Mailam Engineering College

Mailam (Po), Villupuram (Dt). Pin: 604 304
(Approved by AICTE, New Delhi, Affiliated to Anna University, Chennai
& Accredited by TATA Consultancy Services & NBA)

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

UNIT IV REAL TIME SYSTEMS

Structure of a Real Time System – Estimating program run times – Task Assignment and Scheduling – Fault Tolerance Techniques – Reliability, Evaluation – Clock Synchronization.

Prepared by

1. **Ramathilagam.B, Asso.Prof/ECE**
2. **Mohanadevi.C, Asso.Prof/ECE**

UNIT-4

PART A

1. What is Real-Time Operating Systems?

RT systems require specific support from OS Conventional OS kernels are inadequate w.r.t. RT requirements

a. Multitasking/scheduling
   i. provided through system calls
   ii. does not take time into account (introduce unbounded delays)
b. Interrupt management
   i. achieved by setting interrupt priority > than process priority
   ii. increase system reactivity but may cause unbounded delays on process execution even due to unimportant interrupts
c. Basic IPC and synchronization primitives
   i. may cause *priority inversion* (high priority task blocked by a low priority task)
d. No concept of RT clock/deadline

2. What are the Characteristics of Real-Time System?

- Timeliness
  - Achieved through proper scheduling algorithms
- Core of an RTOS!
- Predictability
  - Affected by several issues
- Characteristics of the processor (pipelinig, cache, DMA, ...)
- I/O & interrupts
- Synchronization & IPC
- Architecture
- Memory management
- Applications
- Scheduling!

3. What are the Features of Real-Time System?

- Timeliness
- OS has to provide mechanisms for

- time management
- handling tasks with explicit time constraints

– Predictability
- to guarantee in advance the deadline satisfaction
- to notify when deadline cannot be guaranteed

– Fault tolerance
- HW/SW failures must not cause a crash

– Design for peak load
- All scenarios must be considered

– Maintainability

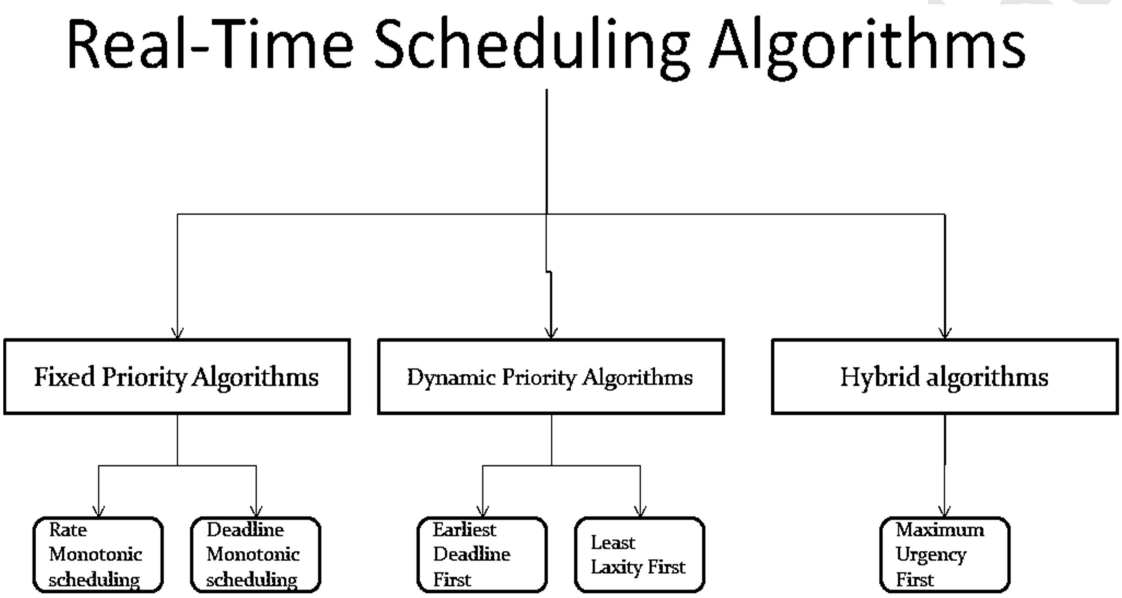
4. Write the RTOS timings?
- Interrupt Latency
- Scheduling Latency
- Cortex switch time
- Maximum system call time

5. Why we need for scheduling?
- Each computation (task) we want to execute needs resources
- Resources: processor, memory segments, communication, i/o devices etc.)
- The computation must be executed in particular order (relative to each other and/or relative to time)
- The possible ordering is either completely or statistically a priori known (described)
- Scheduling: assignment of processor to computations;
- Allocation: assignment of other resources to computations;

6. What are the task categories in real time system?
- Invocation
  - Periodic (time-triggered)
  - Aperiodic (event-triggered)
- Creation
  - Static
  - Dynamic

- Multi-Tasking System
  - Preemptive: higher-priority process taking control of the processor from a lower-priority
  - Non-Preemptive : Each task can control the CPU for as long as it needs it.

7. Write the real time scheduling algorithm?



8. Define initiation time and deadline for periodic and aperiodic process.

❖ **The initiation time** is the time at which the process goes from the **waiting to the ready state**. *An aperiodic process* is by definition *initiated by an event*, such as external data arriving or data computed by another process. The initiation time of *periodic process* is generally measured from that event, although the system may want to make the *process ready at some interval after the event itself.*

❖ **A deadline** specifies when a computation must be finished. *The deadline for an aperiodic* process is generally measured from the initiation time because that is the only reasonable time reference. *The deadline for a periodic process* may in general occur at some time other than the end of the period.

9. Define jitter

The **jitter** of a task, which is the *allowable variation* in the completion of the task.

10. Define DAG(Directed Acyclic Graph) and task graph or task set.

- The data dependencies must form *a directed acyclic graph (DAG)*—a cycle in the data dependencies is difficult to interpret in a periodically executed system.
- A set of processes with data dependencies is known as a **task graph or task set.**

11. Differentiate scheduling policy and Scheduling overhead

Scheduling policy	Scheduling overhead
Scheduling policy defines how **processes are selected for promotion from the ready state to the running state.** Every multitasking operating system implements some type of scheduling policy. Choosing the right scheduling policy not only ensures that the system will meet all its timing requirements, but it also has a profound influence on the CPU horsepower required to implement the system's functionality.	**Scheduling overhead**—the execution time **required to choose the next execution process**, which is incurred in addition to any context switching overhead

12. What is Rate-monotonic scheduling (RMS) (Nov/Dec 18)

- **Rate-monotonic scheduling (RMS)**was one of the first scheduling policies developed for real-time systems and is still very widely used.
- RMS is a **static scheduling policy** because *it assigns fixed priorities to processes*.
- It turns out that these fixed priorities are sufficient to efficiently schedule the processes in many situations.

