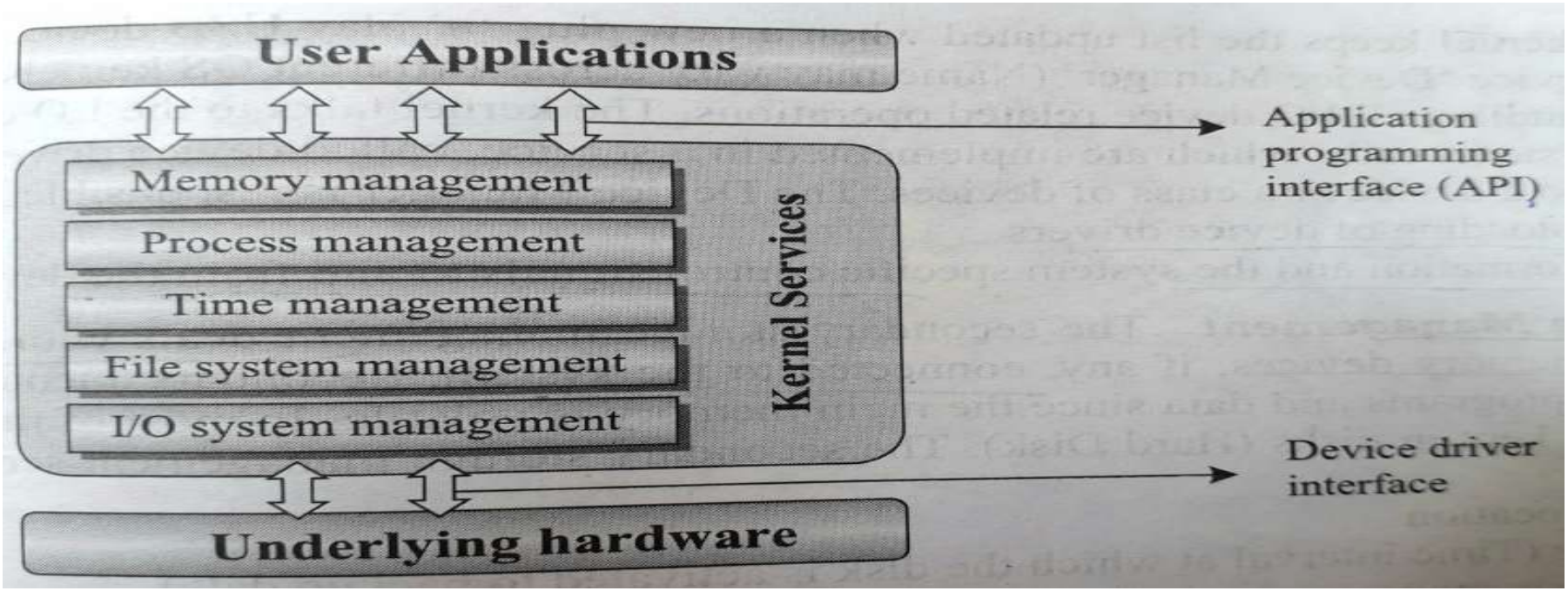
MODULE 5

RTOS AND IDE FOR EMBEDDED SYSTEM DESIGN

Operating System Basics

- **What is an Operating System?**
- The Operating System acts as a bridge between **the user applications/tasks and the underlying system resources** through a set of system functionalities and services. The OS manages the system resources and make the system available to the user on a need basis. The primary functions of an Operating system is
  - Make the system convenient to use
  - Organize and manage the system resources efficiently and correctly



- **What is a Kernel?**
  - The **kernel** is core of the OS.
  - Responsible for managing the system resources.
  - And the communication among the hardware & the system services.
  - Kernel acts **as the abstraction layer** between system resources & user applications.

- **What are the functions handled by the general purpose kernel? Explain.**
  - Process Management
  - Primary Memory Management
  - I/O System(Device) Management
  - File System Management
  - Secondary Storage Management

- **Process Management**
  - Managing the process/tasks
  - Setup the memory space for process
  - Load program/code into space(memory)
  - Scheduling & managing the execution of the process
  - Setting up & managing the PCB
  - Inter process communication & system synchronization process termination & deletion
- **Primary Memory Management**
  - The MMU of kernel - Keeping track of which part of memory area is correctly used by which process.
  - Allocating & de allocating the memory spaces .
- **I/O System(Device) Management**
  - Kernel is responsible for routing the I/O request coming from different user applications .
  - Device manager of the kernel is responsible for handling I/O device related operations.
  - Exchanging the information.

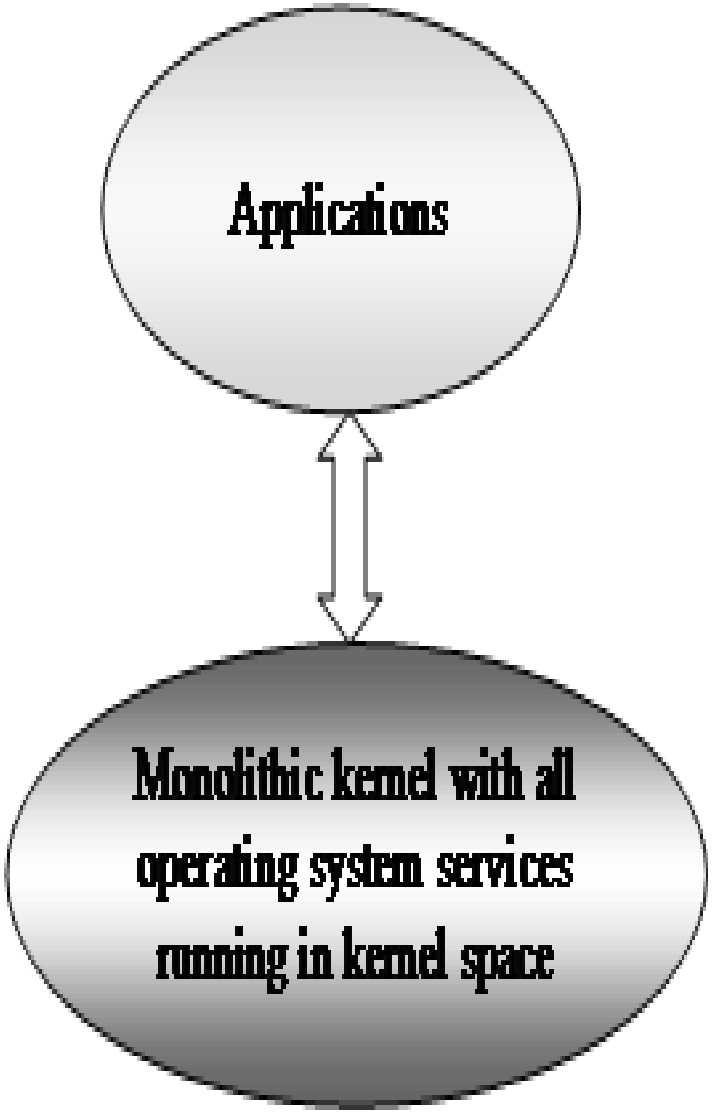
- **File System Management**

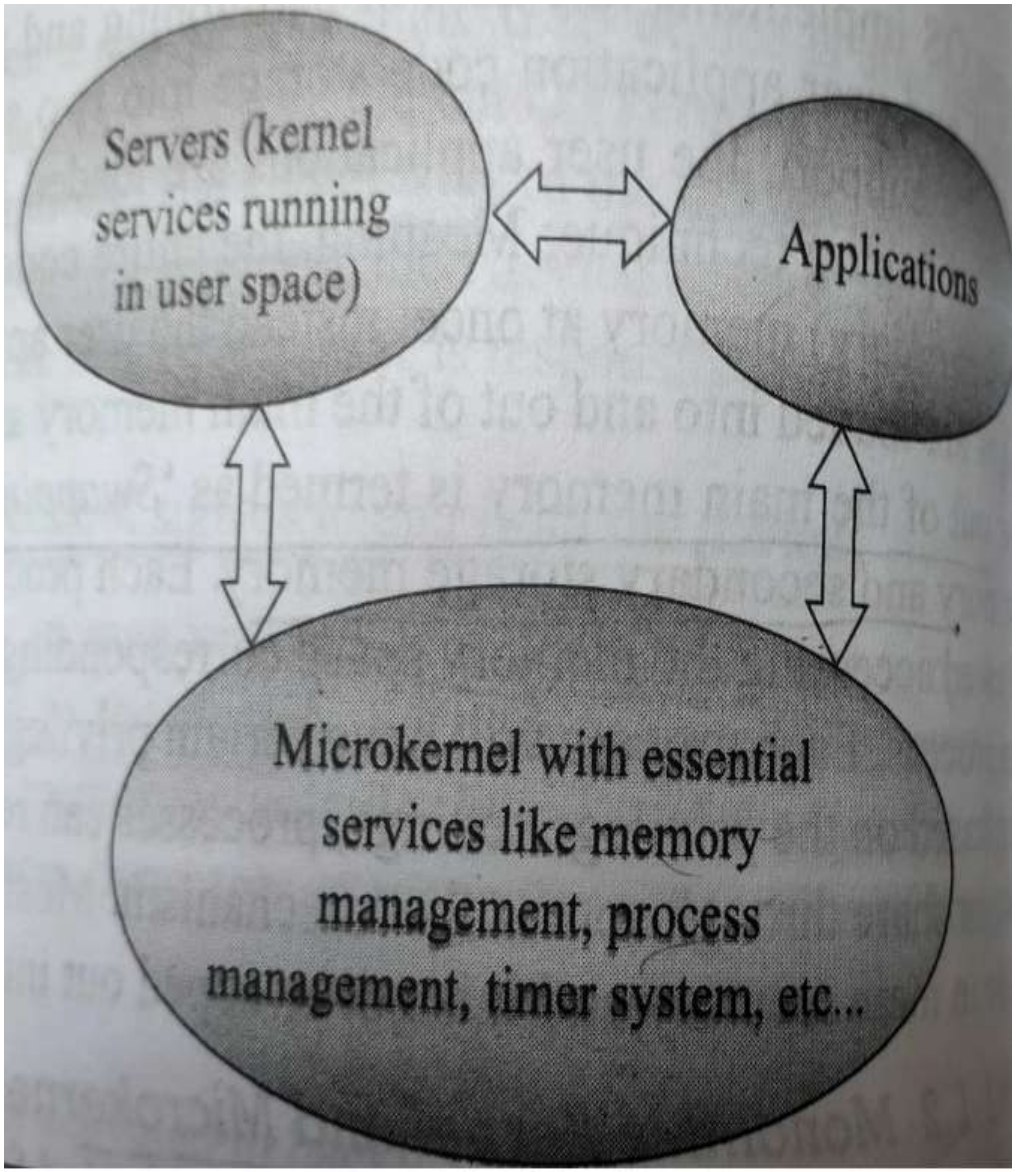
The file system management service of kernel is responsible for

  - The creation & deletion of files.
  - Creation and deletion and alterations of directories.
  - Saving the file on secondary storage.
  - Providing automatic allocation of spaces.
  - Providing a flexible naming convention for the files.

- **Secondary Storage Management**
  - Disk storage allocation
  - Disk scheduling
  - Free disk space management
- In addition kernel also ensures various **protection policies** for protecting the system.
- Provides **interrupt handler** mechanisms for all external /internal interrupt generated by the system.

- **What are Kernel space and user space?**
  - Memory space at which the kernel code is located is known as kernel space.
  - Memory space at which user applications are located known as user space.
- **What is Swapping?**
  - The act of loading code into and out of the Main memory.
  - Swapping happens between main memory and Secondary Storage memory.
- **What are Monolithic kernel and Microkernel?**
- **Monolithic kernel**
  - In monolithic kernel architecture, all kernel services run in kernel space.
  - **Disadvantage** of this design is that any failure or error leads to the crashing of the entire kernel application,
  - Examples are **LINUX,SOLARIS,MS-DOS** kernels.
- **Microkernel**
  - In Microkernel design, only essential applications run in kernel space other operating system services run in user space as programs called as **servers**.
  - Examples are **mininx3 and qnx.**
- **Benefits of Microkernel based design**
  - Robustness
  - Configurability


Applications
Monolithic kernel with all operating system services running in kernel space


Servers (kernel services running in user space)
Applications
Microkernel with essential services like memory management, process management, timer system, etc...

Types of an operating systems

- **Distinguish between GPOS and RTOS?**
- **General Purpose Operating System[GPOS]**
  - **A GPOS is** used for systems/applications that are not time critical.
  - The OS which are deployed in general computing systems are referred as general purpose OS.[Desktop, Laptop, Tablet computers].
  - The kernel of such OS is more generalized & it contains all kinds of services required for executing generic applications
  - GPOS are non-deterministic in behavior and Response Times not Guaranteed
  - Examples: Windows, Linux, Unix etc.
- **Real Time Operating System[RTOS]**
  - **An RTOS is** used for time critical systems. RTOS is an OS intended to serve real-time application requests.[Embedded Computing].
  - Real-time implies **deterministic timing behavior and** Response Time is Guaranteed
  - RTOS decides which application should run in which order & how much time needs to be allocated for each applications.
  - Examples: WindowsCE,UNIX,VxWorks,Micros/OS-II

Real time kernel?

