DEPARTMENT

OF

ELECTRONICS & COMMUNICATION ENGINEERING

EMBEDDED SYSTEM AND DESIGN

(Theory Notes)

Autonomous Course

Prepared by

Prof. SANTHOSH KUMAR R

Module – 5 Contents
Real Time Operating Systems: Real Time and Embedded, Operating Systems, Schedule Management for Multiple Tasks by an RTOS, Interrupt Routines in RTOS Environment, RTOS Task Scheduling models, Interrupt latency and response times for the Tasks as Performance Metrics.

Dayananda Sagar College of Engineering

Shavige Malleshwara Hills, Kumaraswamy Layout,
Banashankari, Bangalore-560078, Karnataka
Tel : +91 80 26662226 26661104 Extn : 2731 Fax : +90 80 2666 0789
Web - http://www.dayanandasagar.eduEmail :hod-ece@dayanandasagar.edu

UNIT V

REALTIME OPERATING SYSTEM

Operating System Basics

❖ Bridges between user application/tasks & system resources

❖ The OS manages the system resources & make them available to the user application/task. Computing system is collection of different I/O sub-system working & storage memory.

Primary Function of OS

❖ Make the system convenient to use

❖ Organizes & manage the system resources efficiently & correctly

The Kernel

❖ The kernel is core of the OS & is responsible for managing the system & the communication among the hardware & the system services. Kernel acts as the abstraction layer between system resources & user application. Kernel contains a set of system libraries of services. For general purpose OS the kernel contains different services for handling the management.

1. **Process Management:**
   - ❖ Managing the process/tasks
   - ❖ Setup the memory space for process Load
   - ❖ Program/code into space(memory)
   - ❖ Scheduling & managing the execution of the process
   - ❖ Setting up & managing the PCB
   - ❖ Inter process communication & system synchronization process termination & deletion

2. Primary Memory Management

- It is volatile memory
- The MMU of kernel is responsible for Keeping track of which part of memory area is correctly used by which process Embedded Computing system.
- Allocating &deal locating the memory spaces

3. File System Management

- File is a collection of related information
- A file could be a program, tent files, image file, word documents, audio/video files etc.
- The file system management service of kernel is responsible for
  - ✓ The creation & deletion of files
  - ✓ Creation , deletion & alteration directly
  - ✓ Saving the file in secondary storage memory
  - ✓ Providing automatic allocation of spaces
  - ✓ Providing a flexible naming conversion for the files

4. I/O System (Device) Management

- Kernel is responsible for routing the I/O request coming from different user applications Direct access of I/O devices are not allowed , we can access the I/O devices through the API imposed by kernel

- ❖ Dynamically update the available devices
- ❖ Device manager of the kernel is responsible for handling I/O device related operations The kernel talks to the I/O device through the device driver , this is responsible for
  - ✓ Loading & unloading of device drivers
  - ✓ Exchanging the information's

5. Secondary Storage Management

- ❖ The secondary storage management deals with secondary storage memory device, if any connected to the system
- ❖ The main memory is volatile
- ❖ The secondary storage management services of kernel deals with
  - ✓ Disk storage allocation
  - ✓ Disk scheduling
  - ✓ Free disk space management

What is RTOS?

1) A real time operating system (RTOS) is an operating system that guarantees a certain **Capability** within a **specified time constraint**.

2) An OS is a system program that provides an **interface** between **Application programs** and the computer system (**hardware**)

3) The applications where dependability that a certain task will finish before a particular **deadline** is just as obtaining the **correct results**.

4) Besides meeting deadlines RTOS must also be able to respond **predictably** to unpredictable **events** and process **multiple events** concurrently.

A **Real-Time Operating System** (RTOS) is a computing environment that reacts to input within a specific time period. A real-time deadline can be so small that system reaction appears instantaneous. The term real time computing has also been used, however, to describe "slow real- time" output that has a longer, but fixed, time limit.

Learning the difference between real-time and standard operating systems is as easy as imagining yourself in a computer game. Each of the actions you take in the game is like a program running in that environment. A game that has a real-time operating system for its environment can feel like an extension of your body because you can count on a specific "lag time:" the time between your request for action and the computer's noticeable execution of your request. A standard operating system, however, may feel disjointed because the lag time is unreliable. To achieve time reliability, real-time programs and their operating system environment must prioritize deadline actualization before anything else. In the gaming example, this might result in dropped frames or lower visual quality when reaction time and visual effects conflict.

Real-Time Kernel

The heart of a real-time OS (and the heart of every OS, for that matter) is the **kernel**. A kernel is the central core of an operating system, and it takes care of all the OS jobs:

1. Booting
2. Task Scheduling
3. Standard Function Libraries

In an embedded system, frequently the kernel will boot the system, initialize the ports and the global data items. Then, it will start the scheduler and instantiate any hardware timers that need to be started. After all that, the Kernel basically gets dumped out of memory (except for the library functions, if any), and the scheduler will start running the child tasks. Standard function libraries: In an embedded system, there is rarely enough memory (if any) to maintain a large function library. If functions are going to be included, they must be small, and important.

Basic Kernel Services

Kernel which is a major part of an operating system that provides the most basic services to application software running on a processor. The "kernel" of a real-time operating system ("RTOS") provides an "abstraction layer" that hides from application software the hardware details of the processor (or set of processors) upon which the application software will run.

Scheduling

In typical designs, a task has three states:

1. Running (executing on the CPU);
2. Ready (ready to be executed);
3. Blocked (waiting for an event, I/O for example).

Most tasks are blocked or ready most of the time because generally only one task can run at a time per CPU. The number of items in the ready queue can vary greatly, depending on the number of tasks the system needs to perform and the type of scheduler that the system uses. On simpler non-pre-emptive but still multitasking systems, a task has to give up its time on the CPU to other tasks, which can cause the ready queue to have a greater number of overall tasks in the ready to be executed state (resource starvation).Usually the data structure of the ready list in the scheduler is designed to minimize the worst-case length of time spent in the scheduler's critical section, during which pre-emption is inhibited, and, in some cases, all interrupts are disabled. But the choice of data structure depends also on the maximum number of tasks that can be on the ready list.

If there are never more than a few tasks on the ready list, then a doubly linked list of ready tasks is likely optimal. If the ready list usually contains only a few tasks but occasionally contains more, then the list should be sorted by priority. That way, finding the highest priority task to run does not require iterating through the entire list. Inserting a task then requires

walking the ready list until reaching either the end of the list, or a task of lower priority than that of the task being inserted.

