Process Management

Date:
Page:

Unit-2

A process is an instance of a program running in computer with which a particular set of data is allocated to that it's tracked can be kept. Furth a process is defined as an entity which represents the basic unit of work to be implemented on the system. A process can initiate a sub process which is kno as child process. A child process is a replica of pa process and share some of its resources. but cannot e if the parent is terminated.

IPC → Process can exchange information or synch ize their operation through several methods of inter process communication.

Program → Program is an executable file that contai the set of instructions to perform specific job in computer system. Program resides in secondary memo and sometimes it is known as passive entity. Progra are never stored in primary memory but they are rea into primary memory executed by kernel when requir A program is a piece of code which may be th line or millions of which performs specific task. Some programs are : operating system, web brows like Mozilla fire fox, office package, video games

Note : All the process are programs but all the prog are not processed.

Date:
Page:

(*) Write the differences between process and program

→ The differences between process and program are:-

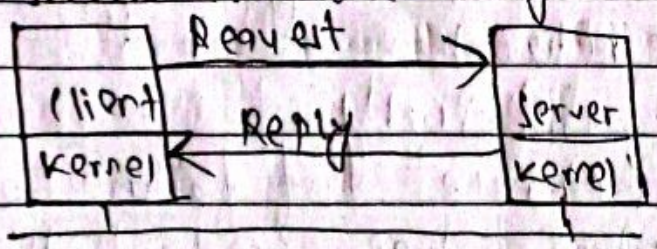
Process	Program
(i) Process is an active entity of a program.	Program is a passive/inactive entity.
(ii) It is stored or located in the primary memory i.e. RAM	It is stored or located in the secondary memory i.e. Hard disk
(iii) All processes are program	All program are not process
(iv) It has a high resource requirement.	It doesn't have any resource requirement.
(v) It has its own control block called process control block.	It doesn't have any control block.
(vi) It exists for a limited span of time as it gets terminated after the completion of task.	It exists at a single place & continues to exist until it is deleted.

Unit 1 continue ↓

(#) Client-Server Model

→ The most important and most widely used distributed application software.

→ Client & servers roles are assigned and changeable.



- Also known as Request - Response Architecture.
- Client makes a request to the server and server fulfill the response.

Working mechanism of Client - Server Model:

- Client : A Client is a computer host (Host) i.e. capable of receiving information or using a particular service from the service providers (server).
- Servers → server is a remote computer which provides information (data) or access to a particular service.
- to, it is basically the client requesting something and the server serving it as long as present in database.

Advantages:

⇒ Centralized with all data in a single place.
⇒ Less maintenance cost.
⇒ Data Recovery is possible.
⇒ The capacity of the client and server can be changed separately.

Disadvantages:

⇒ High costs (The cost of setting up and maintaining the server is typically than cost of running the network.
⇒ Maintenance difficulty
⇒ Robustness (chance of failure or breakable)
⇒ phishing, capturing login credentials of the user are common.

Date
Page

(1) Virtual Machine (VM)

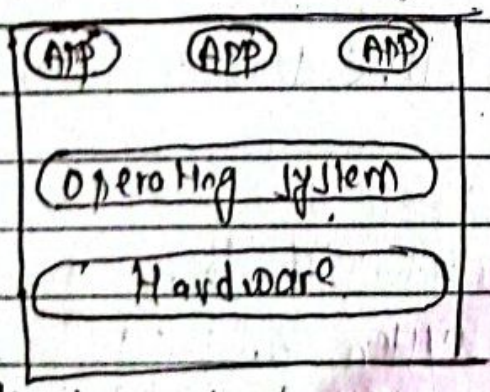
→ Virtual Machine is a virtual environment which functions as a virtual computer system with its own CPU, Memory, network interface and storage, created on a physical hardware system.

→ They are isolated from the rest of the system & multiple VMs can exist on a single piece of hardware, like a server

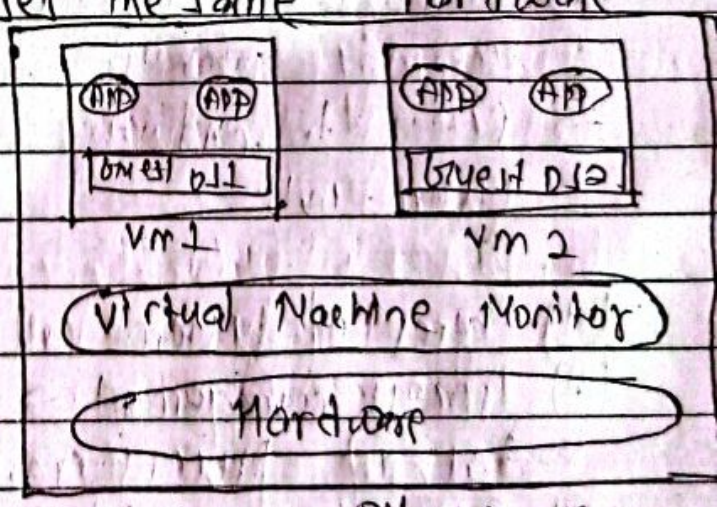
→ It has its own OS & software that will facilitate the resources to virtual computer

→ Separate security & configuration identity.

→ Multiple OS system uses the same hardware



Single OS: No VM



Multiple OS with sharing resources on VM

(11) System calls (Acts as an interface from shell to kernel)

→ A system call is a mechanism that provides the interface between a process and the operating system.

→ It is a programming method in which a computer program requests a service from kernel of the OS

→ Way for programs ~~that~~ to interact with the OS.

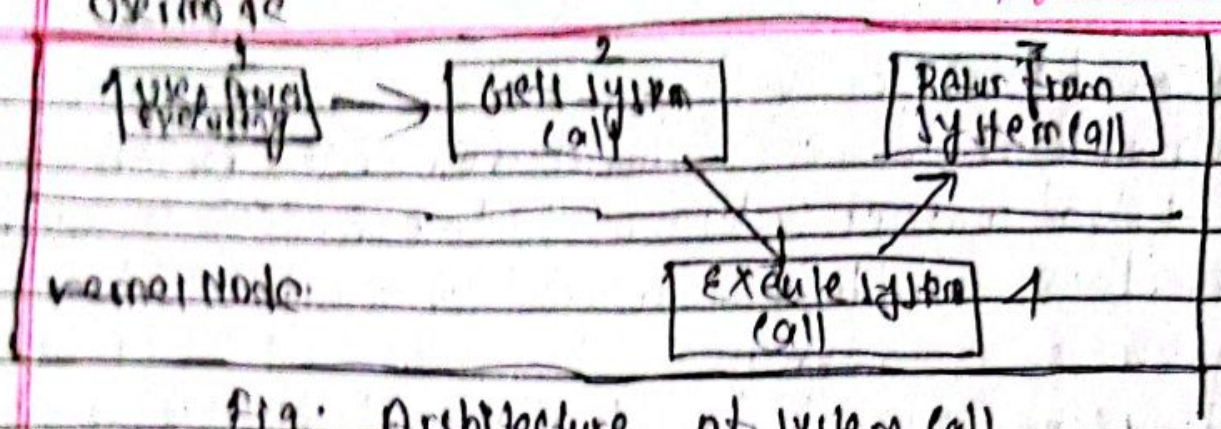


Fig: Architecture of system call

(ii) Why system call is needed in OS?

→ Reading and writing from files, demand system call

→ Creation & deletion of files requires " "

→ For creation and management of new process

→ Network connections need system calls for sending & receiving packets.

→ Access to hardware devices like scanner printer require system call.

(iii) Types of system calls in OS with their function.

(i) Process management → This system calls perform the task of process creation, process termination, etc.

(ii) File Management → It handle file manipulation jobs like creating, reading, writing a file, etc.

(iii) Device Management → It does device manipulation like reading from device buffers, writing into device buffers, etc.

(iv) Information Maintenance → It handles information and its transfer between OS and user program.

Date:
Page:

→ Functions:

Process Control	File Management	Device Management	Information Maintenance	Communication
→ End and Abort	→ Create a file	→ Request & release device	→ Get or set time date	→ Send and receive message
→ Load and Execute	→ Delete a file	→ Logically attach detach devices	→ Get process and device attributes	→ Create delete communication connections
→ Wait and signal Event	→ Open and close file	→ Get and set device attributes		→ Attach or detach remote devices
→ Create & terminate process	→ Get and set file attributes			→ Help to transfer status information
→ Allocate the memory	→ Read, write & reposition			

(v) Communication: These types of system calls are specially used for interprocess communication.

(II) Important system calls used in OS.

(i) wait(): It suspends execution of the current process until one of its children terminates.

(ii) fork(): It is used to create processes that are a copy of themselves.

(iii) exec(): When an executable file replaces an earlier executable file within the context of an process, exec() is used.

(iv) kill(): It is used to urge process to terminate by sending them a signal kill()

Date:
Page:

(5) exit() : An exit system call is used when the programmable must be stopped.

(ii) Shell

→ A shell is a special user program which provide an interface to user to use ~~program~~ operating system services.

→ A shell accepts human readable commands from users and convert them into something which kernel can understand.

Two types of shell:

(i) Command Line shell : Eg (Bash (sh), cmd, (shell)

(ii) GUI shell : Eg (Windows explorer or windows shell)

(iii) GCLI (Graphical Command LI) shell : It is recently developed shell which has features of both CLI and GUI shell.

→ provides an interface which is both user friendly and powerful : Eg : Windows 10.

(x) Process Management ⇒ It is the technique of creating the process, allocating the required resources, scheduling those processes and executing them systematically. The process after creation gets terminated after its execution.

(x) Process architecture ⇒ The simplified layout of process inside the memory is drawn below :

Date.....
Page.....

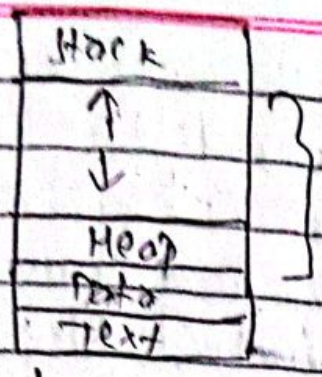


Fig: Layout of process inside Memory

→ Stack: The process stack contains the temporary data such as method / function, parameters, return address and local variables.

→ Heap: This is ~~the~~ a dynamically allocated memory to a process during its run time.

→ Data: This section contains global and static variable.

→ Text: This includes the current activity which is represented by the value of program counter and the contents of processor's ~~and~~ register.

(#) Process lifecycle → The figure showing process lifecycle is drawn below:

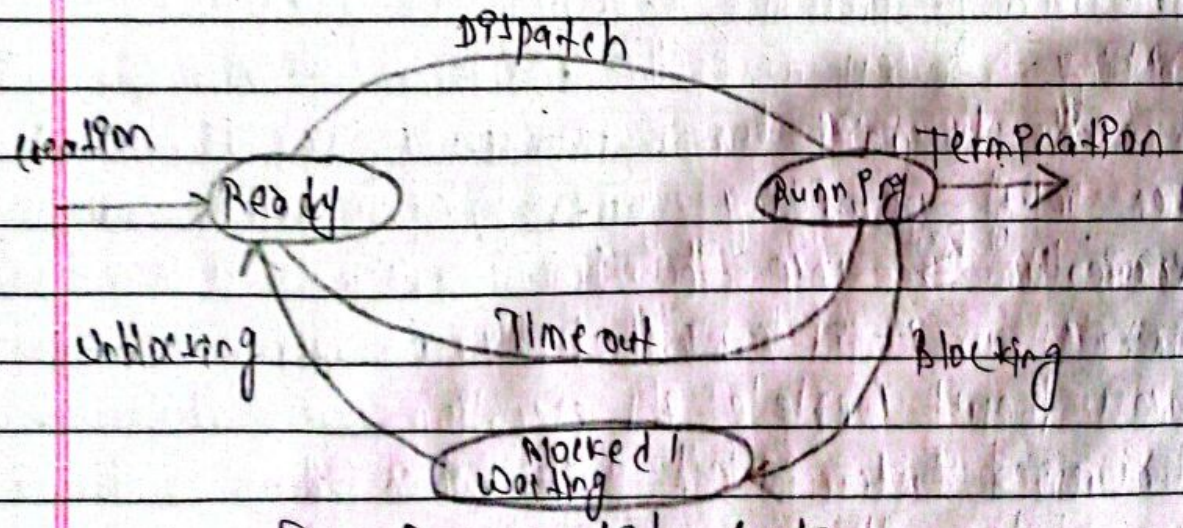


Fig: Process Life cycle.

When a process executes, it passes through different stages. These stages may different OS and the name of these stages are not standardize. The OS maintains management information about the process in a PCB. (Process Control Block). Typically a process can have one of the following five stages at a time.