- Kernel of the real time operating system is referred to as real time kernel.
- Compared to conventional OS Kernel, It is highly Specialized and contain minimal set of services
- The Basic functions of Real time kernel are listed below
- A **real-time kernel** is software that manages the **time** of microprocessor to ensure that **time-**critical events are processed as efficiently as possible.

What are the Basic functions of a real time kernel? Explain.

- Task/Process Management
- Task/Process Scheduling
- Task/Process Synchronization
- Error/Exception Handling
- Memory Management
- Interrupt Handling
- Time Management

❑ Task/Process Management ?

- ❑ Deals with setting up the memory space for the tasks.
- ❑ Loading the tasks code into memory space.
- ❑ Allocating system resources, setting up a Task Control Block [TCB] for the task.
- ❑ Task/process termination/deletion.

❑ TCB?

- ❑ A task control block (TCB) is a data structure used by kernels to maintain information about a task.
- ❑ Each task requires its own TCB
- ❑ **Task Control Block** is used for holding the information corresponding to a task.
  - ❑ **Task ID**
  - ❑ **Task State**
  - ❑ **Task Type**
  - ❑ **Task Priority**
  - ❑ **Task Context Pointer**
  - ❑ **Task Memory Pointer**
  - ❑ **Task System Resource Pointers**
  - ❑ **Task Pointers**
  - ❑ **Other Parameters**

- **When task is created real time kernel,**
  - Creates TCB for a task on creating task.
  - Delete/Remove the TCB when the task is terminated or deleted.
  - Reads the TCB to get the state of a task.
  - Updates the TCB with updated parameters on need basics.
  - Modify the TCB to change the priority of task dynamically.

- **Task/Process Scheduling?**
  - CPU is shared among various tasks/process.
  - The task scheduling is the activity of assigning the tasks in the system in a manner, that will optimize the overall performance of the application, while assuring the correctness of the result.
  - A kernel applications called scheduler, handles the task scheduling.

- **Task/Process Synchronization?**
  - Deals with concurrent accessing of system a resource.

- **Error/Exception Handling?**
  - Ex: Insufficient memory, time outs, dead locks, dead line missing, bus error, divide by zero, unknown instruction execution.

- **Memory Management?**
  - RTOS makes use of **'Block Based Memory'** allocation techniques instead of the usual dynamic memory allocation technique used by GPOS.
  - RTOS kernel uses blocks of fixed size dynamic memory.
  - Block is allocated for a task on a need of basis.
  - A few RTOS kernel implements **Virtual Memory concepts**.
  - Block memory avoid the garbage collection overhead.
- **Interrupt Handler?**
  - Interrupts can be either Synchronous Asynchronous. Interrupts which occurs in synch with the currently executing task is known as **synchronous interrupts.**
  - **Asynchronous interrupts** are those which occurs at any point of execution of any task & are not in sync with currently executing tasks.
  - Most of the real time kernel implements **nested interrupts.**
- **Time management?**
  - Implemented with RTC(real time clock)
  - Handles timer tick interrupt
- **Hard real time?**
- **Soft real time?**

- **Describe the difference between the soft real-time systems and hard real-time systems with examples?**
- **Hard real time systems?**
  - Real time operating systems that strictly adhere to the timing constraint for a task is referred to as **Hard real time systems.**
  - They are time critical, missing any deadline may produce **catastrophic** results.
  - **"A late answer is a wrong answer"**
  - Example: **ABS ,Air Traffic Control System**

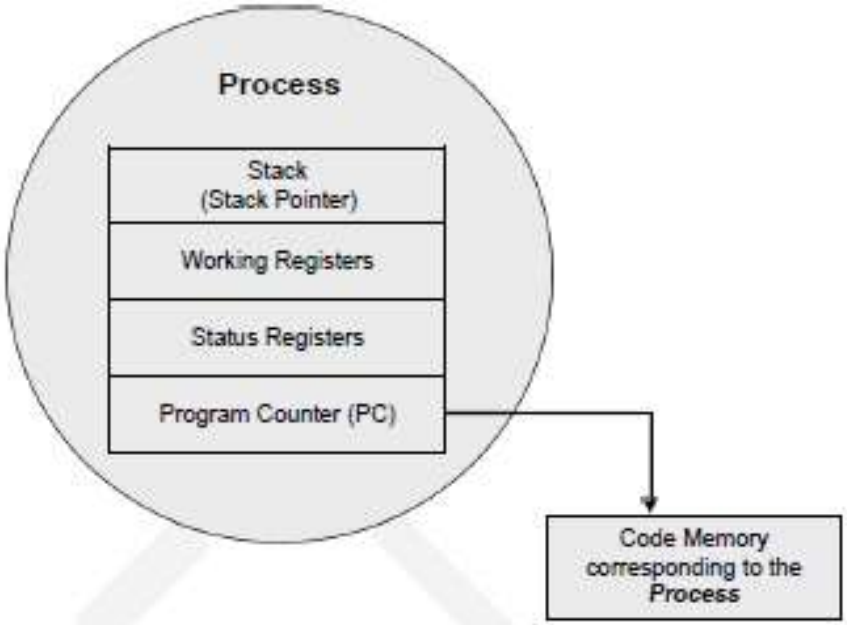
- **Soft real time systems?**
  - Real time operating systems that does not guarantee meeting deadlines but offer the best effort to meet the deadline are referred to as **soft real time systems.**
  - They are not time critical, missing deadlines for a tasks is acceptable.
  - **"A late answer is a an acceptable answer"**
  - Example: **ATM**

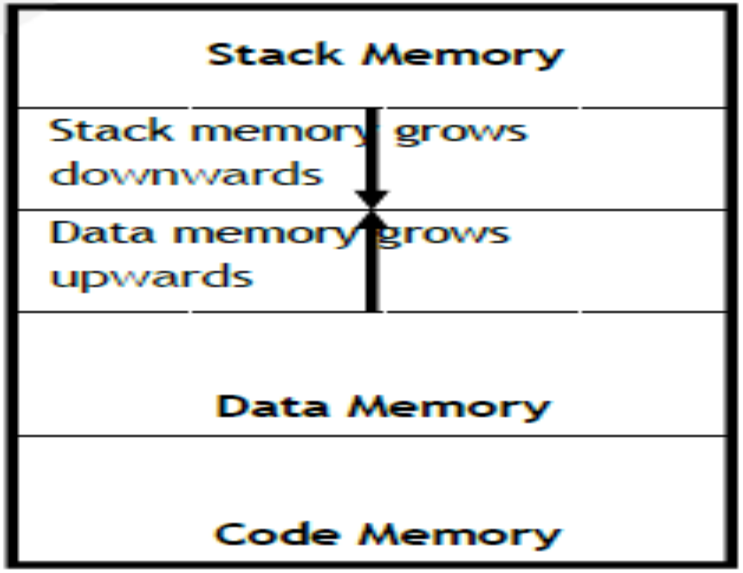
- **Hard real time systems(Example Description)**
- **Air traffic controllers** monitor the location of aircraft in their assigned airspace by radar and communicate with the pilots by radio. To prevent collisions
- **Air traffic control** keeps aircraft from colliding with each other.
- Any delay in response causes a severe damage.
- In general, Suitable for applications that are time critical and where the response time is so important (Embedded systems where missing deadlines are not acceptable).

- **Soft real time systems(Example Description)**
- **ATM** (**Automatic Teller Machine**) is a banking terminal that accepts deposits and dispenses cash.
- When money is needed, insert **ATM** card, and take the required amount.
- Any delay in response does not cause any physical damages.
- In general, Suitable for applications that are not time critical and where the response time is not so important (Embedded systems where missing deadlines are acceptable).

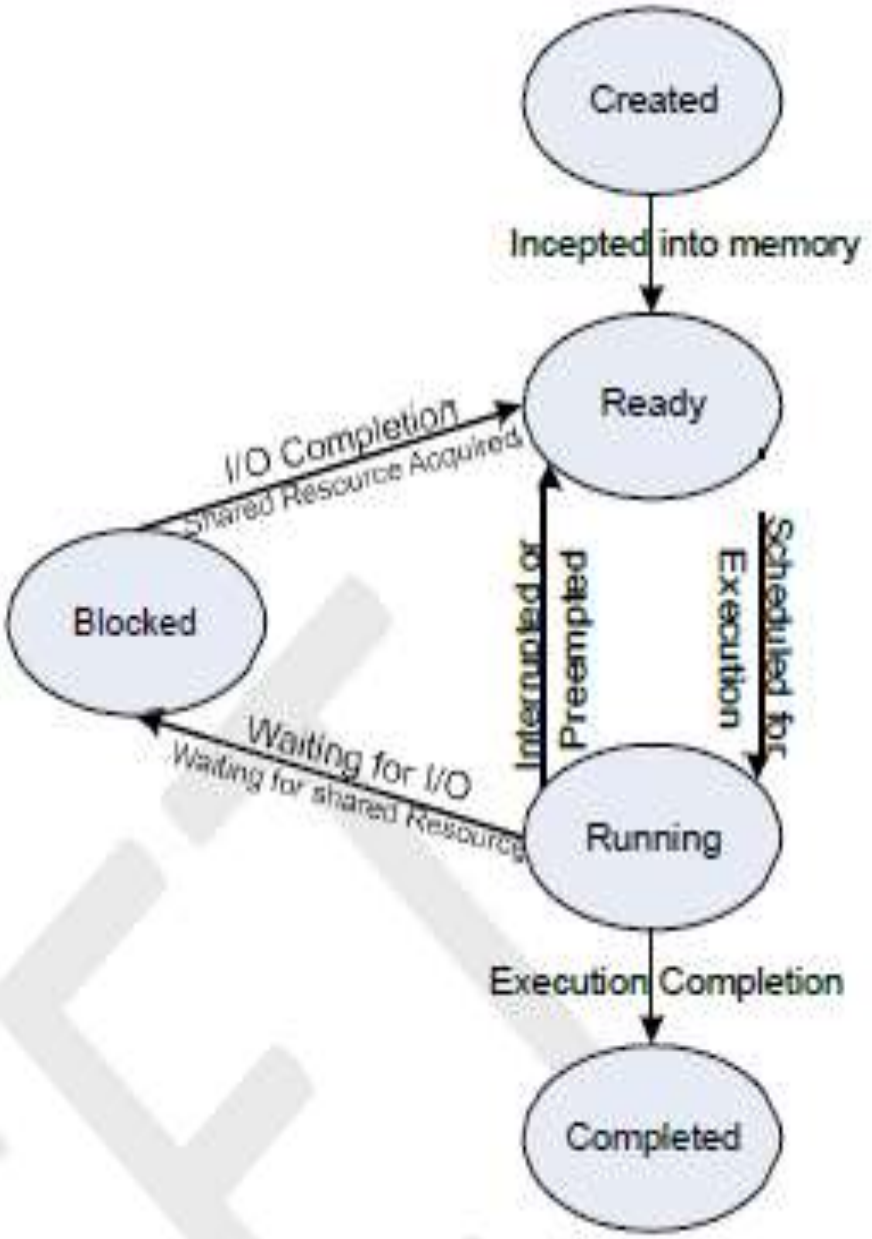
- **What are Tasks, Processes & Threads ?**
- In the Operating System context, a task is defined as the program in execution Task is also known as 'Job' in the operating system context.
- A program or part of it in execution is called as 'Process'.
- The terms 'Task', 'job' and 'Process' refer to the same entity in the Operating System context and most often they are used interchangeably.
- A **thread** is a path of execution within a process. A process can contain multiple **threads**.

- **Describe the structure of a Process?**
- The concept of 'Process' leads to concurrent execution (pseudo parallelism) of tasks and thereby the efficient utilization of the CPU and other system resources.
- A process mimics a processor in properties and holds a set of registers, process status, a Program Counter (PC) to point to the next executable instruction of the process, a stack for holding the local variables associated with the process and the code corresponding to the process.
- A process, which inherits all the properties of the CPU, can be considered as a virtual processor.





- **Describe the memory organization of a process?**
- The memory occupied by the process is segregated into three regions namely; Stack memory, Data memory and Code memory.
- The 'Stack' memory holds all temporary data such as variables local to the process.
- Data memory holds all global data for the process.
- The code memory contains the program code (instructions) corresponding to the process



- **Describe process states and state transition with neat representation**
- Created State: The state at which a process is being created is referred as 'Created State'.
- Ready State: The state, where a process is incepted into the memory and awaiting the processor time for execution, is known as 'Ready State'.
- Running State: The state where in the source code instructions corresponding to the process is being executed is called 'Running State'.
- Blocked State/Wait State: Refers to a state where a running process is temporarily suspended from execution
- Completed State: A state where the process completes its execution

- **In detail explain the concept of threads?**
- Thread: A thread is a path of execution or flow of control within a process.
- Multithreading: A process can contain multiple **threads**. multiple processes can be executed Parallelly by increasing number of threads.
- Types of Thread: **User Threads** and **Kernel Threads**
- Multithreading Models: Many to One Model ,One to One Model and Many to Many Model
- Many to One Model: In the **many to one** model, many user-level threads are all mapped to a single kernel thread
- One to One Model: The **one to one** model creates a separate kernel thread to handle each and every user thread.
- Many to Many Model: The **many to many** model multiplexes any number of user threads to an equal or smaller number of kernel threads
- Three types of Thread: **POSIX Pitheads, Win32 threads and Java threads**
- Advantages
  - Better memory utilization
  - Efficient CPU utilization
  - Speeds up the execution process.

