process due to which problem with data occurs. There might be chance of getting inconsistent of data.
- Therefore, to maintain the consistency of data, we need process synchronization.
- Cooperative process share (variable, memory, code, resources(e.g CPU, printer, scanner)).

Suppose we have three processes p1, p2, p3. Let assume variable a=10.

P1=a+1; a=11

P2=a+2; a=12

P3=a-1;a=9

Also, assume, all three processes are doing same job at the same time.

This brings inconsistency of data, meaning that there is a Race Condition.

59

CSC259: Operating System

Race condition:

A race condition occurs when two or more process can access shared data and they try to change it at the same time. Because the process scheduling algorithm can swap between process at any time. All processes are racing to access/change the data.

Code:

P1

```
int x=shared;
x++;
sleep(1);
Shared = x;
```

P2

```
int y=shared;
y--;
Sleep(1);
Shared=y;
```

Assume two process compute together at the same time.

60

CSC259: Operating System

Critical Solution problem:

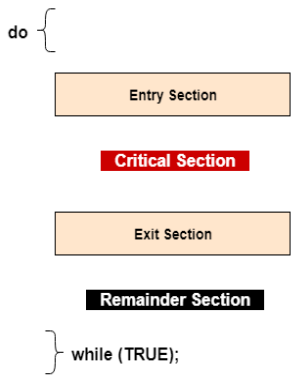
Assume that processes are in a race condition. To avoid race condition, we have to solve the critical section problem.

- The critical section is a **code segment** where the shared variables can be accessed. An atomic action is required in a critical section i.e. only one process can execute in its critical section at a time. All the other processes have to wait to execute in their critical sections.
- The critical section cannot be executed by more than one process at the same time; operating system faces the difficulties in allowing and disallowing the processes from entering the critical section.
- The critical section problem is used to design a set of protocols which can ensure that the Race condition among the processes will never arise.
- In order to synchronize the cooperative processes, our main task is to solve the critical section problem. We need to provide a solution in such a way that the following conditions can be satisfied.

61

CSC259: Operating System

A diagram that demonstrates the critical section is as follows −



62

CSC259: Operating System

In the above diagram, the entry section handles the entry into the critical section. It acquires the resources needed for execution by the process. The exit section handles the exit from the critical section. It releases the resources and also informs the other processes that the critical section is free.

Solution to the Critical Section Problem

In order to synchronize the cooperative processes, our main task is to solve the critical section problem. We need to provide a solution in such a way that the following conditions can be satisfied.

•**Mutual Exclusion: Mutual** exclusion implies that only one process can be inside the critical section at any time. If any other processes require the critical section, they must wait until it is free.

•**Progress: Progress** means that if a process is not using the critical section, then it should not stop any other process from accessing it. In other words, any process can enter a critical section if it is free.

•**Bounded Waiting: Bounded** waiting means that each process must have a limited waiting time. Its should not wait endlessly to access the critical section.

63

CSC259: Operating System

Avoiding Critical Region:

- To avoid critical region, we need process synchronization. Process synchronization is the task of coordinating the execution of processes in a way that no two processes can have access to the same shared data and resources.
- It is specially needed in multi process system when multiple processes are running together, and more than one process try to gain access to the same shared resource or data at the same time.
- Generally, process need to be synchronized with each other to avoid inconsistency of data.
- Mutual exclusion and serializability is the best way of avoiding critical regions.

64

CSC259: Operating System

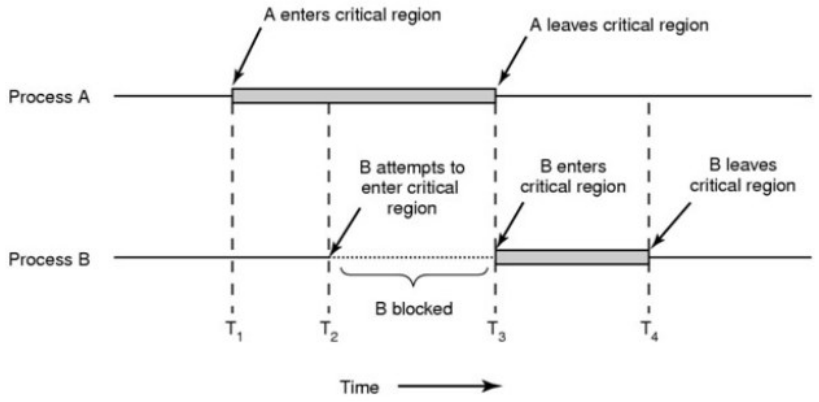


Figure: Mutual exclusion in critical region

65

CSC259: Operating System

Mutual Exclusion:

- **Mutual Exclusion** is a property of process synchronization that states that "no two processes can exist in the critical section at any given point of time".
- Any process synchronization technique being used must satisfy the property of mutual exclusion, without which it would not be possible to get rid of a race condition.
- The requirement of mutual exclusion is that when process P1 is accessing a shared resource R1, another process should not be able to access resource R1 until process P1 has finished its operation with resource R1.
- Examples of such resources include files, I/O devices such as printers, and shared data structures.

66

CSC259: Operating System

Conditions Required for Mutual Exclusion

According to the following four criteria, mutual exclusion is applicable:

1.When using shared resources, it is important to ensure mutual exclusion between various processes. There cannot be two processes running simultaneously in either of their critical sections.

2.It is not advisable to make assumptions about the relative speeds of the unstable processes.

3.For access to the critical section, a process that is outside of it must not obstruct another process.

