UNIT-I: REALTIME OPERATING SYSTEMS AND CONCEPTS

Introduction -Operating system (OS) : An Operating system (OS) is a piece of software that controls the overall operation of the Computer. It acts as an interface between hardware and application programs .It facilitates the user to format disks, create ,print ,copy , delete and display files , read data from files ,write data to files ,control the I/O operations , allocate memory locations and process the interrupts etc. It provides the users an interface to the hardware resources. In a multiuser system it allows several users to share the CPU time ,share the other system resources and provide inter task communication ,Timers , clocks , memory management and also avoids the interference of different users in sharing the resources etc. Hence the OS is also known as a resource manager.

So, the Operating system can also be defined as a collection of system calls or functions which provide an interface between hardware and application program.

It manages the hardware resources of a computer and hosting applications that run on the computer. Hence it is also called a resource Manager.

An OS typically provides multitasking, synchronization, Interrupt and Event Handling, Input/Output, Inter-task Communication, Timers and Clocks and Memory Management.

The core of the OS is the Kernel which is typically a small, highly optimised set of libraries.

The Kernel is a program that constitutes the central core of an operating system. It has complete control over everything that occurs in the system.

The Kernel is the first part of the operating system to load into memory during booting (i.e., system startup), and it remains there for the entire duration of the session because its services are required continuously.

The kernel provides basic services for all other parts of the operating system, typically including memory management, process management, file management and I/O (input/output) management (i.e., accessing the peripheral devices). These services are requested by other parts of the operating system or by application programs through a specified set of program interfaces referred to as system calls.

Popular Operating Systems: Windows (from Microsoft), MacOS, MS-Dos, Linux(Open source),Unix (Multi user-Bell Labs) , Xenix (Microsoft),Android (Mobile) ..

REAL TIME SYSTEMS : Real-time systems are those systems in which the correctness of the system depends not only on the Output , but also on the time at which the results are produced(Time constraints must be strictly followed).

Real time systems are two types. (i) Soft real time systems and (ii) Hard real time systems. A Soft real time system is one in which the performance of the system is only degraded but, not destroyed if the timing deadlines are not met .

For Ex: Air conditioner, TV remote or music player, Bus reservation ,automated teller machine in a bank , A Lift etc.

A hard Real time system is one in which the failure to meet the time dead lines may lead to a complete catastrophe or damage to the system.

For Ex: Air navigation system, Nuclear power plant , Failure of car brakes , Gas leakage system ,RADAR operation ,Air traffic control system etc.

Typical Real Time Applications: Real Time systems find applications in various fields of science and technology. The prominent applications are (i) Digital Control (ii) command and control, (iii) Signal processing (iv) Telecommunication systems and (v) Defense etc .

Examples:

- In automobile engineering, the real time systems control the engine and brakes of the vehicle and regulate traffic lights for smooth travel.
- In air craft monitoring, the real time systems schedule and monitor the takeoff and landing of the planes, make it fly, maintain the flight path, and avoid accidents.
- The real time patient care system monitor and regulate the blood pressure and heart beats of the patient and also , they can entertain people with electronic games ,TV and music.
- The real time systems are found in Air Traffic Control system also. The Air Traffic Control (ATC) system regulates the flow of flights to each destination airport. It does so by assigning to each aircraft an arrival time and en route to the destination
- The real time systems are important in industries also.For example a system of robots perform assembly tasks and repairs in a factory or chemical industries where human beings cannot enter.

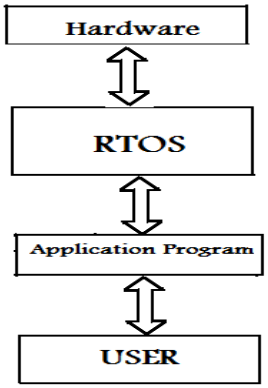
- An avionics system for a military aircraft ,the real time systems perform the tracking and ballistic computations and coordinates the RADAR and weapon control systems.
- Dgital filtering, video and voice compressing/decompression, and radar signal processing are the major applications of real time systems in signal processing.
- Another interesting application is the real-time database systems that refers to a diverse spectrum of information systems, ranging from stock price quotation systems, to track records databases, to real-time file systems.
- Real time systems are also found in Supervisory Control and Data Acquisition (SCADA). In SCADA systems the sensors are placed at different geographical points to collect the raw data and this data are processed and stored in a Real time data base.
- Robots used in nuclear power stations ,to handle the radioactive material and other dangerous materials .
- Real time system applications are also found in office automation where LASER printers and FAX machines are used .

REAL TIME OPERATING SYSTEM (RTOS) : It is an operating system that supports real-time applications by providing logically correct result within the deadline set by the user. A real time operating system makes the embedded system into a real time embedded system.

The basic structure of RTOS is similar to regular OS but, in addition, it provides mechanisms to allow real time scheduling of tasks.

Though the real-time operating systems may or may not increase the speed of execution, but they provide more precise and predictable timing characteristics than general-purpose OS.

The figure below shows the embedded system with RTOS



All the embedded systems are not designed with RTOS. Low end application systems do not require the RTOS but only High end application oriented embedded systems which require scheduling alone need the RTOS.

For example an embedded system which measures Temperature or Humidity etc. do not require any operating system.

Where as a Mobile phone , RADAR or Satellite system used for high end applications require an operating system.

Popular Real-Time Operating Systems:

RTOS	Applications/Features
Windows CE (Microsoft Widows)	Used small foot print mobile and connected devices Supported by ARM,MIPS, SH4 & x86 architectures
LynxOS	Complex, hard real-time applications ·POSIX- compatible, multiprocess, multithreaded OS. ·Supported by x86, ARM, PowerPC architectures
VxWorks (Wind river)	· Most widely adopted RTOS in the embedded industry. · Used in famous NASA rover robots Spirit and Opportunity ·Certified by several agencies and international standards for real time systems, reliability and security-critical applications.
Micrium µC/OS-II	· Ported to more than a hundred architectures including x86, Mainly used in microcontrollers with low resources. · Certified by rigorous standards, such as RTCADO-178B
QNX	· Most traditional RTOS in the market. · Microkernel architecture; completely compatible with the POSIX · Certified by FAADO-278 and MIL-STD-1553 standards.
Symbian	Designed for Smartphones Supported by ARM, x86 architecture
VRTX	·Suitable for traditional board based embedded systems and SoC architectures ·Supported by ARM, MIPS, PowerPC & other RISC architectures
RTLINUX	Open source
Neutrino	

DIFFERENCES BETWEEN RTOS and GENERAL PURPOSE OS :

The basic differences are Determinism, Task scheduling and preempting

Determinism : The key difference between general-computing operating systems and real-time operating systems is the “deterministic " timing behavior in the real-time operating systems. "Deterministic" timing means that OS consume only known

and expected amounts of time. RTOS have their worst case latency defined. Latency is not of a concern for General Purpose OS.