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

- Creation / Start ⇒ This is the initial state when the process is first created / started by operating system.
- Ready: The process is waiting to be assigned to a processor. Ready process are waiting to have the processor allocated to them by OS so that they can run. Process may come into this state after start state or while running it but interrupted by the scheduler to assign CPU to some other process.
- Running ⇒ Once the process has been assigned to the processor by OS scheduler the process state is set to running state and the processor executes its instructions.
- Waiting ⇒ Process moves the waiting / blocked state, if it needs to wait for a resource, such as waiting for user input or waiting for a file to become available.
- Termination or Exit ⇒ Once the process finishes its execution, it terminates or it is terminated by OS when the running process doesn't complete its execution even after long time. It is moved to terminated state when it waits to be removed from the main memory.

Date:
Page:

Suspended Process → The process that is not immediately available for an execution is known as suspended process.

Characteristics of suspend process

- The process may or may not be waiting on an event.
- For preventing the execution process is suspended by OS, parent process and an agent.
- Process may not be removed from the suspended state until the agent orders the removal.

(*) Swapping → Swapping is the process of moving all the processes from the main memory to disk and and loading the programs from disk to main memory.

(*) PCB (Process control Block) → While creating a process, the OS perform several operation. To identify these processes it must identify each processes, Hence it assigns a process identification number (PID) to each processor. As OS supports multiprogramming it needs to keep track of all the processes and for this process control block is used. A PCB is a data structure that contains the information about the process which is maintained by OS. Each process has its own PCB. When the process moves transition from one state to another, the OS must update the information in their respective PCB.

Role of PCB.

→ The role or work of PCB in process management is that it can access or modify by most of utilities including

Date:
Page:

these are involved with memory, scheduling and I/O resource access. Thus it can be said that, the set of process control block gives the information of current of OS. The various information that is contained in PCB is listed below:

- Name of the process
- State of the process
- Resources allocated to the process.
- Memory allocated to the process.
- Scheduling information
- Input-output devices associated with process.

(*) Components of PCB: A PCB keeps all the information needed to keep the track of process which are listed below:

- Process State → we know that, process state on any process can be new, running, waiting or blocked, suspend and terminated. PCB is used to define process state of process.
- Process Privileges → This is required to allow or disallow access to system resources.
- Process Id ⇒ In computer system there are various process running simultaneously and each process has its own unique id and this id helps the system in scheduling the process and is provided by PCB.
- Pointer → It is used to point the parent process or next PCB to be executed.

Date:
Page:

- Program counter → It is the pointer to the address of next instructions to be executed for the process.
- CPU Register → This provides information regarding various regarding index and stack that are associated with process and such information are managed by CPU.
- CPU scheduling information ⇒ scheduling information is used to set the priority of different processes which is very useful information managed by PCB. The priority of primary feature of RAM is higher than secondary feature
- Memory management information → This includes the information of page table, memory limits, segment table depending on memory used by OS.
- Accounting information → This includes the amount of CPU used for process execution, time limits, execution id etc.
- I/O status information: This includes the information regarding a list of I/O devices allocated to the process

Process Id
State
Pointer
Priority
Program counter
CPU Registers
I/O Information
Account Information

Fig: PCB diagram

Date:
Page:

(7) Operations on process

⇒ There are many operations that can be performed on the processes. Some of these are process creation, process preemption, process-blocking and process termination which are described below.

(a) Process Creation → Process need to be created in the system for different operation. This can be done by the following events:

- User request for process creation.
- System initialization.
- Execution of a process creation system call by running process (parent process)
- Batch job initialization.
- Note: A process may be created by another process by using fork(). The creating process is called parent process and the created process is called child process.

(b) Process preemption ⇒ An interrupt mechanism is used in preemption that suspends the process executing currently and the next process to execute is determined by short term scheduler. Preemption guarantees that all the process get some CPU time time for execution. A diagram showing process preemption is given below:

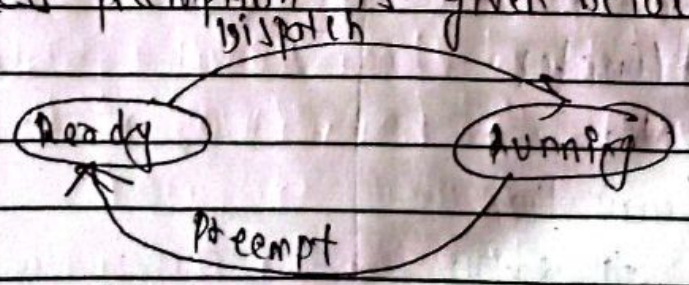


Fig: Process Preemption

(c) Process Blocking ⇒ The process is blocked if it is waiting for some event to occur. This event may be I/O as the I/O events are executed in the main memory and doesn't require the processor. A diagram showing process blocking is drawn below:

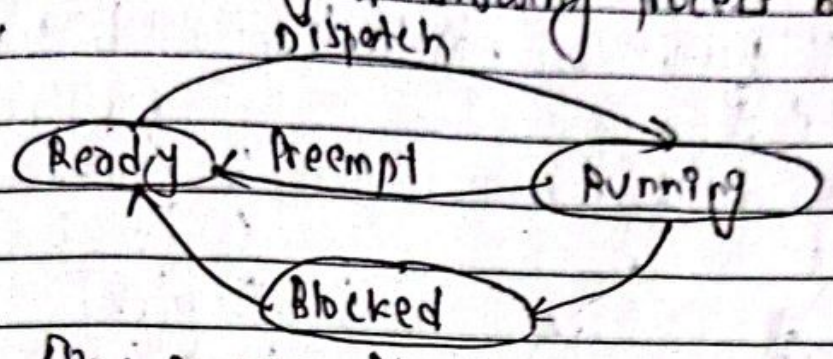


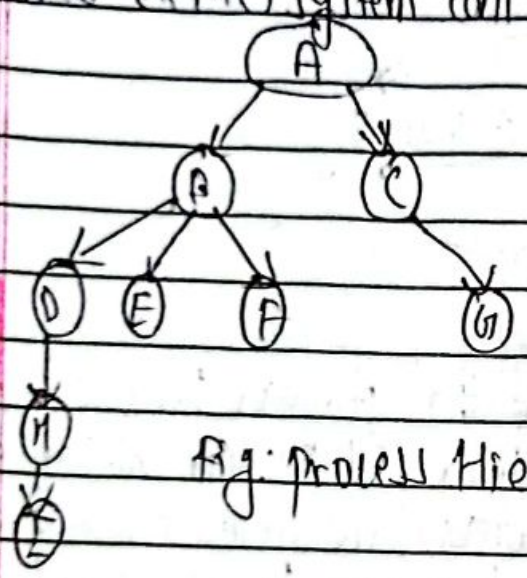
Fig: Process Blocking

(d) Process Termination ⇒ After the process has completed the execution of its last instruction it is terminated normally. The resource held by a process are released after its termination. Some of the causes of process termination are as follows:

- A process may be terminated after its execution is naturally completed leaving the processor and releasing the resources.
- A child process may be terminated if its parent process requests for its terminating.
- A process can be terminated if it tries to use a resource that is not allow to it.
- If a process requires more memory than currently available in the system it is terminated because of memory scarcity.

(X) Process Hierarchies ⇒ The process of creating a new process from their parent process and again creating a new process from previously newly created process in a tree like structure is

called process hierarchy. Windows has no concept of process Hierarchy. Modern general purpose OS permits a user and destroy the process. In UNIX OS, this task is by the fork() system call which creates a child process and exit() system call which terminates the current process



A forks B and C
B forks D, E and F
C forks G
D forks H
H forks I

Fig: process Hierarchy

(*) Process Implementation → Process model is implemented by process table and process control block which keeps the track of all the information regarding the process. At the time creation of new process, operating system allocates a memory for it, loads the process code into the allocated memory and setup data space for it. The state of a process is stored as new in its PCB and when this process move to ready state, its state is also changed in PCB. When a running process needs to wait for an input/output device its state is changed into blocked.

There are various queues used for process implementation which are implemented as linked list.

(a) Ready queue → This queue is used for storing the process

that are in ready state.

(b) Blocked queue → It is used for storing the processes that need to wait for an input/output device or resource.

(c) Suspended Queue → It is used for storing the blocked process that has been suspended.

(d) Free process queue → It is used for the information of empty space in the memory where a new PCB can be created

Note: Each PCB has a pointer that points to next PCB. There is a header for each type of queue. The header stores the information about the first and last PCB in that queue.

Co-operating process:

→ Cooperating process are those that coordinates with each other by using shared data or messages. Co-operating process can affect or be affected by other process running on the system. In the computer system there are many process which may be either independent process or cooperating process that run in the system. A process is said to be independent when it cannot affect or affected by another process that are running on the system. There are several reasons for providing an environment that allows process co-operation.

(a) Modularity → Dividing the tasks into subtasks (complicated) and completed by different co-operating process. This leads to faster and more efficient of required tasks

(b) Information sharing → Sharing the information betn one and other process. Mechanism required so that the process can access file ll to each other

Date:
Page:

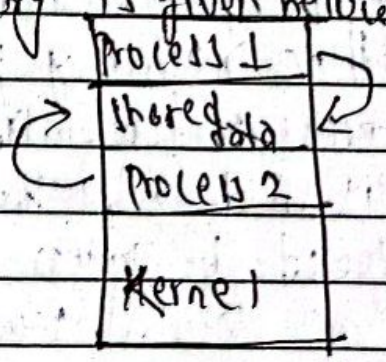
(c) Convinence → (Good / Comfort / Ease) is made by us cooperating process.

(d) Computation speedup → Subtask of a task can be perf parallel along cooperating process. The increase the exec speed and this is only possible if the system has multiple processing element.

(*) Methods of co-operation:

Cooperating process can co-ordinate with each other by using shared data or messages. The methods of co-oper are described below:

(a) Co-operation by sharing → The co-operating process can co-op with each other by using shared data such as memory, variable, files, database, etc. Critical section is used to provide data integrity and ~~acting~~ writing is mutually exclusi to prevent inconsistent data. A figure demonstrating co-o ation by sharing is given below.



In above diagram, process P1 and P2 can co-operate wi each other using shared data such as memory, variables, f etc.

(b) Cooperation by communication → The cooperating process

co-operate with each other by using messages. This may lead to deadlock if each process is waiting for a message from the other to perform a certain operation. Starvation is also possible if a process never receives a message. The diagram demonstrating co-operation by communication is given below

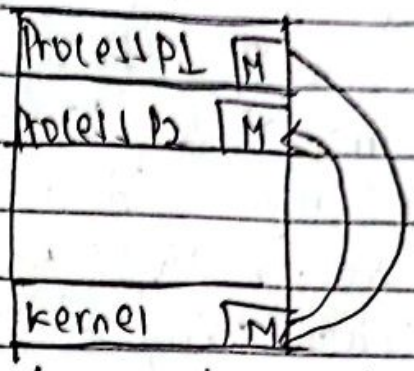


Fig: Co-operation by communication.

In above diagram, process P1 and P2 can co-operate with eachother by using message by the help of ~~control~~ kernel.

System calls → The interface between a process and an operating system is provided by system call. In general, system calls are available as assembly language instructions. System calls are usually made when a process in a user mode requires access to resource. Then it requests the kernel to provide the resource with the help of system call. A diagram representing execution of a system call is given below.

