UNIT -3 RTOS AND SCHEDULING

Introduction OF RTOS and Scheduling

An operating system acts as an intermediary between the user of a computer and computer hardware. In short, it's an interface between computer hardware and user**.**

Input / output devices to different processes that need the resources. assignment of resources has to be fair and secure.

- **Functionalities of Operating System**
- **Resource Management**
- **Process Management**
- **Storage Management**
- **Memory Management**
- **Security/Privacy Management**

Resource Management: When multiple processes run on the system and need different resources like memory, input/output devices, the OS works as Resource Manager, its responsibility is to provide hardware to the user. It decreases the load in the system.

Process Management: It includes various tasks like scheduling and synchronization of processes. Process scheduling is done with the help of CPU Scheduling algorithms. Process Synchronization is mainly required because processes need to communicate with each other. When processes communicate different problems arise like two processes can update the same memory location in incorrect order.

Storage Management: The file system mechanism used for the management of the secondary storage like Hard Disk. **NIFS, CIFS**, **CFS**, **NFS**, etc. are some file systems that are used by operating systems to manage the storage. All the data is stored in various tracks of Hard disks that are all managed by the storage

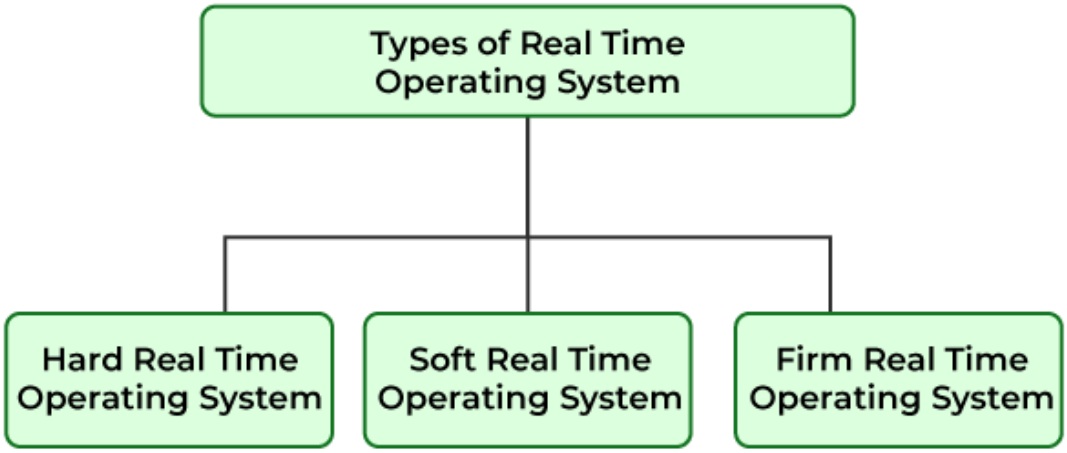
Memory Management: Refers to the management of primary memory, mainly allocation and de-allocation of memory to processes. The operating system has to keep track of how much memory has been used

and by which process. It has to decide which process needs memory space and how much.

Security/Privacy Management: Privacy is also provided by the Operating system using passwords so that unauthorized applications can't access programs or data. For example, Windows uses **Kerberos** authentication to prevent unauthorized access to data.

Types of Real-Time Operating System

The real-time operating systems can be of 3 types



Hard Real-Time Operating System
These operating systems guarantee that critical tasks are completed within a range of time. For example, a robot is hired to weld a car body. If the robot welds too early or too late, the car cannot be sold, so it is a hard real-time system that requires complete car welding by the robot hardly on time., scientific experiments, medical imaging systems, 1

- Defense systems likeRADAR .
- Air traffic control system.
- Networked multimedia systems.
- Medical devices like pacemakers**.**
- Stock trading applications.

Advantages
The advantages of real-time operating systems are as follows:

Maximum Consumption: Maximum utilization of devices and systems. Thus, more output from all the resources.

Task Shifting: Time assigned for shifting tasks in these systems is very less. For example, in older systems, it takes about 10 microseconds.

Focus On Application: Focus on running applications and less importance to applications that are in the queue.

Real-Time Operating System In Embedded System: Since the size of programs is small, RTOS can also be embedded systems like in transport and others.

Error Free: These types of systems are error-free.

