UNIT I - **INTRODUCTION AND INTERNALS**

Introduction to Real-Time Systems, Classification of real time systems, Difference between GPOS and RTOS- Real Time Kernels - RTOS Architecture- Features of RTOS- POSIX-RT standard

Introduction to Real-Time Systems

Real-time operating system (RTOS) is an operating system intended to serve real time application that process data as it comes in, mostly without buffer delay. The full form of RTOS is Real time operating system. In a RTOS, Processing time requirement are calculated in tenths of seconds increments of time. It is time-bound system that can be defined as fixed time constraints. In this type of system, processing must be done inside the specified constraints. Otherwise, the system will fail.

Usage of RTOS

Here are important reasons for using RTOS:

- **It offers priority-based scheduling, which allows you to separate analytical processing from non-critical processing.**
- **The Real time OS provides API functions that allow cleaner and smaller application code.**
- **Abstracting timing dependencies and the task-based design results in fewer interdependencies between modules.**
- **RTOS offers modular task-based development, which allows modular task-based testing.**
- **The task-based API encourages modular development as a task, will typically have a clearly defined role. It allows designers/teams to work independently on their parts of the project.**
- **An RTOS is event-driven with no time wastage on processing time for the event which is not occur**

Components of RTOS

Here, are important Component of RTOS

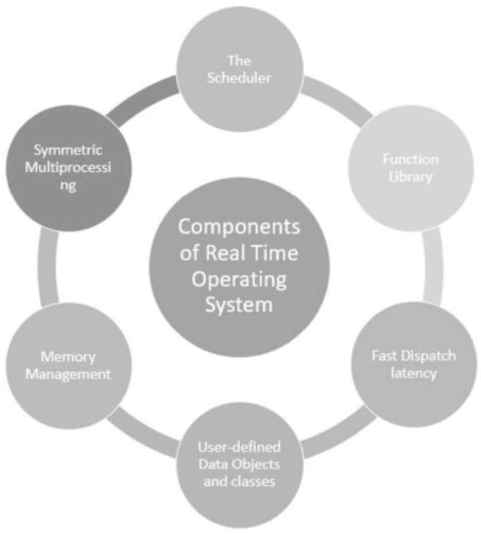
Components of Real Time Operating System
The Scheduler
Function Library
Fast Dispatch latency
User-defined Data Objects and classes
Memory Management
Symmetric Multiprocessing

Fig:1.1: Components of RTOS

The Scheduler: This component of RTOS tells that in which order, the tasks can be executed which is generally based on the priority.

Symmetric Multiprocessing (SMP): It is a number of multiple different tasks that can be handled by the RTOS so that parallel processing can be done.

Function Library: It is an important element of RTOS that acts as an interface that helps you to connect kernel and application code. This application allows you to send the requests to the Kernel using a function library so that the application can give the desired results.

Memory Management: this element is needed in the system to allocate memory to every program, which is the most important element of the RTOS.

Fast dispatch latency: It is an interval between the termination of the task that can be identified by the OS and the actual time taken by the thread, which is in the ready queue, that has started processing.

User-defined data objects and classes: RTOS system makes use of programming languages like C or C++, which should be organized according to their operation.

Types of RTOS

Three types of RTOS systems are:

Hard Real Time :

In Hard RTOS, the deadline is handled very strictly which means that given task must start executing on specified scheduled time, and must be completed within the assigned time duration.

Example: Medical critical care system, Aircraft systems, etc.

Firm Real time:

These type of RTOS also need to follow the deadlines. However, missing a deadline may not have big impact but could cause undesired affects, like a huge reduction in quality of a product.

Example: Various types of Multimedia applications.

Soft Real Time:

Soft Real time RTOS, accepts some delays by the Operating system. In this type of RTOS, there is a deadline assigned for a specific job, but a delay for a small amount of time is acceptable. So, deadlines are handled softly by this type of RTOS.

Example: Online Transaction system and Livestock price quotation System.