13. Define kernel and time quantum

- ✓ The **kernel** is the part of the operating system that **determines what process is running**. The kernel is activated periodically by the timer.

- ✓ The **length of the timer** period is known as the **time quantum** because it is the smallest increment in which we can control CPU activity. The kernel determines what process will run next and causes that process to run. On the next timer interrupt, the kernel may pick the same process or another process to run.

14. How do we switch between processes before the process is done or define Context switching mechanism (or) Define Context switching in RTOS APR/MAY 18

Context switching mechanism:

- ✓ The timer interrupt causes control to change from the currently executing process to the kernel; assembly language can be used to save and restore registers.
- ✓ The set of registers that defines a process is known as its context, and **switching from one process's** register set **to another is known as context switching**.
- ✓ The data structure that holds the state of the process is known as the record.

15. Define Round-robin scheduling

- ✓ A common scheduling algorithm in general-purpose operating systems is round robin.
- ✓ All the **processes are kept on a list and scheduled one after the other**.
- ✓ This is generally combined with pre-emption so that one process does not grab all the CPU time.
- ✓ Round-robin scheduling provides a form of fairness in that all processes get a chance to execute.
- ✓ However, it does not guarantee the completion time of any task; as the number of processes increases, the response time of all the processes increases.

16. Define Race condition and how you will avoid race condition using critical section

- ✓ Consider the case in which an I/O device has a flag that must be tested and modified by a process.
- ✓ **Problems can arise when other processes may also want to access the device**.
- ✓ If combinations of events from the two tasks operate on the device in the wrong order, we may create a **critical timing race** or **race condition** that causes erroneous operation.

For example:

1. Task 1 reads the flag location and sees that it is 0.

2. Task 2 reads the flag location and sees that it is 0.

3. Task 1 sets the flag location to 1 and writes data to the I/O device's data register.

4. Task 2 also sets the flag to 1 and writes its own data to the device data register, overwriting the data from task 1.

In this case, both devices thought they were able to write to the device, but the task 1's write was never completed because it was overridden by task 2.

Critical sections

- ✓ To prevent this type of problem we need to control the order in which some operations occur.
- ✓ For example, we need to be sure that a **task finishes an I/O operation before allowing another task to starts its own operation** on that I/O device.
- ✓ We do so by enclosing sensitive sections of code in a **critical sectionthat executes without interruption.**

17. How will you call and release the critical section using Semaphores?(April 17)

- ✓ The semaphore is used to guard a resource.
- ✓ We start a critical section by calling a semaphore function that does not return until the resource is available.
- ✓ When we are done with the resource we use another semaphore function to release it. The semaphore names are, by tradition, **P() to gain access to the protected resource and V() to release it.**

/* some nonprotected operations here */

P(); / wait for semaphore */*

/* do protected work here */

V(); / release semaphore */*

This form of semaphore assumes that all system resources are guarded by the same P()/V() pair.

18. What is the function of Test-and-set

- ❖ To implement P() and V(), the microprocessor bus must support an **atomic read/write** operation, which is available on a number of microprocessors.
- ❖ The test-and-set allows us to implement semaphores. The P() operation uses a test and- set to repeatedly test a location that holds a lock on the memory block. The P() operation does not exit until the lock is available; once it is available, the test-and-set automatically sets the lock. Once past the P() operation, the process can work on the protected memory block.
- ❖ The V() operation resets the lock, allowing other processes access to the region by using the P() function.

19. Define priority inversion and priority inheritance(April 17)(Apr 19)

Shared resources cause a new and subtle scheduling problem:*a low-priority process blocks execution of a higher-priority process* by keeping hold of its resource, a phenomenon known as **priority inversion**.

Priority inheritance

The most common method for dealing with priority inversion is **priority inheritance:** promote the priority of any process when it requests a resource from the operating system. *The priority of the process temporarily becomes higher* than that of another process that may use the resource. This ensures that the process will continue executing once it has the resource so that it can finish its work with the resource, return it to the operating system, and allow other processes to use it. Once the process is finished with the resource, its priority is demoted to its normal value.

20. Define Earliest-deadline-first scheduling(EDF)

Earliest deadline first (EDF) is another well-known scheduling policy that was also studied by Liu and Layland. It is a *dynamic priority scheme—it changes process priorities during execution based on initiation times*. As a result, it can achieve higher CPU utilizations than RMS.

21. What do you know about RMS versus EDF

EDF can extract higher utilization out of the CPU, but it may be difficult to diagnose the possibility of an imminent overload. Because the scheduler does take some overhead to make scheduling decisions, a factor that is ignored in the schedulability analysis of both EDF and RMS, running a scheduler at very high utilizations is somewhat problematic. *RMS achieves lower CPU utilization* but is easier to ensure that all deadlines will be satisfied. In some applications, it may be acceptable for some processes to occasionally miss deadlines.

22. Define Interrupt latency. What are the several factors in both hardware and software affect interrupt latency?

Interrupt latency for an RTOS is the duration of time from the *declaration of a device interrupt to the completion of the device's requested operation.* Interrupt latency is critical because data may be lost when an interrupt is not serviced in a timely fashion.

Several factors in both hardware and software affect interrupt latency:

- The processor interrupt latency;
- The execution time of the interrupt handler;
- Delays due to RTOS scheduling.

23. What is the functions of POSIX RTOS (Nov/Dec 17)

- ✓ POSIX pipes
- ✓ POSIX message queues

24. List the advantages and limitations of priority based scheduling

Advantages:

Process with high priority executed first. Process are executed depending upon their priority.

Limitations

However process with low priority has to wait for long time until the high priority processes finishes the execution.

25. Define Evaluation.

Evaluation is a systematic determination of a subject's merit, worth and significance, using criteria governed by a set of standards. It can assist an organization, program, design, project or any other intervention or initiative to assess any aim, realisable concept/proposal, or any alternative, to help in decision-making; or to ascertain the degree of achievement or value in regard to the aim and objectives and results of any such action that has been completed. The primary purpose of evaluation, in addition to gaining insight into prior or existing initiatives, is to enable reflection and assist in the identification of future change.