Memory Allocation: Memory allocation is best managed in these types of systems.

Disadvantages

Limited Tasks: Very few tasks run simultaneously, and their concentration is very less on few applications to avoid errors.

Use Heavy System Resources: Sometimes the system resources are not so good and they are expensive as well.

Complex Algorithms: The algorithms are very complex and difficult for the designer to write on

Thread Priority: It is not good to set thread priority as these systems are very less prone to switching tasks

Minimum Switching: RTOS performs minimal task switching

Tasks in Real Time systems

The system is subjected to real-time, i.e. response should be guaranteed within a specified timing constraint or system should meet the specified deadline. For example flight control systems, real-time monitors, etc

There are four types of tasks in real-time systems:

1. Periodic tasks
2. Dynamic tasks
3. Critical task

4. Non-critical task

Periodic Tasks: In periodic tasks, jobs are released at regular intervals. A periodic task is one that repeats itself after a fixed time interval. A periodic task is denoted by four tuples: **Ti = < Φi, Pi, ei, Di >** Where,

- **Φi** - is the phase of the task. Phase is the release time of the first job in the task. If the phase is not mentioned then the release time of the first job is assumed to be zero.
- **Pi** - is the period of the task i.e. the time interval between the release times of two consecutive jobs.
- **ei** - is the execution time of the task.
- **Di** - is the relative deadline of the tas

Dynamic Tasks: It is a sequential program that is invoked by the occurrence of an event. An event may be generated by the processes external to the system or by processes internal to the system. Dynamically arriving tasks can be categorized on their criticality and knowledge about their occurrence times.

Aperiodic Tasks: In this type of task, jobs are released at arbitrary time intervals i.e. randomly. Aperiodic tasks have soft deadlines or no deadlines.

Sporadic Tasks: They are similar to aperiodic tasks i.e. they repeat at random instances. The only difference is that sporadic tasks have hard deadlines. A sporadic task is denoted by three tuples: **Ti = (Ei, Gi, Di)**

Ei -the execution time of the tasks

Gi - the minimum separation between the occurrence of two consecutive instances of task.

Di - the relative deadline of the task.

Critical Tasks: Critical Tasks are those who's opportune executions are basic. In the event that cutoff times are missed, calamities happen.

For instance, life-emotionally supportive networks and the steadiness control of airplane.

Non-critical Tasks:

Non-critical Tasks are genuine times assignments. As the name infers, they are not basic to the application. Be that as it may, they can manage time, changing information, and consequently they are futile on the off chance that not finished inside a cutoff time. The objective of booking these assignments is to expand the level of occupations effectively executed inside their cutoff times.

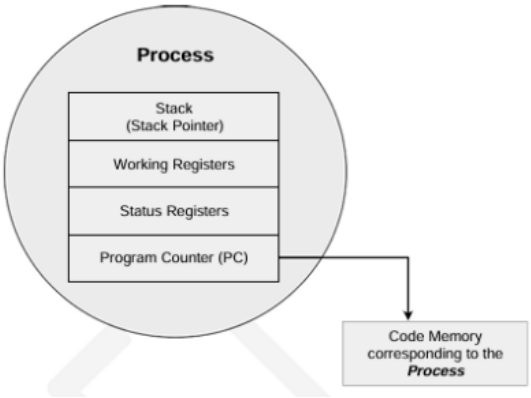
processes and **threads**

Process

A **process** is an instance of a computer program being executed. It's more than just the program code; it's an active entity that includes:

- **Program Code (Text Section):** The executable instructions.
- **Data Section:** Global and static variables.
- **Heap:** Dynamically allocated memory during runtime.
- Stack: Temporary data like function parameters, return addresses, and local variables.
- **Process Control Block (PCB):** A data structure maintained by the operating system that contains information about the process, such as its state (new, ready, running, waiting, terminated), process ID (PID), program counter, CPU registers, CPU scheduling information, memory management information, and I/O status information.

A process, which inherits all the properties of the CPU, can be Considered as a virtual processor, awaiting its turn to have its properties switched into the physical processor



Thread

A **thread** (often called a "lightweight process") is a single sequential flow of execution within a process. A process can have

one or more threads. When a process has multiple threads, it's called a multi-threaded process.