Explain the differences between Thread and process?

THREAD	PROCESS
❑Thread is a single unit of execution and is part of process.	Process is a program in execution and contains one or more threads.
❑A thread does not have its own data memory and heap memory.	Process has its own code memory, data memory and stack memory.
❑A thread cannot live independently; it lives within the process.	A process contains at least one thread.
❑Threads are very inexpensive to create	Processes are very expensive to create. Involves many OS overhead.
❑Context switching is inexpensive and fast	Context switching is complex and involves lot of OS overhead and is comparatively slower.
❑If a thread expires, its stack is reclaimed by the process.	❑If a process dies, the resources allocated to it are reclaimed by the OS and all the associated threads of the process also dies.

- **Define the following terms?**
- Multiprocessing: The ability to execute multiple processes simultaneously is referred as multiprocessing.
- Multiprocessor systems: Systems which are capable of performing multiprocessing are known as multiprocessor systems.
- Multiprogramming: The ability of the Operating System to have multiple programs in memory, which are ready for execution, is referred as multiprogramming.
- Multitasking: Multitasking refers to the ability of an operating system to hold multiple processes in memory and switch the processor (CPU) from executing one process to another process.
- Multitasking involves 'Context switching', 'Context saving' and 'Context retrieval'
- Context switching: Context switching refers to the switching of execution context from task to other task.
- Context saving: When a task/process switching happens, the current context of execution should be saved to (Context saving) retrieve it at a later point of time when the CPU executes the process, which is interrupted currently due to execution switching.
- Context retrieval: During context switching, the context of the task to be executed is retrieved from the saved context list. This is known as Context retrieval

- **Explain the different types of Multitasking?**
- Co-operative Multitasking :Co-operative multitasking is the most primitive form of multitasking in which a task/process gets a chance to execute only when the currently executing task/process voluntarily relinquishes the CPU. In this method, any task/process can avail the CPU as much time as it wants. Since this type of implementation involves the mercy of the tasks each other for getting the CPU time for execution, it is known as co-operative multitasking.

- Preemptive Multitasking: Preemptive multitasking ensures that every task/process gets a chance to execute. When and how much time a process gets is dependent on the implementation of the preemptive scheduling. As the name indicates, in preemptive multitasking, the currently running task/process is preempted to give a chance to other tasks/process to execute.

- Non-preemptive Multitasking: The process/task, which is currently given the CPU time, is allowed to execute until it terminates (enters the 'Completed' state) or enters the 'Blocked/Wait' state, waiting for an I/O. The co- operative and non-preemptive multitasking differs in their behavior when they are in the 'Blocked/Wait' state. In co-operative multitasking, the currently executing process/task need not relinquish the CPU when it enters the 'Blocked/Wait' sate, waiting for an I/O, or a shared resource access or an event to occur whereas in non-preemptive multitasking the currently executing task relinquishes the CPU when it waits for an I/O.

- **Explain the various factors on which scheduling algorithms are selected?**
- CPU Utilization: The scheduling algorithm should always make the CPU utilization high. CPU utilization is a direct measure of how much percentage of the CPU is being utilized.
- Throughput: This gives an indication of the number of processes executed per unit of time. The throughput for a good scheduler should always be higher.
- Turnaround Time: It is the amount of time taken by a process for completing its execution. It includes the time spent by the process for waiting for the main memory, time spent in the ready queue, time spent on completing the I/O operations, and the time spent in execution. The turnaround time should be a minimum for a good scheduling algorithm.
- Waiting Time: It is the amount of time spent by a process in the 'Ready' queue waiting to get the CPU time for execution. The waiting time should be minimal for a good scheduling algorithm.
- Response Time: It is the time elapsed between the submission of a process and the first response. For a good scheduling algorithm, the response time should be as least as possible.

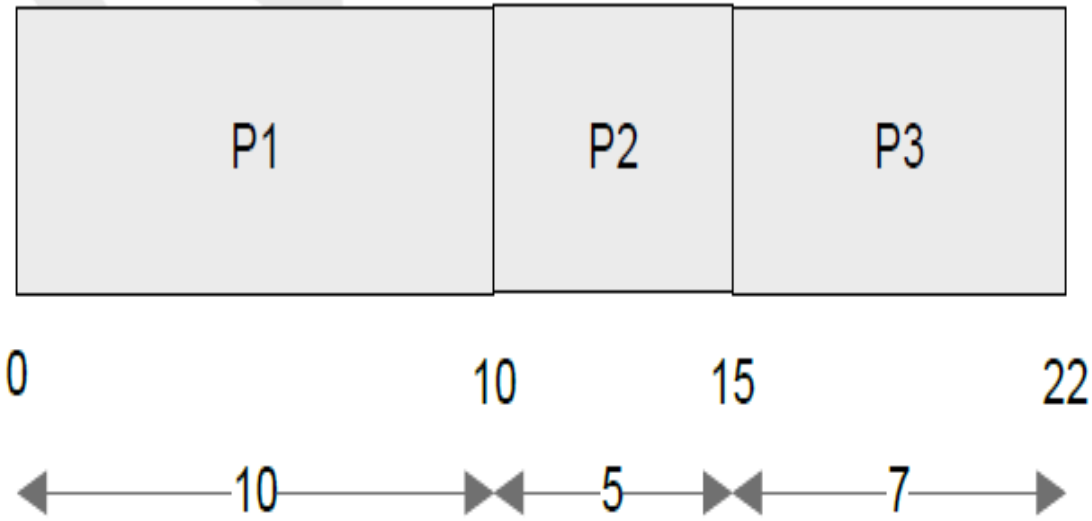
- **Non preemptive Scheduling Examples**
- In **Non-preemptive scheduling**, the CPU is allocated to the processes till it terminates or switches to waiting state.
- **First Come First Served (FCFS)/First In First Out (FIFO) Scheduling**
- **Last Come First Served (LCFS)/Last In First Out (LIFO) Scheduling**
- **Shortest Job First (SJF) Scheduling**
- **Priority based Scheduling**

- **Preemptive Scheduling Examples**
- In **preemptive scheduling** the CPU is allocated to the processes for the limited time.
- **Preemptive SJF Scheduling/ Shortest Remaining Time (SRT)**
- **Round Robin (RR) Scheduling**
- **Priority based Scheduling**

- **NOTE: Students must explain the above scheduling techniques and solve one or two problems with respect to all scheduling algorithms mentioned above.**

- **Non preemptive Scheduling Examples**
- **First Come First Served (FCFS)/First In First Out (FIFO) Scheduling**
- **EXAMPLE:** Three processes with process IDs P1, P2, P3 with estimated completion time 10, 5, 7 milliseconds respectively enters the ready queue together in the order P1, P2, P3. Calculate the waiting time and Turn Around Time (TAT) for each process and the Average waiting time and Turn Around Time (Assuming there is no I/O waiting for the processes).

Solution: The sequence of execution of the processes by the CPU is represented as

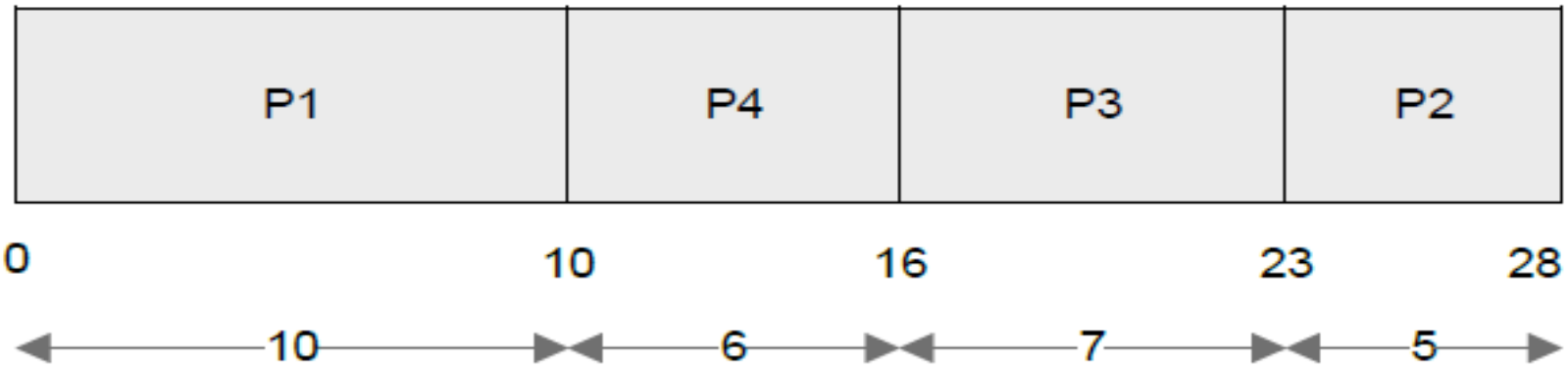


- Waiting Time for P1 = 0 ms (P1 starts executing first)
- Waiting Time for P2 = 10 ms (P2 starts executing after completing P1)
- Waiting Time for P3 = 15 ms (P3 starts executing after completing P1 and P2)

- Average waiting time = (Waiting time for all processes) / No. of Processes
  = (Waiting time for (P1+P2+P3)) / 3
  = (0+10+15)/3 = 25/3 = 8.33 milliseconds
- Turn Around Time (TAT) for P1 = 10 ms (Time spent in Ready Queue + Execution Time)
- Turn Around Time (TAT) for P2 = 15 ms (-Do-)
- Turn Around Time (TAT) for P3 = 22 ms (-Do-)

- Average Turn Around Time= (Turn Around Time for all processes) / No. of Processes
  = (Turn Around Time for (P1+P2+P3)) / 3
  = (10+15+22)/3 = 47/3
  = 15.66 milliseconds

- **Non preemptive Scheduling Examples**
- **Last Come First Served (LCFS)/Last In First Out (LIFO) Scheduling**
- **EXAMPLE:** Three processes with process IDs P1, P2, P3 with estimated completion time 10, 5, 7 milliseconds respectively enters the ready queue together in the order P1, P2, P3 (Assume only P1 is present in the 'Ready' queue when the scheduler picks up it and P2, P3 entered 'Ready' queue after that). Now a new process P4 with estimated completion time 6ms enters the 'Ready' queue after 5ms of scheduling P1. Calculate the waiting time and Turn Around Time (TAT) for each process and the Average waiting time and Turn Around Time (Assuming there is no I/O waiting for the processes).Assume all the processes contain only CPU operation and no I/O operations are involved.



- The waiting time for all the processes are given as Waiting Time for P1 = 0 ms (P1 starts executing first)
- Waiting Time for P4 = 5 ms (P4 starts executing after completing P1.
- But P4 arrived after 5ms of execution of P1.
- Hence its waiting time = (Execution start time – Arrival Time = 10-5 = 5)
- Waiting Time for P3 = 16 ms (P3 starts executing after completing P1 and P4)
- Waiting Time for P2 = 23 ms (P2 starts executing after completing P1, P4 and P3)

- Average waiting time = (Waiting time for all processes) / No. of Processes
  = (Waiting time for (P1+P4+P3+P2)) / 4
  = (0 + 5 + 16 + 23)/4 = 44/4
  = 11 milliseconds

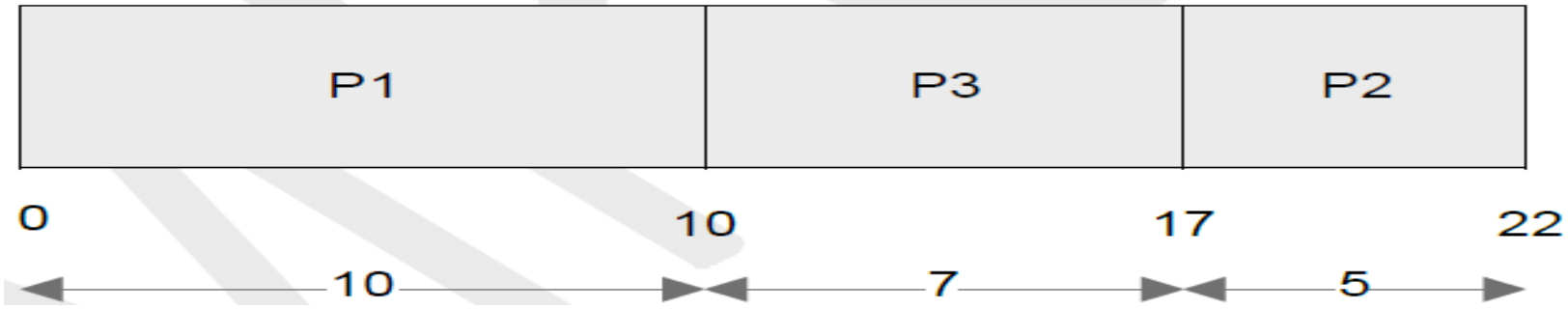
- Turn Around Time (TAT) for P1 = 10 ms (Time spent in Ready Queue + Execution Time)
- Turn Around Time (TAT) for P4 = 11 ms (Time spent in Ready Queue + Execution Time = (Execution Start Time – Arrival Time) + Estimated Execution Time = (10-5) + 6 = 5 + 6)
- Turn Around Time (TAT) for P3 = 23 ms (Time spent in Ready Queue + Execution Time)
- Turn Around Time (TAT) for P2 = 28 ms (Time spent in Ready Queue + Execution Time)
- Average Turn Around Time = (Turn Around Time for all processes) / No. of Processes

  = (Turn Around Time for (P1+P4+P3+P2)) / 4

  = (10+11+23+28)/4 = 72/4

  = 18 milliseconds

- **Non preemptive Scheduling Examples**
- **Shortest Job First (SJF) Scheduling (SOLVE)**
- **EXAMPLE:** Three processes with process IDs P1, P2, P3 with estimated completion time 10, 5, 7 milliseconds respectively enters the ready queue together. Calculate the waiting time and Turn Around Time (TAT) for each process and the Average waiting time and Turn Around Time (Assuming there is no I/O waiting for the processes) in SJF algorithm.
- **Priority based Scheduling**
- **EXAMPLE:** Three processes with process IDs P1, P2, P3 with estimated completion time 10, 5, 7 milliseconds and priorities 0, 3, 2 (0- highest priority, 3 lowest priority) respectively enters the ready queue together. Calculate the waiting time and Turn Around Time (TAT) for each process and the Average waiting time and Turn Around Time (Assuming there is no I/O waiting for the processes) in priority based scheduling algorithm.



