**Real Time Operating Systems**

What is RTOS ?

* "RTOS" stands for Real-Time Operating System.
* It is a type of operating system used for real time applications in embedded systems.

How is it different from GPOS ?

RTOS is known for its characteristics that helps is in many applications.

* Reliability :

RTOS provides more reliability as compared to GPOS.

It has more control over events in real time and they are always available to provide service.

Some systems are required to run for a longer period of time without human intervention, for these purposes RTOS can be very useful.

* Determinism:

RTOS entirely functions over deadlines which makes it more efficient.

It means that for each process a specifc deadline or a time-period is specified within which it has to finish that particular process.

* Scheduling:

In this operating system, user has more control over scheduling a particular task or a process depending on its priority.

So we can define the priority for that particular task and also the frequency with which it should occur ( more like a delay).

In GPOS all the scheduling functions are process based and user has less control on them.

Task defined in RTOS are preemptive.

Generally, in an operating systems there are two types of tasks viz. High priority tasks and Low priority tasks.

High priority task can meet their deadlines consistently because of the preemptive property.

* Scalability:

RTOS is used in wide variety of applications in the field of embedded systems.

So it is scalable depending on the application requirements (i.e we can add or remove modular components depending on our use).

Types of tasks in RTOS:

Hard real-time tasks:

These types of tasks strictly run based on deadlines.

If a particular task is not finished within the predetermined deadlines then the system is considered to be a failed system.

Applications : Anti-missile systems, Air bag mechanisms etc.

Firm real-time tasks:

Similar to hard real-time tasks they should also meet the deadlines.

But if they don't meet then that doesn't make this a failed system, but the results that are produced after the deadlines are discarded and the utility of the system becomes zero.

Applications : Multimedia

Soft real-time systems:

Here the deadlines are not expressed as some absolute value but they are expressed as a average response time required by the task.

If the task is finished then the utility of the task is 100%. But if they fail to meet them, then the utility of the system gradually falls depending on the extra time that is taken past the deadline.

The number of resources available in a system are limited and need to be shared across different processes/tasks. For the proper allocation of resources an operation system provides different techniques by which a user can allocate resources or restrict resources for few particular tasks.

**MultiTasking**

Multitasking is running multiple processes at the same time.

In a multi-processor system it implies that each core of processor is executing different tasks i.e. multiple tasks at the same time

Whereas in a single processor system the operating system schedules tasks in such a way that all the tasks are performed simultaneously i.e. each task gets a limited amount of Processor time, after the time expires the running task is suspended and another task is executed, The original task gets the resources again when all the tasks are given equal amount of processor time.

**Semaphore**

“A semaphore is a value in a designated place in operating system (or kernel) storage that each process can check and then change. Depending on the value that is found, the process can use the resource or will find that it is already in use and must wait for some period before trying again.”

Semaphores basically is a variable which indicates number of units of a particular resource that are available. Each semaphore(S) has two associated functions (P, V) which can only change the value of S.

A simple way to understand wait (P) and signal (V) operations is:

* wait: If the value of semaphore variable is not negative, decrements it by 1. If the semaphore variable is now negative, the process executing wait is blocked (i.e., added to the semaphore's queue) until the value is greater or equal to 1. Otherwise, the process continues execution, having used a unit of the resource.
* signal: Increments the value of semaphore variable by 1. After the increment, if the pre-increment value was negative (meaning there are processes waiting for a resource), it transfers a blocked process from the semaphore's waiting queue to the ready queue.

**Mutex**

“Mutex is a property of concurrency control, which is instituted for the purpose of preventing race conditions; it is the requirement that one thread of execution never enter its critical section at the same time that another, concurrent thread of execution enters its own critical section.”

Mutex refers to a variable that tells if a given resource is available or not. These are useful in cases where the resources are required exclusively by a single process. Suppose if there exists two tasks: writing and reading of data from memory. If both try to access memory at the same time there would be loss of data, for this one process should get exclusive access to the Memory after its execution the other task would gain access. It can be said that the 1st process blocks the resource for its exclusive use.

**Difference between Mutex and Semaphore**

1. Mutex is used to gain exclusive access to the resource whereas Semaphore is used to know number of units of a particular resource available.
2. A process which blocks the access to a resource can only unblock it in case of a Mutex.

**Deadlock**

“Deadlock occurs when two competing actions wait for the other to finish, and thus neither ever does.”

Deadlock is a situation when two different processes require resources owned by each other and cannot release the existing resources unless the required resources are obtained.

A deadlock situation can arise if and only if all of the following conditions (known as the Coffman conditions) hold simultaneously in a system

1. Mutual exclusion: at least one resource must be held in a non-shareable mode. Only one process can use the resource at any given instant of time.
2. Hold and wait or resource holding: a process is currently holding at least one resource and requesting additional resources which are being held by other processes.
3. No pre-emption: a resource can be released only voluntarily by the process holding it.
4. Circular wait: a process must be waiting for a resource which is being held by another process, which in turn is waiting for the first process to release the resource. In general, there is a set of waiting processes, P = {P1, P2, …, PN}, such that P1 is waiting for a resource held by P2, P2 is waiting for a resource held by P3 and so on until PN is waiting for a resource held by P1.

Solutions for Deadlock involve finding solution to anyone or all of the aforementioned conditions.

**LIVELOCK**

Livelock is a situation in which no actual job is done but the resources are being used it is the same as a DeadLock, except that processes are not blocked forever, they are being processed constantly by the CPU, but the system is slow because the lock has too much Containment, it is BottleNeck.

FreeRTOS is a set of files which are used which provides API to design and implement semaphore, Mutex, IPC and preventing Deadlock and livelock.

It also provides API’s for memory management S