• A thread is the primitive that can execute code

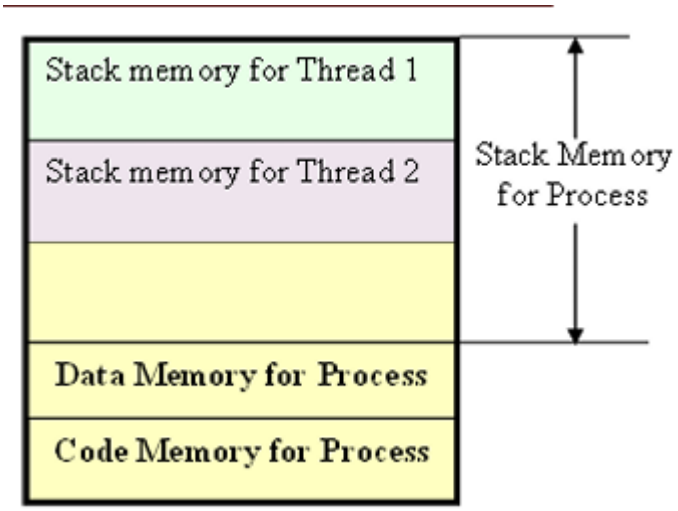
• A thread is a single sequential flow of control within a process

• ‘Thread’ is also known as lightweight process

• A process can have many threads of execution

• Different threads, which are part of a process, share the same address space; meaning they share the data memory, code memory and heap memory area

• Threads maintain their own thread status (CPU register values), Program Counter



Thread v/s process

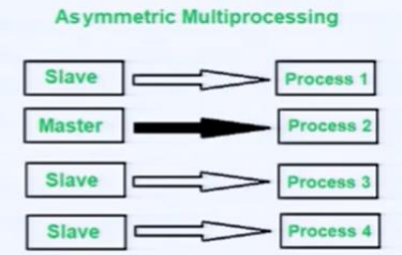
Thread	Process
Thread is a single unit of execution and is part of process	Process is a program in execution and contains one or more thread
A thread does not have its own data memory and heap memory. It shares the data memory and heap memory with other threads of the same process	Process has its own code memory, data memory and stack memory.
A thread cannot live independently; it lives within the process.	A process contains at least one thread
There can be multiple threads in a process. The first thread (main thread) calls the main function and occupies the start of the stack memory of the process	Threads within a process share the code, data and heap memory. Each thread holds separate memory area for stack (share the total stack memory of the process).
Threads are very inexpensive to create	Processes are very expensive to create. Involves many OSS overhead
Context switching is inexpensive and fast	Context switching is complex and involves lot of oss 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.

Multiprocessing and multitasking

- The ability to execute multiple processes simultaneously is referred as multiprocessing
- system capable of performing multiple central processing units (CPUs) and can execute multiple processes simultaneously or simply we can state that Multiprocessing is the use of two or more central processing units within a single computer system

Asymmetric Multiprocessing

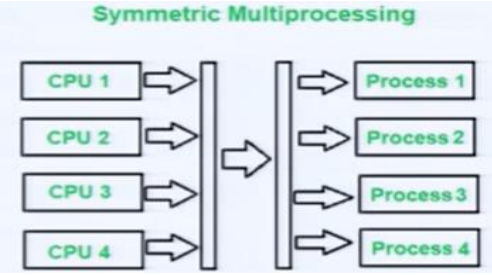
Asymmetric Multiprocessing system is a multiprocessing computer system where not all of the multiple interconnected central processing units (CPUs) are treated equally



In asymmetric multiprocessing only a master processor runs the tasks of the operating system

Symmetric Multiprocessing

Symmetric Multiprocessing involves a multiprocessing computer Hardware and software architecture where two or more identical Processors are connected to a single, shared main memory, have full



Access to all input and output devices
In other words, symmetric Multiprocessing is a type of multiprocessing Where each processor is self-scheduling.
For example, SMP applies multiple processors to that one problem, known as
Parallel programming

Multitasking

The kernel is the core component within an operating system. Operating systems such
As Linux employ kernels that allow users access to the computer seemingly simultaneously

Multiple users can execute multiple programs apparently concurrently. Each executing program is a task (or thread) under control of the operating system