Care must be taken not to inhibit pre-emption during this search. Longer critical sections should be divided into small pieces. If an interrupt occurs that makes a high priority task ready during the insertion of a low priority task, that high priority task can be inserted and run immediately before the low priority task is inserted.

The critical response time, sometimes called the fly back time, is the time it takes to queue a new ready task and restore the state of the highest priority task to running. In a well-designed RTOS, readying a new task will take 3 to 20 instructions per ready-queue entry, and restoration of the highest-priority ready task will take 5 to 30 instructions.

Inmoreadvancedsystems,real-timetasksssharecomputingresourceswithmanynon-real time tasks, and the ready list can be arbitrarily long. In such systems ,a scheduler ready list implemented as a linked list would being adequate.

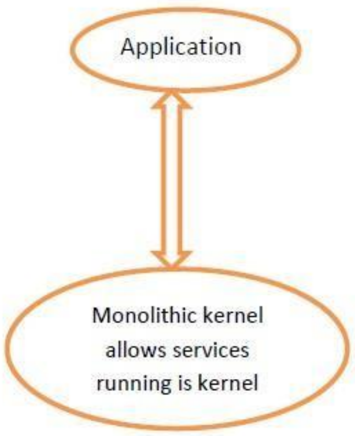
Types of Kernel:

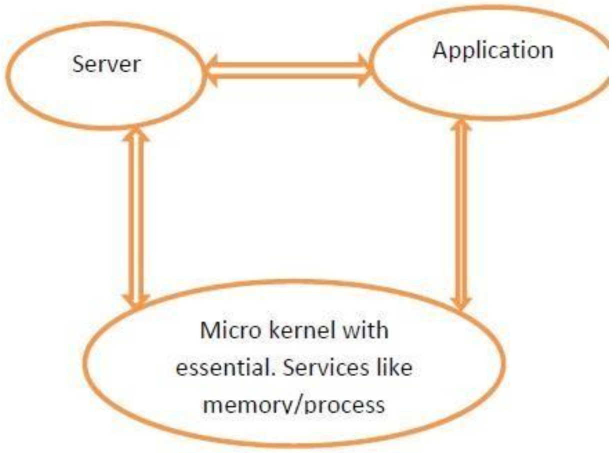
Based on kernel design, kernel can be classified into Monolithic & Micro

❖ In Monolithic kernel architecture all the kernel service run in the kernel space – means all kernel module run with same memory space under single kernelthread

❖ The drawback of Monolithic kernel is that any error or failure in any of the kernel module

i.e. leads to the crashing of entire kernel application. Ex : LINUX,SOLARIS,MS-DOS





Monolithic kernel model **Micro kernel**

Micro kernel: design incorporates only the essential set of OS services into the kernel. The rest of the OS services are implanted in programs known as "servers" which runs in user space.

The essential services of the Micro kernel are

- ✓ Memory management
- ✓ Process management
- ✓ Timer system
- ✓ Interrupt handler

Ex. OS , MACH , ONX , MINIX3

Benefits of Micro kernel

1. Robust

2. Configurability

Types of Operating System

Depending on the type of kernel & kernel services purpose & type of in computing system OS are classified into two types

1. General Purpose Operating System[GPOS]

2. Real Time Operating System[RTOS]

1. General Purpose Operating System[GPOS]

The OS which are deployed in general computing systems are referred as general purpose OS.

The kernel of such OS is more generalized & it contains all kinds of services required for executing generic applications

GPOS are non-deterministic in behavior

Ex: PC/Desktop system, windows XP & MS-DOS

2. Real Time Operating System [RTOS]

There is no universal definition available for the term RTOS.

- ✓ A RTOS is an OS intended to serve real-time application requests. It must be able to process the data as it comes in typically without buffering delays
- ✓ Processing time requirements are measured in tenth of seconds. Hard RTOS has less jitter than soft RTOS.

[Jitter: Variation in time between packet arriving with time congestion & time.]

- ✓ Real-time implies deterministic timing behavior – means the OS services consumes only known & expected amount of time regardless the no of services.
- ✓ RTOS decides which application should run in which order & how much time needs to be allocated for each applications.

Ex: Windows CE, UNIX, VxWorks, Micros/OS-II Real Time Kernel:

The RTOS is referred to as Real Time kernel in complement to conventional OS kernel. The kernel is highly specialized & it contains only the minimal set of services required for running the user application/

Tasks.

The Basic Functions of Real Time Kernel:

1. Task/Process Management

2. Task/Process Scheduling

3. Task/Process Synchronization

4. Error/Exception Handling

5. Memory Management

6. Interrupt Handling

7. Time Management

Task/Process Management: Deals with setting up the memory space for the tasks, loading the tasks code into memory space, allocating system resources, setting up a Task Control Block [TCB] for the task &task/process termination/deletion.

TCB: Task Control Block is used for holding the information corresponding to a task. The TCB contains the following set of information

- ✓ Task ID: Task identification number
- ✓ Task State: The current state of the task
- ✓ Task Type: Indicates what is the type of this task, the task can be hard real time or soft real time or background task
- ✓ Task Priority: Ex task priority=1 for task with priority =1
- ✓ Task Context Pointer: context pointer, pointer for saving context Task Memory Pointer: Pointers to the code memory, data memory &
- ✓ stack memory for the task
- ✓ Task System Resource Pointers: Pointer to system resource used by the task

- ✓ Task Pointers: Pointer to other TCB’s
- ✓ Other Parameters: Other relevant task parameters

The TCB parameters vary across different kernels based on the task management Implementation

Creates TCB for a task on creating task

- ✓ Delete/Remove the TCB when the task is terminated or deleted
- ✓ Reads the TCB to get the state of a task
- ✓ Updates the TCB with updated parameters on need basics
- ✓ Modify the TCB to change the priority of task dynamically

Task/Process Scheduling: Sharing the CPU among various tasks/process. A kernel applications called scheduler, handles the task scheduling. Scheduler is nothing but an algorithm implemented all to , which performs the efficient & optimal scheduling of task to provide a deterministic behaviour.

Task/Process Synchronization: deals with system concurrent accessing a resource which is shared access multiple tasks & Communication between various tasks.