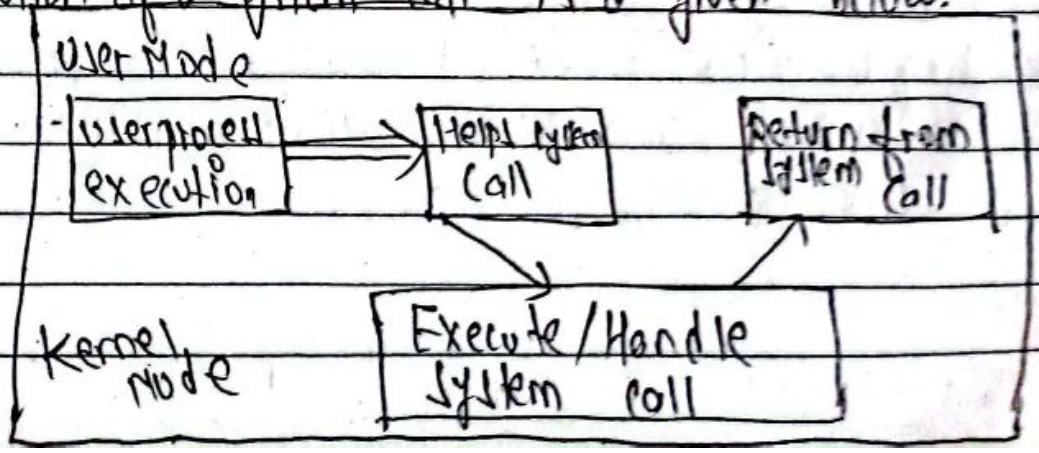


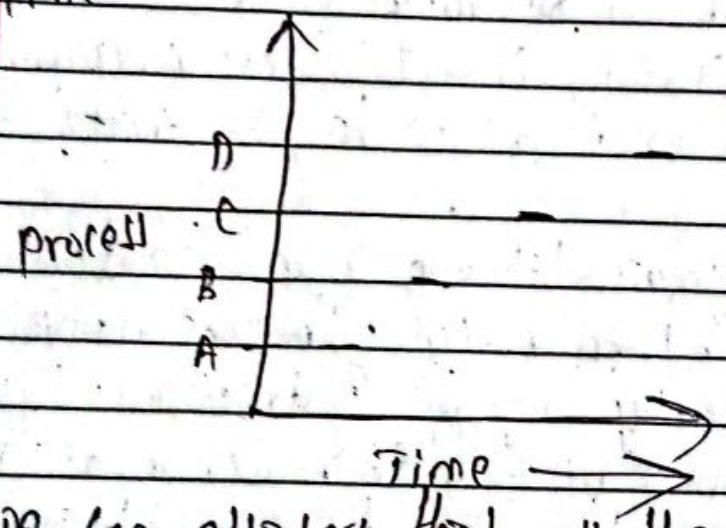
Fig: System call

Date:
Page:

(#) Process Model ⇒ Conceptually, every process has its own CPU but in reality, CPU switches back and forth from process to process but to understand the system, it is much easier to think about a collection of process running in parallel (pseudo) then try to keep the track of how CPU switches from program to program. This rapid switching back and forth is called multiprogramming. There are mainly three types of process model which are given below:

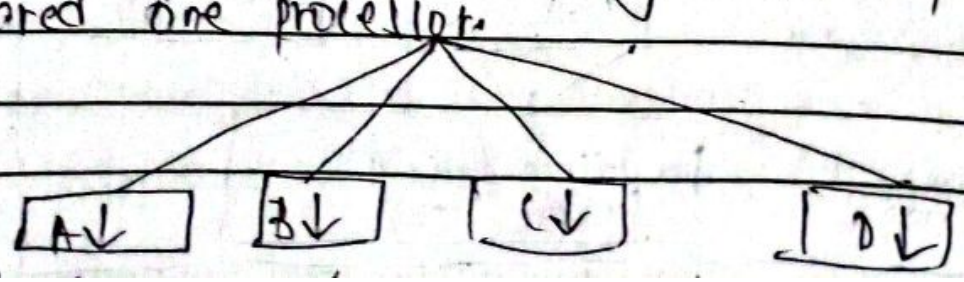
(a) Nonprogramming (one program at a time)

In this model, at any given instance only process run and other process run only after some interval of time.



we can analogy that, all the process progress but only one process actually run at given instance of time.

(b) Multiprogramming ⇒ In this model, there are more than one program in the memory and all programs share one processor.



All the programs, are executed by single CPU. In this program counter is initialized to execute the part of program. And then with the help of process switch it transfers control to program B and execute some part of it. After that program counter for C is active and execute some part of it and so on the program D. All these steps may be random or continuous with the help of process switch untill all the programs are not completed.

(c) Multiprocessing ⇒ In this model, more than one CPU controls and executes various processes loaded in the memory. For example, if there are four process, each with its own flow of control and each running independently then the CPU executes one process at a time and other (n) executes the remining process. It means that at least two process can run at a time by the help of two processor keeping track of all process, maintaining program counter and its respective PCB.

(*) Threads ⇒ A thread is a single sequence stream with in a process. A thread is also known as light weight process. A process is sometimes subdivided depending upon it's nature. Such subdivisions are known as thread. In a process, thread always allows multiple execution of streams. The CPU switches so rapidly back and forth among the threads giving an illusion that the threads are running in parallel. A thread has its own states like running, blocked, ready and terminated.

A thread consist of program counter, register set and stack space. Threads are not independent of one other like process. As a result, threads share with other threads their code section, data section, task, etc.

Threads are very special. In modern programming when ever a process has multiple task to perform independent of others. The major benefits of threading are as follows:

(a) Responsiveness
(b) Resource sharing
(c) Economy
(d) Scalability i.e. utilization of multiprocessor architecture.

Thread Model:

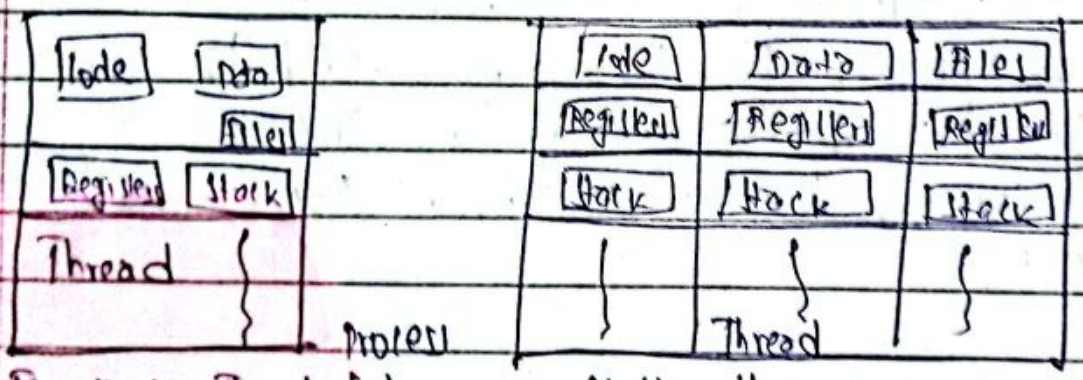


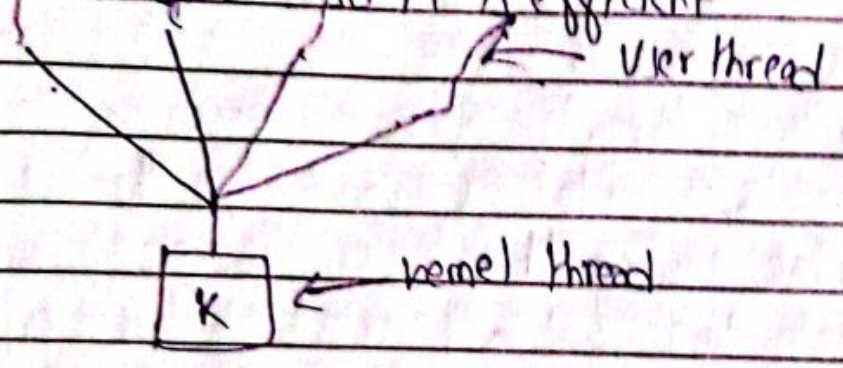
Fig: Single Threaded ^ Multi threaded process

Basically, there are two types of threads to be managed in modern system: User threads and kernel threads. User threads are supported above the kernel without kernel support. User threads are those threads that application programmers would put into their programs. Kernel threads are supported within the kernel of OS itself. All modern OS support kernel level thread allowing the kernel to perform multiple simultaneous task.

In a specific implementation the user threads must be mapped to kernel threads using one of the following strategies (models)

(a) Many - One Model
(b) One - ~~Many~~ One Model
(c) Many - Many Model.

(a) Many - One Model ⇒ This model maps many user level threads to one kernel level threads. Thread management is by thread library in user space. So it is efficient.



Many - One Model

(b) One - One Model ⇒ One user thread is mapped into one kernel thread. It provides more concurrency than previous model by allowing another thread to run during blocking. It also allows parallelism.

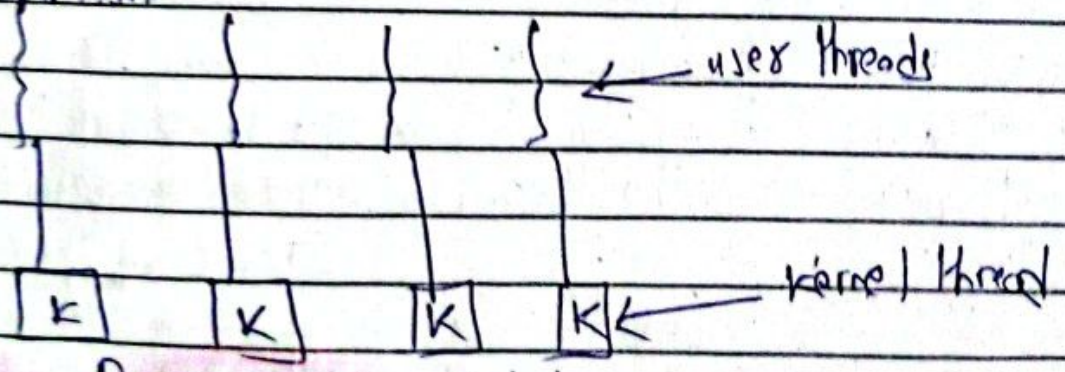


Fig: One - One Model

(c) Many - Many model ⇒ In this model, many user level threads multiplexed to the kernel thread of smaller or equal number

The number of kernel threads may be specific to either a particular application or particular machine. User's have no restriction on the number of threads created - blocking kernel system calls do not block the entire process.

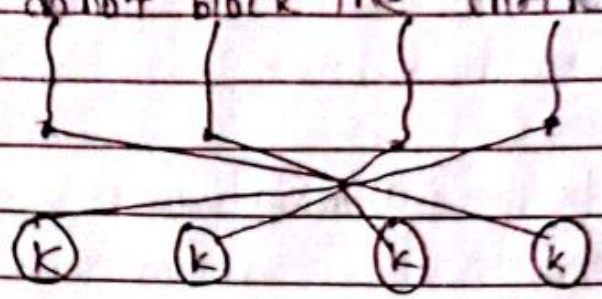


Fig: Many - Many Model

(*) Implementing threads in user space:

The first model is to put the threads package entirely in user space. During this time, the kernel knows nothing about them. The first and most obvious advantage is that a user level threads package can be implemented on an operating system that does not support threads.

The threads run on the top of runtime system which is a collection of procedure that manage threads. When threads are managed in user space, each process needs its own private thread table. This analogous to kernel's process table except that it keeps track only of per-thread properties. The thread table is managed by runtime system.

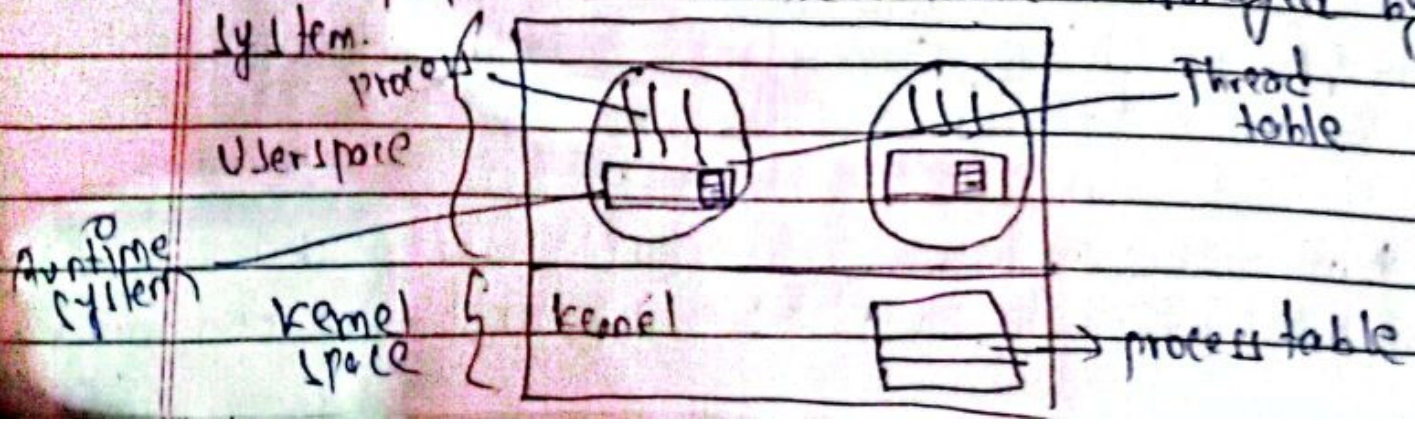


Fig: User level thread package:

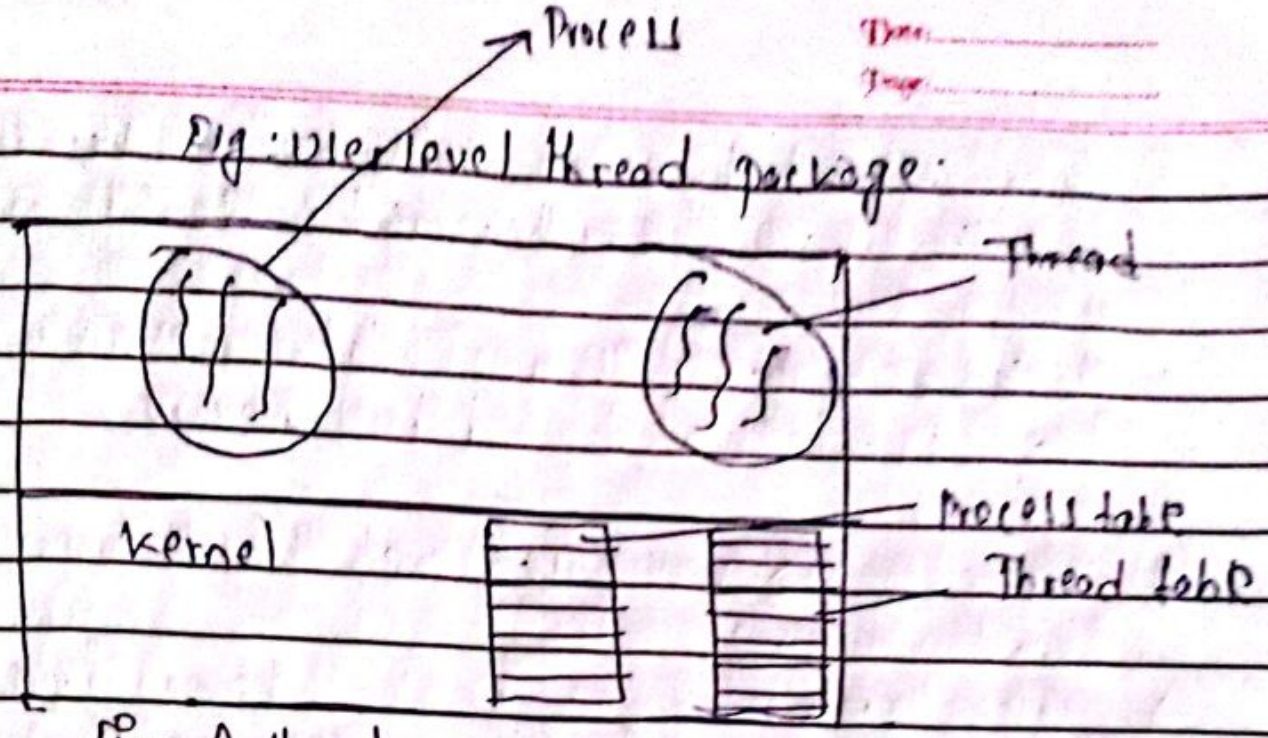


Fig: A thread package manage by kernel

Thread vs process

In many aspect, threads operate in the same way as the process. Some of the similarities and differences between thread & processes are given below.

Similarities.

- Like process, thread shares CPU and only one thread is active (running at a time)
- Like process, threads within a process execute sequentially
- Like process, thread can also create children.
- Like process, if one thread is blocked then other thread can run easily.

Differences.

- All the threads of a program are logically contained within a process.
- A process is a heavy weighted but a thread is light weighted

Date:
Page:

- Thread cannot have an individual existence as it is allied to a process. On the other hand a process can exist individually.
- A program (process) is an isolated execution unit whereas thread is not isolated and shares memory

(#) Differences between user level threads and kernel level thread.

User Level Thread	Kernel Level Thread
They are implemented by user.	They are implemented by OS
OS doesn't recognize user level threads.	Kernel threads are recognized by OS.
Implementation of user threads is easy.	Implementation of kernel threads is complicated.
Context switch requires no hardware support.	Hardware support is needed for concept switching.
If one user level thread perform blocking operation then entire process will be blocked.	If one kernel thread perfors blocking operation then another thread can continue execution
User level threads are designed as dense threads.	Kernel level threads are designed as independent threads
Example: Java thread, POSIX threads, etc.	Example: Windows, Solaris

Date:
Page:

(IPC)

Inter Process Communication ⇒ Inter Process communication is the mechanism that allows the exchange of data between the process. IPC allows one application to control another application there by enabling data sharing without interference. IPC enables data communication by allowing processes to use segments, semaphores (integer variable) and other methods to share memory & information. IPC facilitates efficient message transfer between the process.

Working together with multiple process requires inter process communication method which allows them to exchange data along with various information. There are two primary models of inter process communication and they are as follows :-

(i) Shared Memory
(ii) Message passing

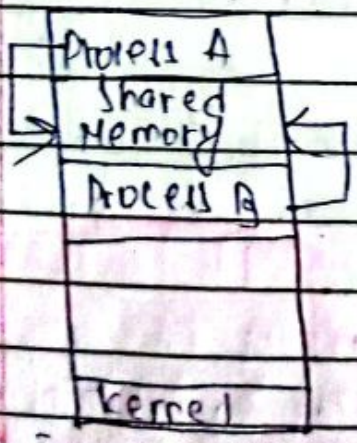


Fig: Shared Memory

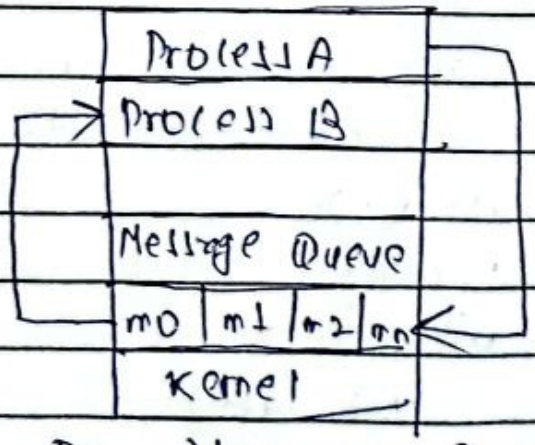


Fig. Message Passing

In the shared memory model, a region of memory, which is shared by cooperating process gets established. Processes can be then able to exchange information by

Date:
Page:

reading and writing of the data to the shared region.
In message passing form, communication takes place by the way of message exchange among the co-operating process by the help of kernel where message queue is allocated.

Race condition ⇒ A race condition is a situation that occurs when two or more process / thread can access shared data and they try to exchange it at the same time. Because the thread scheduling algorithm can swap between ~~them~~ threads will attempt to access the shared data. Therefore, the result of the change in data is dependent on thread scheduling algorithm i.e. both the threads are racing to access or change the data.

(Q) Write in short about kernel level threads and user level threads listing its advantages and disadvantages:-

⇒ Kernel level Threads:

Kernel level threads are the threads that are handled by the operating system directly and the thread management is done by kernel.

advantages:-

- Multiple threads of the same process can be scheduled on different processors in the kernel-level threads.
- The kernel routines can also be multithread.
- If a kernel-level thread is blocked, another thread of the same process can be scheduled by the kernel.

Date:
Page:

Advantages:

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

User Level Threads:

The user-level threads are the threads that are implemented by the users and the kernel is not aware of the existence of these threads.

Advantages:

- User level threads are easier and faster to create and manage than kernel-level threads.
- They can be run on any OS.
- There are no kernel mode privileges required for thread switching in user level threads.

Disadvantages:

- Multithreaded applications in user level threads cannot use multiprocessing to their advantage.
- The entire process is blocked, if one user-level thread performs blocking operation.

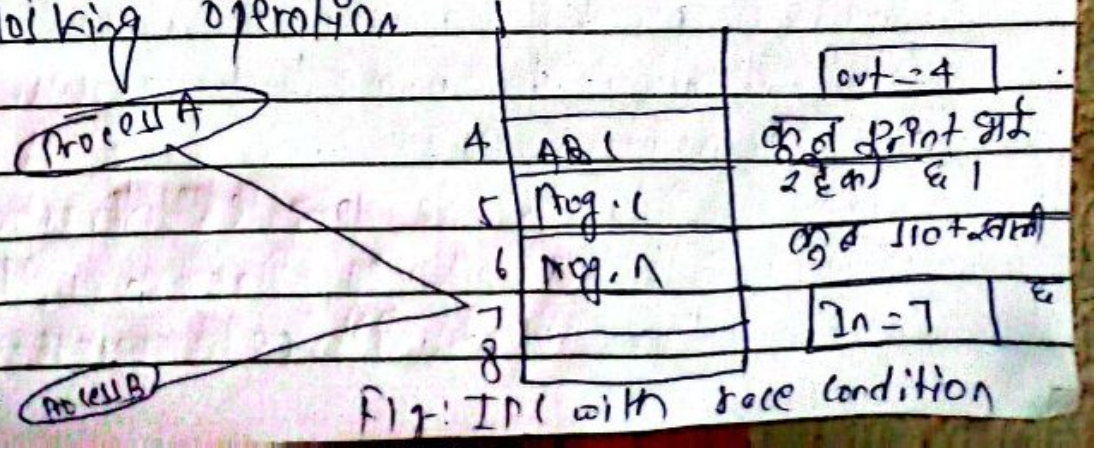


Fig: IPC with race condition

Date:
Page:

(A) Print spooler ⇒ When a process wants to print a file it enters the file name in a special spooler directory. Another process, the printer Daemon periodically checks to see if there are any files names are removed from directory. Imagine that our spooler directory has large number of slots numbered as 0, 1, 2, 3 ... each one capable of holding a file name. Also imagine that there are two shared variable namely out: points to next file to be printed and In: point to next ~~file~~ free slot in the directory.

Slots 0-3 files are already printed & slots 4 to 6 file name has to be printed. Now, the main issue comes at: process A reads in and store the value 7 in ~~some of~~ process a local variable called next free slot. Just then clock interrupt occurs and the CPU decides that process A has run long enough. So it switches to process B. ~~Th~~ run long enough. So it switches to process B. Process B also reads in and gets a 7. So, it stores the name of its in slot 7 and update in to 8. Then it goes off and does other thing. Eventually process A once again starts from the place left off last time. It looks for new free slot and finds a 7 there and writes it filename 7 slot 7 erasing the name that process B just put there.

Then it computes next free slot it which is 8 and sets in to 8. The spooler directory is now internally consistent so the printer Daemon will notice any thing wrong but process B will never receive any output

(#) Critical section ⇒ That part of program where shared memory is accessed is called critical region or critical section.

→ If we could arrange the matters in such a way that no two processes were ever in the critical region at the same time, we could avoid races betn them.

→ A critical section cannot be executed by more than one process at a time because OS faces the difficulties in allowing and disallowing the process from entering the critical section.

→ The critical section problem is used to design a set of protocols that can ensure that race condition among the process ~~that can~~ will never arise.

```
do
{:      [Entry section] → control the entry into critical section & gets lock on required resources.
        critical section → The critical part

        [Exit section] → Remove the lock from the resources and leave the others know that it's over.
        Remainder section ⇒ Rest of section
}
while (TRUE);
```

- In above portion, the entry section handles the entry into critical section.

- It requires the resources needed for execution by the process.
- The exit section handles the exit from critical section.
- It releases the resources and also informs the other processes that critical section is free.
- In order to synchronize, the co-operating processes over main task is to solve the critical section problem.
- The solution to critical section problem must satisfy the following conditions:-

VI

(a) Mutual exclusion → It implies that only process can be inside the critical section at any time. If any other process requires critical section, they must wait until it is free.

primary mechanism

(b) Progress → It means that if a process is not using critical region, then it should not stop any other process from accessing it. In other words, we can say that any process can enter critical if it is free.

Secondary Mechanism

(c) Bounded waiting → It means that each process must have limited waiting time. It should not wait endlessly to access critical section.

(d) Architectural Neutrality → It means that ~~each process~~ if our solution is working fine on one architecture then it should also on other as well. It means that mechanism must

be architecturally neutral.

Avoiding critical region → In order to avoid critical region problem, we need process synchronization process synchronization is the task of coordinating the execution of processes in such a way that no two processes can have access to the same shared data and resources. It is especially needed in multiprocessing 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. This may lead to inconsistency of shared data. To avoid inconsistency of data the process need to be synchronized with each other. Mutual exclusion and serializability is the best way of avoiding critical region which guarantees synchronization.

(#) Mutual Exclusion and Serializability:

→ Mutual exclusion is a concurrency control property which is introduced to prevent race condition. It is the requirement that a process cannot enter its critical section while another concurrent process is currently present or executing in its critical section i.e. only one process is allowed to execute the critical section at given instance of time.

Serializability is the classical concurrency scheme which ensures that the schedules of a process are serializable.