26. What is meant by Clock Synchronization?

Clock synchronization is a topic in computer science and engineering that aims to coordinate otherwise independent clocks. Even when initially set accurately, real clocks will differ after some amount of time due to clock drift, caused by clocks counting time at slightly different rates. There are several problems that occur as a result of clock rate differences and several solutions, some being more appropriate than others in certain contexts.

27. What is meant by fault?

Fault is an erroneous state of software or hardware resulting from failures of its components

a. Design errors
b. Manufacturing Problems
   - External disturbances
   - Harsh environmental conditions

- System Misuse

PART-B

1. Draw the neat structure of Real-Time system?

Real-time systems have been defined as: "*those systems in which the correctness of the system depends not only on the logical result of the computation, but also on the time at which the results are produced*"

- Correct function at correct time
- Usually embedded
- Deadlines
- Hard real-time systems
- Soft real-time systems

Soft RTS: meet timing constraints most of the time, it is not necessary that every time constraint be met. Some deadline miss is tolerated.

Hard RTS: meet all time constraints exactly, Every resource management system must work in the correct order to meet time constraints. No deadline miss is allowed.

- A real-time system is a system whose specification includes both a logical and a temporal correctness requirement.
  - Logical correctness: produce correct output.
    - Can be checked by various means including Hoare axiomatics and other formal methods.
  - Temporal correctness: produces output at the right time.
    - A soft real-time system is one that can tolerate some delay in delivering the result.
    - A hard real-time system is one that can not afford to miss a deadline.

Why we need scheduling ?

- Each computation (task) we want to execute needs resources

- Resources: processor, memory segments, communication, I/O devices etc.)
- The computation must be executed in particular order (relative to each other and/or relative to time)
- The possible ordering is either completely or statistically a priori known (described)
- Scheduling: assignment of processor to computations;

allocation: assignment of other resources to computations

Characteristics of Real time System

- Event-driven, reactive
- High cost of failure
- Concurrency/multiprogramming
- Stand-alone/continuous operation.
- Reliability/fault tolerance requirements
- Predictable behavior
- There is no science in real-time-system design.

– We shall see…

- Advances in supercomputing hardware will take care of real-time requirements.

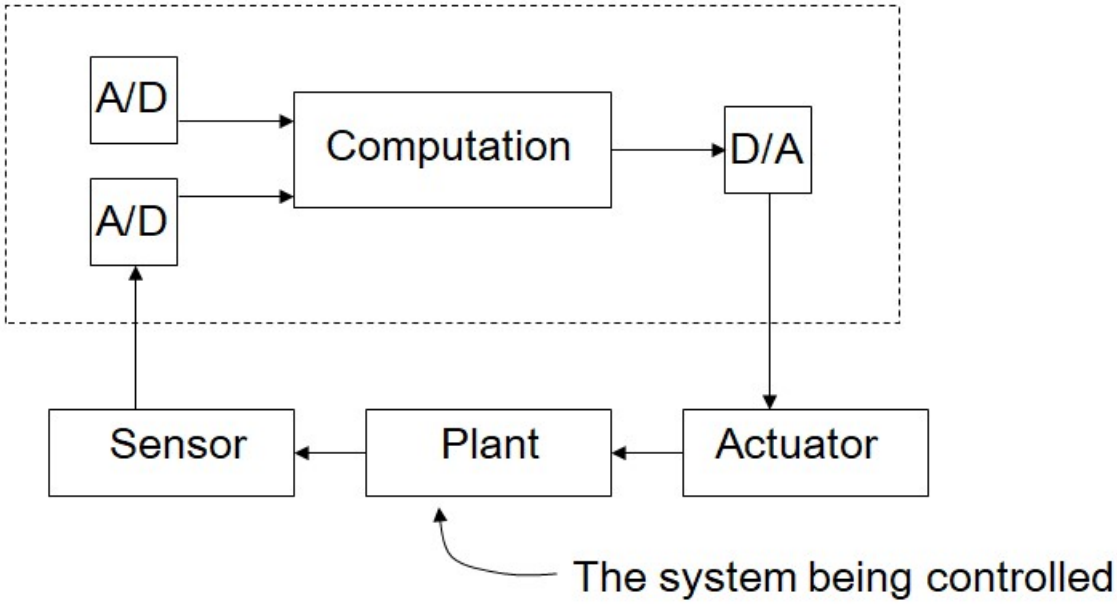
– The old "buy a faster processor" argument…

- Real-time computing is equivalent to fast computing.

– Only to ad agencies. To us, it means PREDICTABLE computing.

- Many real-time systems are **control systems**.

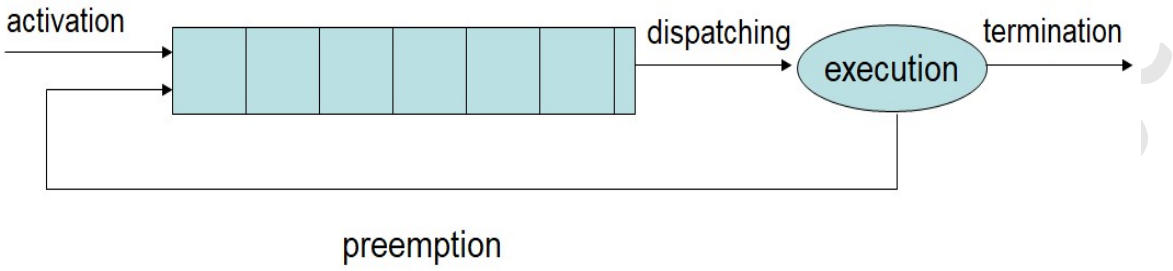
Example: A simple one-sensor, one-actuator control system.



2. Explain in detail about Scheduling Process?

- Scheduling
  define a policy of how to order tasks such that a metric is maximized/minimized
  - Real-time: guarantee hard deadlines, minimize the number of missed deadlines, minimize lateness
- Dispatching
  carry out the execution according to the schedule
  - Preemption, context switching, monitoring, etc.
- Admission Control Filter tasks coming into the systems and thereby make sure the admitted workload is manageable
- Allocation designate tasks to CPUs and (possibly) nodes. Precedes scheduling

- Scheduling is the issue of ordering the use of system resources
  - A means of predicting the worst-case behaviour of the system



Why we need scheduling ?