Error/Exception Handling: Deals with registering& handling the errors occurred during the execution of tasks. Ex: Insufficient memory, time outs, dead locks, dead line missing, bus error, divide by zero, unknown instruction execution. Errors & exceptions can happen at two levels of services * Kernel Level Service * At Task Level

Memory Management: RTOS makes use of 'Block Based Memory' allocation techniques instead of the usual dynamic memory allocation technique used by GPOS. RTOS kernel uses blocks of fixed size dynamic memory & block is allocated for a task on a need of basis. A few RTOS kernel implements Virtual Memory concepts avoid the garbage collection overhead.

Interrupt Handler: Deals with the handling of several of interrupts. Interrupts provide Real Time behaviour embedded system design process

to systems. Interrupts inform the processor that an external device or an associated task required immediate attention of the CPU. Interrupts can

be either Synchronous Asynchronous. Interrupts which occurs in synch with the currently executing task is known as synchronous interrupts. Usually the system interrupts fall under the synchronous category Ex

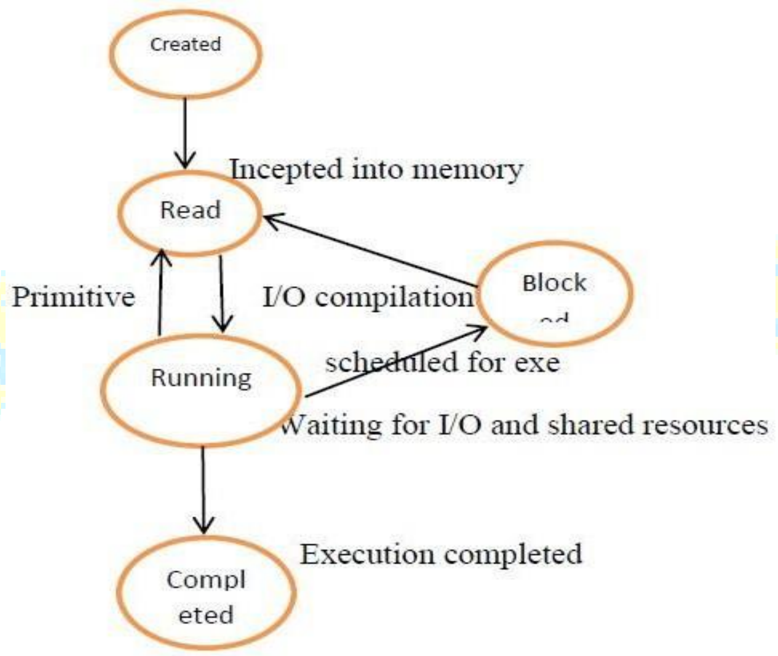
Divide by zero, memory segmentation error. A synchronous interrupts are those which occurs at any point of execution of any task & are not in sync with currently executing tasks.

Task, Process & Threads

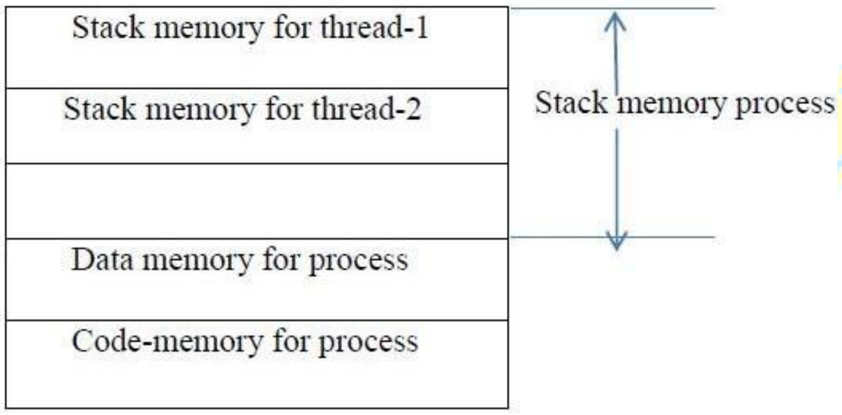
Task: Refers to something that needs to be done. The tasks refers can be one assigned by manages or the one assigned by only one of the processor family needs. In addition we have an order of priority & scheduling timeline for executing these tasks. In OS context, a task is defined as the program in execution & related information maintained by the OS. Task is also known as "JOB" in the OS context.

Process: A process is a program as part of the program, in execution process is also known as instance of the program in execution, multiple instance of the same program can execute simultaneously. Process requires various system resources like CPU for executing the process, memory for storing the code corresponding to the process, I/O devices for in function exchange etc.

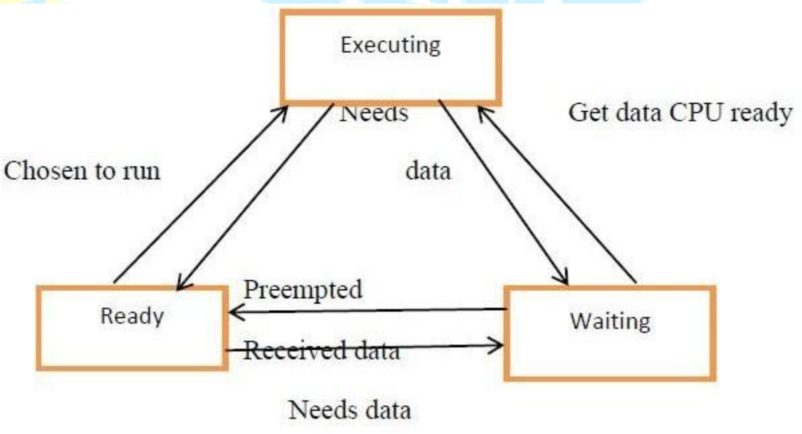
Process States & State Transition



- Memory organization of a process and its associated threads Run: A task enters this state as it starts executing on the process Ready state of those tasks that are ready to execute but
-



- cannot be executed because the processor is assigned to another task Blocked a tasks enters this state when it executes synchronization primitive to wait for an event
- Ex:-a wait primitive on a semaphore this case the task is inserted in a queue associated with the semaphore.
- Created (idle) a periodic job enters this state when it computes its execution and has to wait for the beginning of the next period Process management:
- Process management deals with the creating a process, setting up the memory space for the process, loading the process code with the memory space, allocating the system resources. Setting up a process control block (PCB) for the process and process termination/detection. Task scheduling:
- Multitasking evolves the execution switching among the different tasks. There should be same mechanism in place to show the CPU among the
- different takes 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 give point of time is known as task/process scheduling. Task
- scheduling from the basis of multitasking Scheduling policies from the guidelines for determining which task is to be executed when.
- The work of choosing the order of running process is known as scheduling. A scheduling policy defines how processes are selected for
- promotion from the ready state to the running sate. The process scheduling decision may take place when a process Switches its states Ready state from Running state Blocked/wait sate from Running state Ready state from Blocked/wait state Completed (executed) state.