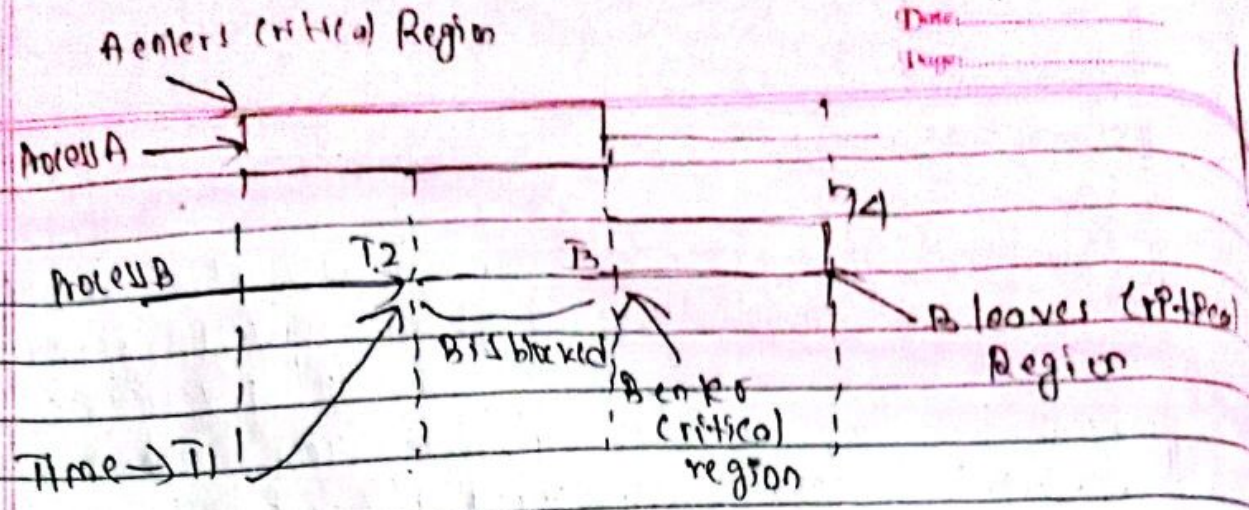


Fig: Mutual Exclusion

From above diagram, we can say that, at time T1, Process A enter its critical region. At time T2, Process B attempts to enter its critical region but fails (i.e. B gets blocked until T3 process B is temporarily blocked (suspended). At time T3, B enters its critical region and at T4, B leaves its critical region.

(*) Mutual Exclusion Conditions:

If we could arrange matters in such a way that no two processes were ever in their critical section simultaneously then we could avoid race conditions. Basically we need four conditions to hold to have a good solution for critical section problem and that solution is mutual exclusion.

- No two processes may be inside their critical sections at the same moment.
- No assumptions are made about relative speed of processes or number of CPU's.
- No process outside its critical section should block other processes.

- No process should wait arbitrary long to enter its critical section.

Topics and with their pseudocode:

(i) Critical section → It is the part of a program or a segment of a program where shared variables/resources can be accessed. Only one process can execute at a time in critical section. All the other process have to wait for execution if they want to enter inside critical region. The critical section problem can solved using Mutual exclusion and serializability.

```
do
{   [Entry section]
    critical section
    [Exit section]
    Remainder section
}
while(TRUE);
```

(ii) Disabling Interrupts ⇒ Disabling interrupts is a common way to implement mutual exclusion in critical sections. When a process or thread enters a critical section, it can disable interrupts to prevent other process or thread from accessing the shared resources. This ensures that the critical section is executed as a single, without any interrupts from other process or threads.

However it is important to note that disabling

for too long can cause system instability, delays in handling important events. So, it is important to keep the critical section as short as possible and to re-enable interrupts as soon as possible.

(iii) Lock variables → A lock variable is a special variable that is used to protect shared resources by allowing only one thread or process at a time to access the resource. A lock variable typically has two states: locked and unlocked. When a thread or process needs to access a shared resource, it first tries to acquire the lock by setting the lock variable to a locked state. If the lock is already held by another thread or process, the requesting thread or process is blocked until the lock is released. Once a thread or process has acquired the lock, it can safely access the shared resource without interference from other threads or processes. When the thread or process is done using the resource, it releases the lock by setting the lock variable to an unlocked state, allowing other threads or process to acquire the lock and access the resource.

Pseudocode:

```
entry section - while (lock==0)
                    lock = 1;
                // critical section
exit section - lock = 0;
```

Approaches for achieving Mutual Exclusion:

(1) Disabling Interrupt ⇒

→ Hardware type solution

→ Runs in kernel mode

→ Ensures mutual exclusion and progress.

→ Not frequently used as it is very risky to disable all interrupt.

→ Busy waiting solution

Entry section : Disable All Interrupt

CS

Exit section : Enable All Interrupt

(2) Lock variable:

→ Executes in user mode (software mechanism)

→ Multiprocess solution.

→ Busy waiting solution → No mutual exclusion guarantee

Pseudocode:

```
Entry section : while (lock != 0);
                lock = 1;
                // critical section
Exit section : lock = 0;
```

(3) Turn variable or strict alternation

→ runs in user mode (software mechanism)

→ busy waiting solution.

→ Implemented only for two processes.

→ Ensures mutual exclusion.

Pseudocode:

for process P0	for process P1
while(turn != 0)	while (turn != 1)
critical section	critical section
turn = 1;	turn = 0;

(4) Peterson's solution:

→ software based solution

→ may not work correctly on modern computer architecture.

→ requires two data items to be shared between the two processes.

• int turn	• boolean flag[2]
indicates whose turn is it to enter its critical region	used to indicate if a process is ready to enter its critical section.

Structure of process Pi	Structure of process Pj
do{	do{
flag[i] = true;	flag[j] = true;
turn = j;	turn = i;
while(flag[j] && turn == [j]);	while(flag[i] && turn == [i])
critical section	critical section;
flag[i] = false;	flag[j] = false;
remainder section	remainder section
}	}
while (TRUE);	while (TRUE);

(5) Test and set lock : (TSL)

→ Hardware type solution

Pseudocode:

Enter-region

TSL REGISTER, lock | copy lock to register and set lock to 1

CMP REGISTER, #0 | was lock 0?

JNE enter region | if it was non-zero, lock was set, so loop

RET | return to caller, critical region entered.

leave-region

MOVE LOCK #0 | store 0 in lock

RET | return to caller

(6) Sleep and wakeup → Both peterson's solution and the solution using TSL are correct, but both have the defect of busy waiting, so these defect of busy waiting is handeled by sleep and wakeup concept.

Sleep is the system call that causes the caller to block (suspended) until another process wakes it up. The wakeup call has one parameter, the process to be awakened.

Considered a computer with two process, H with higher priority and L with lower priority. The scheduling rules are such that H is run whenever it is in ready state. At a certain moment, with L in its critical region H becomes ready to run (Eg. an I/O operation completes) H now begins busy waiting, but since L is never scheduled while H is running L never gets the

chance to leave its critical region, so it lasts for. This situation is sometimes reffered to as the priority inversion problem.

(*) Semaphores:

→ Semaphores is an integer variable that solves the section problem and used to achieve process synchronization in the multiprocessing environment.

After initialization, it can be accessed only through two standard atomic operations, i.e. wait() and signal()

→ wait():

Pseudocode:

```
P (semaphore s)
{ while (s<=0)
    s--;
}
```

signal():

```
V (semaphore s)
{   s++;
}
```

wait: Decrements the value of its argument s, as long as it would become non-negative.

signal: Increments the value of its argument s, as there is no more process blocked on the queue.

Properties of semaphores:

- It is simple and always has a non (-) ve integer value
- Semaphores are machine independent.
- Semaphores are simple to implement
- They works with many processes.
- Each critical section has unique access semaphores.

Date.....
Page.....

Uses of Semaphores :-

- Used for achieving mutual exclusion.
- Used to solve the synchronization problems

Types of Semaphores:

Mainly, there are two types of semaphores and they are as follows :

(i) Binary Semaphores
(ii) Counting Semaphores.

(1) Binary Semaphore => The value of a binary semaphore can range only between 0 and 1. In some systems, binary semaphores are known as mutex locks, as they are locks, that provide mutual exclusion.

(2) Counting Semaphore : It's value can range over an unrestricted domain. It is used to control access to a resource that has multiple instances.

(1) Binary Semaphore

Pseudocode:

```
P (Semaphore S)
{ if (S==1)
  { S=0
  }
  else
  { Block the process and
    place it in suspend
    list,
    sleep()
  }
}
```

Pseudocode

```
V (Semaphore S)
{
  if(suspend list is empty)
  { S=1
  }
  else
  { select a process from suspend
    list and wakeup()
  }
}
```

(2) Counting semaphore:

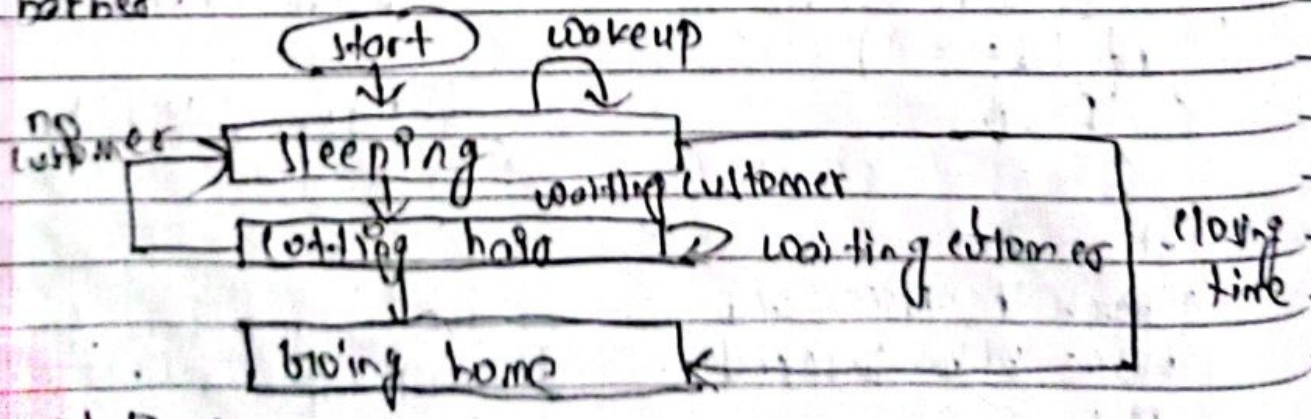
Pseudocode:

```
P(semaphore s)
{ s = s - 1
  if (s < 0)
{ put process in suspend list (sleep())
  }
else
return
  }
```

```
V (semaphore s)
s = s + 1
if (s ≤ 0)
{ select a process from suspend list (wakeup())
  }
}
```

(#) Sleeping Barber problem

1. If there are no customers, the barber goes to sleep
2. If customer enter the shop.
   - If all chairs are occupied, the customer leaves the shop, otherwise he sits in one of the free chairs.
   - If barber is asleep, the customer wakes up the barber.



Solution:

Variables : shared data
semaphore customer = 0;
semaphore barber = 0;
access seat mutex = 1;
int number of free seats = N;
int waiting = 0;

(#) Dining philosopher problem.
→ Pseudocode:

```
void philosopher (void)
{ while (true)
  { thinking ();
    take-fork (i);      // left fork
    take-fork (i+1 % N);  right fork
    EAT();
    put-fork (i);
    put-fork (i+1 % N);
  }
}
```

Process scheduling:
→ Process scheduling is the activity of the process manager that handles the removal of running process from the CPU and the selection of another process on the basis of particular strategies (algorithm). Process scheduling is very essential part of multi programming as because it helps to provide CPU time to all the process in the systematic way without hampering

the smooth functioning of the system.

Process Scheduling Queues.

⇒ The OS maintains all PCBs in process scheduling queues. It maintains a separate queue for each of the process state and PCB's of all process 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 which are given below:

- Job queue ⇒ This queue keeps all the process (programs) in the system.
- Ready queue ⇒ This queue keeps a set of all the process residing in the main memory, ready and waiting to execute. A new process is always put in this queue.
- Device Queue ⇒ A process which are blocked due to unavailability of an input-output device are kept in this queue.

The OS uses different policies (algorithm) to manage each queue. It may use algorithms like FIFO, SJF, Round Robin, Priority, etc. The OS scheduler determines how to move process between the ready and

queues which can only one entry per processor core of the system.

(*) Schedulers ⇒ Schedulers are the special system software which handles the process scheduling the jobs among many to be submitted to the system and it will decide which process to run. Basically, schedulers are of three types which are given below:

- Long-term schedulers
- Short-term schedulers
- Medium-term scheduler

- Long-term scheduler ⇒ Long-term scheduler determines which programs are admitted to the system for processing. It selects the process from the queue and loads them into memory for execution. It is also known as job scheduler. When a process changes the state from new to ready then there is a long term scheduler. The primary objective of job scheduler is to provide a balanced mix of jobs such as I/O bound and process bound and also controls the degree of multi-programming.

- Short-term scheduler ⇒ Short-term scheduler is a scheduler that changes the ready state of a process to a running state which is also known as CPU scheduler. It's main objective is to increase the

system performance according to chosen set of criteria. CPU scheduler selects a process among the processes that are ready to execute and allocate CPU time to one of them. Short term scheduler are also known as dispatcher because they make decisions regarding which process to execute next.

- Medium-term scheduler.