If an operating system can execute multiple tasks in this manner it is said to be multitasking

Thus, multitasking refers to the ability of an OS to hold multiple tasks in memory and switch the processor (CPUs)from executing one tasks to another tasks

Multitasking involves ‘Contesting switching', 'Contesting saving’ and ‘Contesting retrieval Context switching refers to the switching of executing context from task to other

When a task 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

Processes, which is interrupted currently due to execution switching

During context switching, the context of the task to be executed is retrieved from the saved context list. This is known as Context retrieval

Types of multitasking

The various job can be accepted from same user or different users. There are2 types of multitasking systems

1.Single User Multitasking

2.Multi User multitasking

Depending on how the task/processes executing switching act is implemented, multitasking is classified into;

1.Co-operating Multitasking

2. Non-pre-emptive Multitasking

3. Preemptive Multitasking

1.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. If the currently executing task is non-cooperative, the other tasks may have to wait for a long time to get the CPU

2.**Preemptive Multitasking**: Pre empties 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. The preemption of task may be based on time slots or task/process priority

3.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.

Task Scheduling:

• In a multitasking system, there should be some mechanism in place to shathe CPU among the different tasks and to decide which process/task is to be executed at a given point of time

• Determining which task/process is to be executed at a given point of time is known as task/process scheduling

• Task scheduling forms the basis of multitasking

• Scheduling policies forms the guidelines for determining which task is to be executed when

• The scheduling policies are implemented in an algorithm and it is run by the kernel as a service

• The kernel service/application, which implements the scheduling algorithm, is known as 'Scheduler'

• The task scheduling policy can be pre-emptive, non-preemptive or co-operative

• Depending on the scheduling policy the process scheduling decision may take place when a process switches its state to

- ➢ 'Ready' state from 'Running' state
- ➢ 'Blocked/Wait' state from 'Running' state
- ➢ 'Ready' state from 'Blocked/Wait' state

Non-pre-emptive scheduling – First Come First Served (FCFS)/First in First Out (FIFO) Scheduling:

Non-preemptive is a CPU scheduling method where, once a process is allocated the CPU, it retains control until it either voluntarily releases it (by completing its task or entering a waiting state) or terminates

A priority, which is unique or same is associated with each task

• The priority of a task is expressed in different ways, like a priority number, the time required to complete the execution etc.

• In number-based priority assignment the priority is a number ranging from 0 to the maximum priority supported by the OS. The maximum level of priority is OS dependent.

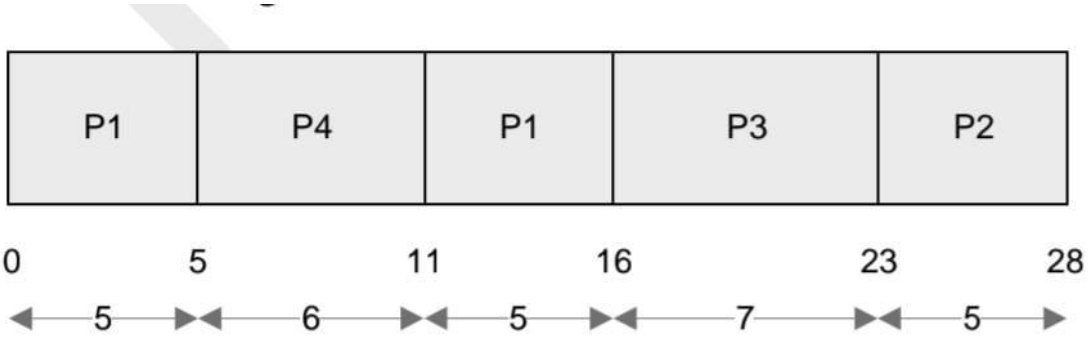
• Windows CE supports 256 levels of priority (0 to 255 priority numbers, with 0 being the highest priority)

The priority is assigned to the task on creating it. It can also be changed dynamically (If the Operating System supports this feature)

• The non-preemptive priority-based scheduler sorts the 'Ready' queue based on the priority and picks the process with the highest level of priority for execution

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 (Summiteer is now waiting for the processes) in priority-based scheduling algorithm.