Scenario 1 is premitive:a process swithches to ready state from the running state

- Scenario 2 is scheduling under can be either pre-emptive or non-pre-emptive When a high priority process in the blocked/wait state complete it's I/O and switches to ready state. The scheduler picks it for execution if the scheduling policy used is priority based pre-emptive.

Task Scheduling Cooperative Models

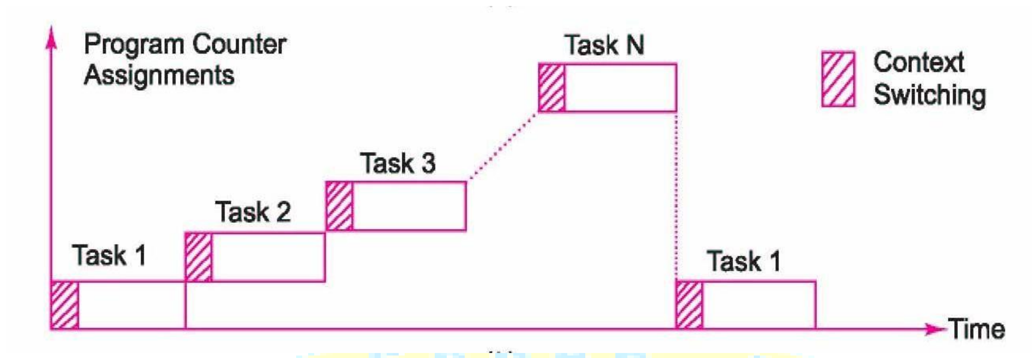
- Cooperative Scheduling of ready tasks in a circular queue. It closely relates to function queue scheduling.
- Cooperative Scheduling with Precedence Constraints
- Cyclic Scheduling of periodic tasks and Round Robin Time Slicing Scheduling of equal priority tasks
- Preemptive Scheduling
- Scheduling using 'Earliest Deadline First' (EDF) precedence.
- Rate Monotonic Scheduling using 'higher rate of events occurrence First' precedence
- Fixed Times Scheduling
- Scheduling of Periodic, sporadic and aperiodic Tasks
- Advanced scheduling algorithms using the probabilistic Timed Petri nets (Stochastic) or Multi Thread Graph for the multiprocessors and complex distributed systems.

Cooperative Scheduling in the cyclic order

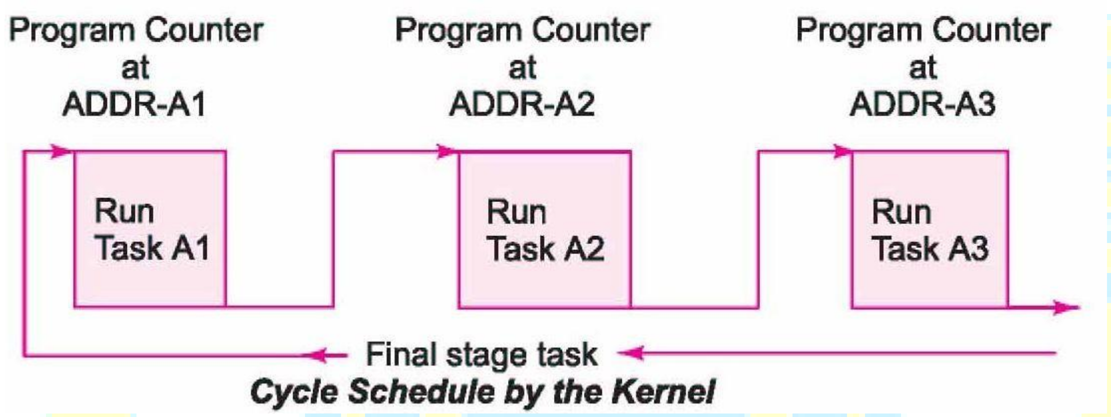
- Each task cooperate to let the running task finish
- Cooperative means that each task cooperates to let the a running one finish.
- None of the tasks does block in-between anywhere during the ready to finish states.
- The service is in the cyclic order.

Worst-case latency

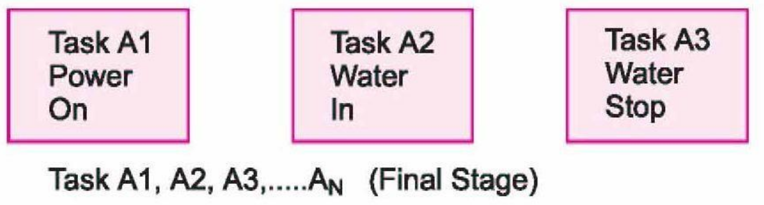
- Same for every task
  - Tworst = {(sti + eti)1 + (sti + eti)2 +...+ (sti + eti)N-1 + (sti + eti)N} + tISR.
- tISR is the sum of all execution times for the ISRs
- For an i-th task, switching time from one task to another be is sti and task execution time be is eti
  - i = 1, 2, …, N 1 , N, when number of tasks = N
- Program counter assignments (switch) at different times, when the on the scheduler calls the o tasks from the list one by one in the circular queue from the list



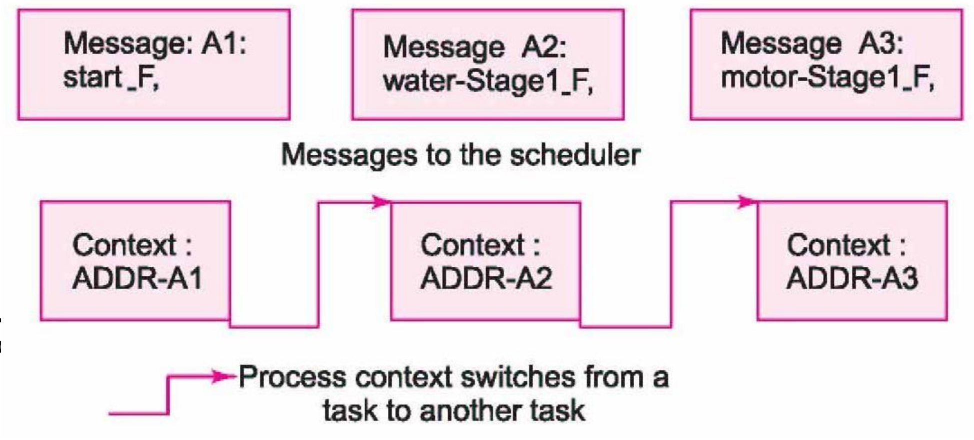
Cyclic scheduling model in tasks scheduling