4.Its critical section must be accessible by multiple processes in a finite amount of time; multiple processes should never be kept waiting in an infinite loop.

67

CSC259: Operating System

How can we achieve Mutual Exclusion ?

Several proposals presented to achieve Mutual Exclusion.

1. **Disabling Interrupts**
2. **Lock Variable**
3. **Turn variable or Strict Alteration**
4. **Peterson's Solution**
5. **The TSL Instruction (Test and Set Lock)**
6. **Sleep and Wakeup**

68

CSC259: Operating System

1. Disabling Interrupts:

Disabling interrupts is one of the **earliest techniques** used in operating systems—particularly on **uniprocessor systems**—to achieve **mutual exclusion**.

How It Works ?

When a process enters its **critical section**, it **disables all interrupts** on the CPU. This prevents the CPU from being interrupted by other processes or system calls, ensuring that the current process can complete its critical section **without being preempted**.

Once the critical section is finished, the process **re-enables the interrupts**, allowing other processes to resume execution.

Steps

- Disable interrupts
- Enter critical section
- Execute critical section
- Enable interrupts

69

CSC259: Operating System

2. Lock Variable:

- It is a software mechanism implemented in user mode(It's a software solution).
- This is a busy waiting solution .
- It can be used for more than two processes.

A **lock variable** is a simple **software mechanism** used to achieve **mutual exclusion** by controlling access to the **critical section** with a shared variable.

In this mechanism, a Lock variable **lock** is used. Two values of lock can be possible, either 0 or 1. Lock value 0 means that the critical section is vacant while the lock value 1 means that it is occupied.

A process which wants to get into the critical section first checks the value of the lock variable. If it is 0 then it sets the value of lock as 1 and enters into the critical section, otherwise it waits.

70

CSC259: Operating System

Entry section - while(lock != 0);
Lock = 1;
//critical section
Exit section - Lock = 0;

If we look at the Pseudo Code, we find that there are three sections in the code. Entry Section, Critical Section and the exit section.

Initially the value of **lock variable** is **0**. The process which needs to get into the **critical section**, enters into the entry section and checks the condition provided in the while loop.

The process will wait infinitely until the value of **lock** is 1 (that is implied by while loop). Since, at the very first-time critical section is vacant hence the process will enter the critical section by setting the lock variable as 1.

When the process exits from the critical section, then in the exit section, it reassigns the value of **lock** as 0.

71

CSC259: Operating System

Simply,

Consider having a single, shared (lock) variable, initially 0.

When a process wants to enter its critical region, it first tests the lock.

- If lock=0, the process sets it to 1 and enters the critical region.
- If the lock is already 1, the process just waits until it becomes 0. Thus, 0 means that no process is in critical region.

Note:

•In the lock variable approach, the process is able to enter into the critical section only when the lock variable is set to 1.

•In the lock variable approach, more than one process has a lock variable value of 1 at the same time.

•Due to such conditions, the lock variable is not able to guarantee mutual execution.

. Hence, the lock variable doesn't provide the mutual exclusion that's why it cannot be used in general.

72

CSC259: Operating System

3. Turn Variable or Strict alteration:

The turn variable is defined as a synchronization mechanism that is implemented in the user mode. The turn variable is also known as the Strict Alternation Approach.

It is a busy waiting solution.

It is a synchronization mechanism that is implemented for synchronizing two processes.

Example: Consider we have two processes process P0 and process P1. For the below two processes P0 and P1,

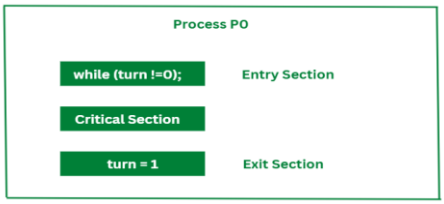
•**Turn value = 0:** When the turn value is set to 0, it means that process P0 will enter into the critical section.

•**Turn value = 1:** When the turn value is set to 1, it means that process P1 will enter into the critical section.

73

CSC259: Operating System

Synchronization for two processes



The above figure represents the code snippet for the Execution of process P0 into the Critical Section

•Initially, the process P0 arrives.

•The block executes the first instruction. i.e. while (turn != 0).

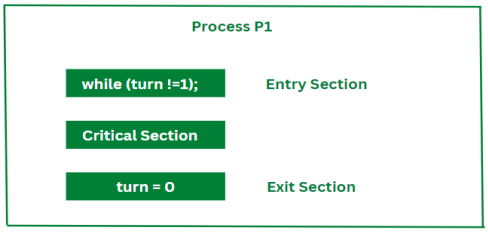
•Initially, the value to turn variable is set to 0, therefore it returns 0 to the while loop.

•As the value of the turn variable is 0, the while loop condition is not satisfied, and process P0 enters into the Critical section.

•Once process P0 has entered into the critical section, process P1 cannot enter into the critical section block unless it completes its execution and set the turn value as 1.

74

CSC259: Operating System



The above figure represents the code snippet for the Execution of process P1 into the Critical Section

•Initially, process P1 arrives.

•The block executes the first instruction. i.e. while (turn != 1).

•Initially, the value to turn variable is set to 1, therefore it returns 1 to the while loop.

•As the value of the turn variable is 1, the while loop condition is not satisfied, and process P1 enters into the Critical section.

•Once process P1 has entered into the critical section, process P0 cannot enter into the critical section block unless it completes its execution and set the turn value as 0

75

CSC259: Operating System

The execution of process P1 into an infinite loop

•When initially the turn value is set to 0 and process P1 arrives.