→ Medium-term scheduler is a scheduler which helps in removing the processes from the memory hence it reduces the degree of multi-programming. This scheduler is responsible for swapping. The medium term scheduler is in charge of handling the swap-out processes.

(*) Differences between scheduler and Dispatcher:

Scheduler	Dispatcher
• A scheduler is a special system software that handles process scheduling by selecting the process to execute	The dispatcher is the module that gives control of CPU to the process selected by the short term scheduler
• There are three types of schedulers known as: long term scheduler, short-term and medium term scheduler.	There is no categorization for dispatcher.
• Scheduler works on various	

algorithm such as FCFS, SJF, RR, etc.	Dispatcher has no specific algo. for its implementation
• Time taken by scheduler is usually negligible. Hence we neglect it.	Time taken by dispatcher is called dispatch latency.
• Scheduler works independently.	Working of dispatcher is dependent on scheduler.

(*) Scheduling Algorithm.

(A) Batch system scheduling → It is a technique in which OS collects programs and the data together in a batch (group) before the processing starts. The major process scheduling based on batch system scheduling are as follows:

(i) First come first serve (FCFS) → FCFS provides an efficient, simple and error free process scheduling algorithm that saves CPU resources. It uses non preemptive scheduling in which a process is automatically queued and processing occurs according to an incoming requests and process order.

(1) Considered the processes P1, P2, P3, P4 with their arrival time 0 ms and burst time given in the table. Find average waiting time and turnaround time using FCFS scheduling algorithm.

Processes	Arrival Time (AT)	Burst Time (BT)
P1	0 ms	21 ms
P2	0 ms	3 ms
P3	0 ms	6 ms
P4	0 ms	2 ms

Solution:

⇒ Representing the above table in Gantt chart.

P1	P2	P3	P4

0 21 24 30 32

Now,

Calculation of Average turn around time (TAT) & Average waiting time (AWT).

Process	Completion Time (CT)	Turnaround time (TAT)	Waiting time
P1	21	21-0 = 21	21-21 = 0
P2	24	24-0 = 24	24-3 = 21
P3	30	30-0 = 30	30-6 = 24
P4	32	32-0 = 32	32-2 = 30

Here,

$$\text{Average TAT} = \frac{21+24+30+32}{4} = 26.75 \text{ ms}$$

$$\text{Average WT} = \frac{(0+21+24+30)}{4} = 18.75 \text{ ms}$$

#

(X) Consider the processes P1, P2, P3, P4 with their arrival time and burst time given in table. Find average waiting time and turn around time using FCFS scheduling algorithm.

→ Solution

Processes	Arrival Time (AT)	Burst Time (BT)
P1	0 ms	21 ms
P2	2 ms	3 ms
P3	2 ms	6 ms
P4	3 ms	2 ms

Solution

→ Representing the above table in Gantt chart.

P1	P2	P3	P4

0 21 24 30 32

Now, calculation of Average turnaround time and Average waiting time.

Process	Completion time	Turnaround time	Waiting time
P1	21	21-0 = 21	21-21 = 0
P2	24	24-2 = 22	22-3 = 19
P3	30	30-2 = 28	28-6 = 22
P4	32	32-3 = 29	29-2 = 27

Here,

Average TAT = $\frac{21+22+28+29}{4}$ = 25 ms

Average WT = $\frac{0+19+22+27}{4}$ = 17 ms

(*) CPU - I/O Burst cycle ⇒ The success of CPU scheduling depends on the observe property of processes. Process execution consists of a cycle of CPU execution and I/O wait. Process

es alternative back-and-forth between these two states. The execution begins with the CPU burst followed by an I/O burst, then another CPU burst and so on. The last CPU burst will end with the system request to terminate execution rather than with another I/O burst. The duration of CPU burst are measured which help to appropriate CPU scheduling algorithm.

Load store add store Read from file	} CPU Burst
[wait for I/O]	} I/O Burst
Store increment index write for file [wait for I/O]	} CPU Burst I/O Burst
Load store add store Read from file	} CPU Burst
[wait for I/O]	} I/O Burst

Context switching ⇒ Context switch is the mechanism to store and restore the state or context of a CPU in process control block so that then the process can be resumed from the same point at a later time. In this technique a context switcher enables multiple processes to share a single CPU. So context switching is an essential features of multitasking OS. Hence, context switching is the process of switching the CPU from

one process to another process by storing and restoring the context or state or stack of current process in PCB before moving to next process.

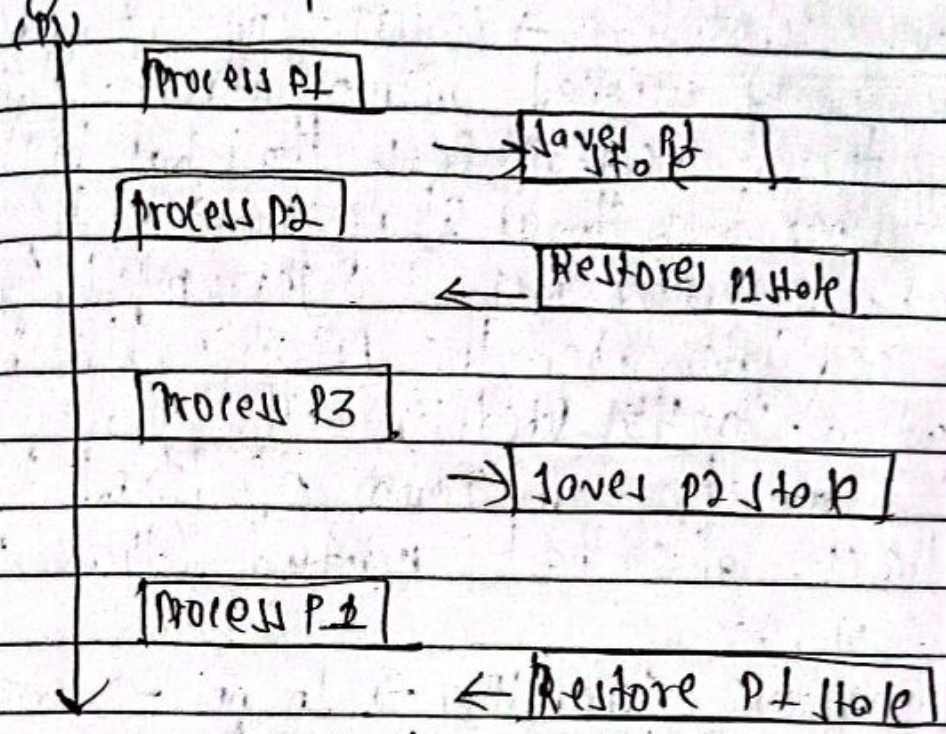


Fig: Context switching

Types of scheduling

There are two types of scheduling process execution in operating system and they are:

- Pre-emptive scheduling
- Non-preemptive scheduling.

CPU scheduling decision may take under the four following circumstances which are given below

(i) when a process switches from running state to the waiting state.

(ii) when a process switches from running state to ready state.

(iii) when a process switches from waiting state to ready state.

(iv) when a process terminates.

In the above conditions (i) and (iv), there is no choice in terms of scheduling. There is a choice of scheduling in conditions (ii) and (iii)

- Preemptive scheduling → In this type of scheduling, the tasks are usually assigned with priorities. At time of it is necessary to run a certain task that has higher priority before another task though it is running. Hence, in this scheduling, the running task is interrupted for sometime and resumed later when the priority task has finished its execution. In this scheduling, the process is picked and let it run for a maximum of fixed time and releases the CPU after that quantum (time slice) whether it finishes or not
- Non-preemptive scheduling → In non-preemptive scheduling once the CPU has been allowed to a process, the process keeps the CPU until it releases the CPU either by terminating or by switching to the waiting state. In this scheduling a process is picked and runs until it releases the CPU.

(*) Write the differences between preemptive and Non-preemptive scheduling.

→

Preemptive scheduling	Non-preemptive scheduling
• Processor can be preempted to execute different process in the middle of execution of any current process.	Once processor starts to execute a process it must finish it before executing the other. It cannot be paused in middle

• CPU utilization is more compared to non-preemptive scheduling.	CPU utilization is less compared to preemptive scheduling
• Waiting time and response time is less.	Waiting time and response time is more
• The preemptive scheduling is prioritized. The higher priority should always be the process that is currently utilized.	When a process enters the state of running the state of that process is not deleted from the scheduler until it finishes its service time.
• Preemptive scheduling is flexible.	Non-preemptive scheduling is rigid.
• Example: SRTF, priority, Roundrobin	Example: FCFS, SJF, etc.

(#) Scheduling Criteria:

→ Different CPU scheduling algorithms have different properties and may favor one class of processes over another. Many criteria have been suggested for comparing CPU scheduling algorithm and the criteria are given below:-

(a) CPU utilization

→ keep the CPU as busy as possible

(b) Throughput - Number of processes that completes its execution per time unit

(c) Turn-around Time - Amount of time to execute a particular process

(d) Waiting time → Amount of time a process has been waiting in the ready state.

- Response Time → Amount of time it takes from the a request was submitted until the first response is produced (not output). For time sharing environment

The goals of scheduling are as follows:-

- Fairness → Each process gets fair share of CPU
- Efficiency → when a CPU is 100% busy, then the efficiency is increased.
- Response Time → Minimizes the response time for the interactive users.
- Throughput → Maximizes the jobs for given time period
- Waiting Time → Minimizes total time spent waiting in the ready queue.
- Turnaround time → Minimizes the time between submission and termination

(X) Scheduling algorithm → Scheduling algorithm is mainly divided into following three categories.

(a) Batch system scheduling
(b) Interactive system scheduling.
(c) Real-time scheduling.

Some major terminologies used in various scheduling algorithms are as follows:-

- Arrival Time → Time at which the process arrives in ready queue.
- Completion time → Time at which process completes its execution

Date:
Page:

- Burst time (CPU time)

⇒ Time required by a process for CPU execution

- Turnaround time: It is the time difference between completion time and arrival time is Turnaround time.

Turnaround Time (TAT) = Completion time - Arrival time

- Waiting time (WT) ⇒ Time differences between turnaround time and burst time is waiting time

Waiting Time (WT) = Turnaround time (TAT) - Burst Time (BT)

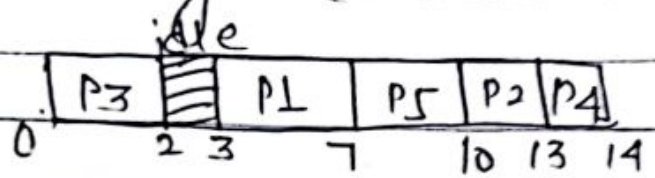
CS CamScanner

(*) Example : (1)

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

Solution

⇒ Representing the above table in Gantt chart



Now,

Calculating the Average turnaround time (ATAT) and Average waiting time.

Now,

Process Id	Completion time	Turn around time	waiting time
P1	7	7 − 3 = 4	4 − 4 = 1
P2	13	13 − 5 = 8	8 − 3 =
P3	2	2 − 0 = 2	2 − 2 =
P4	14	14 − 5 = 9	9 − 1 =

P5	10	10 − 4 = 6	6 − 3 = 3

Again,

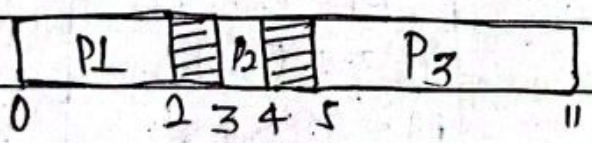
$\therefore$ Average TAT $= \frac{4+8+2+9+6}{5} = 5.8$ ms

$\therefore$ Average WT $= \frac{0+5+0+8+3}{5} = \frac{16}{5} = 3.2$ ms

(#) Example : 2

Process Id	Arrival time	Burst time
P1	0	2
P2	3	1
P3	5	6

⇒ Representing the above table in Gantt chart



Now,

Process Id	Completion time	Turn Around Time	Waiting Time
P1	2	2 − 0 = 2	2 − 2 = 0
P2	4	4 − 3 = 1	1 − 1 = 0
P3	11	11 − 5 = 6	6 − 6 = 0

Again,

$\therefore$ Average TAT $= \frac{2+1+6}{3} = \frac{9}{3} = 3$

$\therefore$ Average WT $= \frac{0+0+0}{3} = 0$ #

(##) Example : 3

Given :

Process	Burst time	Arrival time
P1	6	2
P2	2	5
P3	8	1
P4	3	0
P5	4	4

→ Representing the above table in Gantt chart

P4	P3	P1	P5	P2

0 3 11 17 21 23

Process Id	Completion time	TAT	waiting time
P1	17	17 − 2 = 15	15 − 6 = 9
P2	23	23 − 5 = 18	18 − 2 = 16
P3	11	11 − 1 = 10	10 − 8 = 2
P4	3	3 − 0 = 3	3 − 3 = 0
P5	21	21 − 4 = 17	17 − 4 = 13

Average Turnaround time $= \frac{15+18+10+3+17}{5}$

$= 12.6$

Average waiting time $= \frac{9+16+2+0+13}{5}$

$= 8$

(ii) Shortest Job first (SJF) ⇒ In FCFS, we saw that if a process is having very high burst time and it comes first then the other process with very low burst time

have to wait for its turn. So, to remove this problem, we come with a new approach called shortest job first (SJF)

In SJF technique, the process having the minimum burst at a particular instance of the time will be executed first. If the subsequent (n) burst of two processes are same, then FCFS scheduling is used to break this tie. It is non preemptive approach has the advantage of having minimum average waiting time all scheduling algorithm.