Solution: The scheduler sorts the 'Ready' queue based on the priority and schedules the process with the highest priority (P1 with priority number 0) first and the next high priority process (P3 with priority number 2) as second and so on. The order in which the processes are scheduled for execution is represented as



The waiting time for all the processes are given as

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

Drawbacks:

➢ Similar to SJF scheduling algorithm, non-preemptive priority-based algorithm also possess the drawback of 'Starvation' where a process whose priority is low may not get a chance to execute if more and more processes with higher priorities enter the 'Ready' queue before the process with lower priority starts its execution.

➢ 'Starvation' can be effectively tackled in priority based non-preemptive scheduling by dynamically raising the priority of the low priority task/process which is under starvation (waiting in the ready queue for a longer time for getting the CPU time)

➢ The technique of gradually raising the priority of processes which are waiting in the 'Ready' queue as time progresses, for preventing 'Starvation', is known as 'Aging'

Preemptive scheduling:

Employed in systems, which implements preemptive multitasking model

• Every task in the 'Ready' queue gets a chance to execute. When and how often each process gets a chance to execute (gets the CPU time) is

dependent on the type of preemptive scheduling algorithm used for scheduling the processes

• The scheduler can pre-empt (stop temporarily) the currently executing task/process and select another task from the 'Ready' queue for execution

• When to pre-empt a task and which task is to be picked up from the 'Ready' queue for execution after preempting the current task is purely dependent on the scheduling algorithm Time (SRT)

• A task which is preempted by the scheduler is moved to the 'Ready' queue. The act of moving a 'Running' process/task into the 'Ready' queue by the scheduler, without the processes requesting for it is known as 'Preemption'

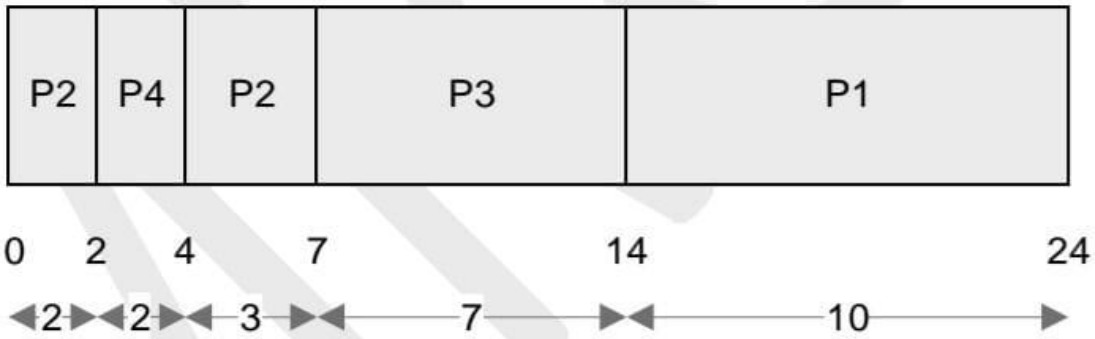
• Time-based preemption and priority-based preemption are the two important approaches adopted in preemptive scheduling

Preemptive scheduling – Preemptive SJF Scheduling/ Shortest Remaining

• The non preemptive SJF scheduling algorithm sorts the 'Ready' queue only after the current process completes execution or enters wait state, whereas the preemptive SJF scheduling algorithm sorts the 'Ready' queue when a new process enters the 'Ready' queue and checks whether the execution time of the new process is shorter than the remaining of the total estimated execution time of the currently executing process
• If the execution time of the new process is less, the currently executing process is preempted and the new process is scheduled for execution

• Always compares the execution completion time (i.e. the remaining execution time for the new process) of a new process entered the 'Ready' queue with the remaining time for completion of the currently executing process and schedules the process with shortest remaining time for execution.
EXAMPLE: Three processes with process IDs P1, P2, P3 with estimated completion time 10, 5, 7 milliseconds respectively enter 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.

Solution: At the beginning, there are only three processes (P1, P2 and P3) available in the 'Ready' queue and the SRT scheduler picks up the process with the Shortest remaining time for execution completion (In this example P2 with remaining time 5ms) for scheduling. Now process P4 with estimated execution completion time 2ms enters the 'Ready' queue after 2ms of start of execution of P2. The processes are re-scheduled for execution in the following order.



The waiting time for all the processes are given as

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 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 Scheduling: = (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