Task Scheduling : General purpose operating systems are optimized to run a variety of applications and processes simultaneously, thereby ensuring that all tasks receive at least some processing time. As a consequence, low-priority tasks may have their priority boosted above other higher priority tasks, which the designer may not want. However, RTOS uses priority-based preemptive scheduling, which allows high-priority threads to meet their deadlines consistently. All system calls are deterministic, implying time bounded operation for all operations and ISRs. This is important for embedded systems where delay could cause a safety hazard. The scheduling in RTOS is time based. In case of General purpose OS, like Windows/Linux, scheduling is process based.

- **Preemptive kernel** - In RTOS, all kernel operations are pre-emptiable
- **Priority Inversion** - RTOS have mechanisms to prevent priority inversion
- **Usage** - RTOS are typically used for embedded applications, while General Purpose OS are used for Desktop PCs or other generally purpose PCs.

Note : **Jitter**: The Timing error of a task over subsequent iterations of a program or loop is referred to as jitter. RTOS are optimized to minimize jitter.

There are four broad categories of kernels.

i.**Monolithic** kernels: provide rich and powerful abstractions of the underlying hardware.

ii.**Microkernels** provide a small set of simple hardware abstractions and use applications called servers to provide more functionality

iii. **Hybrid** (modified Micro kernels) Kernels are much like pure Microkernels , except that they include some additional code in kernel space to increase performance.

iv. **Exo-kernels** provide minimal abstractions, allowing low-level hardware access. In Exokernel systems,library operating systems provide the abstractions typically present in monolithic kernels.

Pre-Emptive and Non-Pre-Emptive : In a normal operating system ,if a task is running ,it will continue to run until its completion .It cannot be stopped by the OS in the middle due to any reason .Such concept is known as non-preemptive.

In real time OS ,a running task can be stopped due to a high priority task at any time with-out the willing of present running task.This is known as pre-emptiveness.

So, Preemptive scheduling involves scheduling based on the highest priority. The highest priority will always be given chance. Non-preemptive scheduling is a process is not interrupted once started until it is finished.

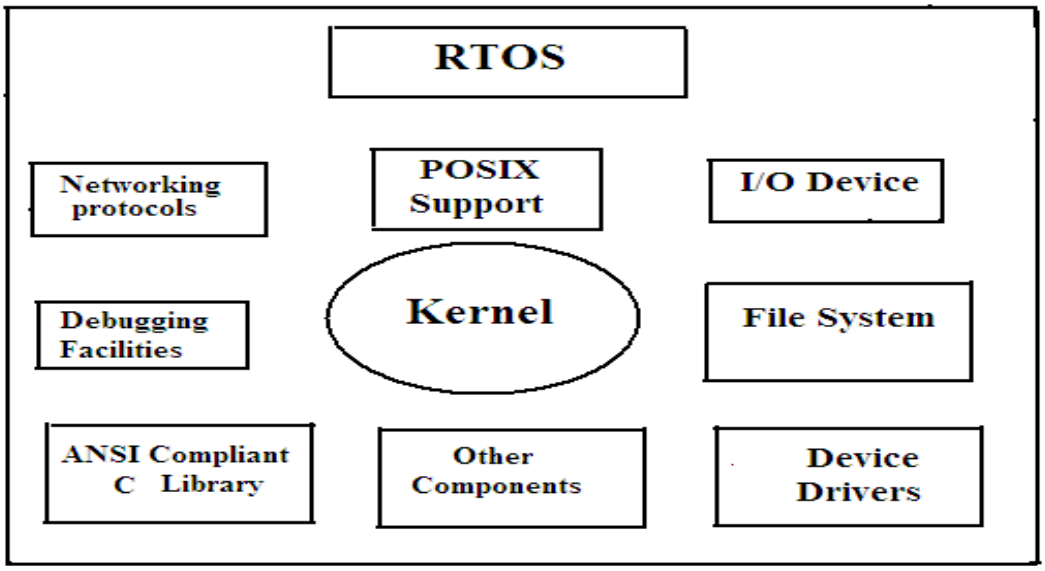
Initialization of RTOS:

RTOS is initialized using the following code.

```
Void main(void)
{
Init RTOS( );          /*Initialize the RTOS*/
Start task (v respond to Button, High _priority);
Start task (v calculate tasklevels , low_priority);
Start_RTOS ( );              /*start RTOS*/
}
```

Architecture of the RTOS :

The heart or nucleus of any RTOS is the kernel. Inside the kernel is the scheduler. It is basically a set of algorithms which manage the task running order. Multitasking definition comes from the ability of the kernel to control multiple tasks that must run within time deadlines. Multitasking may give the impression that multiple threads are running concurrently, as a matter of fact the processer runs task by task, according to the task scheduling.



General Architecture of RTOS

Architecture of the Kernel :

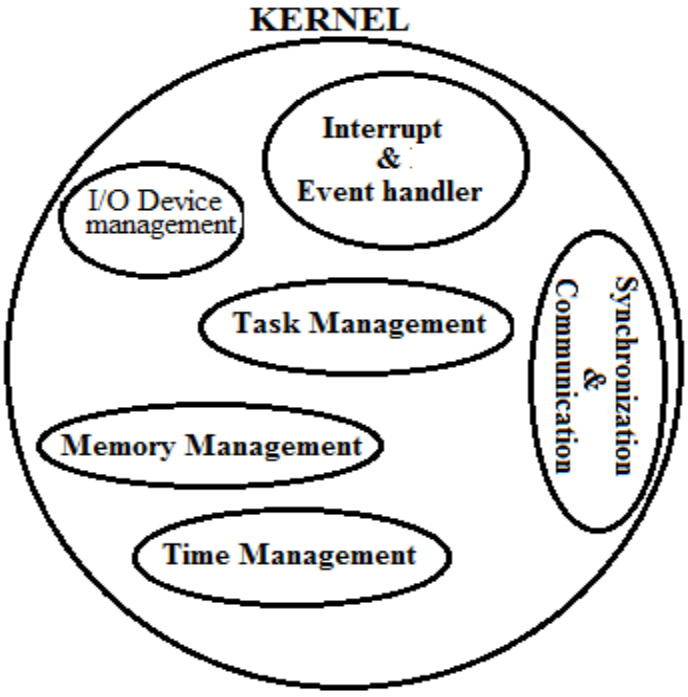
The kernel is the core of an operating system. It is a piece of software responsible for providing secure access to the system's hardware and to running the programs. Kernel is common to every operating system either a real time or non-real time .The major difference lies in its architecture .Since there are many programs, and hardware access is limited, the kernel also decides when and how long a program should run. This is called scheduling. Kernels has various functions such as file management, data transfer between the file system ,hardware management ,memory management and also the control of CPU time. The kernel also handles the Interrupts.