•It executes the first while loop condition that is while (turn != 1)

•Initially, as the turn value is set to 0, it returns 1 to the while loop

•The value 1 does not break the condition present in the while loop

•Therefore, process P1 gets trapped inside the infinite while loop

•The process P1 remains in an infinite loop until the turn value becomes 1.

76

CSC259: Operating System

4. Peterson's Solution:

- This is a software mechanism implemented at user mode.
- It is a busy waiting solution can be implemented for only two processes.
- It uses two variables that are turn variable and interested variable.

77

CSC259: Operating System

```
# define N 2
# define TRUE 1
# define FALSE 0
int interested[N] = FALSE;
int turn;
voidEntry_Section (int process)
{
    1. int other;
    2. other = 1-process;
    3. interested[process] = TRUE;
    4. turn = process;
    5. while (interested [other] =True && TURN=process);
}
voidExit_Section (int process)
{
    6. interested [process] = FALSE;
}
```

78

CSC259: Operating System

Analysis of Peterson Solution

```
voidEntry_Section (int process)
{
   1. int other;
   2. other = 1-process;
   3. interested[process] = TRUE;
   4. turn = process;
   5. while (interested [other] =True && TURN=process);
}

Critical Section

voidExit_Section (int process)
{
   6. interested [process] = FALSE;
}
```

P0……CS……process=0 and other =1
P1……CS……process=1 and other =0

TT=T
TF=F
FT=F
FF=F

79

CSC259: Operating System

This is a two process solution. Let us consider two cooperative processes P1 and P2. The entry section and exit section are shown below. Initially, the value of interested variables and turn variable is 0.

Initially process P1 arrives and wants to enter into the critical section. It sets its interested variable to True (instruction line 3) and also sets turn to 1 (line number 4). Since the condition given in line number 5 is completely satisfied by P1 therefore it will enter in the critical section.

P1 → 1 2 3 4 5 CS

Meanwhile, Process P1 got preempted and process P2 got scheduled.

80

CSC259: Operating System

P2 also wants to enter in the critical section and executes instructions 1, 2, 3 and 4 of entry section. On instruction 5, it got stuck since it doesn't satisfy the condition (value of other interested variable is still true). Therefore it gets into the busy waiting.

P2 → 1 2 3 4 5

P1 again got scheduled and finish the critical section by executing the instruction no. 6 (setting interested variable to false). Now if P2 checks then it are going to satisfy the condition since other process's interested variable becomes false. P2 will also get enter the critical section.

P1 → 6

P2 → 5 CS

Any of the process may enter in the critical section for multiple numbers of times. Hence the procedure occurs in the cyclic order.

Problem: Difficult to program for n processes, may lead to starvation.

81

CSC259: Operating System

5. Test and set lock (TSL Instruction):

Assignment: Write Notes on TSL Instruction

82

CSC259: Operating System

6.Sleep and Wakeup:

When a process wants to enter in its critical section, it checks to see if the entry is allowed. If it is not, the process goes into tight loop and waits(i.e start busy waiting) until it is allowed to enter. This approach waste CPU time.

Lets look at inter-process communication primitives is the part of sleep –wake up.

Sleep: It is a system call that causes the caller to block, i.e suspended until some other process wakes it up.

Wakeup: It is the system call that wakes up the process.

Both sleep and wakeup system calls have one parameter that represents a memory address used to match up sleeps and wakeup.

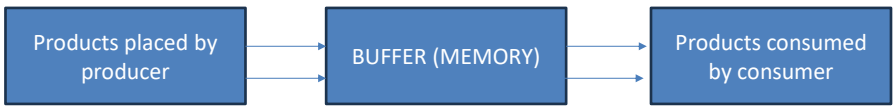
83

CSC259: Operating System

Producer –Consumer Problem:

-works for 2 processes.

1. Producer: Producer produces a product and place it in a buffer (memory).
2. Consumer: Consumes the product which was placed in buffer by the producer



84

CSC259: Operating System

Two conditions:

Condition at Producer Side(Count will be incremented)

- Before placing product into Buffer, producer have to check the buffer is not full

Condition at Consumer Side(Count will be decremented)

- Before consuming product, from Buffer consumer have to check buffer is not empty.

Producer:
Count !=Buffersize
Buffer=product
Count=count+1

Consumer:
Count!=0
Consumed =product
Count=count-1

85

CSC259: Operating System

Producer Consumer problem

Assume that we have two system calls as **sleep** and **wake**. The process which calls sleep will get blocked while the process which calls will get waked up.

There is a popular example called **producer consumer problem** which is the most popular problem simulating **sleep and wake** mechanism.

If the critical section is not empty then the process will go and sleep. It will be waked up by the other process which is currently executing inside the critical section so that the process can get inside the critical section.

In producer consumer problem, let us say there are two processes, one process writes something while the other process reads that. The process which is writing something is called **producer** while the process which is reading is called **consumer**

86

CSC259: Operating System

```
#define N 100                    //maximum slots in buffer
#define count=0                  //items in the buffer
void producer (void)
{
        int item;
        while(True)
 {
          item = produce_item();      //producer produces an item
          if(count == N)              //if the buffer is full then the producer will sleep
            Sleep();
          insert_item (item);          //the item is inserted into buffer
          count=count+1;
          if(count==1)                //The producer will wake up the
                                      //consumer if there is at least 1 item in the buffer
           wake-up(consumer);
  }
}
```

87

CSC259: Operating System

```
void consumer (void)
{
   int item;
   while(True)
   {
      {
        if(count == 0)           //The consumer will sleep if the buffer is empty.
        sleep();
        item = remove_item();
        count = count - 1;
        if(count == N-1)         //if there is at least one slot available in the buffer
                                 //then the consumer will wake up producer
        wake-up(producer);
        consume_item(item);   //the item is read by consumer.
      }
   }
}
```

88

CSC259: Operating System

- The producer produces the item and inserts it into the buffer. The value of the global variable count got increased at each insertion. If the buffer is filled completely and no slot is available then the producer will sleep, otherwise it keep inserting.