Program Counter at ADDR-A1
Program Counter at ADDR-A2
Program Counter at ADDR-A3
Run Task A1
Run Task A2
Run Task A3
Final stage task
Cycle Schedule by the Kernel

First three tasks among N- tasks in washing machine tasks scheduling



Messages from the scheduler and task programs contexts at various instances in washing machine tasks scheduling


Message: A1: start_F,
Message A2: water-Stage1_F,
Message A3: motor-Stage1_F,
Messages to the scheduler
Context : ADDR-A1
Context : ADDR-A2
Context : ADDR-A3
Process context switches from a task to another task

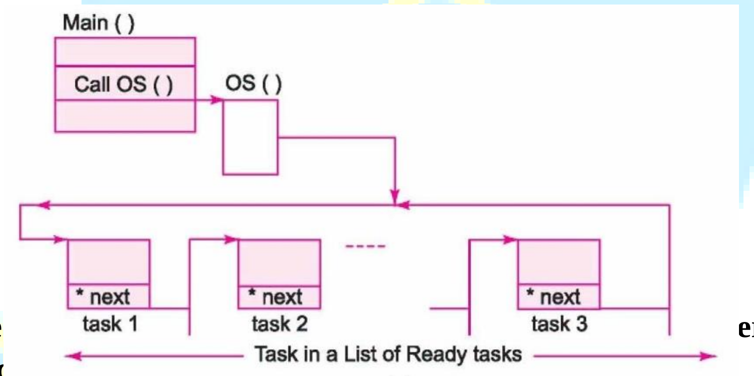
Cooperative Scheduling in the order in which a task is initiated on interrupt

- None of the tasks does block in-between anywhere during the ready to finish states.
- The service is in the order in which a task is initiated on interrupt

Worst-case latency

- Same for every task in the ready list
- Tworst = {(dti + sti + eti)1 + (dti + sti + eti)2
- +...+ (dti + sti + eti)n-1 + (dti + sti + eti)n} + tISR.
- tISR is the sum of all execution times for the ISRs
- For an i-th task, let the event detection time with when an event is brought into a list be is dti , switching time from one task to another be is sti and task execution time be is eti
- i = 1, 2, …, n □1 , n

Scheduler in which the scheduler inserts into a list the ready tasks for a sequential execution in a cooperative mode



Cooperative ered List as per priority prec

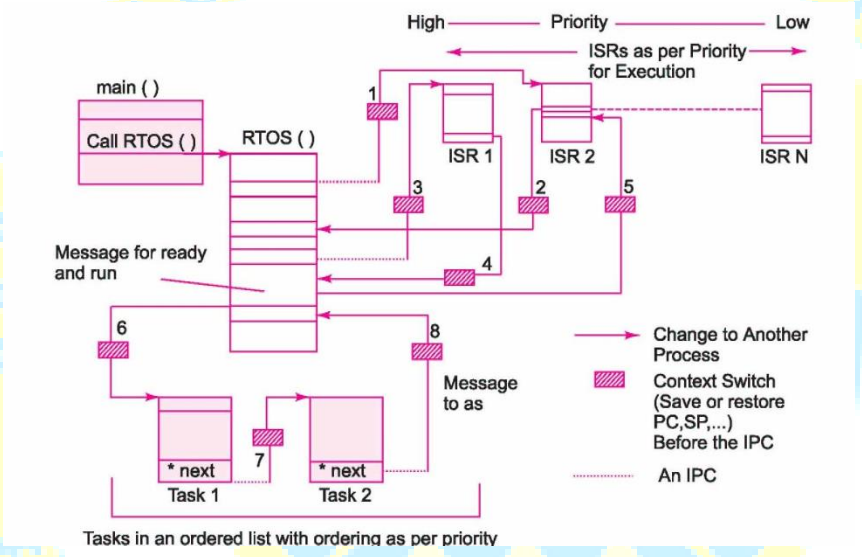
- Scheduler using a priority parameter, taskPriority does the ordering of list of the tasks— ordering according to the precedence of the interrupt sources and tasks.
  - The scheduler first executes only the first task at the ordered list, and the total, equals the to period taken by the first task on at the list. It is deleted from the list after the first task is executed and the next task becomes the first.
- Cooperative Scheduling in the order of Ready Tasks using an Ordered List as per priority precedence
- The insertions and deletions for forming the ordered list are made only at the beginning of the cycle for each list

Worst-case latency

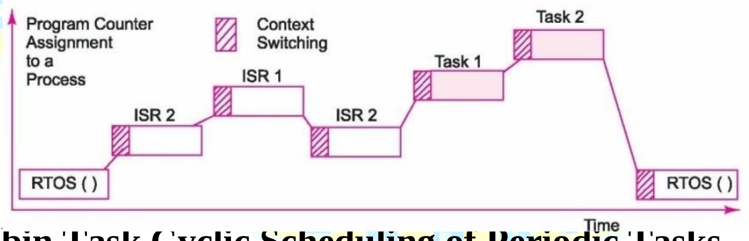
- Not Same for every task. Varies from (dti +
- sti + eti) p(m)} + tISR to{(dti + sti + eti)p1 + (dti + sti + eti) p2 +...+ (dti+ sti + eti) p(m-1) + (dti + sti + eti) p(m)} + tISR.