Kernel Objects : The various kernel objects are Tasks, Task Scheduler, Interrupt Service Routines, Semaphores, Mutexes, Mailboxes, Message Queues, Pipes, Event Registers, Signals and Timers

(i).Task:

A task is a basic unit or atomic unit of execution that can be scheduled by an RTOS to use the system resources like CPU , Memory, I/O devices etc. It starts with reading of the input data and of the internal state of the task, and terminates with the production of the results and updating the internal state. The control signal that initiates the execution of a task is provided by the operating system.

There are two types of tasks. (i)Simple Task(S-Task) and (ii) Complex Task(C-Task).



Simple Task (S-task): A simple task is one which has no synchronization point i.e., whenever an S -task is started, it continues until its termination point is reached. Because an S-task cannot be blocked within the body of the task the execution time of an S-task is not directly dependent on the progress of the other tasks in the node. S-task is mainly used for single user systems.

Complex Task (C-Task): A task is called a complex task (C-Task) if it contains a blocking synchronization statement (e.g., a semaphore operation "wait") within the task body. Such a "wait" operation may be required because the task must wait until a condition outside the task is satisfied, e.g., until another task has finished updating a common data structure, or until input from a terminal has arrived.

Task States:

At any instant of time a task can be in one of the following states :

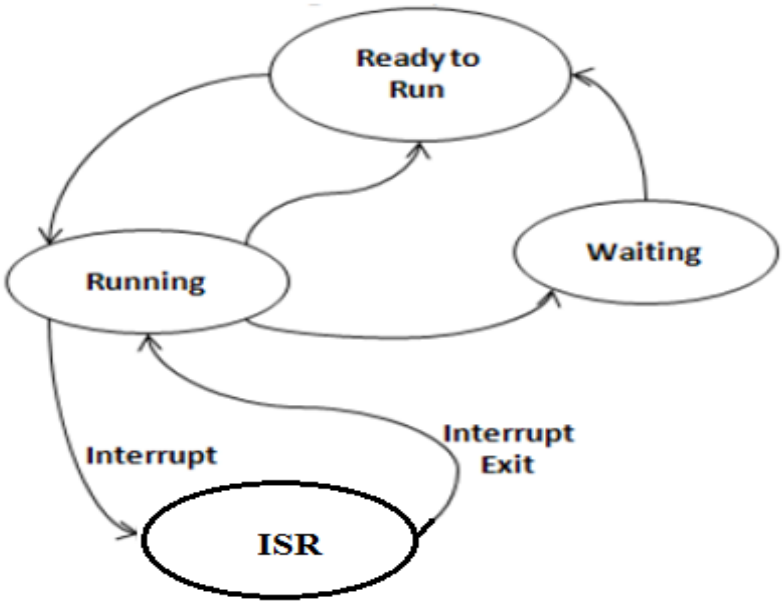
(i) Dormant

(ii).Ready

(iii).Running and

(iv).Blocked.

When a task is first created , it is in the dormant task .When it is added to RTOS for scheduling ,it is a ready task. If the input or a resource is not available ,the task gets blocked.



If no task is ready to run and all of the tasks are blocked, the RTOS will usually run the Idle Task .An Idle Task does nothing .The idle task has the lowest priority.

```
void Idle task(void)
{
While(1);
}
```

Creation of a Task:

A task is characterized by the parameters like task name , its priority , stack size and operating system options .To create a task these parameters must be specified .A simple program to create a task is given below.

```
result = task-create("Tx Task", 100,0x4000,OS_Pre-emptiable);     /*task create*/
if (result = = os_success)
{                    /*task successfully created*/
}
```

Task Scheduler:

Task scheduler is one of the important component of the Kernel .Basically it mis a set of algorithms that manage the multiple tasks in an embedded system. The various tasks are handled by the scheduler in an orderly manner. This produces the effect of simple multitasking with a single processor. The advantage of using a scheduler is the ease of implementing the sleep mode in microcontrollers which will reduce the power consumption considerably (from mA to μA). This is important in battery operated embedded systems.

The task scheduler establishes task time slots. Time slot width and activation depends on the available resources and priorities.

A scheduler decides which task will run next in a multitasking system.

Every RTOS provides three specific functions.

(i).Scheduling (ii) Dispatching and (iii). Inter-process communication and synchronization.

The scheduling determines ,which task ,will run next in a multitasking system and the dispatches perform the necessary book keeping to start the task and Inter-process communication and synchronization assumes that each task cooperate with others.

Scheduling Algorithms: In Multitasking system to schedule the various tasks ,different scheduling algorithms are used..They are

(a).First in First out

(b).Round Robin algorithm

©.Round Robin with priority

(d)Non-preemptive

(e)Pre-emptive.

In FIFO scheduling algorithm, the tasks which are ready-to-run are kept in a queue and the CPU serves the tasks on first-come-first served basis.

In Round-Robin Algorithm the kernel allocates a certain amount of time for each task waiting in the queue. For example, if three tasks 1, 2 and 3 are waiting in the queue, the CPU first executes task1 then task2 then task3 and then again task1.

The round-robin algorithm can be slightly modified by assigning priority levels to the tasks. A high priority task can interrupt the CPU so that it can be executed. This scheduling algorithm can meet the desired response time for a high priority task..This is the Round Robin with priority.

In Shortest-Job First scheduling algorithm, the task that will take minimum time to be executed will be given priority. The disadvantage of this is that as this approach satisfies the maximum number of tasks, some tasks may have to wait forever.

In preemptive multitasking, the highest priority task is always executed by the CPU, by preempting the lower priority task. All real-time operating systems implement this scheduling algorithm.

The various function calls provided by the OS API for task management are given below.

• Create a task

• Delete a task

• Suspend a task

• Resume a task

• Change priority of a task

• Query a task

Interrupt Service Routines:

An interrupt service routine (ISR), also known as an interrupt handler, is a callback subroutine in an operating system or device driver whose execution is triggered by the reception of an interrupt.

In a real-time embedded system

,there are two possible interrupts. one is the Hardware Interrupt and the other is the software Interrupt.

Hardware Interrupts are asynchronous interrupts which are triggered by an electric pulse ,where as software interrupts are synchronous interrupts and these are triggered by a command or instruction.

In hardware driven scheduling, mostly timers,keyboard devices ,I/O ports will take part.