Factors for selecting an RTOS

Here, are essential factors that you need to consider for selecting RTOS:

Performance: Performance is the most important factor required to be considered while selecting for a RTOS.

Middleware: if there is no middleware support in Real time operating system, then the issue of time-taken integration of processes occurs.

Error-free: RTOS systems are error-free. Therefore, there is no chance of getting an error while performing the task.

Embedded system usage: Programs of RTOS are of small size. So we widely use RTOS for embedded systems.

Maximum Consumption: we can achieve maximum Consumption with the help of RTOS.

Task shifting: Shifting time of the tasks is very less.

Unique features: A good RTS should be capable, and it has some extra features like how it operates to execute a command, efficient protection of the memory of the system, etc.

24/7 performance: RTOS is ideal for those applications which require to run 24/7.

Here are important differences between GPOS and RTOS:

Table:1.1:Difference between in GPOS and RTOS

General-Purpose Operating System (GPOS)	**Real-Time Operating System (RTOS)**
It used for desktop PC and laptop	**It is only applied to the embedded application.**
Process-based Scheduling	**Time-based scheduling used like round-robin scheduling**
Interrupt latency is not considered as important as in RTOS	**Interrupt lag is minimal, which is measured in a few microseconds.**
No priority inversion mechanism is present in the system.	**The priority inversion mechanism is current. So it cannot modify by the system.**
Kernel's operation may or may not be preempted.	**Kernel's operation can be preempted.**
Priority inversion remain unnoticed	**No predictability guarantees**

Applications of Real Time Operating System

Real-time systems are used in:

- **Airlines reservation system.**
- **Air traffic control system.**
- **Systems that provide immediate updating.**
- **Used in any system that provides up to date and minute information on stock prices.**
- **Defense application systems like RADAR.**
- **Networked Multimedia Systems**
- **Command Control Systems**
- **Internet Telephony**
- **Anti-lock Brake Systems**
- **Heart Pacemaker**

Disadvantages of RTOS

Here, are drawbacks/cons of using RTOS system:

- **RTOS system can run minimal tasks together, and it concentrates only on those applications which contain an error so that it can avoid them.**
- **RTOS is the system that concentrates on a few tasks. Therefore, it is really hard for these systems to do multi-tasking.**
- **Specific drivers are required for the RTOS so that it can offer fast response time to interrupt signals, which helps to maintain its speed.**
- **Plenty of resources are used by RTOS, which makes this system expensive.**
- **The tasks which have a low priority need to wait for a long time as the RTOS maintains the accuracy of the program, which are under execution.**
- **Minimum switching of tasks is done in Real time operating systems.**
- **It uses complex algorithms which is difficult to understand.**
- **RTOS uses lot of resources, which sometimes not suitable for the system.**

RTOS Architecture – Kernel

RTOS Architecture

For simpler applications, RTOS is usually a kernel but as complexity increases, various modules like networking protocol stacks debugging facilities, device I/Os are includes in addition to the kernel. The general architecture of RTOS is shown in the below fig 1.2