- Each computation (task) we want to execute needs resources
- Resources: processor, memory segments, communication, I/O devices etc.)
- The computation must be executed in particular order (relative to each other and/or relative to time)
- The possible ordering is either completely or statistically a priori known (described)
- Scheduling: assignment of processor to computations;

Real-time Scheduling Taxonomy

- **Job (J_{ij})**: Unit of work, scheduled and executed by system. Jobs repeated at regular or semi-regular intervals modeled as periodic
- **Task (T_i)**: Set of related jobs.
- Jobs scheduled and allocated resources based on a set of scheduling algorithms and access control protocols**.**
- ***Scheduler***: Module implementing scheduling algorithms
- ***Schedule***: assignment of all jobs to available processors, produced by *scheduler*.
- ***Valid schedule***: All jobs meet their deadline

- Clock-driven scheduling vs Event(priority)-driven scheduling
- Fixed Priority vs Dynamic Priority assignment

Scheduling Periodic Tasks

- In hard real-time systems, set of tasks are known *apriori*
- Task T_i is a series of periodic Jobs J_{ij}. Each task has the following parameters
  - t_i - period, minimum interrelease interval between jobs in Task T_i.
  - c_i - maximum execution time for jobs in task T_i.
  - r_{ij} - release time of the j^{th} Job in Task i (J_{ij} in T_i).
  - φ_i - phase of Task T_i, equal to r_{i1}.
  - u_i - utilization of Task $T_i = c_i / t_i$
- In addition the following parameters apply to a set of tasks
  - H - Hyperperiod = Least Common Multiple of p_i for all i: $H = lcm(p_i)$, for all i.
  - U - Total utilization = Sum over all u_i.
- Schedulable utilization of an algorithm U_s
  - If $U < U_s$ the set of tasks can be guaranteed to be scheduled
- Static Scheduling:
  - All scheduling decisions at compile time.
    - Temporal task structure fixed.
    - Precedence and mutual exclusion satisfied by the schedule (implicit synchronization).
    - One solution is sufficient.

- Any solution is a sufficient schedulability test.

- Benefits
  - Simplicity
- Dynamic Scheduling:
  - All scheduling decisions at run time.
    - Based upon set of ready tasks.
    - Mutual exclusion and synchronization enforced by explicit synchronization constructs.
  - Benefits
    - Flexibility.
    - Only actually used resources are claimed.
  - Disadvantages
    - Guarantees difficult to support
    - Computational resources required for scheduling

Scheduling Algorithm
Preemptive vs. Nonpreemptive

- Nonpreemptive Scheduling:
  - Tasks remain active till completion
  - Scheduling decisions only made after task completion.
- Benefits:
    - Reasonable when
    - task execution times ~= task switching times.
    - Less computational resources needed for scheduling

- Disadvantages:
  - Can leads to starvation (not met the deadline) especially for those real time tasks (or high priority tasks).

Non Real time Scheduling

- Primary Goal: maximize performance
- Secondary Goal: ensure fairness
- Typical metrics:
  - Minimize response time
  - Maximize throughput
  - E.g., FCFS (First-Come-First-Served), RR (Round-Robin)

Rate Monotonic scheduling

Example: Non-preemptive FCFS Scheduling

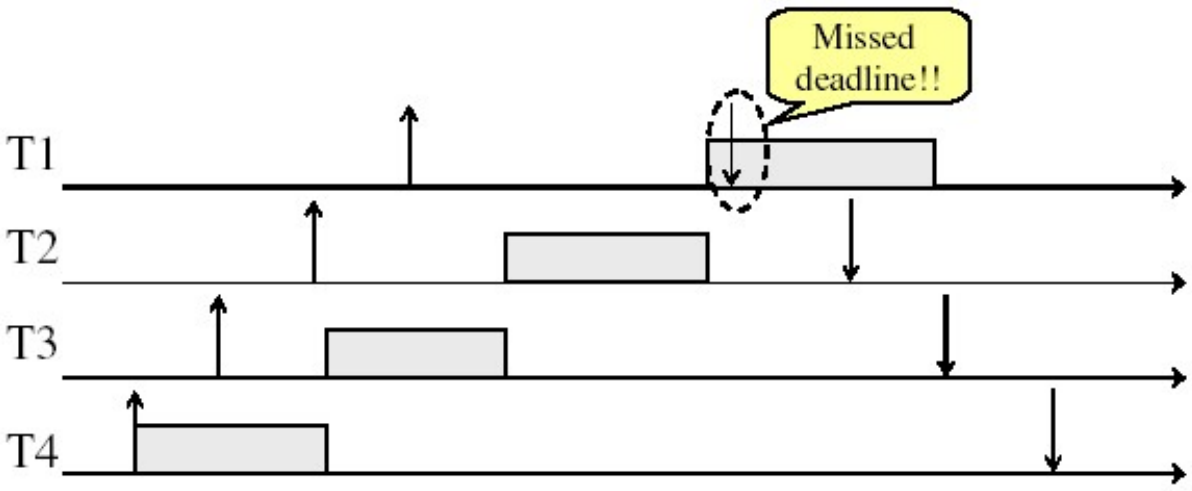
- Priority assignment based on rates of tasks
- Higher rate task assigned higher priority
- Schedulable utilization = 0.693 (Liu and Leyland)

  Where C_i is the computation time, and T_i is the release period
- If U < 0.693, schedulability is guaranteed
- Tasks may be schedulable even if U > 0.693

RM example

Process	Execution Time	Period

P1	1	8
P2	2	5
P3	2	10



Rate Monotonic Scheduling: Principle

- Each process is assigned a (unique) priority based on its period (rate); always execute active job with highest priority
- The shorter the period the higher the priority
- (1 = low priority)
- W.l.o.g. number the tasks in reverse order of priority

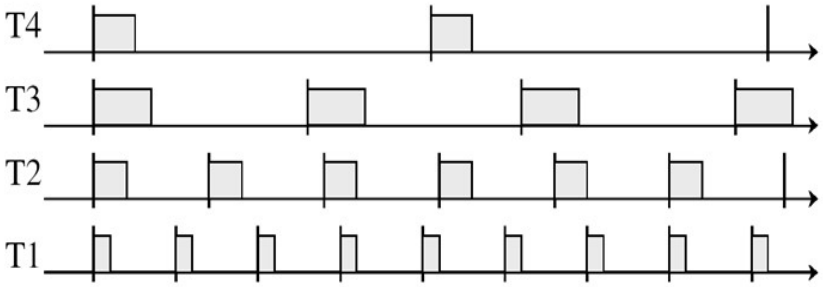
Rate

Process	Period	Priority	Name
A	25	5	T1
B	60	3	T3
C	42	4	T2
D	105	1	T5
E	75	2	T4

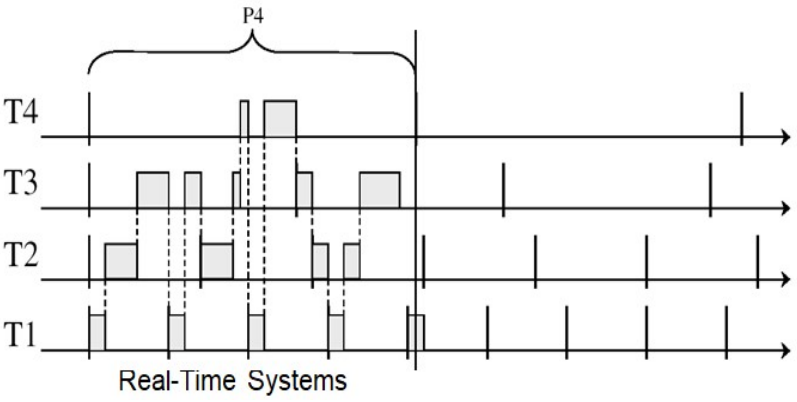
Example:

Monotonic Scheduling

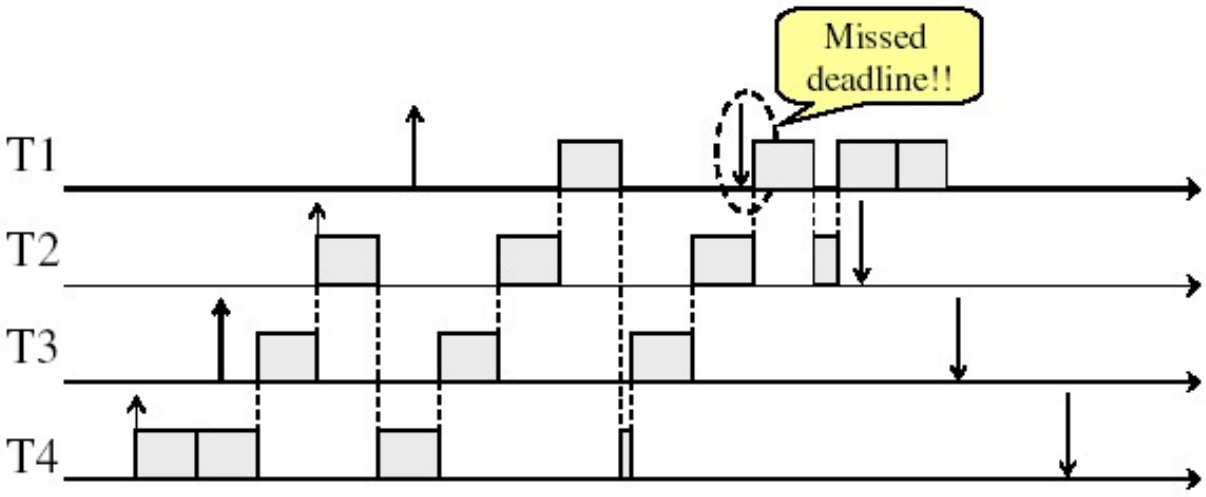
- Example instance



- RMA - Gant chart



Example: Round-Robin Scheduling



The time slots are of equal size, some short processes may have time left over in their time slot. We can use utilization as a schedulability measure: the total CPU time of all the processes must be less than the hyper period. Another scheduling policy that is slightly more sophisticated is ***round robin***. As illustrated in Figure round robin uses the same hyper period as does cyclostatic. It also evaluates the processes in order. But unlike cyclostatic scheduling, if a process

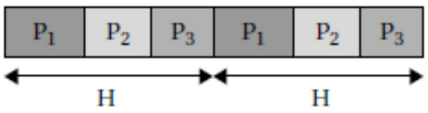


FIGURE 6.7
Cyclostatic scheduling.

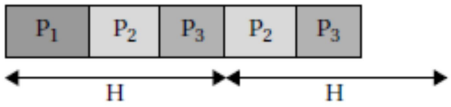


FIGURE 6.8
Round-robin scheduling.

Earliest deadline Scheduling:

- ***Earliest deadline first (EDF)*** is another well-known scheduling policy that was also studied by Liu and Layland [Liu73]. It is a dynamic priority scheme—it changes process priorities during execution based on initiation times.

- As a result, it can achieve higher CPU utilizations than RMS. The EDF policy is also very simple: It assigns priorities in order of deadline. The highest-priority process is the one whose deadline is nearest in time,and the lowestpriority process is the one whose deadline is farthest away. Clearly, priorities must be recalculated at every completion of a process.

- However, the final step of the OS during the scheduling procedure is the same as for RMS—the highest-priority ready process is chosen for execution. Example 6.4 illustrates EDF scheduling in practice Liu and Layland showed that EDF can achieve 100% utilization.A feasible schedule exists if the CPU utilization (calculated in the same way as for RMA) is _1.

- They also showed that when an EDF system is overloaded and misses a deadline, it will run at 100% capacity for a time before the deadline is missed.

The implementation of EDF is more complex than the RMS code. Figure 6.13 outlines one way to implement EDF.

- The major problem is keeping the processes sorted by time to deadline—since the times to deadlines for the processes change during execution, we cannot presort the processes into an array, as we could for RMS. To avoid resorting the entire set of records at every change,
- we can build a binary tree to keep the sorted records and incrementally update the sort. At the end of each period,we can move the record to its newplace in the sorted list by deleting it from the tree and then adding it back to the tree using standard tree manipulation techniques.

- We must update process priorities by traversing them in sorted order, so the incremental sorting routines must also update the linked list pointers that let us traverse the records in deadline order.

- After putting in the effort to building the sorted list of records, selecting the next executing process is done in a manner similar to that of RMS. However, the dynamic sorting adds complexity to the entire scheduling process. Each update of the sorted list requires *O(log n) s*teps. The EDF code is also significantly more complex than the RMS code.Since the system is schedulable!