- Waiting Time for P1 = 0 ms (P1 starts executing first)
- Waiting Time for P3 = 10 ms (P3 starts executing after completing P1)
- Waiting Time for P2 = 17 ms (P2 starts executing after completing P1 and P3)
- Average waiting time = (Waiting time for all processes) / No. of Processes

  = (Waiting time for (P1+P3+P2)) / 3

  = (0+10+17)/3 = 27/3

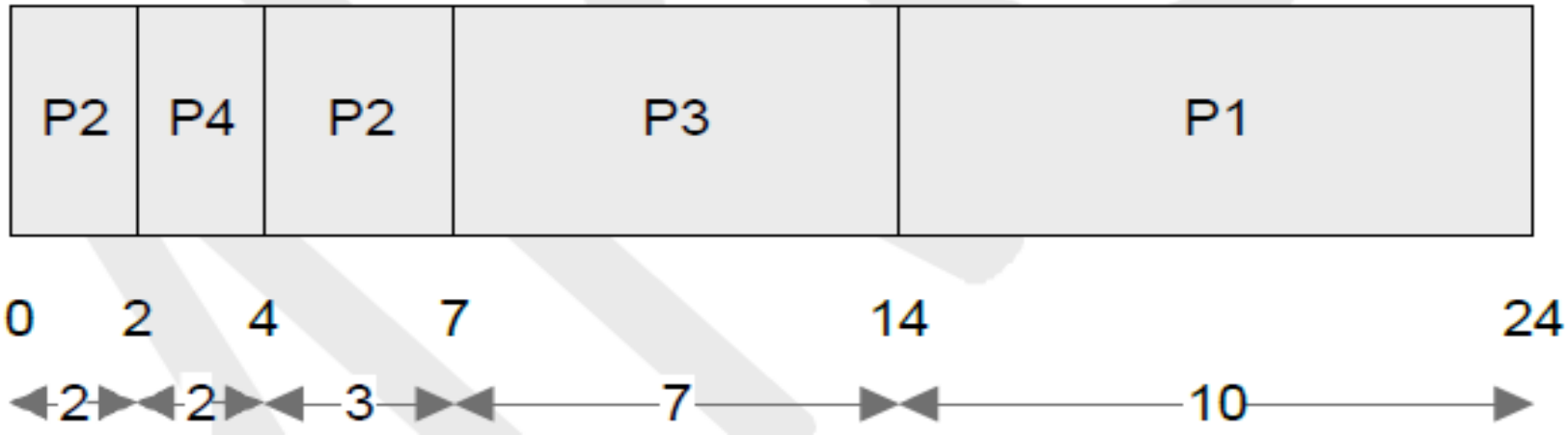
  = 9 milliseconds
- Turn Around Time (TAT) for P1 = 10 ms (Time spent in Ready Queue + Execution Time)
- Turn Around Time (TAT) for P3 = 17 ms (-Do-)
- Turn Around Time (TAT) for P2 = 22 ms (-Do-)
- Average Turn Around Time= (Turn Around Time for all processes) / No. of Processes

  = (Turn Around Time for (P1+P3+P2)) / 3

  = (10+17+22)/3 = 49/3

  = 16.33 milliseconds

- **Preemptive Scheduling Examples**
- **Preemptive SJF Scheduling/ Shortest Remaining Time (SRT)**
- **EXAMPLE:** Three processes with process IDs P1, P2, P3 with estimated completion time 10, 5, 7 milliseconds respectively enters the ready queue together. A new process P4 with estimated completion time 2ms enters the 'Ready' queue after 2ms. Assume all the processes contain only CPU operation and no I/O operations are involved.



- Waiting Time for P2 = 0 ms + (4 -2) ms = 2ms (P2 starts executing first and is interrupted by P4 and has to wait till the completion of P4 to get the next CPU slot)
- Waiting Time for P4 = 0 ms (P4 starts executing by preempting P2 since the execution time for completion of P4 (2ms) is less than that of the Remaining time for execution completion of P2 (Here it is 3ms))
- Waiting Time for P3 = 7 ms (P3 starts executing after completing P4 and P2)
- Waiting Time for P1 = 14 ms (P1 starts executing after completing P4, P2 and P3)

- Average waiting time = (Waiting time for all the processes) / No. of Processes
  = (Waiting time for (P4+P2+P3+P1)) / 4
  = (0 + 2 + 7 + 14)/4 = 23/4
  = 5.75 milliseconds

- Turn Around Time (TAT) for P2 = 7 ms (Time spent in Ready Queue + Execution Time)
- Turn Around Time (TAT) for P4 = 2 ms
- (Time spent in Ready Queue + Execution Time = (Execution Start Time – Arrival Time) + Estimated Execution Time = (2-2) + 2)
- Turn Around Time (TAT) for P3 = 14 ms (Time spent in Ready Queue + Execution Time)
- Turn Around Time (TAT) for P1 = 24 ms (Time spent in Ready Queue + Execution Time)

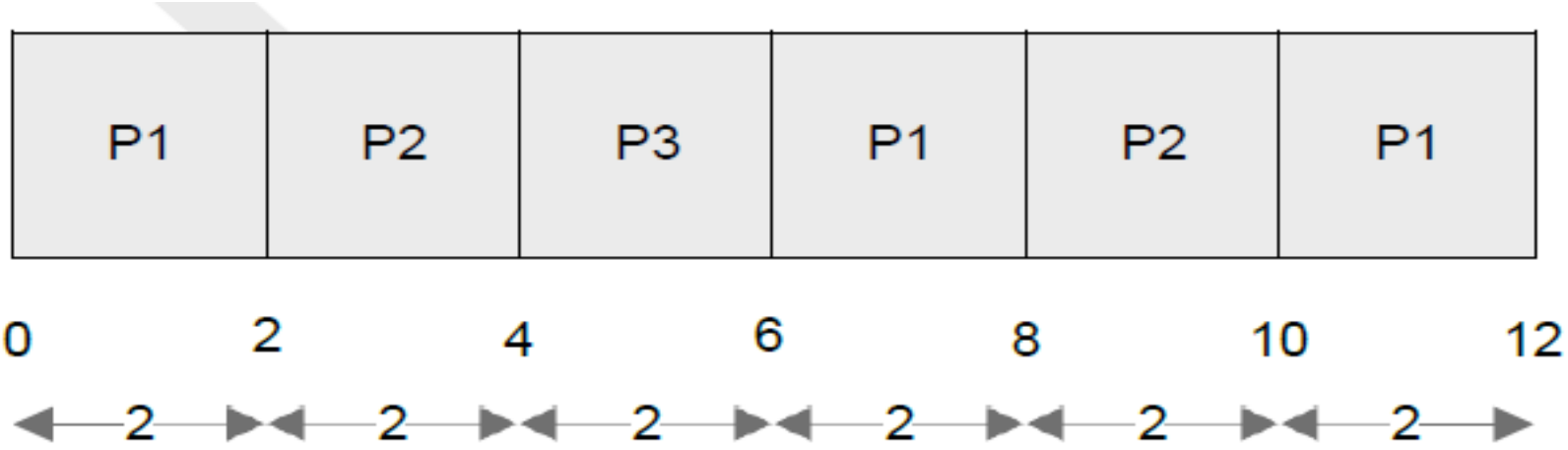
- Average Turn Around Time = (Turn Around Time for all the processes) / No. of Processes

= (Turn Around Time for (P2+P4+P3+P1)) / 4

= (7+2+14+24)/4 = 47/4

= 11.75 milliseconds

- **Preemptive Scheduling Examples**
- **Round Robin (RR) Scheduling**
- **EXAMPLE:** Three processes with process IDs P1, P2, P3 with estimated completion time 6, 4, 2 milliseconds respectively, enters the ready queue together in the order P1, P2, P3. Calculate the waiting time and Turn Around Time (TAT) for each process and the Average waiting time and Turn Around Time (Assuming there is no I/O waiting for the processes) in RR algorithm with Time slice= 2ms.



- Waiting Time for P1 = 0 + (6-2) + (10-8) = 0+4+2= 6ms (P1 starts executing first

  and waits for two time slices to get execution back and again 1 time slice for getting CPU time)
- Waiting Time for P2 = (2-0) + (8-4) = 2+4 = 6ms (P2 starts executing after P1 executes for 1 time slice and waits for two time slices to get the CPU time)
- Waiting Time for P3 = (4 -0) = 4ms (P3 starts executing after completing the first time slices for P1 and P2 and completes its execution in a single time slice.)

- Average waiting time = (Waiting time for all the processes) / No. of Processes

  = (Waiting time for (P1+P2+P3)) / 3

  = (6+6+4)/3 = 16/3

  = 5.33 milliseconds

- Turn Around Time (TAT) for P1 = 12 ms (Time spent in Ready Queue + Execution Time)
- Turn Around Time (TAT) for P2 = 10 ms (-Do-)
- Turn Around Time (TAT) for P3 = 6 ms (-Do-)
- Average Turn Around Time = (Turn Around Time for all the processes) / No. of Processes

  = (Turn Around Time for (P1+P2+P3)) / 3

  = (12+10+6)/3 = 28/3

  = 9.33 milliseconds.

- **Preemptive Scheduling Examples**
- **Priority based scheduling(SOLVE)**
- **EXAMPLE:** Three processes with process IDs P1, P2, P3 with estimated completion time 10, 5, 7 milliseconds and priorities 1, 3, 2 (0- highest priority, 3 lowest priority) respectively enters the ready queue together. A new process P4 with estimated completion time 6ms and priority 0 enters the 'Ready' queue after 5ms of start of execution of P1. Assume all the processes contain only CPU operation and no I/O operations are involved.

- **Explain the various ways of Task Communication**
- **Co-operating Processes:** In the co-operating interaction model, one process requires the inputs from other processes to complete its execution.
- **Competing Processes:** The competing processes do not share anything among themselves but they share the system resources. The competing processes compete for the system resources such as file, display device, etc.
- The co-operating processes exchanges information and communicate through the following methods
  - Co-operation through sharing: Exchange data through some shared resources.
  - Co-operation through Communication: No data is shared between the processes But they communicate for execution synchronization.
- Examples:
  - **Shared Memory**
  - **Message Passing**
  - **Remote Procedure Call(IPC) and Sockets**

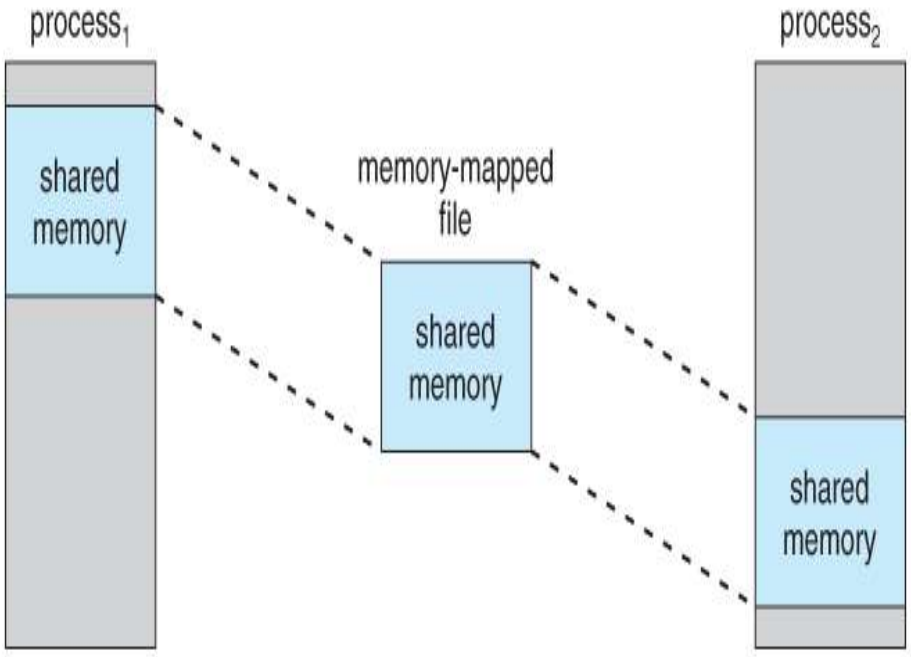
- **Explain the concept of Shared memory in detail?**
- Processes share some area of the memory to communicate among them Information to be communicated by the process is written to the shared memory area.