- On the consumer's end, the value of count got decreased by 1 at each consumption. If the buffer is empty at any point of time then the consumer will sleep otherwise, it keeps consuming the items and decreasing the value of count by 1.

- The consumer will be waked up by the producer if there is at least 1 item available in the buffer which is to be consumed. The producer will be waked up by the consumer if there is at least one slot available in the buffer so that the producer can write that.

89

CSC259: Operating System

Problem:

Consumer is about to go to sleep but got preempted, buffer is >=1 and producer will send the wake up signal but the consumer is not sleeping yet and the signal will get wasted. And then consumer got scheduled and went to sleep. And the producer is thinking that consumer is not sleeping. When count is N, then producer will go to sleep. Now if you see that both producer and consumer are sleeping. And that is deadlock.

Solution:

Whenever someone is about to sleep then a bit is maintained to tell do not sleep. But when there are N producer and consumer then we have to record N wake up calls. So Dijkastra introduced something called as Semaphores which will tell you how many wake up calls happened.

90

CSC259: Operating System

Semaphore:

It is an integer variable which can be used as shared variable to prevent Race Condition.

It is implemented using two procedures

1. **Wait()**
2. **Signal()**

1. **Wait():**
   - It gives the status for the process to enter into critical section.
   - If semaphore value is greater than or equal to 1, then it indicates process can enter into critical section.
   - Semaphore value will be decremented.

CSC259: Operating System

If semaphore value is 0, then it indicates some process is in critical section. No other process should be allowed into critical section.

```
Wait(s)
{
While(s <=0);
s - -;
}
```

2. Signal():

After the completion of process in critical section, semaphore value will be incremented.

```
Signal (s)
{
        S++;
}
```

CSC259: Operating System

Types of Semaphore:

1. **Binary Semaphore:**
   - Value is 0 or 1.
   - Value is 1---implies process can enter critical section
   - Value is 0 --- implies process have to wait to execute.

2. Counting semaphore:

- -Non-negative Integer
- -have more number
- -efficient user of multiple resource.

93

CSC259: Operating System

Properties of Semaphores:

- It is simple and always have a non-negative integer value.
- Semaphores are machine independent and simple to implement
- Works with many processes.
- Can have many different critical sections with different semaphores.
- Each critical section has unique access semaphores.

94

CSC259: Operating System

Advantages of Semaphores:

- To prevent race condition.
- To prevent deadlock
- To implement mutual exclusion
- Efficient use of resources

Disadvantages of Semaphore:

- Improper usage of semaphore may lead to deadlock
- More no. of semaphore may lead to high maintenance
- Difficult to debug if any synchronization problem occurs.

CSC259: Operating System

Monitors:

- Monitors are the synchronization construct that were created to overcome the problems caused by semaphores such as timing errors.
- Monitors are abstract data types and contain shared data variables and procedure.
- The shared data variable cannot be directly accessed by a process and procedures are required to allow a single process to access the shared data variables at a time.

```
Monitor Monitor_name
{
        data variables;
        Condition variables;
        Procedure P1{……..}
        Procedure P2{……..}
        …….
        Procedure Pn{……..}
}
{
        Initilizing_code
}
```

CSC259: Operating System

- Monitor in an operating system is simply a class containing variable_declarations, condition_variables, various procedures (functions), and an initializing_code block that is used for process synchronization.
- Only one process can be active in a monitor at a time.
- Other processes that need to access the shared variables in a monitor have to line up in a queue and are only provided access when the previous processes release the shared variables.

97

CSC259: Operating System

Message Passing:

Message Passing provides a mechanism to allow processes to communicate and to synchronize their actions without sharing the same address space.

- Inter-process communication uses two primitives, send and receive.
- They can easily be put into library procedures, such as **send(destination, &message)**; **receive(source, &message);**
- The former call sends a message to a given destination and the latter one receives a message from a given source.
- If no message is available, the receiver can block until one arrives. Alternatively, it can return immediately with an error code.

98

CSC259: Operating System

Message passing systems have many problems,

- Especially if the communicating processes are on different machines connected by a network. For example, messages can be lost on the network.
- To solve this problem, as soon as a message has been received, the receiver will send back a special acknowledgement message.
- If the sender has not received the acknowledgement within a certain time interval, it retransmits the message.
- Now consider what happens if the message itself is received correctly, but the acknowledgement is lost. The sender will retransmit the message, so the receiver will get it twice.
- It is essential that the receiver is able to distinguish a new message from the retransmission of an old one.

CSC259: Operating System

- Usually, this problem is solved by putting consecutive sequence numbers in each original message.
- If the receiver gets a message bearing the same sequence number as the previous message, it knows that the message is a duplicate that can be ignored.
- There are also design issues that are important when the sender and receiver are on the same machine. One of these is performance. Copying messages from one process to another is always slower than doing a semaphore operation.