(#) Example: Consider four processes P1, P2, P3 and P4 with that time 0 for all the processes and burst time given in the burst SJF scheduling algorithm to determine its average waiting average turnaround time. (The mode is non-preemptive)

⇒

Process	Arrival Time (AT)	Burst time (BT)
P1	0 ms	21
P2	0 ms	3
P3	0 ms	6
P4	0 ms	2

solution

Here,

Representing the above table in Gantt chart.

P4	P2	P3	P1

0 2 5 11 32

Again

Process Id	Completion Time	TAT	Waiting Time
P1	32	32-0=32	32-21 = 11
P2	5	5	5-3 = 2
P3	11	11	11-6 = 5

Date:
Page:

P4 | 2 | 2 | 2-2 = 0

∴ Average Turnaround time = $\frac{32+5+11+2}{4}$ = 12.5

∴ Average waiting time = $\frac{11+2+5+0}{4}$ = 4.5

(X) Shortest Job first Numericals (on time ↓

(#) Considered 5 process P1, P2, P3, P4 and P5 with their corresponding arrival time and CPU time given in the table. Find Average WT and Average TAT using SJF scheduling algorithm:

Process	Arrival time (AT)	Burst Time (BT)
P1	0 ms	10 ms
P2	2 ms	7 ms
P3	1 ms	5 ms
P4	4 ms	2 ms
P5	6 ms	8 ms

Solution:

→ Representing above table in Gantt chart

P1	P4	P3	P2	P5

0 10 12 17 24 32

Now, calculating the Average waiting time and Average turn around time:

So,

Process	Completion time	Turnaround time	Waiting time
P1	10	10 − 0 = 10	10 − 10 = 0
P2	24	24 − 2 = 22	22 − 7 = 15
P3	17	17 − 1 = 16	16 − 5 = 11
P4	12	12 − 4 = 8	8 − 2 = 6
P5	32	32 − 6 = 26	26 − 8 = 18

∴ Average TurnAround time $= \frac{10+22+16+8+26}{5}$

$= 16.4$ ms

Average waiting time $= \frac{0+15+11+6+18}{5}$

$= 10$ ms #

(#) Considered 5 process P1, P2, P3, P4 and P5 with their corresponding arrival time and CPU time given in the table:

Processes	Arrival time (AT)	Burst time (BT)
P1	3 ms	4 ms
P2	5 ms	3 ms
P3	0 ms	2 ms
P4	5 ms	1 ms
P5	4 ms	3 ms

⇒ Solution,

Representing above table in Gantt chart

P3		P1	P4	P5	P2

0 2 3 7 8 11 14

Processes	Completion time	Turnaround time	Waiting time
P1	7	7 − 3 = 4	4 − 4 = 0
P2	14	14 − 5 = 9	9 − 3 = 6
P3	2	2 − 0 = 2	2 − 2 = 0
P4	8	8 − 5 = 3	3 − 1 = 2
P5	11	11 − 4 = 7	7 − 3 = 4

$\therefore$ Average Turnaround time $= \frac{4+9+2+3+7}{5}$

$= \frac{25}{5} = 5.0$

$\therefore$ Average waiting time $= \frac{0+6+0+2+4}{5}$

$= 2.4$ #

(*) Convoy Effect ⇒ Convoy effect is the condition that arises in FCFS scheduling algorithm when one process holds the CPU for a longer period of time and each process can get the CPU only when the process holding the CPU finishes its execution. Due to this effect resources utilization is ~~processed and also poor~~ and also effects the performance of operating system.

(#) Shortest Remaining Time Next (SRTN)

(1) Considered the four process P1, P2, P3 and P4 with the corresponding burst time and arrival time given in the table. Use SRTN scheduling algorithm i.e. preemptive version of SJF to find average waiting time and

average turn around time.

Process	Arrival time (in ms)	Burst time (in ms)
P1	0	21
P2	1	3
P3	2	6
P4	3	2

Solution:

Representing the above table in Gantt chart

P1	P2	P4	P3	P1

0 1 4 6 12 32

Now, calculating the average W.T and average Turnaround time.

Process	Completion time	Turnaround time	Waiting time
P1	32	32 − 0 = 32	32 − 21 = 11
P2	4	4 − 1 = 3	3 − 3 = 0
P3	12	12 − 2 = 10	10 − 6 = 4
P4	6	6 − 3 = 3	3 − 2 = 1

$\therefore$ Average TAT $= \frac{32+3+10+3}{4}$

$= \frac{48}{4} = 12$ ms

Average waiting Time $\frac{11+0+4+1}{4}$

$= \frac{16}{4} = 4$ ms

Date:
Page:

(#1) Considered the set of 4 process whose arrival time and burst time are given in table below. If the CPU scheduling policy is SRTF then calculate Average WT and Average Turnaround time.

→

Process Id	Arrival Time (in ms)	Burst time (in ms)
P0	5	8
* P1	0	5
P2	4	9
* P3	1	2

→ Solution

Representing the above table in Gantt chart

P1	P3	P3	P1	P2	P1	P1	P0	P2

0 1 2 3 4 5 6 7 15 24

Now,

Process Id	Completion time	Turnaround time	Waiting time
P0	15	15-5 = 10	10-8 = 2
P1	7	7-0 = 7	7-5 = 2
P2	24	24-4 = 20	20-9 = 11
P3	3	3-1 = 2	2-2 = 0

$\therefore$ Average TAT $= \frac{10+7+20+2}{4} = 9.75$

Average WT $= \frac{2+2+11+0}{4} = 3.75$ #

Date:
Page:

Shortest Remaining Time Next (SRTN) → SRTN is a preemptive version of SJF scheduling. In this scheduling algorithm, the process with a smallest amount of time remaining until completion is selected to execute. Since the currently executing process is 1 with the shortest amount of time remaining by definition and since that time should only reduce as execution progress, process will always run until they complete or a new process is added that requires a smaller amount of time.

Q) Consider the set of 5 processes whose arrival time and burst time are given below-

→

Process Id	Arrival time	Burst time
P1	3	1
P2	1	4
P3	4	2
P4	0	6
P5	2	3

If the CPU scheduling policy is SJF non preemptive, calculate the average waiting time and average turnaround time.

→ Solution:

Representing the above table in Gantt chart.

P4	P2	P2	P1	P2	P2	P3	P5	P4

0 1 2 3 4 5 6 8 11 16

Process Id	Completion time	TAT	Waiting time
P1	1	1 − 3 = 1	1 − 1 = 0
P2	6	6 − 1 = 5	5 − 4 = 1
P3	8	8 − 4 = 4	4 − 2 = 2
P4	16	16 − 0 = 16	16 − 6 = 10
P5	11	11 − 2 = 9	9 − 3 = 6

Average TAT = $\frac{1+5+4+16+9}{5} = 7$ #

Average WT = $\frac{0+1+2+10+6}{5} = \frac{3+10+6}{5} = \frac{19}{5} = 3.8$ #

(Q)

Process ID	Burst Time	Arrival Time
P0	8	5
P1	5	0
P2	9	4
P3	2	1

⇒ Solution:

Representing the above table in gantt chart

P1	P3	P1	P4	P1	P0	P2

0 1 3 4 5 7 15 24

Now,

Calculating the average WT and average Turn around time.

Process Id	Completion time	Turnaround time	Waiting time
P0	15	15 − 5 = 10	10 − 8 = 2
P1	7	7 − 0 = 7	7 − 5 = 2
P2	24	24 − 4 = 20	20 − 9 = 11
P3	3	3 − 1 = 2	2 − 2 = 0

Average Turnaround time = 9.75 #

Average Waiting Time = $\frac{2+2+11+0}{4}$ = 3.75 #

(Q) Processes

Processes	AT	BT
P1	0	7
P2	1	5
P3	2	3
P4	3	1
P5	4	2
P6	5	1

=) Solution,

Representing the above table in Gantt chart

P1	P2	P3	P4	P3	P3	P6	P5	P2	P1

0 1 2 3 4 5 6 7 9 13 19

Processes	Completion time	Turn Around time	Waiting time
P1	19	19 − 0 = 19	19 − 7 = 12
P2	13	13 − 1 = 12	12 − 5 = 7
P3	6	6 − 2 = 4	4 − 3 = 1
P4	4	4 − 3 = 1	1 − 1 = 0
P5	9	9 − 4 = 5	5 − 2 = 3
P6	7	7 − 5 = 2	2 − 1 = 1

Date:
Page:

Now calculating the Average WT and Average TAT

$$\text{Average WT} = \frac{12+7+1+0+3+1}{6}$$

$$= 4 \text{ ms.}$$

$$\text{Average TAT} = \frac{19+12+4+1+5+2}{6} = 7.16 \text{ ms}$$

Date:
Page:

Round Interactive system scheduling

(a) Robin scheduling ⇒ In this algorithm, the process is allocated the CPU for the specific time period called time slice or time quantum. If the process completes its execution within this time slice, then it is removed from the queue, otherwise it has to wait for the other time slice and preempted process is placed at the back of the ready list. In this approach of CPU scheduling we have a fixed time quantum and CPU will be allocated to a process for that time of amount only.

Example (1)

(#) Considered the set of 5 processes whose arrival time, and burst time is given below in the table. Find Average waiting time and average TAT. using Round Robin scheduling considering time quantum = 2 ms.

Process ID	Arrival Time (in ms)	Burst Time (in ms)
P1	0	~~5~~ 3
P2	1	~~3~~ 1
P3	2	1
P4	3	2
P5	4	3

⇒ Solution:

Representing the above table in Gantt chart.

Ready queue : P1, P2, P3, P1, P4, P5, P2, P1, P5

P1	P2	P3	P1	P4	P5	P2	P1	P5

0 2 4 5 7 9 11 12 13 14

Date:
Page:

Now, calculating the average waiting time and average turnaround time.

Process ID	Completion time	TurnAround time	Waiting time
P1	13	13 − 0 = 13	13 − 5 = 8
P2	12	12 − 1 = 11	11 − 3 = 8
P3	5	5 − 2 = 3	3 − 1 = 2
P4	9	9 − 3 = 6	6 − 2 = 4
P5	14	14 − 4 = 10	10 − 3 = 7

$$\text{Average TurnAround time} = \frac{13+11+3+6+10}{5}$$

$$= 8.6$$

$$\text{Average waiting time} = \frac{8+8+2+4+7}{5}$$

$$= 5.8$$

(1) Time Slice = 3ms

Processes	Arrival time	Burst Time
P1	0	5
P2	1	6
P3	2	3
P4	3	1
P5	4	5
P6	5	4

=> Solution

Representing the above table in Gantt chart

Ready queue: ~~P1, P2, P3, P4, P5, P1~~

P1, P2, P3, P1, P4, P5, P2, P6, P3, P1, P5, P2, P6, P5

(#) TQ = 2

Process Id	AT (Arrival Time)	Burst Time (BT)
P1	0	5
P2	1	7
P3	2	3
P4	3	4

Representing the given table in Gantt chart

⇒

P1	P2	P3	P1	P4	P2	P3	P1	P4	P2	P2

0 2 4 6 8 10 12 13 14 16 18 19

Ready queue : P1, P2, P3, P1, P4, P2, P3, P1, P4, P2

Process	CT	TAT	WT
P1	14	14-0=14	14-5 = 9
P2	19	19-1 =18	18-7 = 11
P3	13	13-2=11	11-3 = 8
P4	16	16-3 = 13	13-4 = 9

Average WT = $\frac{9+11+8+9}{4}$ = 9.25

Average TAT = $\frac{14+18+11+13}{4}$ = 14 #

(2)

Process Id	AT	BT
P1	5	5
P2	4	6
P3	3	7
P4	1	9
P5	2	2
P6	6	3

Represent the given table in Gantt chart

Ready Queue : ~~P4~~, ~~P5~~, ~~P3~~, ~~P2~~, ~~P4~~, ~~P1~~, ~~P6~~, ~~P3~~, ~~P5~~, P4 ~~P1~~, P3