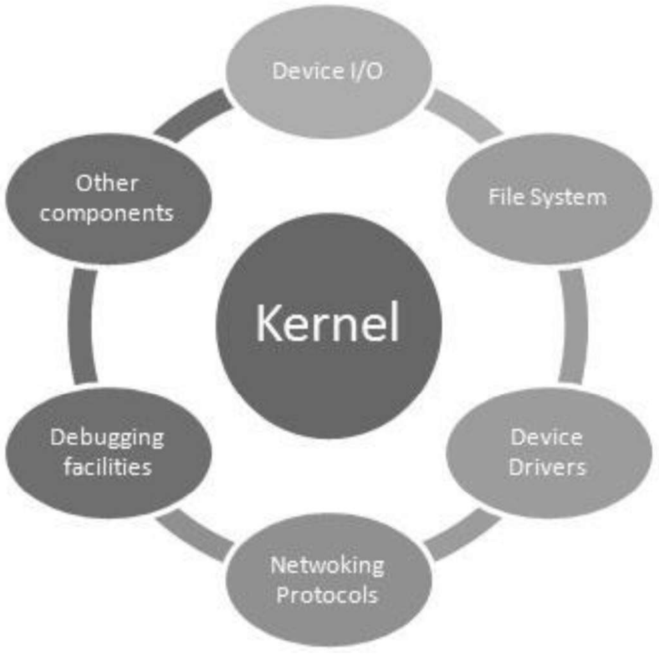

Fig:1.2: **RTOS Architecture**

Kernel

RTOS kernel acts as an abstraction layer between the hardware and the applications. There are three broad categories of kernels

·

Monolithic kernel

Monolithic kernels are part of Unix-like operating systems like Linux, FreeBSD etc. A monolithic kernel is one single program that contains all of the code necessary to perform every kernel related task. It runs all basic system services (i.e. process and memory management, interrupt handling and I/O communication, file system, etc) and provides powerful abstractions of the underlying hardware. Amount of context switches and messaging involved are greatly reduced which makes it run faster than microkernel.

· **Microkernel**

It runs only basic process communication (messaging) and I/O control. It normally provides only the minimal services such as managing memory protection, Inter process communication and the process management. The other functions such as running the hardware processes are not handled directly by micro kernels. Thus, micro kernels provide a smaller set of simple hardware abstractions. It is more stable than monolithic as the kernel is unaffected even if the servers failed (i.e., File System). Micro kernels are part of the operating systems like AIX, BeOS, Mach, Mac OS X, MINIX, and QNX. Etc

·

Hybrid Kernel

Hybrid kernels are extensions of micro kernels with some properties of monolithic kernels. Hybrid kernels are similar to micro kernels, except that they include additional code in kernel space so that such code can run more swiftly than it would were it in user space. These are part of the operating systems such as Microsoft Windows NT, 2000 and XP. Dragon Fly BSD, etc

·

Exokernel

Exokernels provides efficient control over hardware. It runs only services protecting the resources (i.e. tracking the ownership, guarding the usage, revoking access to resources, etc) by providing low-level interface for library operating systems and leaving the management to the application.

Six types of common services are shown in the following figure below and explained in subsequent sections

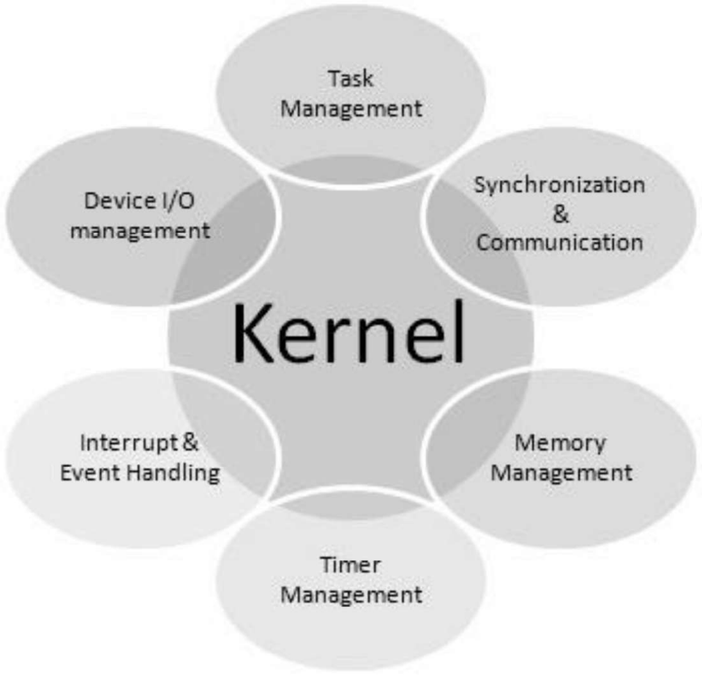

Fig:1.3: **Representation of Common Services Offered By a RTOS System**

Architecture – Task Management

Task Management

In RTOS, The application is decomposed into small, schedulable, and sequential program units known as "Task", a basic unit of execution and is governed by three time-critical properties; release time, deadline and execution time. Release time refers to the point in time from which the task can be executed. Deadline is the point in time by which the task must complete. Execution time denotes the time the task takes to execute.