CSC259: Operating System

2.6 Process Scheduling:

- Process scheduling is a technique that is used when there are limited resources and many processes are competing for them.
- Multiprogramming tries to ensure that there is some process running at all times. This is done to utilize the CPU as much as possible.
- In time sharing system, the CPU switches so frequently between jobs that the user not feel that the machine is being shared by many processes or even many users.
- If the system has more than one processor, then it is possible to execute more than one process at the same time. In a single system, only one process can executed at any given time.
- If there are more processors, then the operating system must schedule the processes. There are many strategies for deciding which process should be assigned to the CPU.

101

CSC259: Operating System

Categories of Scheduling

There are two categories of scheduling:

1.Non-preemptive: Here the resource can't be taken from a process until the process completes execution. The switching of resources occurs when the running process terminates and moves to a waiting state.

2.Preemptive: Here the OS allocates the resources to a process for a fixed amount of time. During resource allocation, the process switches from running state to ready state or from waiting state to ready state. This switching occurs as the CPU may give priority to other processes and replace the process with higher priority with the running process.

102

CSC259: Operating System

Difference between preemptive and non-preemptive scheduling:

Preemptive Scheduling	Non-Preemptive Scheduling
The resources are assigned to a process for a long time period.	Once resources are assigned to a process, they are held until it completes its burst period or changes to the waiting state.
Its process may be paused in the middle of the execution.	When the processor starts the process execution, it must complete it before executing the other process, and it may not be interrupted in the middle.
When a high-priority process continuously comes in the ready queue, a low-priority process can starve.	When a high burst time process uses a CPU, another process with a shorter burst time can starve.
It has overheads associated with process scheduling.	It doesn't have overhead.
It affects the design of the operating system kernel.	It doesn't affect the design of the OS kernel.
Its CPU utilization is very high.	Its CPU utilization is very low.
Examples: Round Robin and Shortest Remaining Time First	FCFS and SJF are examples of non-preemptive scheduling.

103

CSC259: Operating System

Process Scheduling Queues

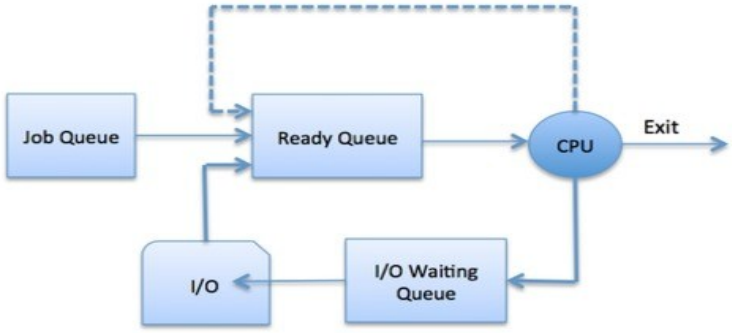
The OS maintains all Process Control Blocks (PCBs) in Process Scheduling Queues. The OS maintains a separate queue for each of the process states and PCBs of all processes in the same execution state are placed in the same queue. When the state of a process is changed, its PCB is unlinked from its current queue and moved to its new state queue.

The Operating System maintains the following important process scheduling queues –

- **Job queue** − This queue keeps all the processes in the system.
- **Ready queue** − This queue keeps a set of all processes residing in main memory, ready and waiting to execute. A new process is always put in this queue.
- **Device queues** − The processes which are blocked due to unavailability of an I/O device constitute this queue.

104

CSC259: Operating System



The OS can use different policies to manage each queue (FIFO, Round Robin, Priority, etc.). The OS scheduler determines how to move processes between the ready and run queues which can only have one entry per processor core on the system; in the above diagram, it has been merged with the CPU.

105

CSC259: Operating System

Schedulers

Schedulers are special system software which handle process scheduling in various ways. Their main task is to select the jobs to be submitted into the system and to decide which process to run. Schedulers are of three types −

- Long-Term Scheduler
- Short-Term Scheduler
- Medium-Term Scheduler

106

CSC259: Operating System

Long Term Scheduler

It is also called a **job scheduler**. A long-term scheduler determines which programs are admitted to the system for processing. It selects processes from the queue and loads them into memory for execution. Process loads into the memory for CPU scheduling.

The primary objective of the job scheduler is to provide a balanced mix of jobs, such as I/O bound and processor bound. It also controls the degree of multiprogramming. If the degree of multiprogramming is stable, then the average rate of process creation must be equal to the average departure rate of processes leaving the system.

On some systems, the long-term scheduler may not be available or minimal. Time-sharing operating systems have no long term scheduler. When a process changes the state from new to ready, then there is use of long-term scheduler.

CSC259: Operating System

Short Term Scheduler

It is also called as **CPU scheduler**. Its 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.

Short-term schedulers, also known as dispatchers, make the decision of which process to execute next. Short-term schedulers are faster than long-term schedulers.

Medium Term Scheduler

Medium-term scheduling is a part of **swapping**. It removes the processes from the memory. It reduces the degree of multiprogramming. The medium-term scheduler is in-charge of handling the swapped out-processes.

A running process may become suspended if it makes an I/O request. A suspended processes cannot make any progress towards completion. In this condition, to remove the process from memory and make space for other processes, the suspended process is moved to the secondary storage. This process is called **swapping**, and the process is said to be swapped out or rolled out. Swapping may be necessary to improve the process mix.