- tISR is the sum of all execution times for the ISRs
- For an i-th task, let the event detection time with when an event is brought into a list be is dti , switching time from one task to another be is sti and task execution time be is eti
- i = 1, 2, …, m 1 , m; m is number of ISRs

Cooperative Priority based scheduling of the ISRs executing in the first layer and Priority based ready tasks at an ordered list executing in the second layer


High
Priority
Low
ISRs as per Priority for Execution
main ( )
Call RTOS ( )
RTOS ( )
1
ISR 1
ISR 2
ISR N
3
2
5
4
Message for ready and run
6
8
Change to Another Process
Context Switch (Save or restore PC,SP,...) Before the IPC
Message to as
An IPC
7
* next
Task 1
* next
Task 2
Tasks in an ordered list with ordering as per priority

Program counter assignments at different times on the scheduler calls to the ISRs and the corresponding



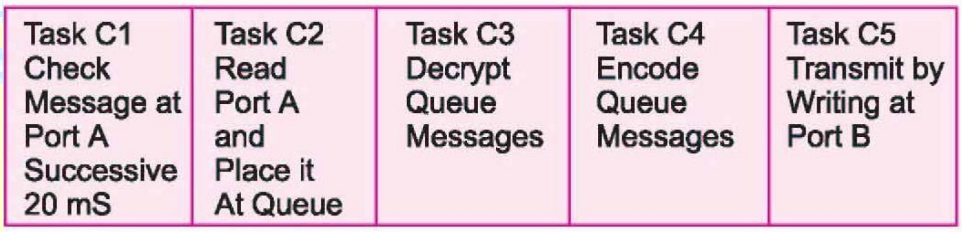
Round Robin Task Cyclic Scheduling of Periodic Tasks

Cyclic Scheduling Periodic tasks/

- Time Periodic Scheduling in the cyclic order
- Assume periodically occurring three tasks
- Let in time-frames allotted to the first task, the task executes at t1, t1 + Tcycle, t1+ 2 Tcycle, .., second task frames at t2, t2 + Tcycle, t2+ 2 Tcycle and third task at t3, t3 + Tcycle, t3+ 2 Tcycle, … .

- Start of a time frame is the scheduling point for the next task in the cycle.
- Tcycle is the cycle for repeating cycle of execution of tasks in order 1, 2 and 3 and equals start of task 1 time frame to end of task 3 frame.
- Tcycle is period after which each task time frame allotted to that repeats
- Case : tc Cycle = N □ Sum of the maximum times for each task
- Then each task is executed once and finishes in one cycle itself.
- When a task finishes the execution before the maximum time it can takes, there is a waiting period in-between period between two cycles.
- The worst-case latency for any task is then N □ Sum of the maximum times for each task. A task may periodically need execution. A task period for the its need ofrequired repeat execution of a task is an integral multiple of tcycle

Tasks C1 to C5 Cyclic Scheduling



Task C1 Check Message at Port A Successive 20 mS	Task C2 Read Port A and Place it At Queue	Task C3 Decrypt Queue Messages	Task C4 Encode Queue Messages	Task C5 Transmit by Writing at Port B

Task C1 to Task C5

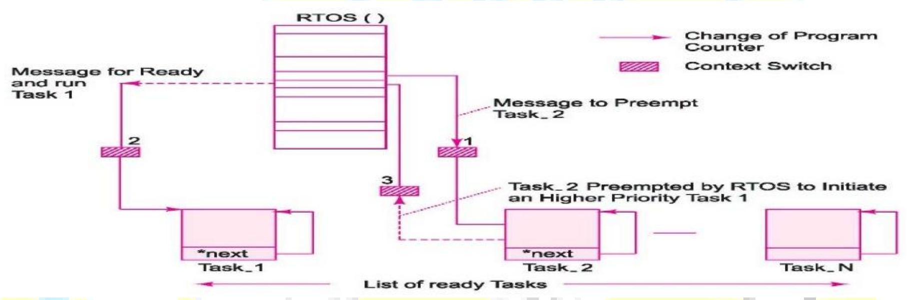
- Example of Video and audio signals
- Signals reaching at the ports in a multimedia system and processed
- The video frames reach at the rate of 25 in one second.
- The cyclic scheduler is used in this case to process video and audio with Tcycle = 40 ms or in multiples of 40 ms.

Preemptive Scheduling Model

- Preemptive Scheduling
- Cyclic Scheduling of periodic tasks and Round Robin Time Slicing Scheduling of equal priority tasks
- Cooperative Scheduling of ready tasks in a circular queue. It closely relates to function queue scheduling.
- Cooperative Scheduling with Precedence Constraints
- Scheduling using 'Earliest Deadline First' (EDF) precedence.

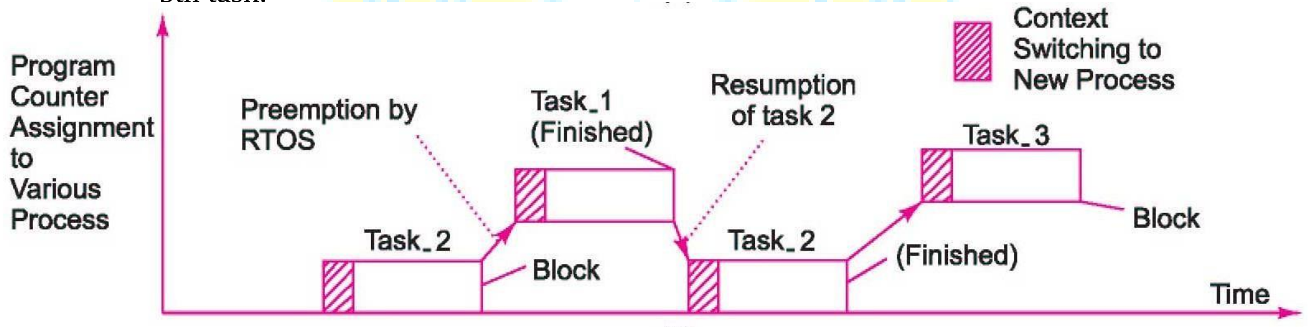
RTOS Preemptive Scheduling

- Processes execute such that scheduler provides for preemption of lower priority process by higher priority process.
- Assume priority of task_1 > task_2> task_3 > task_4…. > task N
- Each task has an infinite loop from start (Idle state) up to finish.
- Task 1 last instruction points to the next pointed address, *next. In case of the infinite loop, *next points to the same task 1 start.


RTOS ( )
Change of Program Counter
Context Switch
Message for Ready and run Task 1
Message to Preempt Task_2
2
1
3
Task_2 Preempted by RTOS to Initiate an Higher Priority Task 1
*next
Task_1
*next
Task_2
Task_N
List of ready Tasks

Worst-case latency

- Not Same for every task
- Highest priority task latency smallest
- Lowest priority task latency highest
- Different for different tasks in the ready list
- Tworst = {(dti + sti + eti)1 + (dti + sti + eti)2
- +...+ (dti + sti + eti)p-1 + (dti + sti + eti)p} + tISR.
- tISR is the sum of all execution times for the ISRs
- For an i-th task, let the event detection time with when an event is brought into a list be is dti , switching time from one task to another be is sti and task execution time be is eti
- i = 1, 2, …, p □1when number of higher priority tasks = p □1 for the pth task.