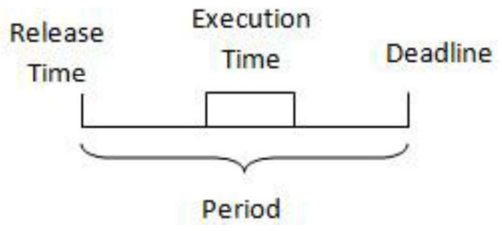

Fig:1.4: Use of RTOS for Time Management Application

Each task may exist in following states

Dormant : Task doesn't require computer time

Ready: Task is ready to go active state, waiting processor time

Active: Task is running

Suspended: Task put on hold temporarily

Pending: Task waiting for resource.

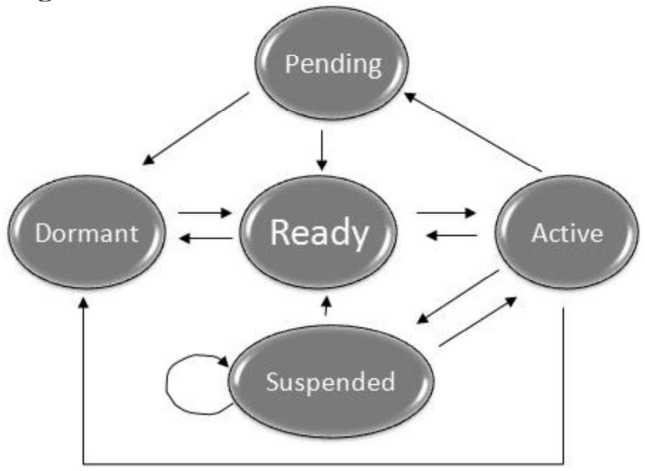

Fig:1.5: **Representation of Different Time Management Tasks Done by an RTOS**

During the execution of an application program, individual tasks are continuously changing from one state to another. However, only one task is in the running mode (i.e. given CPU control) at any point of the execution. In the process where CPU control is change from one task to another, context of the to-be-suspended task will be saved while context of the to-be-executed task will be retrieved, the process referred to as context switching. A task object is defined by the following set of components:

Task Control block: Task uses TCBs to remember its context. TCBs are data structures residing in RAM, accessible only by RTOS

Task Stack: These reside in RAM, accessible by stack pointer.

Task Routine: Program code residing in ROM

Scheduler: The scheduler keeps record of the state of each task and selects from among them that are ready to execute and allocates the CPU to one of them. Various scheduling algorithms are used in RTOS.

Polled Loop: Sequentially determines if specific task requires time.

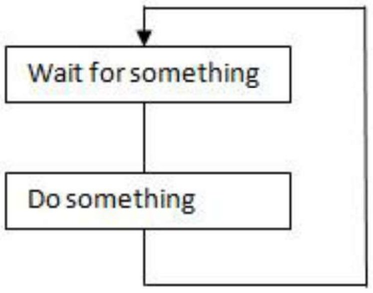

Fig:1.6: Process Flow of a Scheduler

Polled System with interrupts. In addition to polling, it takes care of critical tasks.