- Different mechanisms are adopted by different kernels for implementing this, few among are as follows:
  - Pipes
  - Memory Mapped Objects

- **Pipes:** Pipe is a section of the shared memory used by processes for communicating. Pipes follow the client-server architecture.
- A process which creates a pipe is known as pipe server and a process which connects to a pipe is known as pipe client.
- A pipe can be considered as a medium for information flow and has two conceptual ends. It can be unidirectional, allowing information flow in one direction or bidirectional allowing bi-directional information flow.

- A unidirectional pipe allows the process connecting at one end of the pipe to write to the pipe and the process connected at the other end of the pipe to read the data, whereas a bi-directional pipe allows both reading and writing at one end.

- Two types of pipes are:
- **Anonymous Pipes**: The anonymous pipes are unnamed, unidirectional pipes used for data transfer between two processes.
- **Named Pipes**: Named pipe is a named, unidirectional or bi-directional pipe for data exchange between processes.



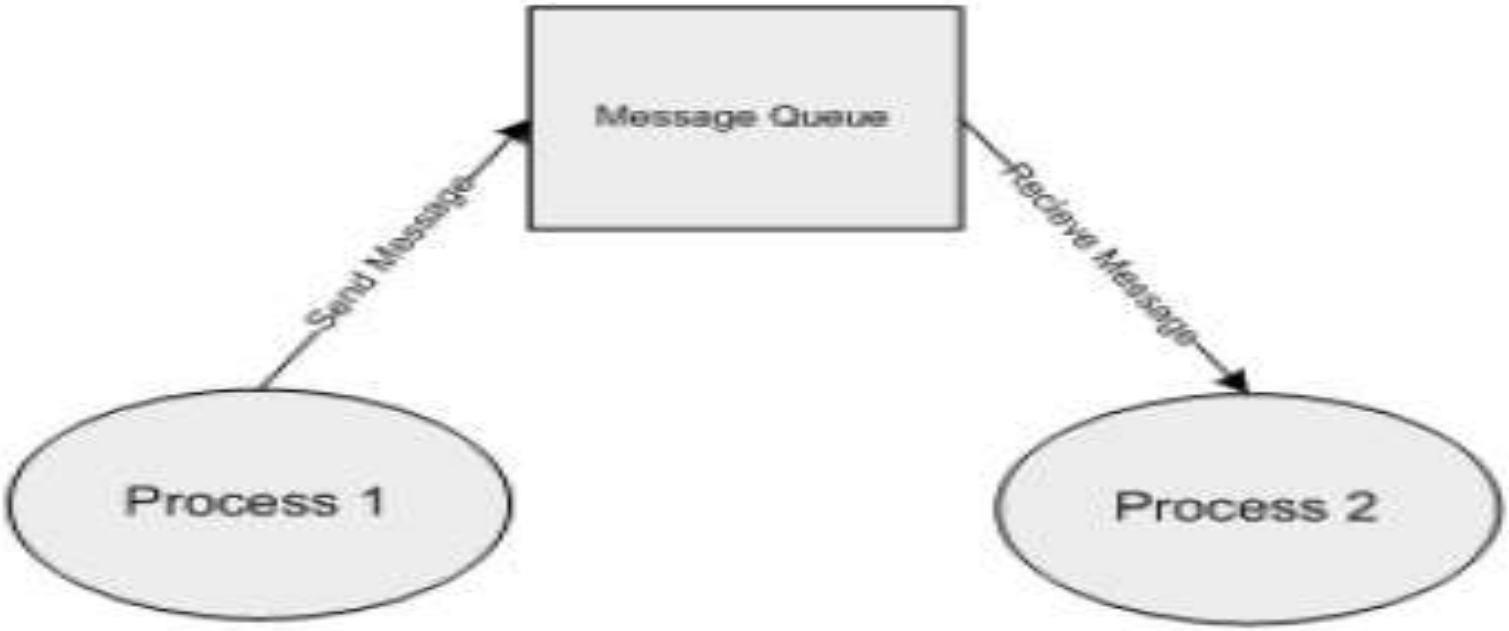
❑ **Memory Mapped Objects:** In this approach, a mapping object is created and physical storage for it is reserved and committed.

❑ A process can map the entire committed physical area or a block of it to its virtual address space.

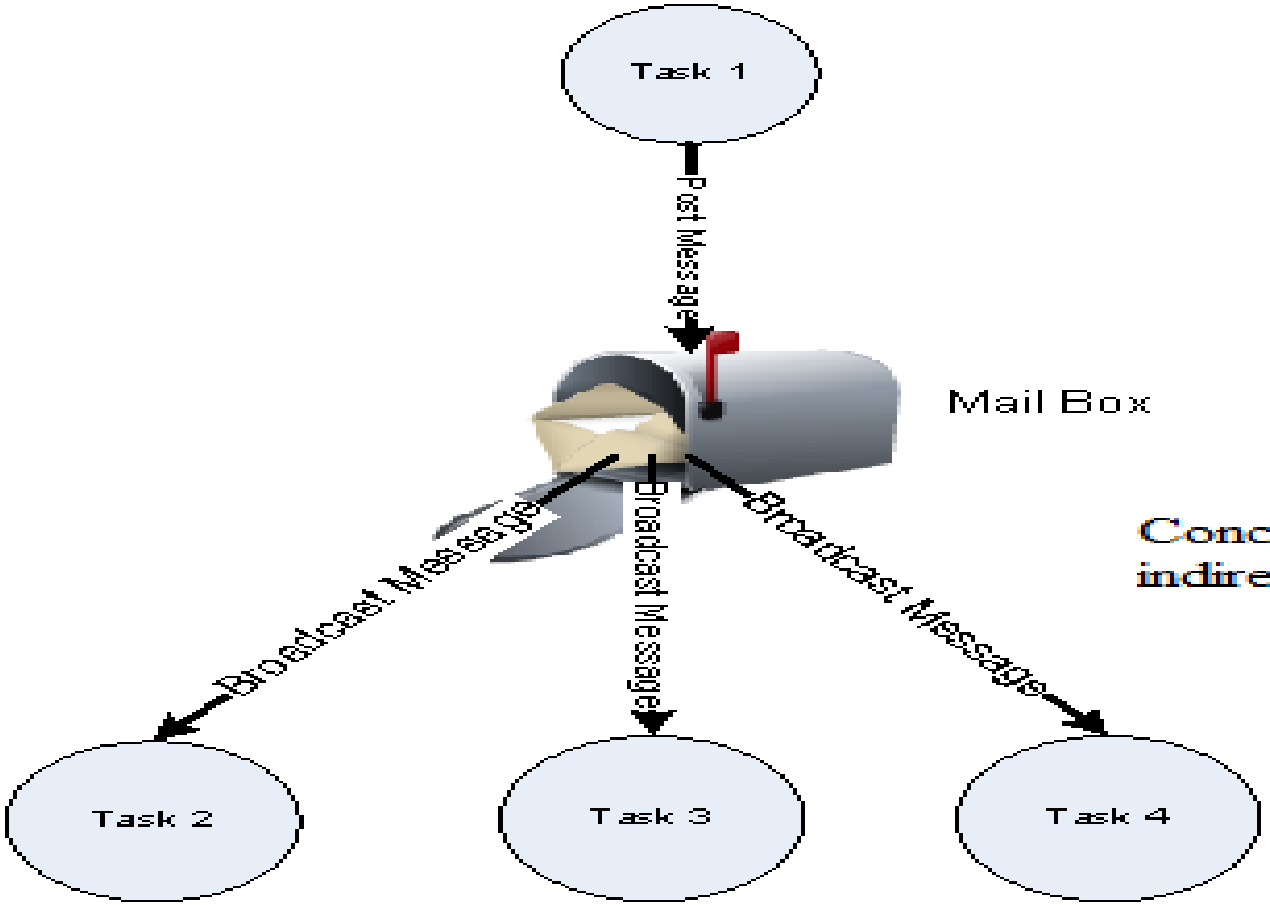
❑ All read and write operation to this virtual address space by a process is directed to its committed physical area.

❑ Any process which wants to share data with other processes can map the physical memory area of the mapped object to its virtual memory space and use it for sharing the data.

- **What is Message passing? Explain the different techniques of message passing?**
- Message passing: Message passing is a/ an synchronous/ asynchronous information exchange mechanism for Inter Process/ Thread Communication.
- Based on the message passing operation between the processes, message passing is classified into
  - Message Queue
  - Mailbox
  - Signaling



- **Message Queue:** Process which wants to talk to another process posts the message to a First-In-First-Out (FIFO) queue called message queue which stores the messages temporarily.
- The messages are exchanged through a message queue.
- Messages are sent and received through send and receive methods.



Concept of Mailbox based indirect messaging for IPC

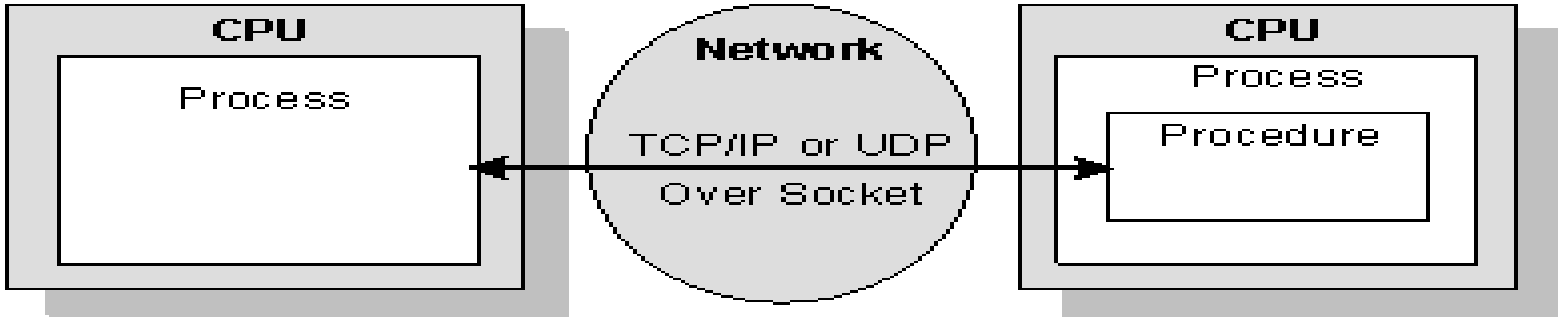
- **Mailbox:** Mailbox is a special implementation of message queue. Usually used for one way communication, only a single message is exchanged through mailbox .

- One task/process creates the mailbox and other tasks/process can subscribe to this mailbox for getting message notification.

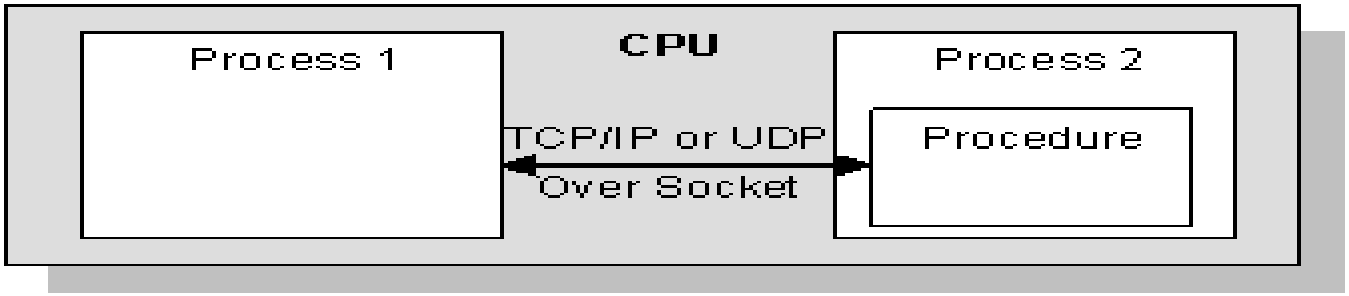
- **Signaling**
- Signals are used for an asynchronous notification mechanism.
- The signal mainly used for the execution synchronization of tasks process/ tasks.
- Signals do not carry any data and are not queued.
- The implementation of signals is OS kernel dependent

Explain Remote Procedure Call also describe the importance of Sockets?