CSC259: Operating System

Comparison among Scheduler

S.N.	Long-Term Scheduler	Short-Term Scheduler	Medium-Term Scheduler
1	It is a job scheduler	It is a CPU scheduler	It is a process swapping scheduler.
2	Speed is lesser than short term scheduler	Speed is fastest among other two	Speed is in between both short and long term scheduler.
3	It controls the degree of multiprogramming	It provides lesser control over degree of multiprogramming	It reduces the degree of multiprogramming.
4	It is almost absent or minimal in time sharing system	It is also minimal in time sharing system	It is a part of Time sharing systems.
5	It selects processes from pool and loads them into memory for execution	It selects those processes which are ready to execute	It can re-introduce the process into memory and execution can be continued.

CSC259: Operating System

Dispatcher

- A dispatcher is a special program which comes into play after the scheduler.
- When the scheduler completes its job of selecting a process, it is the dispatcher which takes that process to the desired state/queue.
- The dispatcher is the module that gives a process control over the CPU after it has been selected by the short-term scheduler. This function involves the following:
  - Switching context
  - Switching to user mode
  - Jumping to the proper location in the user program to restart that program

CSC259: Operating System

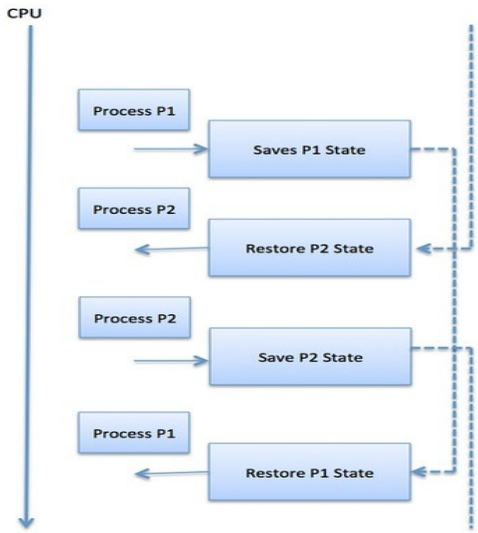
Context Switching

A context switching is the mechanism to store and restore the state or context of a CPU in Process Control block so that a process execution can be resumed from the same point at a later time. Using this technique, a context switcher enables multiple processes to share a single CPU. Context switching is an essential part of a multitasking operating system features.

When the scheduler switches the CPU from executing one process to execute another, the state from the current running process is stored into the process control block. After this, the state for the process to run next is loaded from its own PCB and used to set the PC, registers, etc. At that point, the second process can start executing.

111

CSC259: Operating System



112

CSC259: Operating System

Context switches are computationally intensive since register and memory state must be saved and restored. To avoid the amount of context switching time, some hardware systems employ two or more sets of processor registers. When the process is switched, the following information is stored for later use.

- Program Counter
- Scheduling information
- Base and limit register value
- Currently used register
- Changed State
- I/O State information
- Accounting information

CSC259: Operating System

Goals/Objectives of Process Scheduling Algorithm

- **Utilization of CPU at maximum level**. **Keep CPU as busy as possible**.
- **Allocation of CPU should be fair**.
- **Throughput should be Maximum**. i.e. Number of processes that complete their execution per time unit should be maximized.
- **Minimum turnaround time**, i.e. time taken by a process to finish execution should be the least.
- There should be a **minimum waiting time** and the process should not starve in the ready queue.
- **Minimum response time.** It means that the time when a process produces the first response should be as less as possible.

CSC259: Operating System

Things to Take Care While Designing a CPU Scheduling Algorithm

Different **CPU Scheduling algorithms** have different structures and the choice of a particular algorithm depends on a variety of factors. Many conditions have been raised to compare CPU scheduling algorithms.

The criteria include the following:

•**CPU Utilization:** The main purpose of any CPU algorithm is to keep the CPU as busy as possible. Theoretically, CPU usage can range from 0 to 100 but in a real-time system, it varies from 40 to 90 percent depending on the system load.

•**Throughput:** The average CPU performance is the number of processes performed and completed during each unit. This is called throughput. The output may vary depending on the length or duration of the processes.

115

CSC259: Operating System

•**Turn Round Time:** For a particular process, the important conditions are how long it takes to perform that process. The time elapsed from the time of process delivery to the time of completion is known as the conversion time. Conversion time is the amount of time spent waiting for memory access, waiting in line, using CPU, and waiting for I / O.

•**Waiting Time:** The Scheduling algorithm does not affect the time required to complete the process once it has started performing. It only affects the waiting time of the process i.e. the time spent in the waiting process in the ready queue.

•**Response Time:** In a collaborative system, turn around time is not the best option. The process may produce something early and continue to computing the new results while the previous results are released to the user. Therefore, another method is the time taken in the submission of the application process until the first response is issued. This measure is called response time.

116

CSC259: Operating System

Terminologies Used in CPU Scheduling

•**Arrival Time (AT):** The time at which the process enters into the ready queue is called as arrival time.

Burst Time (BT): The amount of time required by the CPU to execute the process is called Burst Time. This does not include waiting time.

•**Completion Time (CT):** The time at which the process completes its execution is called as completion time.

•**Turn Around Time (TAT):** The amount of time spent by the process from its arrival to its completion is called as turn around time.

TAT = CT-AT

TAT=WT+BT

Therefore, CT-AT=WT+BT

•**Waiting Time(W.T):** Time Difference between turn around time and burst time.