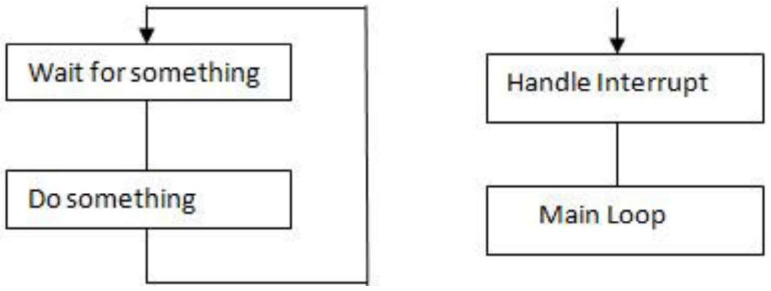

Fig:1.7: A Figure Illustrating Polled Systems with Interrupts

Round Robin : Sequences from task to task, each task getting a slice of time

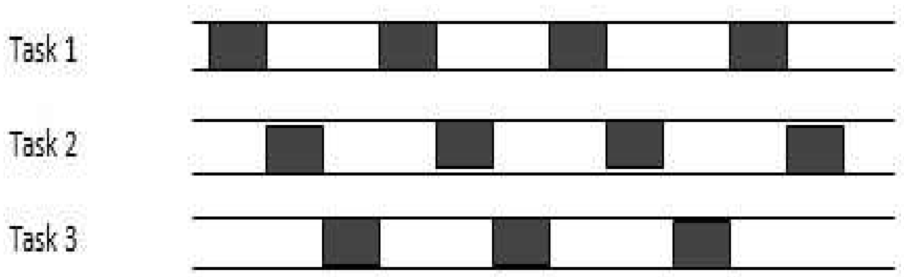

Fig:1.8: Round Robin Sequences From Task to Task

Hybrid System: Sensitive to sensitive interrupts, with Round Robin system working in background

Interrupt Driven: System continuously wait for the interrupts

Non pre-emptive scheduling or Cooperative Multitasking: Highest priority task executes for some time, then relinquishes control, re-enters ready state

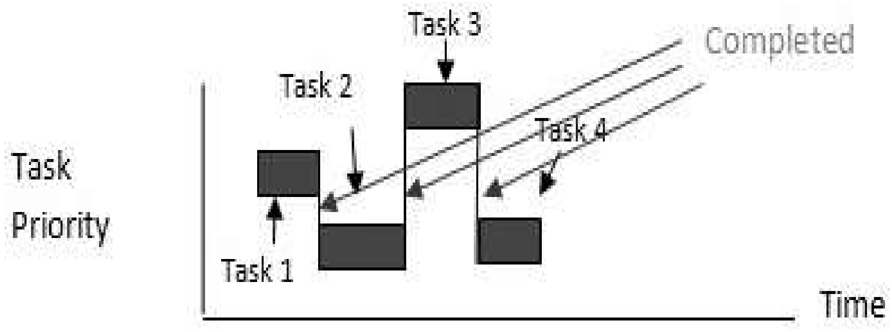

Fig:1.9: Non-Preemptive Scheduling or Cooperative Multitasking

Preemptive scheduling Priority multitasking: Current task is immediately suspended Control is given to the task of the highest priority at all time.

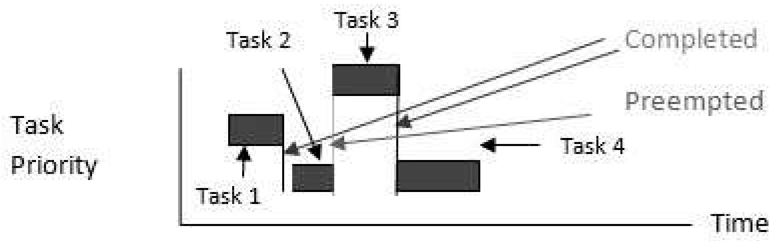

Fig:1.10: Preemptive Scheduling or Priority Multitasking

Dispatcher : The dispatcher gives control of the CPU to the task selected by the scheduler by performing context switching and changes the flow of execution.

Synchronization and communication:

Task Synchronization & inter task communication serves to pass information amongst tasks.

Task Synchronization

Synchronization is essential for tasks to share mutually exclusive resources (devices, buffers, etc) and/or allow multiple concurrent tasks to be executed (e.g. Task A needs a result from task B, so task A can only run till task B produces it.
Task synchronization is achieved using two types of mechanisms:

Event Objects

Event objects are used when task synchronization is required without resource sharing. They allow one or more tasks to keep waiting for a specified event to occur. Event object can exist either in triggered or non-triggered state. Triggered state indicates resumption of the task.