ISRs Handling

- Hardware polls to determine whether an ISR with a higher priority ISR than the present one needs the service at the end of an instruction during execution of an ISR, if yes, then the higher priority ISR is executed.

RTOS method for Preemptive Scheduling of tasks

An infinite loop in each task

- Each task design is like as an independent program, in an infinite loop between the task ready place and the finish place.
- The task does not return to the scheduler, as a function does.
- Within the loop, the actions and transitions are according to the events or flags or tokens.

When priority of task_1 > task_2 > task_3

- (1) At RTOS start, scheduler sends a message (Task_Switch_Flag) to task 1 to go to un-blocked state and run, and thus highest priority task 1 runs at start.
- (2) When task 1 blocks due to need of some input or wait for IPC or delay for certain period, a message (Task_Switch_Flag) will be sent to RTOS, task 1 context saves and the RTOS now sends a message (Task_Switch_Flag) to task 2 to go to un- blocked state and run.
- (3) Task 2 now runs on retrieving the context of task 2. When it blocks due to need of some input or wait for IPC or delay for certain period, a message (Task_Switch_Flag) will be sent to RTOS, task 2 context saves andan RTOS message (Task_Switch_Flag) makes the task 3 in un- blocked state. Task 3 will run now after retrieving the context of task 3.
- (4) If during running of task 3, either task 2 or task 1 becomes ready with the required input or IPC or delay period is over, task 3 is preempted, a message (Task_Switch_Flag) will be sent to RTOS, task 3 context saves, and task 1, and if task 1 not ready, then task 2 runs after retrieving the context of task 2.
- (5) A message (Task_Switch_Flag) is sent to RTOS after task 2 blocks due to wait of IPC or need of sum input and task 2 context saves and task 1 if ready then task 1 runs on retrieving the context of task 1
- (6) task 1 if not ready then task 3 runs on retrieving the context of task 3
- (7) Task 1 when ready to run preempts tasks 2 and 3, and Task 2 when ready to run preempts task 3
- Specifying timeout for waiting for the token or event
- Specify timeout for waiting for the token or event.
- An advantage of using timeout intervals while designing task codes is that worst- case latency estimation is possible.
- There is deterministic latency of each tasks

- Another advantage of using the timeouts is
- the error reporting and handling
- Timeouts provide a way to let the RTOS run even the preempted lowest priority task in needed instances and necessary ca

INTERRUPT ROUTINES IN RTOS EN VIRONMENT HANDELING OF INTERRUPT SOURCE CALLS

In a system the ISR should functions as following. 1.ISR have higher priorities over the OS functions and the applications tasks.an ISR does not wait for a semaphore mailbox message or queue message 2.An ISR does not also wait for mutex else it has to wait for other critical sections code to finish before the critical codes in the ISR can run. only the accept functon for these events can be used

There are three alternative systems for the used to respond to the hardware source calls from the interrupts

• The following sections explain the OS is to respond to the hardware source calls from the interrupts

1.direct call to an ISR by an interrupting.

2.RTOS first interrupting on an interrupt then 0S calling the corresponding ISR.

3.RTOS first interrupting on an interrupt then RTOS initiating the ISR and then an ISR

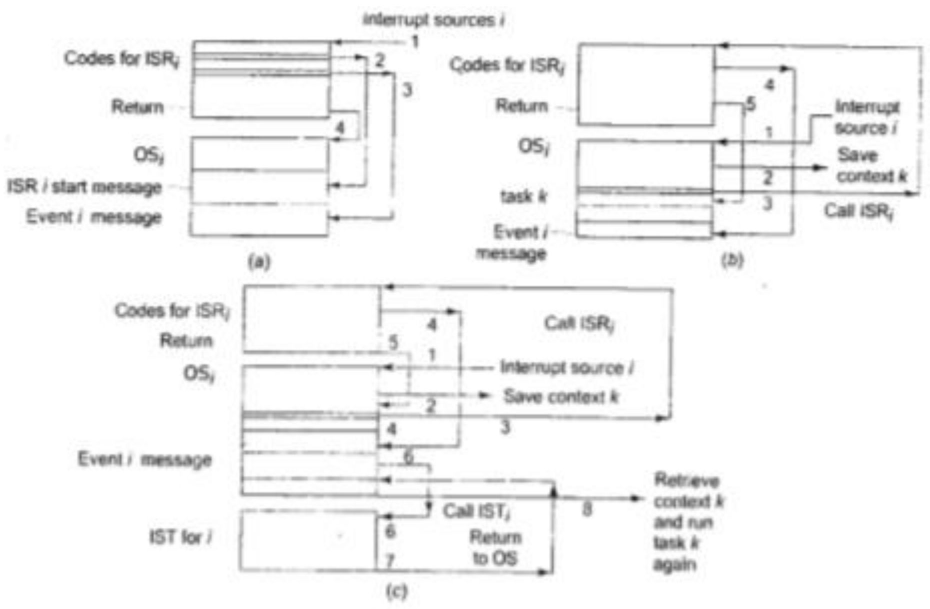
1.DIRECT CALL TO AN ISR BY AN INTERRUPTING

• On an interrupt the process running at the CPU is interrupted and the ISR corresponding to hat source starts executing(step1)

• A hardware source calls an ISR directly the ISR just sends an ISR enter to the OS (step2) • Later the ISR code can send in to a mailbox or queue(step3)

• But the task for the mailbox or queue does not start before the return from the ISR(step4)

• The ISR continues of the codes needed for the interrupt service till the ISR exit message the return(step4).



2.RTOS FIRST INTERRUPTING ON AN INTERRUPT,THEN OS CALLING THE CORRESPOMDING

• On an interrupt of a task system k-th task the OS first gets the hardware source call(step1).

• And initiates the corresponding ISR after saving present process status (step2)

• An i-th interrupt source causes the OS to get the notes of that after step1 it finish the critical code till the pre-emption point and calls the i-th ISR –I executs (step3).

•The preemptions point is the last instructions of the presently running OS function after which the ISR being of highest priority is called the ISR(step4). .Can post the event or mailbox messages to the 0S for initiating the j-th task or k-th task after the return(step5).