- Waiting Time = Turn Around Time (TAT) – Burst Time(BT)

117

CSC259: Operating System

Scheduling Algorithm:

Scheduling algorithm mainly divides into 3 categories:

1. **Batch system scheduling**
   a. First Come First Serve (FCFS) (Non-Preemptive)
   b. Shortest Job First (SJF) (Non-Preemptive)
   c. Shortest Remaining Time Next (SRTN) (Preemptive)
2. **Interactive system scheduling**
   a. Round Robin Scheduling (Preemptive)
   b. Priority Scheduling (Preemptive)
   c. Multiple Queue (Multilevel Queue Scheduling) (Non-Preemptive)
3. **Real Time scheduling**

118

CSC259: Operating System

a. First Come First Serve (FCFS):

- FCFS is the simplest non-preemptive algorithm. Processes are assigned the CPU in the order they request it. That is the process that requests the CPU first is allocated the CPU first.
- The implementation of FCFS is policy is managed with a FIFO(First in first out) queue.
- When the first job enters the system from the outside in the morning, it is started immediately and allowed to run as long as it wants to.
- As other jobs come in, they are put onto the end of the queue.
- When the running process blocks, the first process on the queue is run next.
- When a blocked process becomes ready, like a newly arrived job, it is put on the end of the queue.

CSC259: Operating System

Advantages

•Very easy to perform by the CPU

•Follows FIFO Queue Approach

Disadvantages

•First Come First Serve is not very efficient.

Problem:

- No guarantee of good response time
- Large average waiting time.

Numerical Example :

See in Copy note.

CSC259: Operating System

b. Shortest Job First (SJF):

- This is also non preemptive scheduling algorithm.
- It is the best method to minimize the waiting time.
- It selects the waiting process with smallest execution time to execute next. This is the advantage as shortest process are handled very quickly.

Characteristics:

- The processing times are known in advanced.
- SJF selects the process with shortest expected processing time.
- The decision policies are based on the CPU burst time.

Problem:

- Not applicable in timesharing system.

CSC259: Operating System

c. Shortest Remaining Time Next (SRTN):

- This is preemptive scheduling algorithm.
- The process with the smallest amount of time remaining until completion is selected to execute.
- Shortest remaining time is advantageous because shortest process are handled very quickly.
- When a new process is added the algorithm only needs to compare the currently executing process with the new process ignoring all other processed currently waiting to execute.

Advantages:

- Useful in timesharing.
- Low average waiting time than SJF.

Disadvantages:

- Very high overhead than SJF.
- Required additional computation.

123

CSC259: Operating System

2. Interactive System Scheduling:

a) Round Robin (RR) Scheduling:

- Round Robin CPU Scheduling is the most important CPU Scheduling Algorithm which uses Time Quantum (TQ). The Time Quantum is something which is removed from the Burst Time and lets the chunk of process to be completed.
- Time Sharing is the main emphasis of the algorithm. Each step of this algorithm is carried out cyclically. The system defines a specific time slice, known as a time quantum.
- First, the processes which are eligible to enter the ready queue enter the ready queue. After entering the first process in Ready Queue is executed for a Time Quantum chunk of time. After execution is complete, the process is removed from the ready queue. Even now the process requires some time to complete its execution, then the process is added to Ready Queue.
- The Ready Queue does not hold processes which already present in the Ready Queue. The Ready Queue is designed in such a manner that it does not hold non unique processes. By holding same processes Redundancy of the processes increases.

124

CSC259: Operating System

Advantages

The Advantages of Round Robin CPU Scheduling are:

1.A fair amount of CPU is allocated to each job.

2.Because it doesn't depend on the burst time, it can truly be implemented in the system.

3.It is not affected by the convoy effect or the starvation problem as occurred in First Come First Serve CPU Scheduling Algorithm.

Disadvantages

The Disadvantages of Round Robin CPU Scheduling are:

1.Low Operating System slicing times will result in decreased CPU output.

2.Round Robin CPU Scheduling approach takes longer to swap contexts.

3.Time quantum has a significant impact on its performance.

CSC259: Operating System

b) **Priority Scheduling**

In Priority scheduling, there is a priority number assigned to each process. In some systems, the lower the number, the higher the priority. While, in the others, the higher the number, the higher will be the priority. The Process with the higher priority among the available processes is given the CPU. There are two types of priority scheduling algorithm exists. One is Preemptive priority scheduling while the other is Non Preemptive Priority scheduling.

The priority number assigned to each of the process may or may not vary. If the priority number doesn't change itself throughout the process, it is called static priority, while if it keeps changing itself at the regular intervals, it is called dynamic priority

CSC259: Operating System

Advantages of priority scheduling (Non preemptive)

- Higher priority processes like system processes are executed first.

Disadvantages of priority scheduling (Non preemptive)

- It can lead to starvation if only higher priority process comes into the ready state. Low priority processes may never be executed.
- If the priorities of more two processes are the same, then we have to use some other scheduling algorithm.

127

CSC259: Operating System

What is Starvation in Operating System ?

Starvation is a problem with resource management where a process runs out of resources in the OS because those resources are being utilized by other processes. This issue primarily arises in a priority-based scheduling system where requests with high priority are processed first and those with low priority take longer to process

What exactly does starvation mean in operating systems?

Problems arise when high-priority requests lead low-priority processes to become stalled for an extended period of time. The low-priority process cannot get the processor or resources because of a constant stream of high-priority demands. Starvation typically occurs when a task is postponed for an endless amount of time. The operating system requires the following resources to respond to process requests:

•I/O access to disk or network

•Memory, Disk space, Bandwidth of network, CPU time

128

CSC259: Operating System

•**Causes of Starvation in OS:**

•If a higher priority operation uses a CPU continuously, a low priority process may wait indefinitely in starvation.