ISR is a small program, which is executed to develop an interface between the user and the hardware. The CPU will execute the ISR subroutine when it receives either a hardware or software interrupt.

The synchronization mechanism cannot be used in an ISR , because it is not possible in an ISR to wait indefinitely for a resource to be available.

The faster the ISR can do its job ,the better the real time performance of the RTOS .Hence the ISR should be always as small as possible.

When the CPU receives either a software or hardware interrupts, it will try to execute the corresponding ISR. Before that all the other interrupt sources are disabled and the interrupts are enabled only after the completion of the ISR .Hence the CPU must execute the ISR as fast as possible and also the ISR must be always as small as possible.

In real-time operating systems, the interrupt latency, interrupt response time and the interrupt recovery time are very important.

Interrupt Latency: It is the time between the generation of an interrupt by a device and the servicing of the device which generated the interrupt.

For many operating systems, devices are serviced as soon as the device's interrupt handler is executed. Interrupt latency may be affected by interrupt controllers, interrupt masking, and the operating system's (OS) interrupt handling methods.

Interrupt Response Time : Time between receipt of interrupt signal and starting the code that handles the interrupt is called interrupt response time.

Interrupt Recovery Time: Time required for CPU to return to the interrupted code/highest priority task is called interrupt recovery time.

Semaphores:

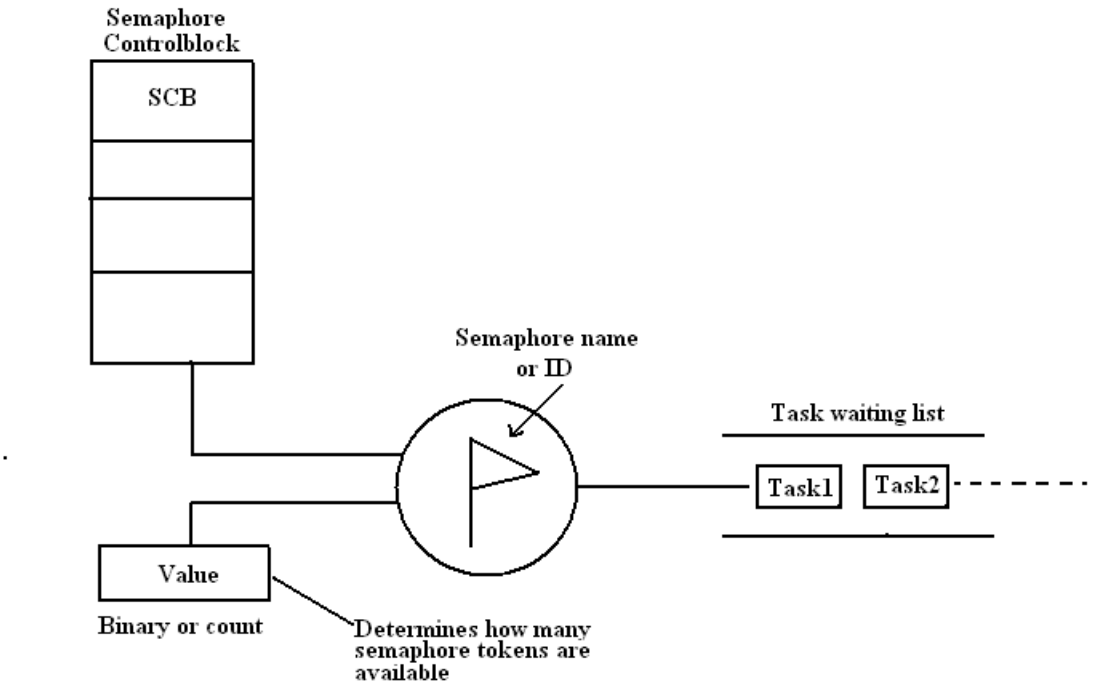
A semaphore is nothing but a value or variable or data which can control the allocation of a resource among different tasks in a parallel programming environment.

The concept of semaphore was first proposed by the Dutch computer scientist Edsger Dijkstra in the year 1965.

So, Semaphores are a useful tool in the prevention of race conditions and deadlocks; however, their use is by no means a guarantee that a program is free from these problems.

Semaphores which allow an arbitrary resource count are called counting semaphores, whilst semaphores which are restricted to the values 0 and 1 (or locked/unlocked, unavailable/available) are called binary semaphores.

The operation of a semaphore can be understood from the following diagram.

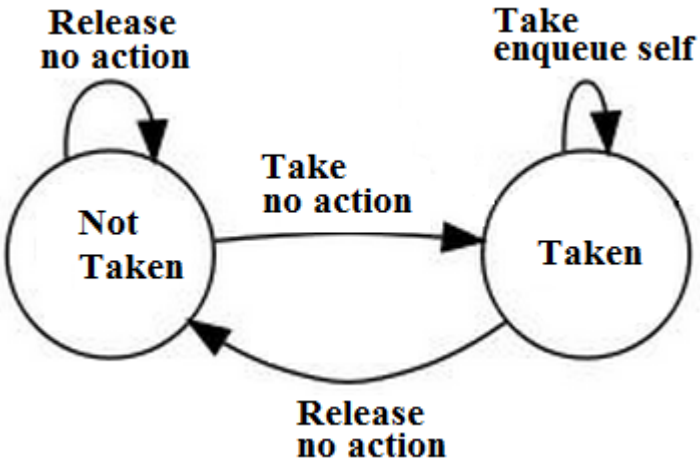


Types of Semaphores: There are three types of semaphores (i).Binary Semaphores, (ii) Counting Semaphores and (iii). Mutexes.

A binary semaphore is a synchronization object that can have only two states 0 or 1. i.e not taken and taken.

Take : Taking a binary semaphore brings it in the "taken" state, trying to take a semaphore that is already taken enters the invoking thread into a waiting queue.

Release: Releasing a binary semaphore brings it in the "not taken" state if there are not queued threads. If there are queued threads then a thread is removed from the queue and resumed, the binary semaphore remains in the "taken" state. Releasing a semaphore that is already in its "not taken" state has no effect.



Binary semaphores have no ownership attribute and can be released by any thread or interrupt handler regardless of who performed the last take operation. Because of this binary semaphores are often used to synchronize threads with external events implemented as ISRs, for example waiting for a packet from a network or waiting that a button is pressed. Because there is no ownership concept a binary semaphore object can be created to be either in the "taken" or "not taken" state initially.

Counting Semaphores:

A counting semaphore is a synchronization object that can have an arbitrarily large number of states. The internal state is defined by a signed integer variable, the counter.

The counter value (N) has a precise meaning: The Negative value indicates that , there are exactly -N threads queued on the semaphore.