	P4	P5	P3	P2	P4	P1	P6	P3	P2	P4	P1	P3

0 1 4 6 9 12 15 18 21 24 27 30 32 33

Process	Completion Time	Turn Around Time	Waiting time
P1	32	32 − 5 = 27	27 − 5 = 22
P2	27	27 − 4 = 23	23 − 6 = 17
P3	33	33 − 3 = 30	30 − 7 = 23
P4	30	30 − 1 = 29	29 − 9 = 20
P5	6	6 − 2 = 4	4 − 2 = 2
P6	21	21 − 6 = 15	15 − 3 = 12

$$\text{Average TAT} = \frac{27+23+30+29+4+15}{6}$$

$$= 21.33$$

$$\text{Average WT} = \frac{22+17+23+20+2+12}{6}$$

$$= 16$$

Q no (1)

Time UPCC = 3ms

Processes	Arrival time	Burst Time
P1	0	5
P2	1	6
P3	2	3

P5	3	1
P6	4	5
P7	5	4

→ Represent the given table in Gantt chart:

Ready Queue = P1, P2, P3, P5, P1, P6, P7, P2, P6, P1

P1	P2	P3	P5	P1	P6	P7	P2	P6	P1

0 3 6 9 10 12 15 18 21 23 24

Processes	Completion Time	Turn Around Time	Waiting Time
P1	12	12-0=12	12-2=10
P2	21	21-1=20	20-3=17
P3	9	9-2=7	7-3=4
P5	10	10-3=7	7-1=6
P6	23	23-4=19	19-5=14
P7	24	24-5=19	19-4=15

∴ Average TAT = $\frac{12+20+7+7+19+19}{6} = 14$

∴ Average WT = $\frac{10+17+4+6+14+15}{6} = 11.4$

(*) Priority Scheduling ⇒ In this scheduling algorithm, the priority is assigned to all the processes and the processes with highest priority executes first. Priority assignment is done on the basis of internal factor such as CPU and memory requirements or sometimes external factor such as user's choice. We can consider lower the number and higher the number higher the priority. This algorithm is non preemptive approach.

(*) Example: Considered four processes P1, P2, P3 and P4 with their corresponding arrival time, burst time and priority order given in the table. Find avg. WT and avg. TAT using priority scheduling.

⇒

Processes	Arrival time	Burst time	Priority
P1	0	5	1
P2	1	3	2
P3	2	8	1
P4	3	6	3

Note: Lower the number, higher the priority

⇒ Representing the above table in Gantt chart

P1	P3	P2	P4

0 5 13 16 22

Processes	Completion time	Turn Around time	Waiting time
P1	5	5 − 0 = 5	5 − 5 = 0
P2	16	16 − 1 = 15	15 − 3 = 12
P3	13	13 − 2 = 11	11 − 8 = 3

P4	22	22-3 = 19	19-6 = 13

Average Waiting Time = $\frac{0+12+3+13}{4}$

= 7 ms

Average Turnaround time = $\frac{5+15+11+19}{4}$ = 12.5 ms

(1)(*) Considered the set of 5 processes whose arrival time and burst time are given below. If the CPU scheduling policy is priority scheduling, calculate average waiting time and average TAT. (lower number represents higher priority)

(i)

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

(ii)

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

(Note: higher ~~priority~~ represents number higher priority)

Date:
Page:

~~Average TAT = 4+12+7+18+26+14+36 / 7 = 16.71~~ wrong

~~Average WT = 0+6+2+11+16+9+25 / 7 = 10.42~~

Q) Represent the above table in Gantt chart

P1	P4	P5	P3	P2

0 4 9 11 12 15

Process Id	Completion time	Turnaround time	Waiting time
P1	4	4-0=4	4-4=0
P2	15	15-1=14	14-3=11
P3	12	12-2=10	10-1=9
P4	9	9-3=6	6-5=1
P5	11	11-4=7	7-2=5

Average TAT = 8.2

Average WT = 5.2

Q) Represen the above table in Gantt chart

P1	P2	P3	P4	P5

0 4 7 8 13 15

Id	Completion time	TAT	WT
1	4	4-0=4	4-4=0
2	7	7-1=6	6-3=3
3	8	8-2=6	6-1=5

CamScanner

4	13	13-3 = 10	10-5 = 5
5	15	15-4 = 11	11-2 = 9

Averog WT = 4.4 Average TAT = 7.4

(*) Considered the set of 7 processes whose arrive time and burst time are given below. If the scheduling policy is priority non preemptive then calculate Average WT and Average TAT.

Sol:

Process Id	Arrival time	Burst time	Priority
P1	0	4	1
P2	3	6	5
P3	2	5	2
P4	5	3	4
P5	7	10	6
P6	6	5	3
P7	8	11	9

→ Represent the above table in Gantt chart

P1	P3	P6	P4	P2	P5	P7

0 4 9 14 17 23 33

Process Id	Completion time	Turnaround Time	Waiting Time
P1	4	4-0 = 4	4-4 = 0
P2	23	23-3 = 20	20-6 = 14
P3	9	9-2 = 7	7-5 = 2
P4	17	17-5 = 12	12-3 = 9
P5	33	33-7 = 26	26-10 = 16
P6	14	14-6 = 8	8-5 = 3

P7 | 44 | 44-8=36 | 36-11=25

Now,

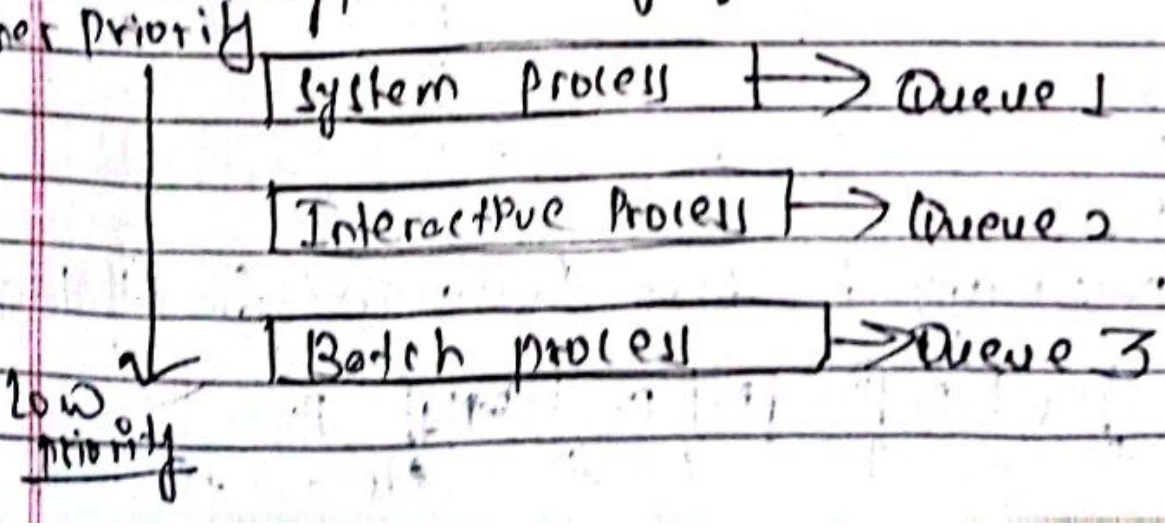
Average TAT = $\frac{4+20+7+12+26+8+36}{7}$ = 16.14

Average WT = $\frac{0+14+2+9+16+3+25}{7}$ = 9.85

(X) Multiple Queue (Multi-level Queue) Scheduling

⇒ It may happen that processes in the ready queue can be divided into different classes where each class has its own scheduling needs. For Example: a common division is foreground (interactive) process and background (batch process). For this kind of situation multi-level scheduling is used. It's working mechanism is describe below:-

Ready queue is divided into separate queues for each class of process. For example, let us take 3 diff types of process namely; System process, Interactive process and Batch process. All three processes have their own queue. The below given figure illustrates the type level of queue.



Date:
Page:

All three diff^n type of process have their own queue. Each queue has its own scheduling algorithm. For Example Queue 1 and Queue 2 uses Round Robin while Queue 3 can use FCFS for scheduling the process.

* Scheduling among the queues.

→ What will happen if all the queues have some process? Which process should get the CPU?

⇒ To determine this, scheduling among queue is very necessary. There are two ways to do so:

(i) Fixed priority Preempt scheduling method.

(ii) Time slicing

(#) Considered four processes under multilevel queue scheduler as given below in the table. Queue number denotes the queue of the process. Priority of queue 1 is greater than queue 2 where queue 1 uses Round Robin with time slice = 2ms and queue 2 uses FCFS. Find average TAT and average WT.

Processes	Arrival time	CPU Time	Queue Number
P1	0	4	1
P2	0	3	1
P3	0	8	2
P4	10	5	1

→ Solution:

Representing the above table in Gantt chart

P1	P2	P1	P2	P3	P4	P3

0 2 4 6 7 10 15 20

Process Id	Completion time	TAT	Waiting time
P1	6	6 − 0 = 6	6 − 4 = 2
P2	7	7 − 0 = 7	7 − 3 = 4
P3	20	20 − 0 = 20	20 − 8 = 12
P4	15	15 − 10 = 5	5 − 5 = 0
Average	TAT = $\frac{6+7+20+5}{4}$ =		9.5 ms #
Average	WT = $\frac{2+4+12+0}{4}$ =		4.5 ms #

(c) Real time system scheduling:

→ A real time system scheduling is composed of scheduling clock and the processing hardware elements. In such system a task has schedulability task are accepted by real time and completed as specified by the task deadline depending characteristics of scheduling algorithm.

One of the common real-time system scheduling is Highest Response Ratio Next (HRRN) scheduling.

- Highest Response Ratio Next (HRRN) scheduling:

→ Highest Response Ratio Next scheduling is non-preemptive scheduling algorithm which is one of the optimal algorithms scheduling. It is the modification of SJF in order to reduce problem of starvation. In this scheduling algorithm, CPU allocate the next process that has highest response ratio and no process having less burst time.

Date:
Page:

(*) Highest Response Ratio Next (HRRN) scheduling

→ Formula to calculate Response Ratio (RR) is given below:

$$\text{Response Ratio (RR)} = \frac{w+s}{s}$$

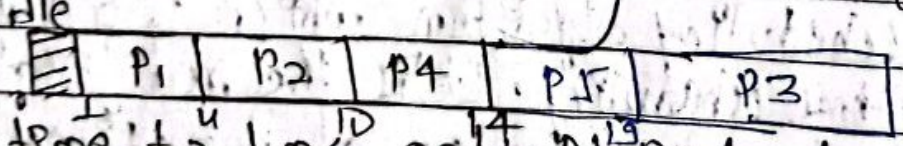
where, w = it indicates waiting time currently
s = it indicates service time i.e. burst time or CPU time.

(#) Example:

→ Considered the following table given below and find average TAT and average WT by using Highest Response Ratio Next (HRRN) scheduling algorithm

Process	Arrival Time (ms)	Burst Time
P1	1	3
P2	3	6
P3	5	8
P4	7	4
P5	8	5

Represent the above table in Gantt chart



At time t = 1 ms only P1 is kept in the running queue and after that at time t = 4 ms only P2 is available and gets scheduled and kept in the running queue.

At time t = 10 ms, the processes P3, P4 and P5

Date:
Page:

are available in the ready queue. So, in order to schedule these processes keep in the running queue, we must first find the response ratio of those processes which is given below

$$RR \text{ of } P_3 = \frac{[(10-5)+8]}{8} = \frac{13}{8} = 1.62 \text{ ms}$$

$$RR \text{ of } P_4 = \frac{[(10-7)+4]}{4} = \frac{7}{4} = 1.75 \text{ ms}$$

$$RR \text{ of } P_5 = \frac{[(10-8)+5]}{5} = \frac{7}{5} = 1.4 \text{ ms}$$

From above results, it is clear that P4 has highest response ratio. So P4 is kept in the running queue.

Now,

At time t = 14 ms, two processes P3 and P5 are in the ready queue. So, finding their response ratio at this time, we get

$$RR \text{ of } P_3 = \frac{[(14-5)+8]}{8} = \frac{17}{8} = 2.125$$

$$RR \text{ of } P_5 = \frac{[(14-8)+5}{5} = \frac{11}{5} = 2.2 \text{ ms}$$

From the above result, it is clear that P5 has the highest response ratio. So, P5 is kept in the running queue.

After that at time t = 19 ms. only one process remains so remaining process P3 is directly kept in the running queue.

Process Id	Completion	Waiting Time	TAT
P1	4	3 − 3 = 0	4 − 0 = 4
P2	10	7 − 6 = 1	10 − 3 = 7
P3	27	22 − 8 = 14	27 − 5 = 22
P4	14	7 − 4 = 3	14 − 7 = 7
P5	19	11 − 5 = 6	19 − 8 = 11

Now,

Average TAT = $\frac{3+7+22+7+11}{5}$ = 10 ms

Average WT = 4.8 ms