•Resources are not available in sufficient quantities to meet all needs.

•There is a chance that a process may have to wait for a long time if a process selection is random.

•Due to improper resource allocation, a process is never given a resource to use for execution.

129

CSC259: Operating System

•**Various OS Starvation Management Techniques**

•To make sure that resources are distributed equally, a freelancing manager should be in charge of CPU resource allocation.

•It is best to avoid selecting a process approach at random.

•It is important to take processes' ageing requirements into account.

•There is a chance that a process may have to wait for a long time if a process selection is random.

•Starvation can also happen when an improper resource allocation prevents a process from ever receiving a resource for use.

130

CSC259: Operating System

c. Multiple Queues (Multilevel Queue Scheduling):

Operating systems use a particular kind of scheduling algorithm called multilevel queue scheduling to control how resources are distributed across distinct tasks. It is an adaptation of the conventional queue-based scheduling method, in which processes are grouped according to their priority, process type, or other factors

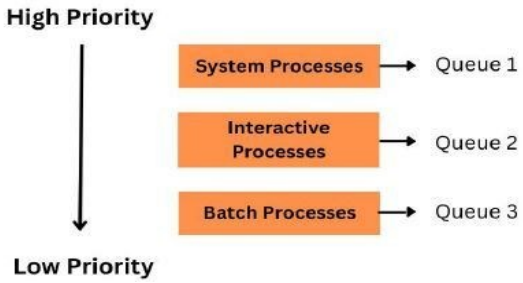
The system can allocate system resources based on the priority and needs of the processes by assigning a separate scheduling algorithm to each queue. For instance, the background queue may employ first-come-first-serve scheduling to maximize the usage of system resources for longer-running activities, while the foreground queue might use Round Robin scheduling to prioritize interactive processes and speed up reaction time.

The goal of multilevel queue scheduling is to strike a compromise between fairness and performance. The system can increase overall performance while making sure that all processes are treated equitably by giving some processes more priority than others and distributing resources accordingly

131

CSC259: Operating System

Components of the MLQ architecture



In an MLQ (Multilevel Queue) architecture, the system is divided into multiple queues or levels based on the type of processes. The MLQ architecture aims to efficiently handle different types of processes and prioritize their execution based on their characteristics.

132

CSC259: Operating System

•**System Processes** − System processes refer to operating system tasks or processes that are essential for the proper functioning of the system. These processes handle critical operations such as memory management, process scheduling, I/O handling, and other system-level functions. System processes typically have the highest priority in the MLQ architecture and are executed before other types of processes.

•**Interactive Processes** − Interactive processes are user-oriented processes that require an immediate or near-real-time response. These processes are typically initiated by user interaction, such as keyboard input or mouse clicks. Examples of interactive processes include running a text editor, web browser, or a graphical application. Interactive processes are given priority in the MLQ architecture to ensure a smooth and responsive user experience.

CSC259: Operating System

•**Batch Processes** − Batch processes are non-interactive processes that are executed in the background without direct user interaction. These processes often involve executing a series of tasks or jobs that can be automated and executed without user intervention. Batch processes are typically resource-intensive and can include tasks such as data processing, large-scale computations, or scheduled backups. In the MLQ architecture, batch processes are allocated resources based on their priority, but they are generally given lower priority compared to interactive processes.

CSC259: Operating System

Example Problem

Let's take an example of a multilevel queue-scheduling (MQS) algorithm that shows how the multilevel queue scheduling work. Consider the four processes listed in the table below under multilevel queue scheduling. The queue number denotes the process's queue.

Process	Arrival Time	CPU Burst Time	Queue Number
P1	0	5	1
P2	0	3	2
P3	0	8	2
P4	0	6	1

In the above example, we have two queues .i.e queue1 and queue2. Queue 1 is having higher priority and using FCFS approach and queue 2 is using Round Robin approach **(Time Quantum = 2),**

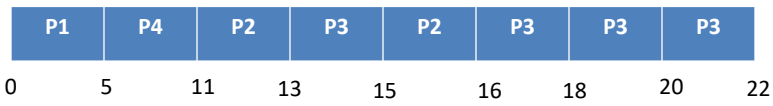
CSC259: Operating System

Since the priority of queue 1 is higher, so queue 1 will be executed first.

In the queue 1, we have two processes i.e P1 and P4 and we are using FCFS. So, P1 will be executed followed by P4. Now the job of queue 1 is finished.

After this, the execution of the processes of queue 2 will be started by using the round robin approach.

Gantt chart of above problem



CSC259: Operating System

Overview of Real Time System Scheduling:

- A real time scheduling system is composed of the scheduler, clock and the processing hardware elements.
- In a real time system, a process or task has schedulable, tasks are accepted by a real time system and completed as specified by the task deadline depending on the characteristic of the scheduling algorithm.
- Modeling and evaluation of a real time scheduling system concern is on the analysis of the algorithm capability to meet a process deadline.
- A task in a real time system must be completed neither too early nor too late.
- A real time scheduling algorithm can be classified as static or dynamic. For a static scheduler , task priorities are determined before the system runs. A dynamic scheduler determines task priorities as it runs.

137

CSC259: Operating System

Task are accepted by the hardware elements in a real time scheduling system from the computing environment and processed in real time. An output signal indicates the processing status. A task deadline indicates the time set to complete for each task like

- Priority fair share scheduling
- Guaranteed scheduling
- Lottery scheduling
- Highest response ratio next (HRN)

138

CSC259: Operating System

Thank you