Semaphores.

A semaphore has an associated resource count and a wait queue. The resource count indicates availability of resource. The wait queue manages the tasks waiting for resources from the semaphore. A semaphore functions like a key that define whether a task has the access to the resource. A task gets an access to the resource when it acquires the semaphore.

There are three types of semaphore:

- **Binary Semaphores**
- **Counting Semaphores**
- **Mutually Exclusion (Mutex) Semaphores**

Semaphore functionality (Mutex) represented pictorially in the following figure

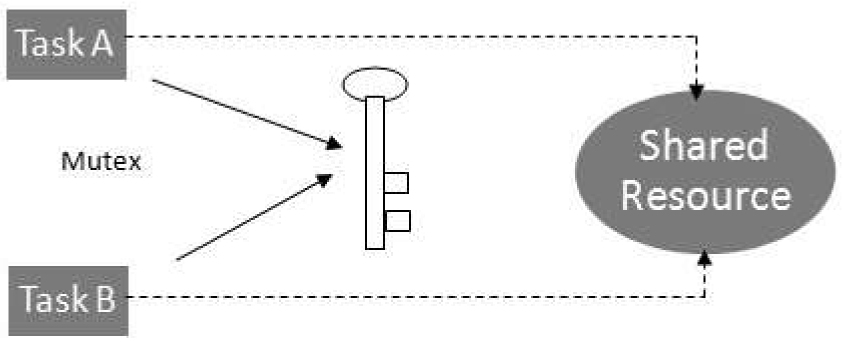

Fig:1.11*:* Architecture of Semaphore Functionality

Inter task communication

Inter task communication involves sharing of data among tasks through sharing of memory space, transmission of data, etc. Inter task communications is executed using following mechanisms

Message queues

A message queue is an object used for inter task communication through which task send or receive messages placed in a shared memory. The queue may follow 1) First In First Out (FIFO), 2) Last in First Out(LIFO) or 3) Priority (PRI) sequence. Usually, a message queue comprises of an associated queue control block (QCB), name, unique ID,

memory buffers, queue length, maximum message length and one or more task waiting lists. A message queue with a length of 1 is commonly known as a mailbox.

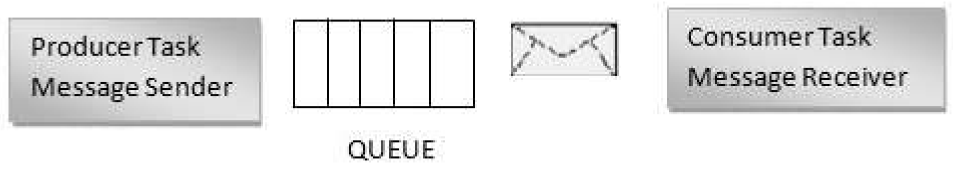

Fig:1.12: Flow of a Message Queue in a Mailbox

Pipes

A pipe is an object that provide simple communication channel used for unstructured data exchange among tasks. A pipe does not store multiple messages but stream of bytes. Also, data flow from a pipe cannot be prioritized.

Remote procedure call (RPC)

It permits distributed computing where task can invoke the execution of another task on a remote computer.

Memory Management

Two types of memory managements are provided in RTOS – Stack and Heap. Stack management is used during context switching for TCBs. Memory other than memory used for program code, program data and system stack is called heap memory and it is used for dynamic allocation of data space for tasks. Management of this memory is called heap management.

Timer Management

Tasks need to be performed after scheduled durations. To keep track of the delays, timers- relative and absolute are provided in RTOS.

Interrupt and event handling

RTOS provides various functions for interrupt and event handling, viz., Defining interrupt handler, creation and deletion of ISR, referencing the state of an ISR, enabling and disabling of an interrupt, etc. It also restricts interrupts from occurring when modifying a data structure, minimize interrupt latencies due to disabling of interrupts when RTOS is performing critical operations, minimizes interrupt response times.