• From the ISR and after returning the j-th or k-th context.

• The system priorities are ISR and then tasks or (ISTS).IST is just a task on signal or message from an ISR .

3.RTOS FIRST INTERRUPTING ON AN INTERRUPT THEN RTOS INITIATING THE ISR AND THEN AN ISR

• An RTOS can provided for two-levels of ISR

1)Fast level ISR ,FLISR

2)Slow level ISR ,SLISR

• The fast level ISR can also be called hard ware interrupt ISR and SLISR as software interrupt ISR

• FLISR is called just the ISR in RTOS windows CE SLISR is called interrupt service thread (IST)in windows CE.

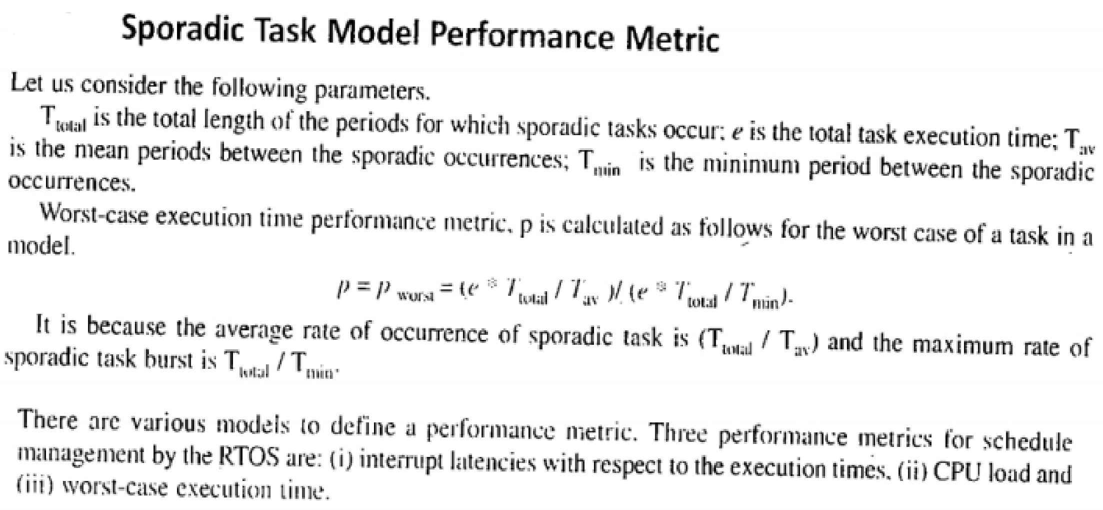
• The use of FLISR reduce the interrupt latency for an interrupt service and jitter (worst case and best case latencies difference)for an interrupt service.

• An IST functions as a deferred procedure call (DPC) of the ISR

• An i-th IST is thread to service an i-th interrupt source call. The ISR during executing then can send one or more outputs for the events and messages in to the mailbox for the IST

• For the per-emption from the same or other hardware sources

• The ISR in the FIFO that have received the message from the ISR executes(step6).



Sporadic Task Model Performance Metric

Let us consider the following parameters.

T_{total} is the total length of the periods for which sporadic tasks occur; e is the total task execution time; T_{av} is the mean periods between the sporadic occurrences; T_{min} is the minimum period between the sporadic occurrences.

Worst-case execution time performance metric, p is calculated as follows for the worst case of a task in a model.

$$p = p_{worst} = (e * T_{total} / T_{av}) / (e * T_{total} / T_{min}).$$

It is because the average rate of occurrence of sporadic task is (T_{total} / T_{av}) and the maximum rate of sporadic task burst is T_{total} / T_{min}.

There are various models to define a performance metric. Three performance metrics for schedule management by the RTOS are: (i) interrupt latencies with respect to the execution times, (ii) CPU load and (iii) worst-case execution time.

Latency and Deadlines as Performance Metric in Scheduling Models For Periodic, Sporadic and Aperiodic Tasks

An RTOS should quickly and predictably respond to the event. It should have minimum interrupt latency and fast context switching latency.

Different models have been proposed for measuring performances. Three performance metrics are as follows.

1. Ratio of the sum of interrupt latencies with respect to the sum of the execution times.
2. CPU load.
3. Worst-case execution time with respect to the mean execution time.

'Interrupt latencies' in various task models can be used for evaluating performance metrics. The latencies for various task scheduling models are described in Sections . . . The CPU load is another way to look at the performance. It is explained in Section . Worst-case performance calculation for a sporadic task is explained in Section

CPU Load as Performance Metric

Each task gives a load to the CPU that equals the task execution time divided by the task period. [Task period means period allocated for a task.] Recall In_AOut_B intranetwork of Example 4.1. Receiver port A expects another character before 172 μs. Task period is 172 μs. If the task execution time is also 172 μs, the CPU load for this task is 1 (100%). The task execution time when a character is received must be less than 172 μs. The maximum load of the CPU is 1 (less than 100%).

The CPU load or *system load* estimation in the case of multitasking is as follows. Suppose there are *m* tasks. For the multiple tasks, the *sum* of the CPU loads for all the tasks and ISRs should be less than 1. The time outs and fixed-time limit definitions for the tasks reduce the CPU load for the higher-priority tasks so that even the lower-priority tasks can be run before the deadlines. What does it mean when the sum of the CPU loads equal to 0.1 (10%)? It means that the CPU is underutilized and spends 90% of its time in a waiting mode. As the execution times and the task periods vary, the CPU loads can also vary.

When a task needs to run only once, then it is aperiodic (one shot) in an application. Scheduling of the tasks that need to run periodically with the fixed periods can be periodic and can be done with a CPU load very close to 1. An example of a periodic task is as follows. There may be inputs at a port with predetermined periods, and the inputs are in succession without any time gap.

When a task cannot be scheduled at fixed periods, its schedule is called sporadic. For example, if a task is expected to receive inputs at variable time gaps, then the task schedule is sporadic. An example is the packets from the routers in a network. The variable time gaps must be within defined limits.

A preemptive scheduler must take into account three types of tasks (aperiodic, periodic and sporadic) separately.

1. An aperiodic task needs to be preempted only once.
2. A periodic task needs to be preempted after the fixed periods and it must be executed before its next preemption is needed.
3. A sporadic task needs to be checked for preemption after a minimum time period of its occurrence. Usually, the strategy employed by the software designer is to keep the CPU load between (0.7 ± 0.25) for sporadic tasks.