The Zero value indicates that no waiting threads, a wait operation would put in queue the invoking thread.

The Positive value indicates that no waiting threads, a wait operation would not put in queue the invoking thread.

Two operations are defined for counting the semaphores.

Wait: This operation decreases the semaphore counter .If the result is negative then the invoking thread is queued.

Signal: This operation increases the semaphore counter .If the result is nonnegative then a waiting thread is removed from the queue and resumed.

Counting semaphores have no ownership attribute and can be signaled by any thread or interrupt handler regardless of who performed the last wait operation .Because there is no ownership concept a counting semaphore object can be created with any initial counter value as long it is non-negative.

The counting semaphores are usually used as guards of resources available in a discrete quantity. For example the counter may represent the number of used slots into a circular queue, producer threads would “signal” the semaphores when inserting items in the queue, consumer threads would “wait” for an item to appear in queue, this would ensure that no consumer would be able to fetch an item from the queue if there are no items available.

The OS function calls provided for Semaphore management are

• Create a semaphore

• Delete a semaphore

• Acquire a semaphore

• Release a semaphore

• Query a semaphore

Mutexes :

Mutex means mutual exclusion A mutex is a synchronization object that can have only two states. They are not-owned and owned. Two operations are defined for mutexes

Lock: This operation attempts to take ownership of a mutex, if the mutex is already owned by another thread then the invoking thread is queued.

Unlock: This operation relinquishes ownership of a mutex. If there are queued threads then a thread is removed from the queue and resumed, ownership is implicitly assigned to the thread.

Mutex is basically a locking mechanism where a process locks a resource using mutex. As long as the process has mutex, no other process can use the same resource. (Mutual exclusion). Once process is done with resource, it releases the mutex. Here comes the concept of ownership. Mutex is locked and released by the same process/thread. It cannot happen that mutex is acquired by one process and released by other.

So, Unlike semaphores, mutexes have owners. A mutex can be unlocked only by the thread that owns it . Most RTOSs implement this protocol in order to address the Priority Inversion problem.

Semaphores can also handle mutual exclusion problems but are best used as a communication mechanism between threads or between ISRs and threads.

The OS functions calls provided for mutex management are

• Create a mutex

• Delete a mutex
• Acquire a mutex
• Release a mutex
• Query a mutex
• Wait on a mutex

Difference between Mutex & Semaphore: Mutexes are typically used to serialize access to a section of re-entrant code that cannot be executed concurrently by more than one thread. A mutex object only allows one thread into a controlled section, forcing other threads which attempt to gain access to that section to wait until the first thread has exited from that section.

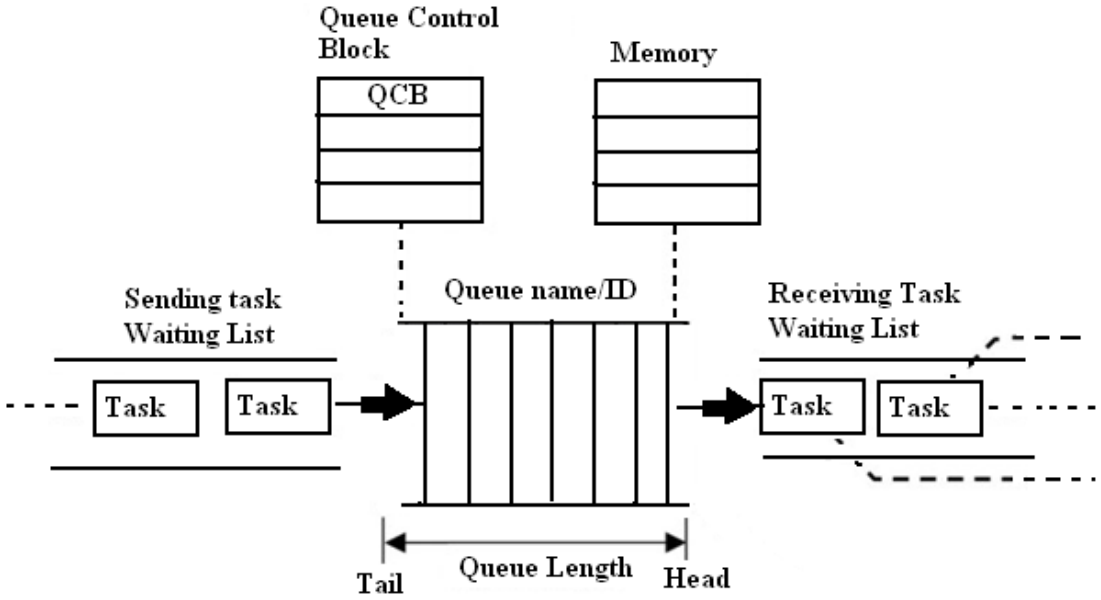
A semaphore restricts the number of simultaneous users of a shared resource up to a maximum number. Threads can request access to the resource (decrementing the semaphore), and can signal that they have finished using the resource (incrementing the semaphore).

Mailboxes:

One of the important Kernel services used to sent the Messages to a task is the message mailbox. A Mailbox is basically a pointer size variable. Tasks or ISRs can deposit and receive messages (the pointer) through the mailbox.

A task looking for a message from an empty mailbox is blocked and placed on waiting list for a time (time out specified by the task) or until a message is received. When a message is sent to the mail box, the highest priority task waiting for the message is given the message in priority-based mailbox or the first task to request the message is given the message in FIFO based mailbox.

The operation of a mailbox object is similar to our postal mailbox. When someone posts a message in our mailbox , we take out the message.



A task can have a mailbox into which others can post a mail. A task or ISR sends the message to the mailbox.

To manage the mailbox object, the following function calls are provided in the OS API:

• Create a mailbox

• Delete a mailbox

• Query a mailbox

• Post a message in a mailbox

• Read a message form a mailbox.

Message Queues:

The Message Queues ,are used to send one or more messages to a task i.e the message queues are used to establish the Inter task communication. Basically Queue is an array of mailboxes. Tasks and ISRs can send and receive messages to the Queue through services provided by the kernel. Extraction of messages from a queue follow FIFO or LIFO structure.

Applications of message queue are

• Taking the input from a keyboard

• To display output

• Reading voltages from sensors or transducers

• Data packet transmission in a network

In each of these applications, a task or an ISR deposits the message in the message queue. Other tasks can take the messages. Based on our application, the highest

priority task or the first task waiting in the queue can take the message. At the time of creating a queue, the queue is given a name or ID, queue length, sending task waiting list and receiving task waiting list.