3. Write a short note on Task Assignment?

- Cyclic executive scheduling (-> later)
- Cooperative scheduling
  - scheduler relies on the current process to give up the CPU before it can start the execution of another process
- A *static priority-driven scheduler* can **preempt** the current process to start a new process. Priorities are set pre-execution
  - E.g., Rate-monotonic scheduling (RMS), Deadline Monotonic scheduling (DM)
- A *dynamic priority-driven* scheduler can assign, and possibly also redefine, process priorities at run-time.
  - E.g., Earliest Deadline First (EDF), Least Laxity First (LLF)

- Fixed set of processes (tasks)
- Processes are periodic, with known periods
- Processes are independent of each other
- System overheads, context switches etc, are ignored (zero cost)
- Processes have a deadline equal to their period
  - i.e., each process must complete before its next release
- Processes have fixed worst-case execution time (WCET)

Temporal Scope of a Task

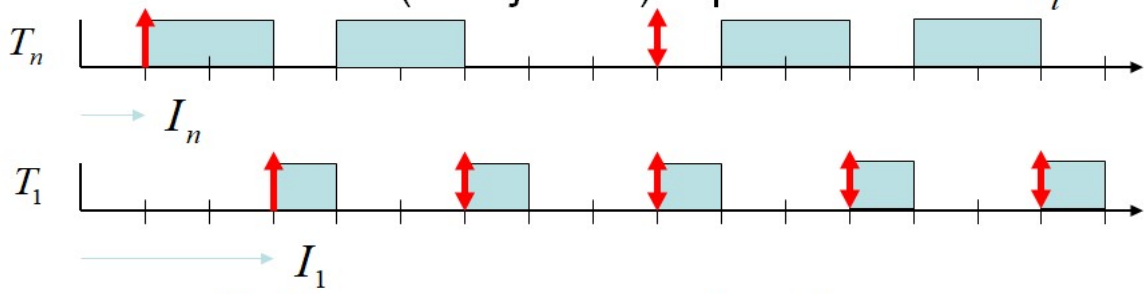
- *C* - Worst-case execution time of the task
- *D* - Deadline of tasks, latest time by which the task should be complete
- *R* - Release time
- *n* - Number of tasks in the system
  - Priority of the task
- *P* - Minimum inter-arrival time (period) of the task
  - Periodic: inter-arrival time is fixed
  - Sporadic: minimum inter-arrival time
  - Aperiodic: random distribution of inter-arrival times
- *J* - Release jitter of a process

Performance Metrics

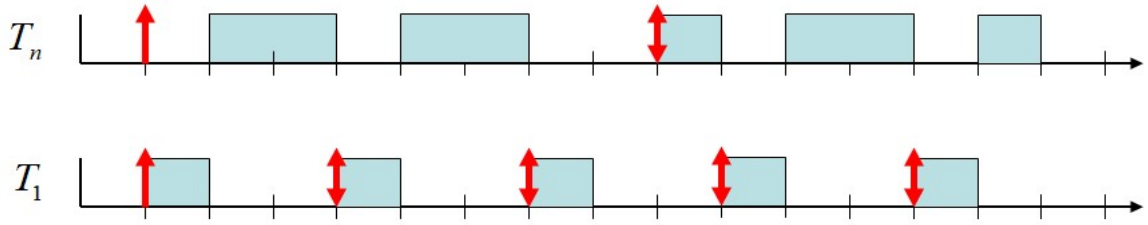
- Completion ratio / miss ration
- Maximize total usefulness value (weighted sum)
- Maximize value of a task
- Minimize lateness
- Minimize error (imprecise tasks)
- Feasibility (all tasks meet their deadlines)

Task Phases

- **Phase:** I_i
  release time of the (first job of) a periodic task T_i



- Two tasks T_i, T_j are <u>in phase</u> if $I_i = I_j$



Two Periodic Tasks

- Execution profile of two periodic tasks
  - Process A
    - Arrives 0 20 40 …
    - Execution Time 10 10 10 …
    - End by 20 40 60 …
  - Process B
    - Arrives 0 50 100 …
    - Execution Time 25 25 25 …
    - End by 50 100 150 …