- Remote Procedure Call is the Inter Process Communication (IPC) mechanism used by a process, to call a procedure of another process running on the same CPU or on a different CPU which is interconnected in a network.
- In the object oriented language terminology, RPC is also known as Remote Invocation or Remote Method Invocation (RMI).
- The CPU/ process containing the procedure which needs to be invoked remotely is known as server.
- The CPU/ process which initiates an RPC request is known as client.
- The RPC communication can be either synchronous or Asynchronous.



Processes running on different CPUs which are networked



Processes running on same CPU

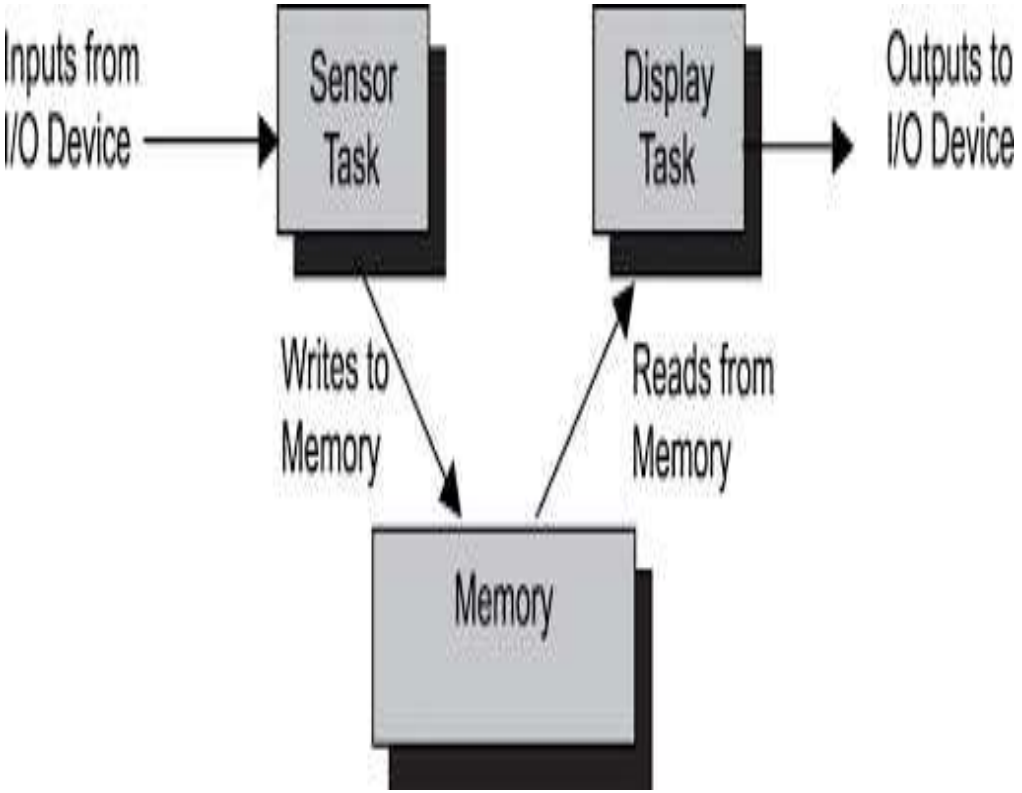
Concept of Remote Procedure Call (RPC) for IPC

- **Sockets : Sockets** are used for RPC communication.

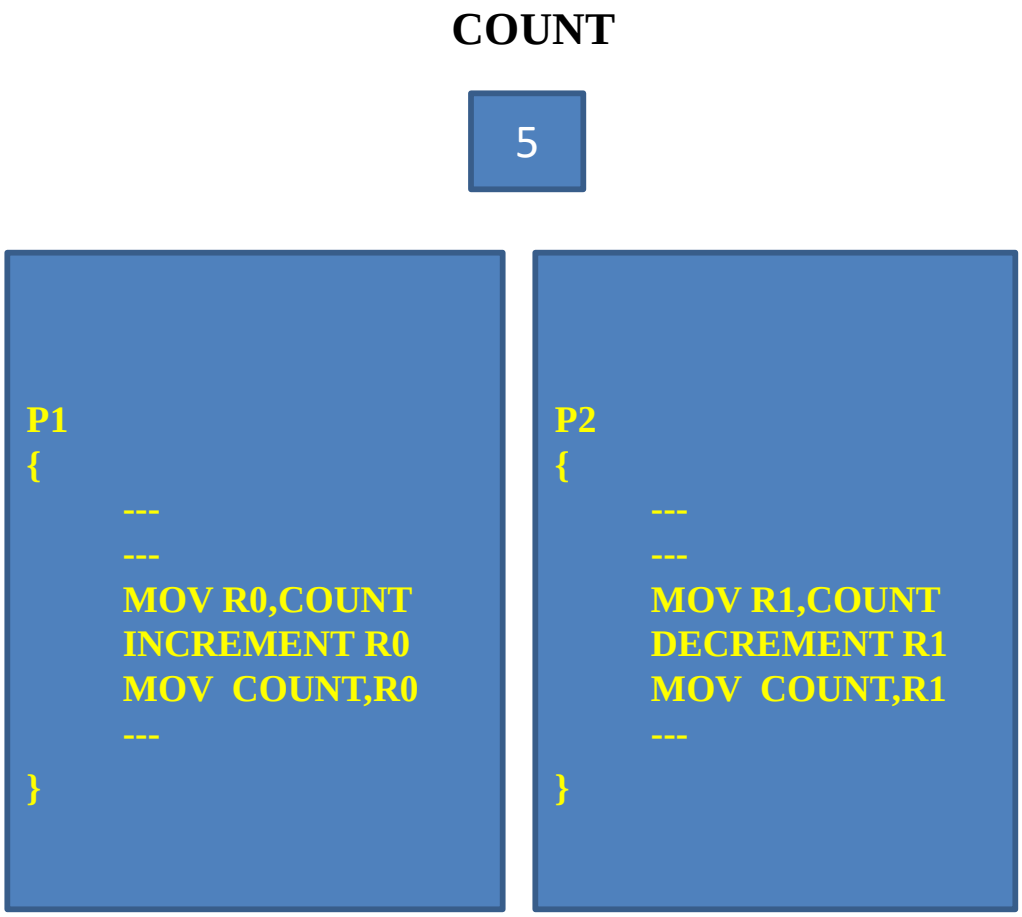
- **Socket** is a logical endpoint in a two-way communication link between two applications running on a network.

- A port number is associated with a socket so that the network layer of the communication channel can deliver the data to the designated application.
- Sockets are of different types namely; Internet sockets (INET), UNIX sockets, etc.

- **What is Task Synchronization? Explain.**
- The act of making the tasks/processes aware of the access of shared resources by each process to avoid conflicts is known as **"Task/ Process Synchronization.**
- Avoiding conflicts in resource access (racing, deadlock, etc.) in multitasking environment.
- Ensuring proper sequence of operation across processes.
- Establish proper communication between processes.
- The code memory area which holds the program instructions (piece of code) for accessing a shared resource is known as **Critical Section.**



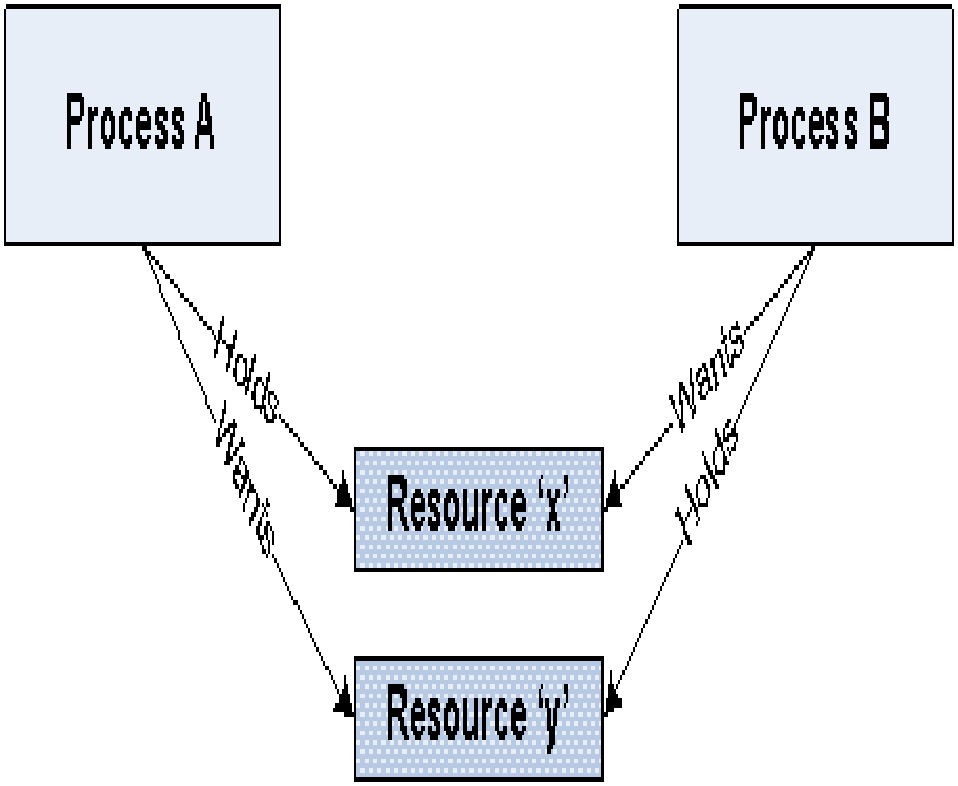
- **Illustrate the concept of Racing or Race Condition.**
- When multiple processes execute concurrently sharing system resources, then inconsistent results might be produced.
- Consider the following
- Two processes P_1 and P_2 are executing concurrently.
- Both the processes share a common variable named "count" having initial value = 5.
- Process P1 tries to increment the value of count.
- Process P2 tries to decrement the value of count.
- Now, when these processes execute concurrently without synchronization, different results may be produced.



What is deadlock? Explain.

- Deadlock is the condition in which a process is waiting for a resource held by another process which is waiting for a resource held by the first process; hence, none of the processes are able to make any progress in their execution.

- Process A holds a resource "x" and it wants a resource "y" held by Process B. Process B is currently holding resource "y" and it wants the resource "x" which is currently held by Process A. Both hold the respective resources and they compete each other to get the resource held by the respective processes.



- **Mention the various conditions favoring Deadlock.**
- Mutual Exclusion: The criteria that only one process can hold a resource at a time. Meaning processes should access shared resources with mutual exclusion.

- Hold & Wait: There must exist a process which holds some resource and waits for another resource held by some other process.

- No Resource Preemption: The criteria that Operating System cannot take back a resource from a process which is currently holding it and the resource can only be released voluntarily by the process holding it.

- Circular Wait: All the processes must wait for the resource in a cyclic manner where the last process waits for the resource held by the first process.

- **Explain the different methods of handling Deadlock.**
- **Ignore Deadlocks:** Always assume that the system design is deadlock free.

- **Detect and Recover:** Use "Back up cars " technique.

- **Back up cars technique:**
- When the vehicles from different directions compete to cross the junction, deadlock (traffic jam) condition is resulted. Once a deadlock (traffic jam) is happened at the junction, the only solution is to back up the vehicles from one direction and allow the vehicles from opposite direction to cross the junction. If the traffic is too high, lots of vehicles may have to be backed up to resolve the traffic jam. This technique is known as **"back up cars "**technique.