Device I/O Management

RTOS generally provides large number of APIs to support diverse hardware device drivers.

Features of RTOS

Here are important features of RTOS:

- **Occupy very less memory**
- **Consume fewer resources**
- **Response times are highly predictable**
- **Unpredictable environment**
- **The Kernel saves the state of the interrupted task ad then determines which task it should run next.**
- **The Kernel restores the state of the task and passes control of the CPU for that task.**

POSIX (Portable Operating System Interface)

POSIX (Portable Operating System Interface) is a set of standard operating system interfaces based on the Unix operating system. The need for standardization arose because enterprises using computers wanted to be able to develop programs that could be moved among different manufacturer's computer systems without having to be recoded. Unix was selected as the basis for a standard system interface partly because it was "manufacturer-neutral." However, several major versions of Unix existed so there was a need to develop a common denominator system.

Informally, each standard in the POSIX set is defined by a decimal following the POSIX. Thus, POSIX.1 is the standard for an application program interface in the C language. POSIX.2 is the standard shell and utility interface (that is to say, the user's command interface with the operating system). These are the main two interfaces, but additional interfaces, such as POSIX.4 for thread management, have been developed or are being developed. The POSIX interfaces were developed under the auspices of the Institute of Electrical and Electronics Engineers (IEEE).

POSIX.1 and POSIX.2 interfaces are included in a somewhat larger interface known as the X/Open Programming Guide (also known as the "Single UNIX Specification" and "UNIX 03"). The Open Group, an industry standards group, owns the UNIX trademark and can thus "brand" operating systems that conform to the interface as "UNIX" systems. IBM's OS/390 is an example of an operating system that includes a branded UNIX interface.

Terms used in RTOS

Here, are essential terms used in RTOS:

- **Task – A set of related tasks that are jointly able to provide some system functionality.**

- **Job – A job is a small piece of work that can be assigned to a processor, and that may or may not require resources.**
- **Release time of a job – It's a time of a job at which job becomes ready for execution.**
- **Execution time of a job: It is time taken by job to finish its execution.**
- **Deadline of a job: It's time by which a job should finish its execution.**
- **Processors: They are also known as active resources. They are important for the execution of a job.**
- **Maximum it is the allowable response time of a job is called its relative deadline**
- **Response time of a job: It is a length of time from the release time of a job when the instant finishes.**
- **Absolute deadline: This is the relative deadline, which also includes its release time.**

Summary

RTOS is an operating system intended to serve real time application that process data as it comes in, mostly without buffer delay. It offers priority-based scheduling, which allows you to separate analytical processing from non-critical processing. Important components of RTOS system are:

- **The Scheduler**
- **Symmetric Multiprocessing**
- **Function Library**
- **Memory Management**
- **Fast dispatch latency and**
- **User-defined data objects and classes.**

RTOS system occupy very less memory and consume fewer resources. Performance is the most important factor required to be considered while selecting for a RTOS. General-Purpose Operating System (GPOS) is used for desktop PC and laptop while Real-Time Operating System (RTOS) only applied to the embedded application. Real-time systems are used in Airlines reservation system, Air traffic control system, etc. The biggest drawback of RTOS is that the system only concentrates on a few tasks.

TEXT / REFERENCE BOOKS

1. Jane W. S Liu, "Real Time Systems" Pearson Higher Education, 3rd Edition, 2000.
2. Raj Kamal, "Embedded Systems- Architecture, Programming and Design" Tata McGraw Hill, 2nd Edition, 2014.
3. Jean J. Labrosse, "Micro C/OS-I : The real time kernel" CMP Books, 2nd Edition,2015.
5. Richard Barry, "Mastering the Free RTOS: Real Time Kernel", Real Time Engineers Ltd, 1st Edition, 2016.
6. David E. Simon, " An Embedded Software Primer", Pearson Education Asia Publication