To use a message queue ,first it must be created.The creation of a Queue return a queue ID .So,if any task wish to post some message to a task ,it should use its queue ID.

q_{id} = queue_create("MyQueue" , Queue_options) ; //*Queue name and OS specification options*//

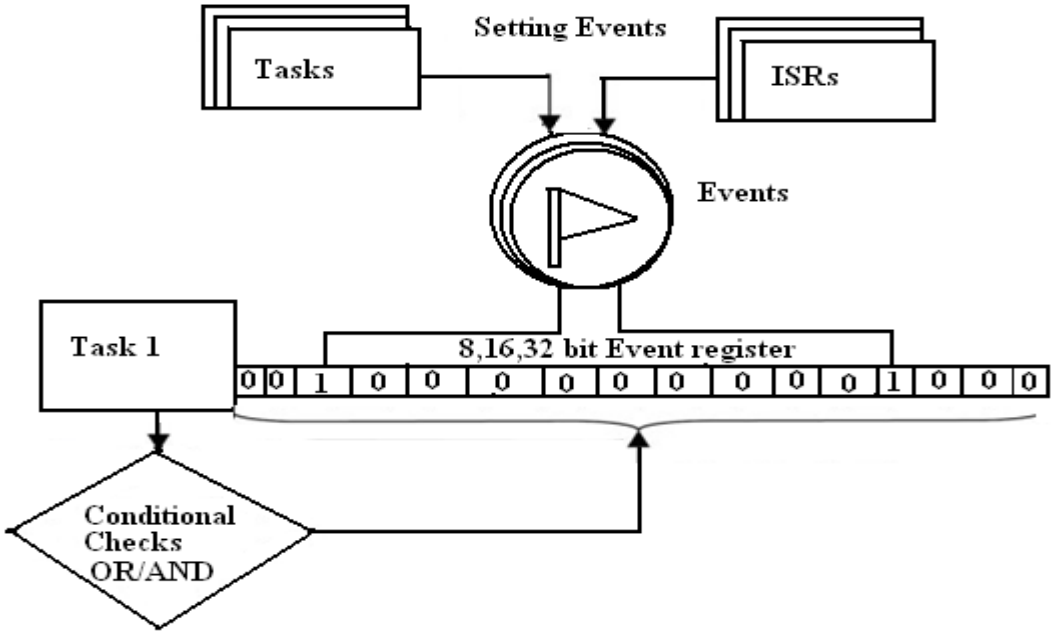
Each queue can be configured as a fixed size/variable size.

The following function calls are provided to manage message queues

- Create a queue
- Delete a queue
- Flush a queue
- Post a message in queue
- Post a message in front of queue
- Read message from queue
- Broadcast a message
- Show queue information
- Show queue waiting list.

Event Registers:

Some kernels provide a special register as part of each tasks control block .This register, called an event register. It consists of a group of binary event flags used to track the occurrence of specific events. Depending on a given kernel's implementation of this mechanism, an event register can be 8 or 16 or 32 bits wide, may be even more.



Each bit in the event register treated like a binary flag and can be either set or cleared. Through the event register, a task can check for the presence of particular events that can control its execution. An external source, such as a task or an ISR, can set bits in the event register to inform the task that a particular event has occurred.

For managing the event registers, the following function calls are provided:

• Create an event register

• Delete an event register

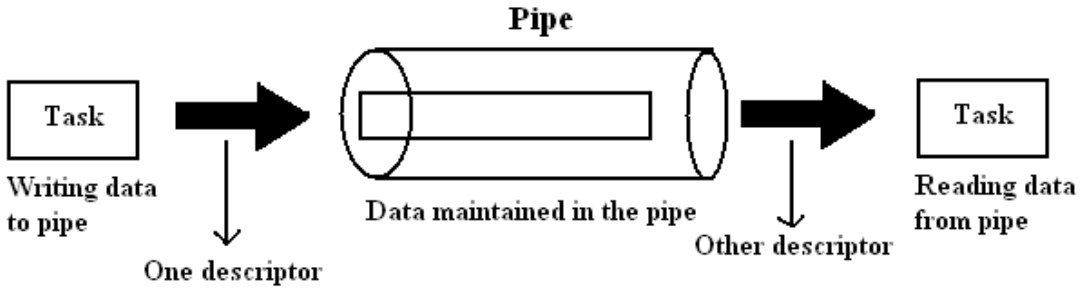
• Query an event register

• Set an event register

• Clear an event flag

Pipes:

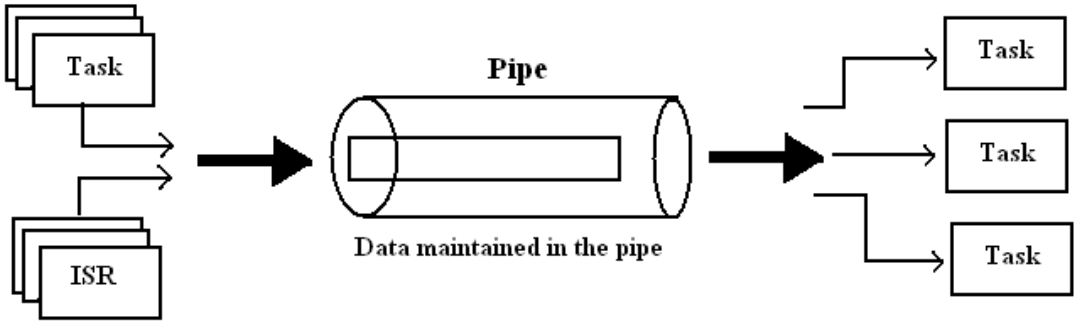
Pipes are kernel objects that are used to exchange unstructured data and facilitate synchronization among tasks. In a traditional implementation, a pipe is a unidirectional data exchange facility, as shown in below Figure.



Two descriptors, one for each end of the pipe (one end for reading and one for writing), are returned when the pipe is created. Data is written via one descriptor and read via the other. The data remains in the pipe as an unstructured byte stream.

Data is read from the pipe in FIFO order. A pipe provides a simple data flow facility so that the reader becomes blocked when the pipe is empty, and the writer becomes blocked when the pipe is full. Typically, a pipe is used to exchange data between a data-producing task and a data-consuming task, as shown in the below

Figure. It is also permissible to have several writers for the pipe with multiple readers on it.