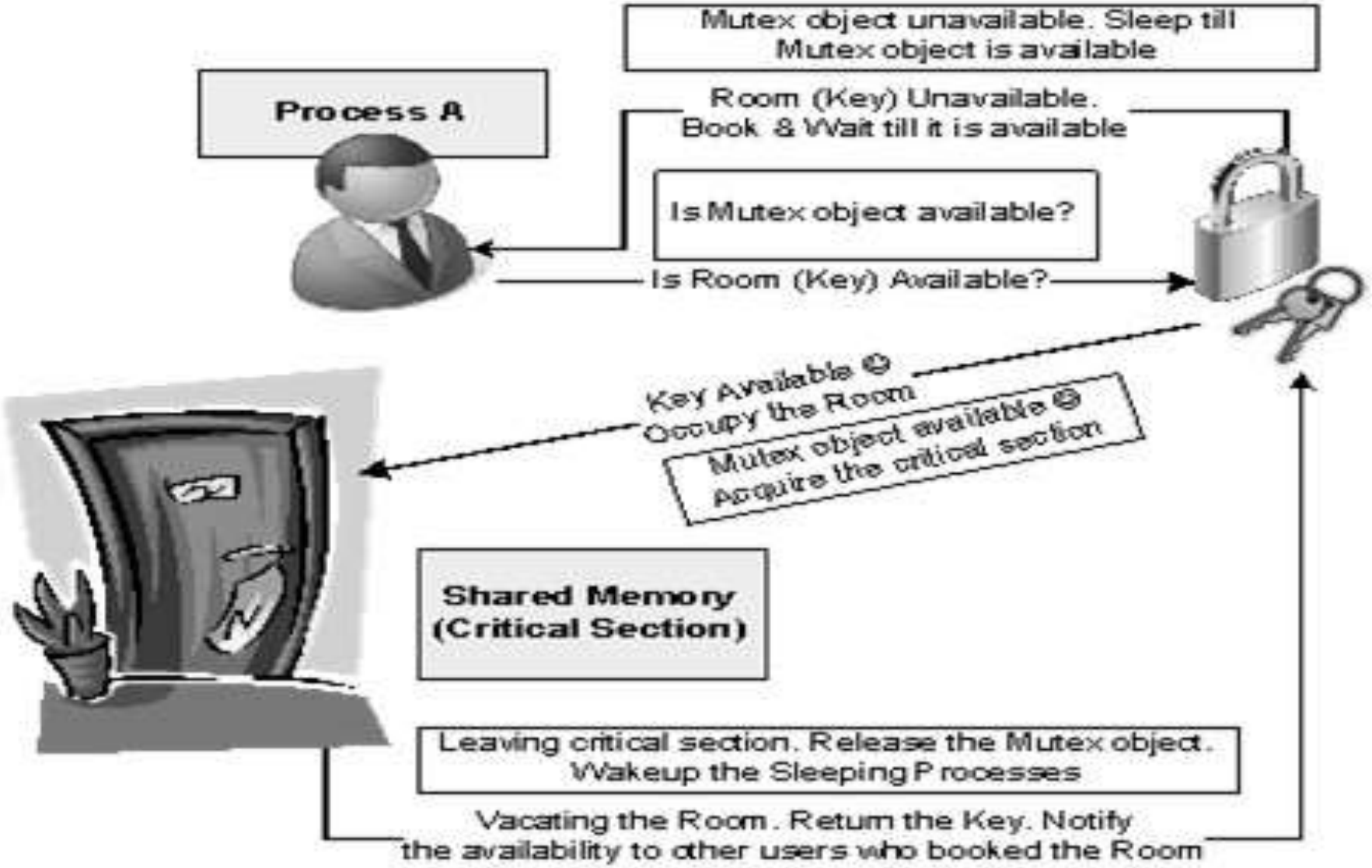
- **Avoid Deadlocks:** Deadlock is avoided by the careful resource allocation techniques by the Operating System. It is similar to the traffic light mechanism at junctions to avoid the traffic jams.

- **Prevent Deadlocks**
- Ensure that a process does not hold any other resources when it requests a resource. This can be achieved by implementing the following set of rules/ guidelines in allocating resources to processes.
  - A process must request all its required resource and the resources should be allocated before the process begins its execution.
  - Grant resource allocation requests from processes only if the process does not hold a resource currently.

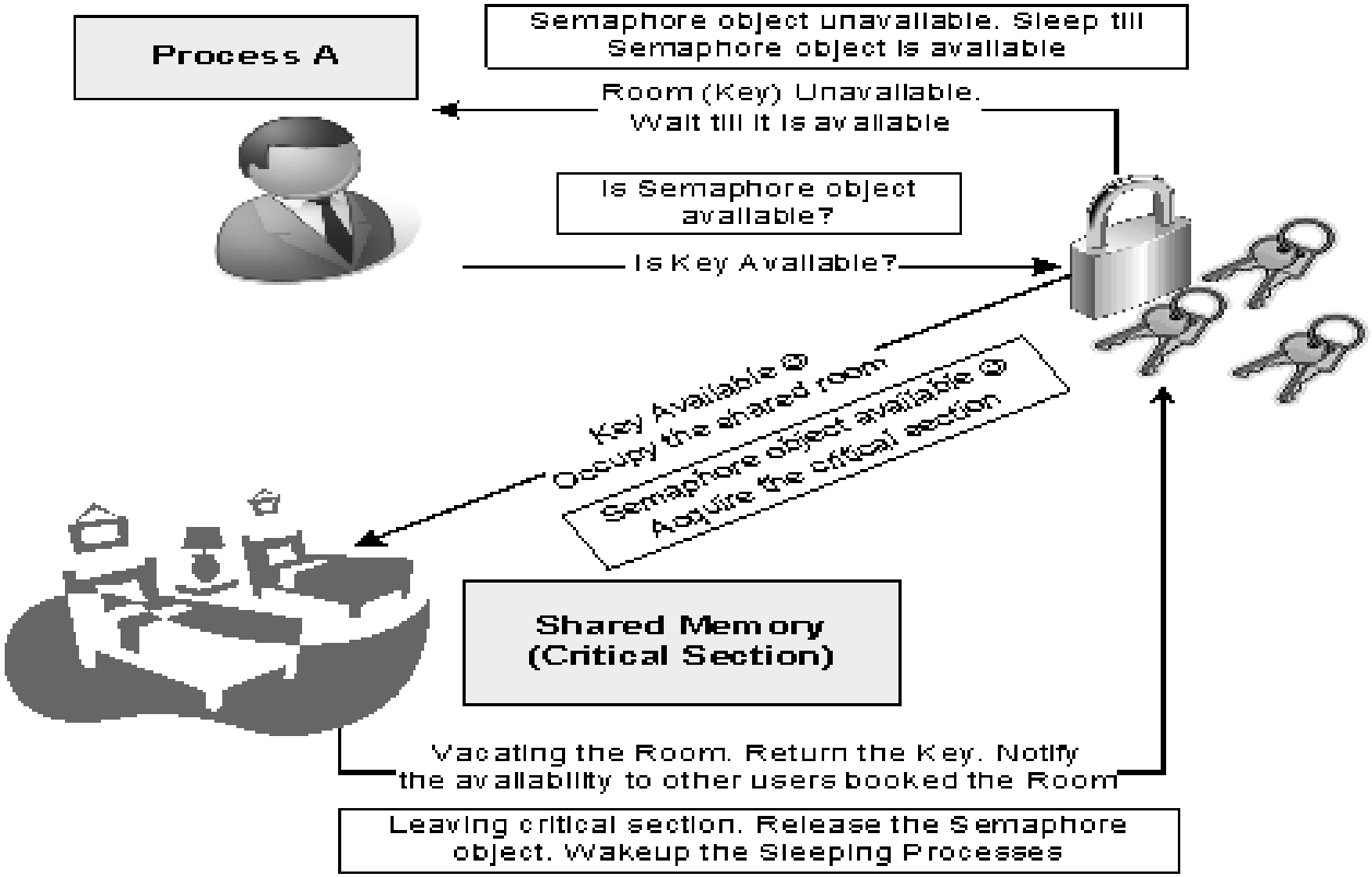
- Ensure that resource preemption (resource releasing) is possible at operating system level. This can be achieved by implementing the following set of rules/ guidelines in resources allocation and releasing
- Release all the resources currently held by a process if a request made by the process for a new resource is not able to fulfill immediately.
- Add the resources which are preempted (released) to a resource list describing the resources which the process requires to complete its execution.
- Reschedule the process for execution only when the process gets its old resources and the new resource which is requested by the process.

- **Explain the different Task Synchronization techniques.**
- **Mutual Exclusion(Through Busy waiting/Spin Lock)**
- The act of preventing the access of a shared resource by a task/process when it is currently held by another task/process.
- Mutual Exclusion can be implemented through Busy waiting (Spin Lock)or Sleep and Wake Up Technique.
- **Busy waiting (Spin Lock)**
- This Technique uses a Lock variable for implementing Mutual exclusion.
- Before Accessing the Shared Resources, Each process must use a variable called lock Variable.
- Lock variable must be set to “1”, if any Processes using the resource ,otherwise it is set to “0”.
- **Sleep and Wake Up Technique**
- When a process/task is not allowed to access a shared resources, which is currently held by other process, then that process undergoes ”sleep” and enters blocked state. Later it gets a wakeup message from the process that has already accessed the Shared resources .

- **Explain the Concept of Semaphore in Detail.**
- **Semaphore:** Semaphore is a sleep and wakeup based mutual exclusion implementation for shared resource access.
- Semaphore is a system resource; and a process which wants to access the shared resource can first acquire this system object to indicate the other processes which wants the shared resource that the shared resource is currently in use by it.
- Based on the implementation, Semaphores can be classified into Binary Semaphore and Counting Semaphore.
- **Binary Semaphore:** This is also known as Mutex lock. It can have only two values – 0 and 1. Its value is initialized to 1. In this type of semaphore, the wait operation works only if semaphore = 1, and the signal operation succeeds when semaphore= 0.
- **Counting Semaphore:** Maintains a count between zero and a maximum value. It limits the usage of resource by a fixed number of processes/ threads.



The concept of Binary Semaphore



The concept of Counting Semaphore

- **Explain the Functional and Non Functional requirements that needs to be evaluated in the selection of RTOS.**
- These **requirements** can be either functional or non-functional
- **Functional requirements**
- Processor Support: It is essential to ensure the processor support by the RTOS.

- Memory Requirements: Since embedded systems are memory constrained, it is essential to evaluate the minimal ROM and RAM requirements for the OS under consideration.

- Real-time Capabilities: It is not mandatory that the operating system for all embedded systems need to be Real-time and all embedded Operating systems-are 'Real-time' in behavior

- Kernel and Interrupt Latency: For an embedded system whose response requirements are high, this latency should be minimal.

- Inter Process Communication and Task Synchronization: The implementation of Inter Process Communication and Synchronization is OS kernel dependent.

- Modularization Support: It is very useful if the OS supports modularization where in which the developer can choose the essential modules and re-compile the OS

- Support for Networking and Communication: Ensure that the OS under consideration provides support for all the interfaces required by the embedded product.

- Development Language Support: It is also necessary that RTOS must support different programming languages

❑ Non Functional requirements

❑ Custom Developed or Off the Shelf: Depending on the OS requirement, it is possible to go for the complete development of an operating system suiting the embedded system needs or use an off the shelf, readily available operating system, which is either a commercial product or an Open Source product, which is in close match with the system requirements.

❑ Cost: This is the total cost for developing or buying the OS and maintaining it in terms of commercial product

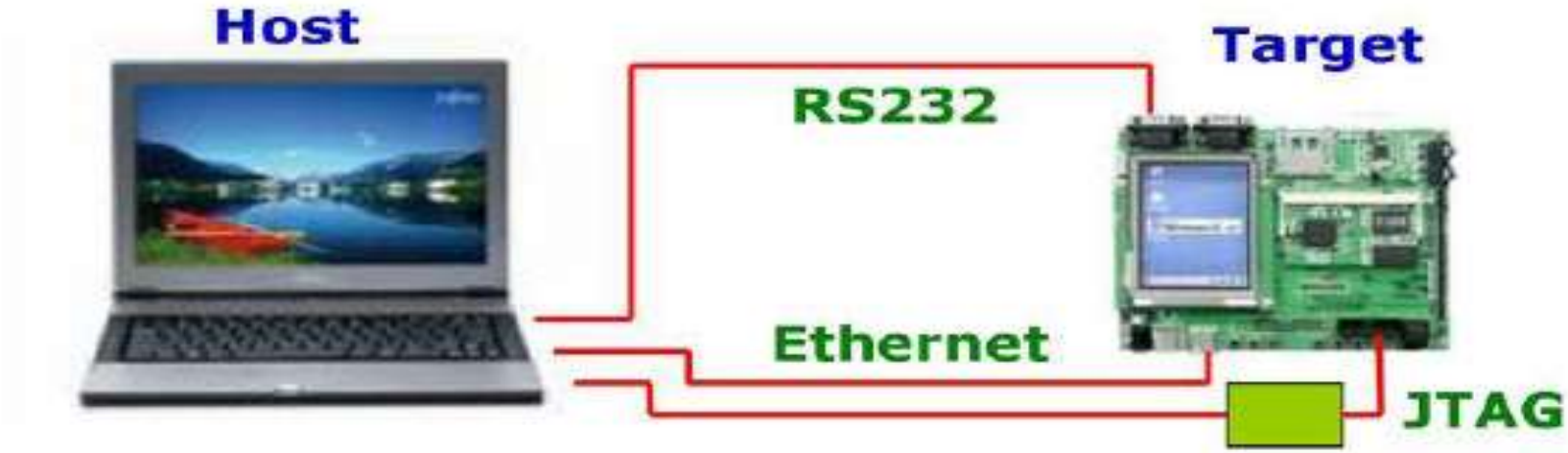
❑ Development and Debugging Tools Availability: The availability of development and debugging tools is a critical decision making factor in the selection of an OS for embedded design.

❑ Ease of Use: How easy it is to use a commercial RTOS is another important feature that needs to be considered in the RTOS selection.