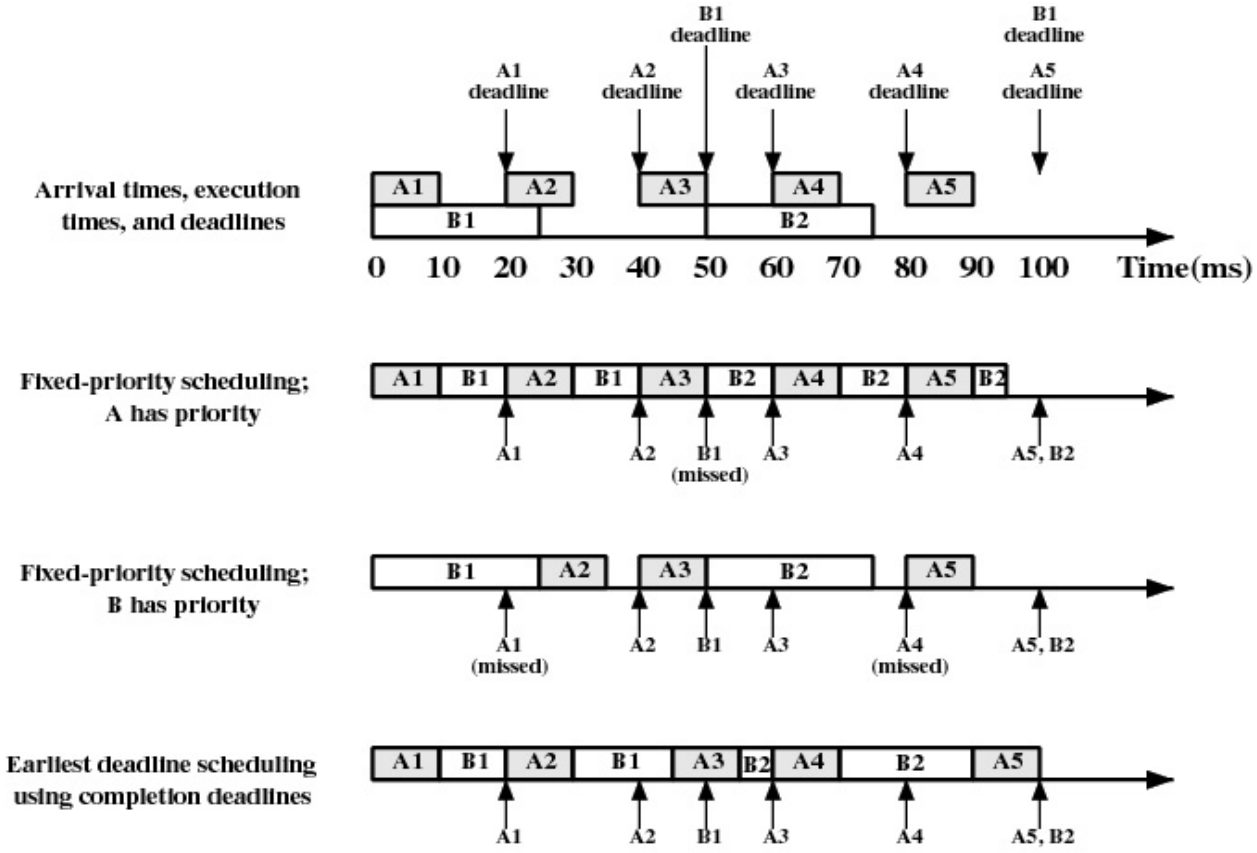


Figure 10.5 Scheduling of Periodic Real-time Tasks with Completion Deadlines

Five Periodic Tasks

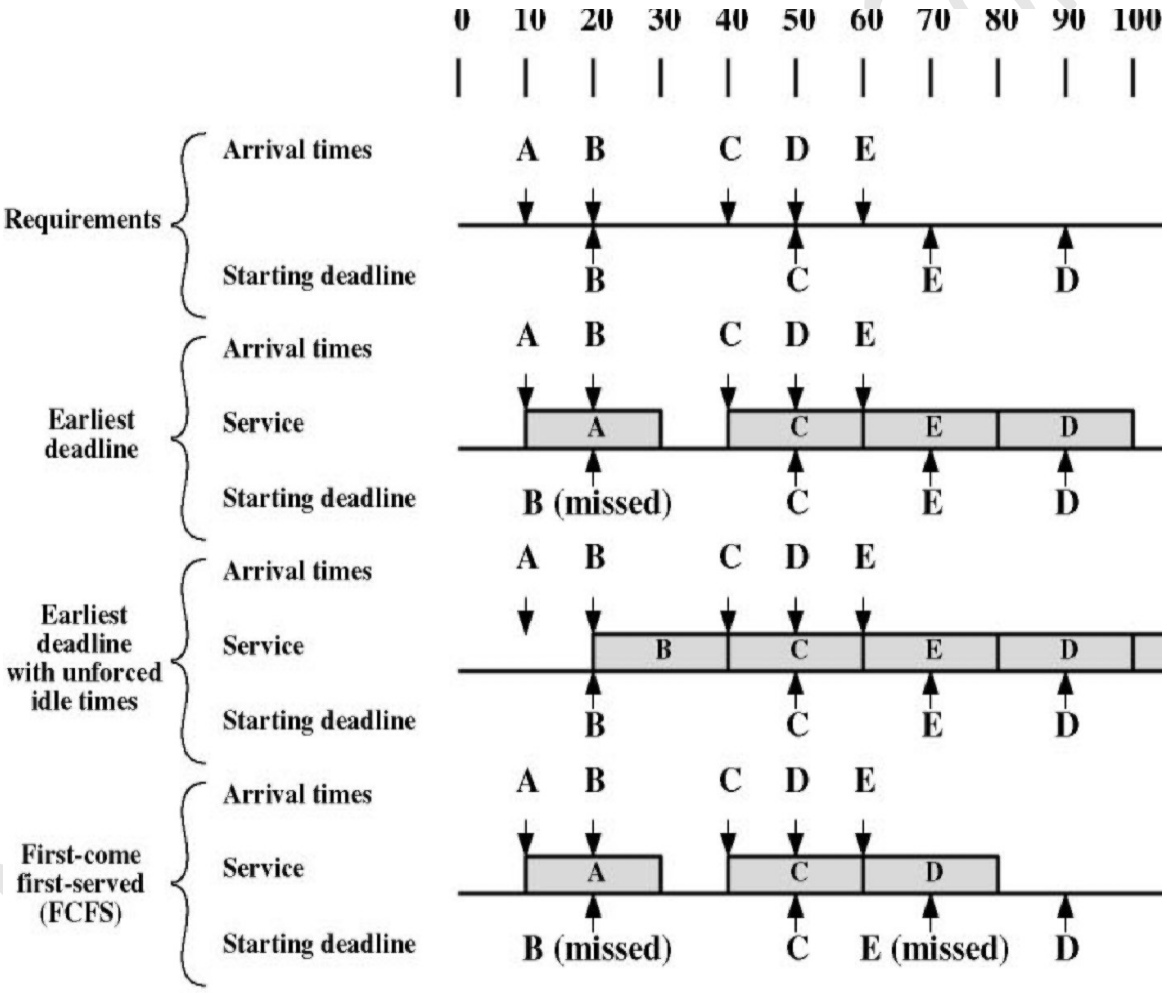
Process	Arrival Time	Execution Time	Starting Deadline
A	10	20	110
B	20	20	20
C	40	20	50
D	50	20	90

E	**60**	**20**	**70**

Least Laxity First (LLF)

- Dynamic preemptive scheduling with dynamic priorities
- Laxity : The difference between the time until a tasks completion deadline and its

-



- Remaining processing time requirement.
- a laxity is assigned to each task in the system and minimum laxity tasks are

executed first.

- Larger overhead than EDF due to higher number of context switches caused by laxity changes at run time
  - Less studies than EDF due to this reason

4. Write briefly about Fault Tolerance Technique?

What is a fault?

Fault is an erroneous state of software or hardware resulting from failures of its components

- Design errors
- Manufacturing Problems
- External disturbances
- Harsh environmental conditions
- System Misuse

Fault Sources

Mechanical -- "wearsout"

a. Deterioration: wear, fatigue, corrosion
b. Shock: fractures, overload, etc.

Electronic Hardware -- "bad fabrication; wearsout"

c. Latent manufacturing defects
d. Operating environment: noise, heat, ESD, electro-migration
e. Design defects

Software -- "baddesign"

f. Design defects
g. "Code rot" -- accumulated run-time faults

People

h. Can take a whole lecture content...

Fault and Classification

Failure: Component does not provide service
Fault: A defect within a system
Error: A deviation from the required operation of the system or subsystem

Extent: Local (independent) or Distributed (related)