The function calls in the OS API to manage the pipes are:

• Create a pipe

• Open a pipe

• Close a pipe

• Read from the pipe

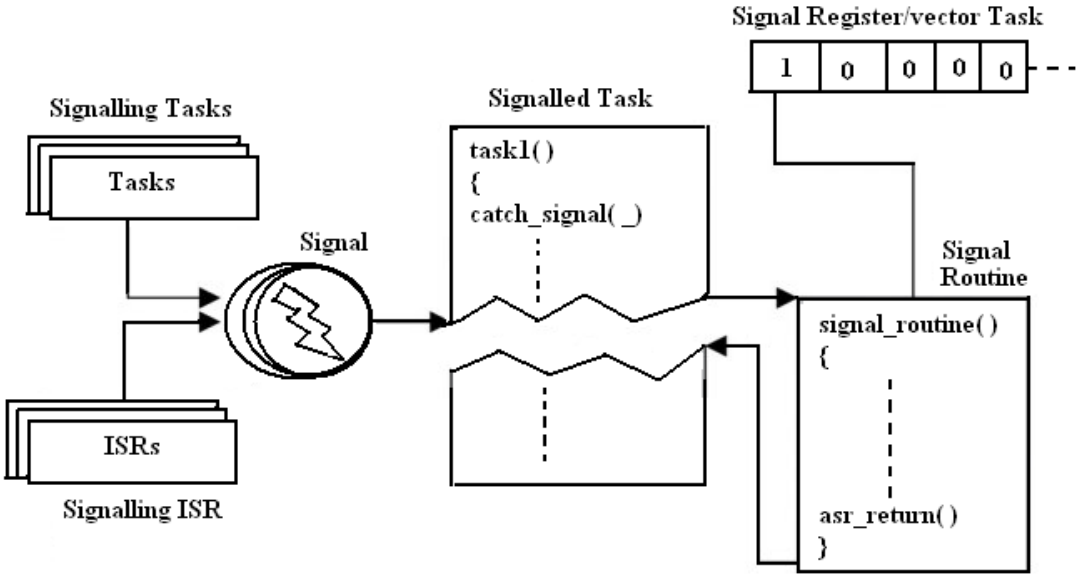
• Write to the pipe

Signals-Signal Handler

A signal is an event indicator. It is a software interrupt that is generated when an event occurs. It diverts the signal receiver from its normal execution path and triggers the associated asynchronous processing. Mainly the , signals notify tasks of events that occurred during the execution of other tasks or ISRs. The difference between a signal and a normal interrupt is that signals are so-called software interrupts, which are generated via the execution of some software within the system. By contrast, normal interrupts are usually generated by the arrival of an interrupt signal on one of the CPU's external pins. They are not generated by software within the system but by external devices.

The number and type of signals defined is both system-dependent and RTOS-dependent. An easy way to understand signals is to remember that each signal is

associated with an event. The event can be either unintentional, such as an illegal instruction encountered during program execution, or the event may be intentional, such as a notification to one task from another that it is about to terminate. While a task can specify the particular actions to undertake when a signal arrives, the task has no control over when it receives signals. Consequently, the signal arrivals often appear quite random,



When a signal arrives, the task is diverted from its normal execution path, and the corresponding signal routine is invoked. The terms signal routine, signal handler, asynchronous event handler, and asynchronous signal routine are inter-changeable. Each signal is identified by an integer value, which is the signal number or vector number.

The function calls to manage a signal are

• Install a signal handler

• Remove an installed signal handler

• Send a signal to another task

• Block a signal from being delivered

• Unblock a blocked signal

• Ignore a signal.

Timers:

A timer is the scheduling of an event according to a predefined time value in the future, like setting an alarm clock. For instance, the kernel has to keep track of different times

• A particular task may need to be executed periodically, say, every 10ms. A timer is

used to keep track of this periodicity.

• A task may be waiting in a queue for an event to occur. If the event does not occur for a specified time, it has to take appropriate action.

• A task may be waiting in a queue for a shared resource. If the resource is not available for a specified time, an appropriate action has to be taken.

The following function calls are provided to manage the timer:

• Get time

• Set time

• Time delay (in system clock ticks)

• Time delay (in seconds)

• Reset timer

Memory Management

It is a service provided by a kernel which allots the memory needed ,either static or dynamic for various processes. The manager optimizes the memory needs and memory utilization. The memory manager allocates memory to the processes and manages it with appropriate protection. There may be static and dynamic allocations of memory. The manager optimizes the memory needs and memory utilization. An RTOS may disable the support to the dynamic block allocation, MMU support to the dynamic page allocation and dynamic binding as this increases the latency of servicing the tasks and ISRs.

Hence, the two instructions “Malloc” and “free” ,although available in C language , are not used by the embedded engineer ,because of the latency problem.

So, an RTOS may or may not support memory protection in order to reduce the latency and memory needs of the processes.

The API provides the following function calls to manage memory

• Create a memory block

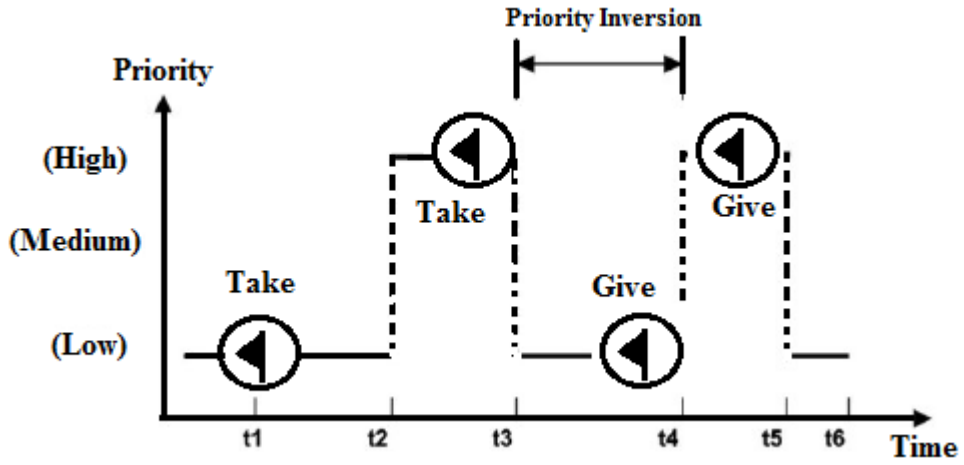
• Get data from memory

• Post data in the memory

• Query a memory block

• Free the memory block.

Priority Inversion Problem In any real time embedded system ,if a high priority task is blocked or waiting and a low priority task is running or under execution ,this situation is called Priority Inversion. This priority Inversion is shown in the diagram below.



In Scheduling, priority inversion is the scenario where a low priority Task holds a shared resource that is required by a high priority task. This causes the execution of the high priority task to be blocked until the low priority task releases the resource, effectively "inverting" the relative priorities of the two tasks.

Suppose some other medium priority task, one that does not depend on the shared resource, attempts to run in the interim, it will take precedence over both the low priority task and the high priority task.

The consequences of the priority Inversion are

(i) Reduce the performance of the system (ii) May reduce the system responsiveness which leads to the violation of response time guarantees (iii) Create problems in real-time systems.