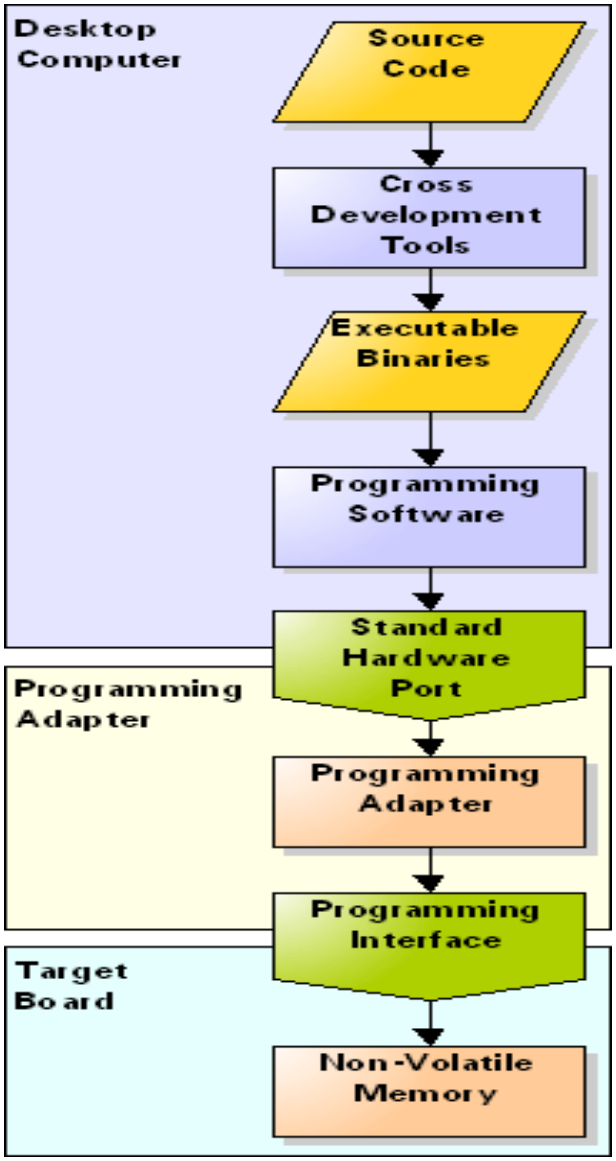
❑ After Sales: For a commercial embedded RTOS, After Sales information must be analyzed thoroughly.

Embedded System Development Environment

- Target Device
  - Embedded System Target Board
- Host System
  - Cross-Development Toolchain
- Connections for Host and Target
  - JTAG: Parallel port for download bootloader
  - UART: Serial port for target terminal (minicom)
  - Ethernet: tftp & NFS services protocols



Embedded System Development Environment

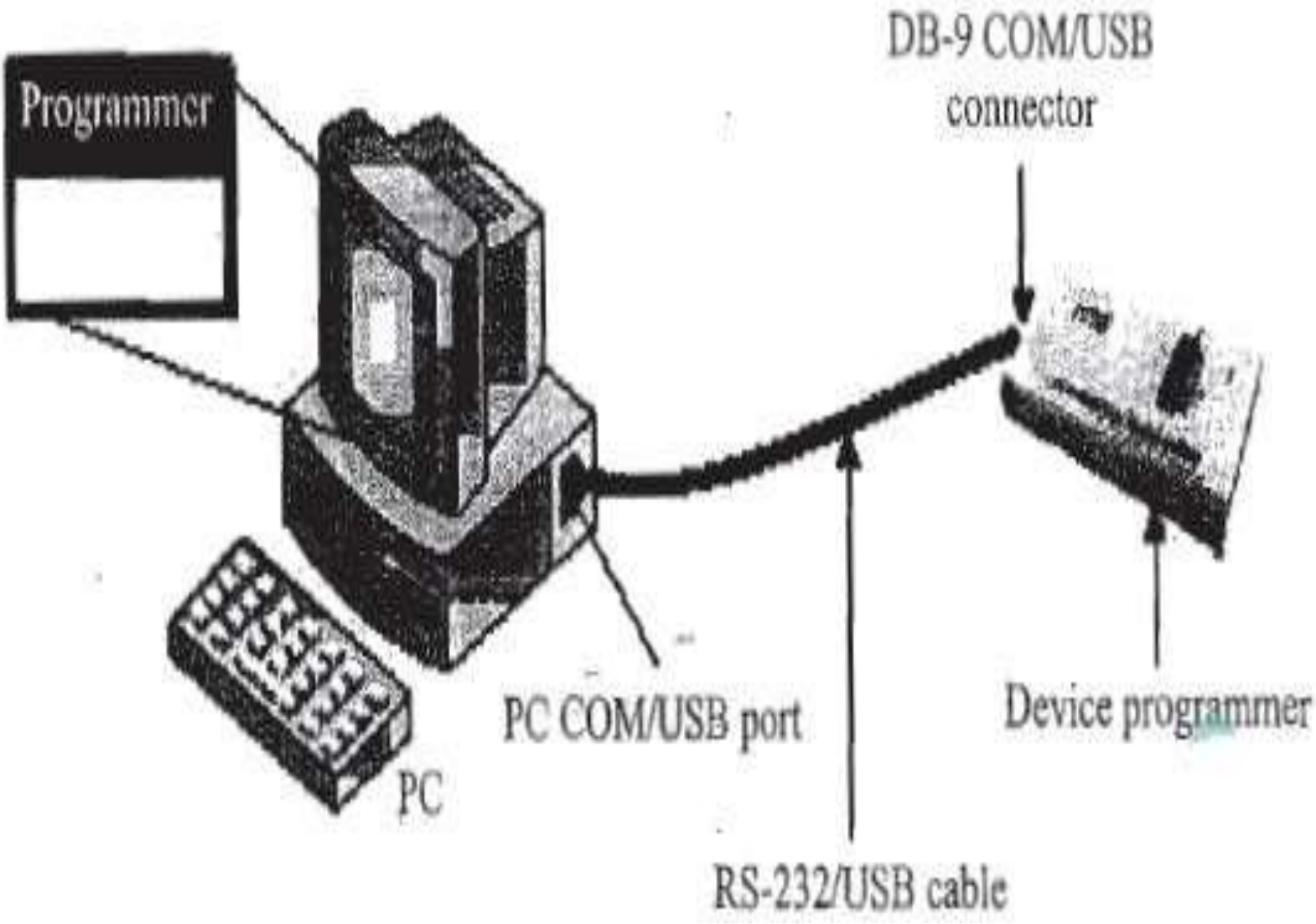


- **Explain the different techniques for embedding the firmware into target board for non OS Embedded system.**

- **Out-of-circuit programming:** It is performed outside the target board. The processor or memory chip into which the firmware needs to be embedded is taken out of the target board and it is programmed with the help of a programming device.

- The sequence of operations for embedding the firmware is listed below
  - Connect the programming device to the specified port of PC
  - Power up the device
  - Execute the programming utility on the PC and ensure proper connectivity is established between PC and programmer.
  - Unlock the ZIF socket by turning the lock pin
  - Insert the device to be programmed into the open socket
  - Lock the ZIF socket
  - Select the device name from the list of supported devices
  - Load the hex file which is to be embedded into the device

- Program the device by 'Program' option of utility program
- Wait till the completion of programming operation
- Ensure that programming is success by checking the status LED
- Unlock the ZIF socket and take the device out of programmer.
- The major drawback of out-of-circuit programming is the high development time.


DB-9 COM/USB
connector
Programmer
PC COM/USB port
Device programmer
PC
RS-232/USB cable

- **In System Programming (ISP):** With ISP, programming is done 'within the system', meaning the firmware is embedded into the target device without removing it from the target board. It is the most flexible and easy way of firmware embedding. The only pre- requisite is that the target device must have an ISP support.
- Implemented with SPI Protocol, SPI (Serial Peripheral Interface) is an interface bus commonly used for communication with flash memory
- The primary I/O lines involved in SPI-In System Programming are listed below
  - MOSI – Master Out Slave In
  - MISO – Master In Slave Out
  - SCK – System Clock
  - RST – Reset of Target Device
  - GND – Ground of Target Device
- PC acts as the master and target device acts as the slave in ISP. The program data is sent to the MOSI pin of target device and the device acknowledgement is originated from the MISO pin of the device. SCK pin acts as the clock for data transfer.

- **In Application Programming (IAP)**
- In Application Programming is a technique used by the firmware running on the target device for modifying a selected portion of the code memory.
- It is not a technique for first time embedding of user written firmware. It modifies the program code memory under the control of the embedded application.
- Updating calibration data, look-up tables, etc., which are stored in code memory, are typical examples of IAP.
- **Use of Factory Programmed Chip**
- It is possible to embed the firmware into the target processor/ controller memory at the time of chip fabrication itself. Such chips are known as 'Factory Programmed Chips'.
- Once the firmware design is over and the firmware achieved operational stability, the firmware files can be sent to the chip fabricator to embed it into the code memory.
- **Firmware Loading for Operating System Based Devices:** The OS based embedded systems are programmed using the In System Programming (ISP) technique. OS based embedded systems contain a special piece of code called 'Boot loader' program which takes control of the OS and application firmware embedding and copying of the OS image to the RAM of the system for execution.

❑ What is Board power up?

❑ Once the firmware is embedded into the target board using one of the programming techniques, then power up the board.

❑ What is BootROM?

❑ A program which contains libraries for implementing In system Programming which is embedded into the memory at the time of chip fabrication.

❑ What is Bootloader?

❑ Program which takes control of the OS and application firmware embedding and copying of the OS image to the RAM.

- **Explain IDE with respect to Embedded System.**
- In embedded system development context, Integrated Development Environment (IDE) stands for an integrated environment for developing and debugging the target processor specific embedded firmware.
- IDE is a software package which bundles
  - a "Text Editor (Source Code Editor)",
  - "Cross-complier (for cross platform development and complier for same platform development)",
  - "Linker", and
  - a "Debugger".
- Some IDEs may provide
  - interface to target board emulators,
  - target processor's/ controller's Flash memory programmer, etc.
  - IDE may be command line based or GUI based.
- **NOTE:** The Keil µVision IDE is an Example

- **What are Disassembler and Decompiler?**
- **Disassembler:** It is a utility program which converts machine codes into target processor specific Assembly codes/ instructions.
- **Decompiler:** It is the utility program for translating machine codes into corresponding high level language instructions.
- **Write a Short Notes on**

(i)Simulators: Simulator is a software tool used for simulating the various conditions for checking the functionality of the application firmware.

Features:

- Purely software based
- Doesn't require a real target system
- Very primitive(Lack of I/O Support)
- Lack of Real-time behavior.

Advantages:

- Simulator based debugging techniques are simple and straightforward
- No Need for Original Target Board
- Simulate I/O Peripherals
- Simulates Abnormal Conditions

Limitations: Deviation from Real Behavior and Lack of Real Timeliness

(ii) Emulator: An emulator is a piece of hardware that ideally behaves exactly like the real microcontroller chip with all its integrated functionality

Features:

- It is the most powerful debugging tool of all
- A microcontroller's functions are emulated in real-time and non-intrusively
- The emulator controls the logic
- The Emulator controls Memory

(iii) Debugger : Debugging in embedded application is the process of diagnosing the firmware execution, monitoring the target processor's registers and memory, and checking the signals from various buses of the embedded hardware.

- Debugging process in embedded application is broadly classified into two, namely; hardware debugging and firmware debugging.
- **Hardware debugging** deals with the monitoring of various bus signals and checking the status lines of the target hardware.
- **Firmware debugging** deals with examining the firmware execution, execution flow, changes to various CPU registers and status registers.

❑ **What are the different technique available for embedded firmware debugging? Explain.**

(i) Incremental EEPROM Burning Technique: This is the most primitive type of firmware debugging technique where the code is separated into different functional code units. Instead of burning the entire code into the EEPROM chip at once, the code is burned in incremental order, where the code corresponding to all functionalities are separately coded, cross-complied and burned into the chip one by one.

(ii) Inline Breakpoint Based Firmware Debugging: Inline Breakpoint Based Firmware Debugging is another primitive method of debugging. Within the firmware where you want to ensure that the firmware execution is reaching up to specified point, insert an inline debug code immediately after the point.

(iii) Monitor program based firmware debugging: It is invasive method of firmware debugging. In this approach a monitor program which acts as a supervisor is developed. The monitor program controls the downloading of user code into code memory, inspects and modifies register/memory locations, allows single stepping of source code etc.

(iv) In Circuit Emulator (Ice) Based Firmware Debugging: 'Emulator' is a self-contained Hardware device which emulates (copy of) the target CPU.

(v) On Chip Firmware Debugging (OCD): Today almost all processor/controllers incorporate built in debug modules called On chip Debug (OCD) support.

- **Explain the different tools for hardware Debugging**
- **Magnifying Glass (Lens):**Similar to a watch repairer, magnifying glass is the primary hardware debugging tool for an embedded hardware debugging professional. A magnifying glass is a powerful visual inspection tool.
- **Multimeter:** A multimeter is used for measuring various electrical quantities like voltage (Both AC and DC), current (DC as well as AC), resistance, capacitance, continuity checking, transistor checking, cathode and anode identification of diode, etc.
- **Digital CRO:** Cathode Ray Oscilloscope (CRO) is a little more sophisticated tool compared to a multimeter. CRO is used for waveform capturing and analysis, measurement of signal strength, etc.
- **Logic Analyzer:** A logic analyzer is similar to digital CRO. Logic analyzer is used for capturing digital data (logic 1 and 0) from a digital circuitry whereas CRO is employed in capturing all kinds of waves including logic signals.
- **Function Generator:** A function generator is capable of producing various periodic waveforms.

- **Explain the concept of Boundary Scan.**
- As the complexity of the hardware increase, the number of chips present in the board and the interconnection among them may also increase.
- Boundary scan is a method for testing interconnectivity on printed circuit boards or sub-blocks and paths inside an integrated circuit.
- BSDL boundary Scan description language is used.
- BSDL is a subset of VHDL and describes JTAG implementation.