Value*:*

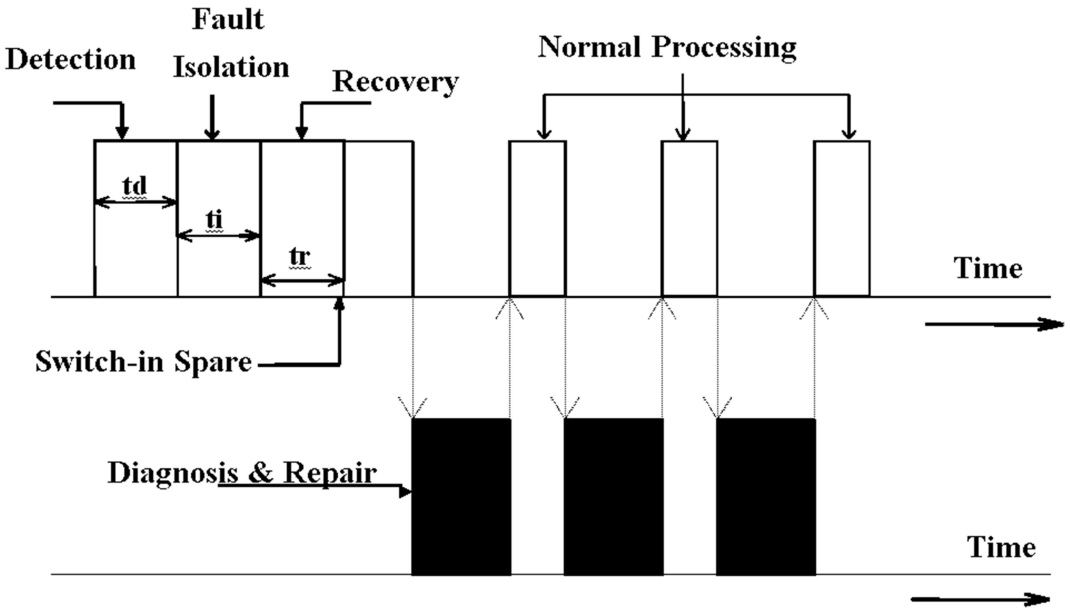
- Determinate
- Indeterminate (varying values)

Duration*:*

i. Transient
j. Intermittent
k. Permanent

Fault-Tolerant Computing

Main aspects of Fault Tolerant Computing (FTC): Fault detection, Fault Isolation and containment, System recovery, Fault Diagnosis Repair



Tolerating Faults

There is four-fold categorisation to deal with the system faults and increase system reliability and/or availability.

Methods for Minimising Faults

l. **Fault Avoidance**: How to prevent the fault occurrence. Increase reliability by

conservative design and use high reliability components.

Fault Tolerance: How to provide the service complying with the specification in spite of faults having occurred or occurring.

Fault Removal: How to minimise the presence of faults.

Fault Forecasting: How to estimate the presence, occurrence, and the consequences of faults.

Fault-Tolerance is the ability of a computer system to survive in the presence of faults.

Fault Tolerance Technique

- Hardware Fault Tolerance
- Software Fault Tolerance

Hardware Fault Tolerance

Fault Detection

Redundancy (masking, dynamic)

Use of extra components to mask the effect of a faulty component. (Static and Dynamic)

Redundancy alone does not guarantee fault tolerance. It guarantee higher fault arrival rates (extra hardware).

Redundancy Management is Important

A fault tolerant computer can end up spending as much as 50% of its throughput in managing redundancy.

Detection of a failure is a challenge

Many faults are latent that show up later

Fault detection gives warning when a fault occurs.

Duplication: Two identical copies of hardware run the same computation and compare each other results. When the results do not match a fault is declared.

Error Detecting Codes: They utilise information redundancy

Redundancies

Static and Dynamic Redundancy
Extra components mask the effect of a faulty component.

Masking Redundancy

Static redundancy as once the redundant copies of an element are installed, their interconnection remains fixed e.g. N-tuple modular redundancy (nMR), ECC, TMR (Triple Modular Redundancy) 3 identical copies of modules provide separate results to a voter that produces a majority vote at its output.

Dynamic Redundancy System configuration is changed in response to faults. Its success largely depends upon fault detection ability.

Hardware based fault-tolerance

Hardware based fault-tolerance provides tolerance against physical i.e.

hardware faults.

How to tolerate design/software faults?

It is virtually impossible to produce fully correct software.

We need something:

Toprevent software bugs from causing system disasters. Tomask out software bugs.

Tolerating unanticipated design faults is much more difficult than

tolerating anticipated physical faults.

Software Fault Tolerance is needed as:

Software bugs will occur no matter what we do. No fully dependable way of eliminating these bugs. These bugs have to be tolerated.

Software Fault Tolerance

How to Tolerate Software Faults?

Software fault-tolerance uses design redundancy to mask residual design faults of software programs.

Software Fault Tolerance Strategy

Defensive Programming

If you can not be sure that what you are doing is correct.

Do it in many ways.

Review and test the software.

Verify the software.

Execute the specifications.

Produce programs automatically.

Fault Recovery Technique

Fault recovery technique's success depends on the detection of faults accurately and as early as possible.

Three classes of recovery procedures:

Full Recovery

It requires all the aspects of fault tolerant computing.

Degraded recovery: Also referred as graceful degradation. Similar to full recovery but no

subsystem is switched-in.

Defective component is taken out of service.

Suited for multiprocessors.

Safe Shutdown

5. Write the short note on Reliability and Evaluation?

Reliability is **defined** as the probability that a **system** works properly for a specific period of time in the existence of probable failures. Different **reliability** analysis and improving techniques have been proposed in the literature.

- The reliability, $R_F(t)$ of a system is the probability that no fault of the class F occurs (i.e. system survives) during time t.

$$R_F(t) = P(t_{init} \leq t < t_f \forall f \in F)$$

where t_{init} is time of introduction of the system to service,
t_f is time of occurrence of the first failure f drawn from F.

- Failure Probability, $Q_F(t)$ is complementary to $R_F(t)$

$$R_F(t) + Q_F(t) = 1$$

- We can take off the F subscript from $R_F(t)$ and $Q_F(t)$

- When the lifetime of a system is exponentially distributed, the reliability of the system is: $R(t) = e^{-\lambda t}$ where the parameter λ is called the failure rate

Advantage of reliability centered maintenance (RCM)

• Can be most efficient maintenance program.

- Lower costs by eliminating unnecessary maintenance or overhauls.
- Reduced probability of sudden equipment failures.
- Able to focus maintenance activities on critical components.
- Increased component reliability.
- Incorporate root cause analysis.

Disadvantage of reliability centered maintenance (RCM)

- Can have significant start-up cost, training, equipment, etc.
- Savings potential not readily seen by management.

Evaluation is a systematic determination of a subject's merit, worth and significance, using criteria governed by a set of standards. It can assist an organization, program, design, project or any other intervention or initiative to assess any aim, realisable concept/proposal, or any alternative, to help in decision-making; or to ascertain the degree of achievement or value in regard to the aim and objectives and results of any such action that has been completed. The primary purpose of evaluation, in addition to gaining insight into prior or existing initiatives, is to enable reflection and assist in the identification of future change.

Evaluation is often used to characterize and appraise subjects of interest in a wide range of human enterprises, including the arts, criminal justice, foundations, non-profit organizations, government, health care, and other human services. It is long term and done at the end of a period of time.