There are two types of priority inversions.(i) Bounded and (ii).Unbounded.

For example let us consider two tasks T_A and T_B in a real time system. Task A has higher priority than Task B. Initially Task A is under execution. But Task A is blocked after some time due to interruption and Task B is scheduled next. The Task B acquires a mutex corresponding to a resource common to both Task A and Task B. After some time Task A acquire mutex before the completion of Task B. But Task A cannot acquire mutex and it is blocked because already Task B has acquired the mutex. So, the Task A ,though has the higher priority ,is blocked until Task B releases the mutex for the resource. This is called the bounded Priority inversion.

If the time over which the higher priority is blocked is unknown ,then it is called unbounded priority inversion.

The Priority Inversion is avoided by using two protocolas.,namely

(i).Priority Inheritance Protocol (PIP)

(ii) Priority Ceiling Protocol(PCP).

The Priority Inheritance Protocol is a resource access control protocol that raises the priority of a task, if that task holds a resource being requested by a higher priority task, to the same priority level as the higher priority task.

The **priority ceiling protocol** is a synchronization protocol for shared resources to avoid unbounded priority inversion and mutual deadlock due to wrong nesting of critical sections .In this protocol each resource is assigned a priority ceiling, which is a priority equal to the highest priority of any task which may lock the resource.

Types of operating systems :

An Operating system (OS) is nothing but a piece of software that controls the overall operation of the Computer. It acts as an interface between hardware and application programs .It facilitates the user to format disks, create ,print ,copy , delete and display files , read data from files ,write data to files ,control the I/O operations , allocate memory locations and process the interrupts etc. It provides the users an interface to the hardware resources. In a multiuser system it allows several users to share the CPU time ,share the other system resources and provide inter task communication ,Timers , clocks , memory management and also avoids the interference of different users in sharing the resources etc. Hence the OS is also known as a resource manager.

There are three important types of operating systems .They are

(i).Embedded Operating System

(ii). Real time operating system and

(iii).Handheld operating system.

(i).Embedded Operating System

The operating system used for embedded computer systems is known as embedded operating system. These operating systems are designed to be compact, efficient, and reliable.

The embedded operating system uses a preemptive priority based kernel.But this kernel do not meet the strict dead lines.By removing the unnecessary components from the kernel of desktop operating system ,the embedded operating can be obtained. This OS occupies less memory space.The popularly known embedded operating systems are

(a).Embedded NT (b) Windows XP Embedded (c) Embedded Linux

The Embedded NT for its minimal operation without any network support occupies nearly 9MB of RAM and 8 MB of Flash .It is a preemptive ,multitasking operating system. Generally Embedded NT is preferred to other OSs because of its ease in developing the applications.It is suitable for embedded systems built around single board computers for applications,like Internet Kiosks,Automatic Teller Machines (ATM) etc..

Microsoft Windows XP Embedded is the successor to Embedded NT.It is also pre-emptive multitasking operating system like Embedded NT.This OS is widely used in set top boxes, point of sale terminals and Internet Kiosks etc.

Embedded Linux is a open source software and it is covered by GNU General Public License(GPL) and hence the complete source code is available at free of cost. The important features of Embedded Linux are POSIX support and availability of large software resources.

Embedded Linux is used in embedded computer systems such as mobile phones, personal digital assistants, media players, set-top boxes, and other consumer electronics devices, networking equipment, machine control, industrial automation, navigation equipment and medical instruments.

Real-Time Operating System:

A real-time operating system (RTOS) is an operating system (OS) intended to serve te real time application requests .A key characteristic of an RTOS is the level of its consistency concerning the amount of time it takes to accept and complete an application's task .

A hard real-time operating system has less jitter than a soft real-time operating system. The main objective is not the high throughput, but a guarantee of meeting the deadlines. An RTOS that can usually or generally meet a deadline is a soft real-time OS, but if it can meet a deadline deterministically it is a hard real-time OS.

A real-time OS has an advanced algorithm for scheduling Key factors in a real-time OS are minimal interrupt latency and minimal thread switching latency, but a real-time OS is valued more for how quickly or how predictably it can respond.

There are various Real-Time operating systems both commercial and open source in the market

(i).QNX Neutrino (ii)VxWorks (iii) microC/OS-II (iv).RTLinux.

QNX Neutrino is a real time operating system from QNX Software systems limited . It is supported by ARM, MIPS, Power PC, Strong ARM,X86 and Pentium.

This OS supports multiple scheduling algorithms and upto 65535 tasks and can create embedded data base applications.

microC/OS-II is a real time operating system used mainly in academic institutions.It is available in source code form for non-commercial applications.This do not support the Round Robin scheduling algorithm.

RT Linux is a hard real time RTOS microkernel that runs the entire Linux operating system as a fully preemptive process. It was developed by Victor Yodaiken and Michael Barabanov at the New Mexico Institute of Mining and Technology. It was commercialized at FSM Labs.

RT Linux runs underneath the Linux OS. The Linux is an idle task for RT Linux. The real-time software running under RT Linux is given priority as compared to non-real-time threads running under Linux. This OS is an excellent choice for 32-bitprocessor based embedded systems.

Handheld Operating Systems:

Handheld computers are becoming very popular now a days due to their increasing applications. A handheld operating system, also known as a mobile OS, a mobile platform, is the operating system that controls a mobile device.

Typical examples of devices running a mobile operating system are smart phones, personal digital assistants (PDAs), tablet computers and information appliances, or what are sometimes referred to as smart devices, which may also include embedded systems, or other mobile devices and wireless devices.

The important requirements for a mobile operating system are:

- To keep the cost of the handheld computer low,small footprint of the OS is required.
- The OS should have support for soft real time performance.
- TCP/IP stack needs to be integrated along with the OS.
- Communication protocol stacks for Infrared, Bluetooth, IEEE 802.11interfaces need to be integrated.
- There should be support for data synchronization.

The popular handheld operating systems available are (a) Palm OS (b)Symbian OS (iii)Windows CE (iv) Windows CE.NET.

Palm OS is a popular Handheld OS ,where the application development can be done using C,C++ or Java. Most of the Sony based handheld devices use this OS.

The **Windows CE** is another popular handheld operating system widely used in Pocket PCs and PDAs. In addition to this the handheld devices from Thoshiba ,

Hitachi ,Alpha use this OS. Application software can be developed in Visual Basic,VC++using the embedded visual tools.

----------xxxxxxx----------

References:

1.Embedded/Real Time Systems : Concepts,Design &Programming
–Dr.K.V.K.K Prasad : Dreamtech Publs

2. Embedded & Real Time Systems Notes - Mr. Suman Kalyan Oduri