Real Time Systems

Prof. Neelamani Samal
Department Of Computer Science and Engineering

Real Time Systems - Introduction

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

- A real time system has performance deadlines on its computations and actions.
- Deadline: time when execution must be completed. A deadline is either a point in time (time-driven) or a delta-time interval (event- driven) by which a system action must occur.

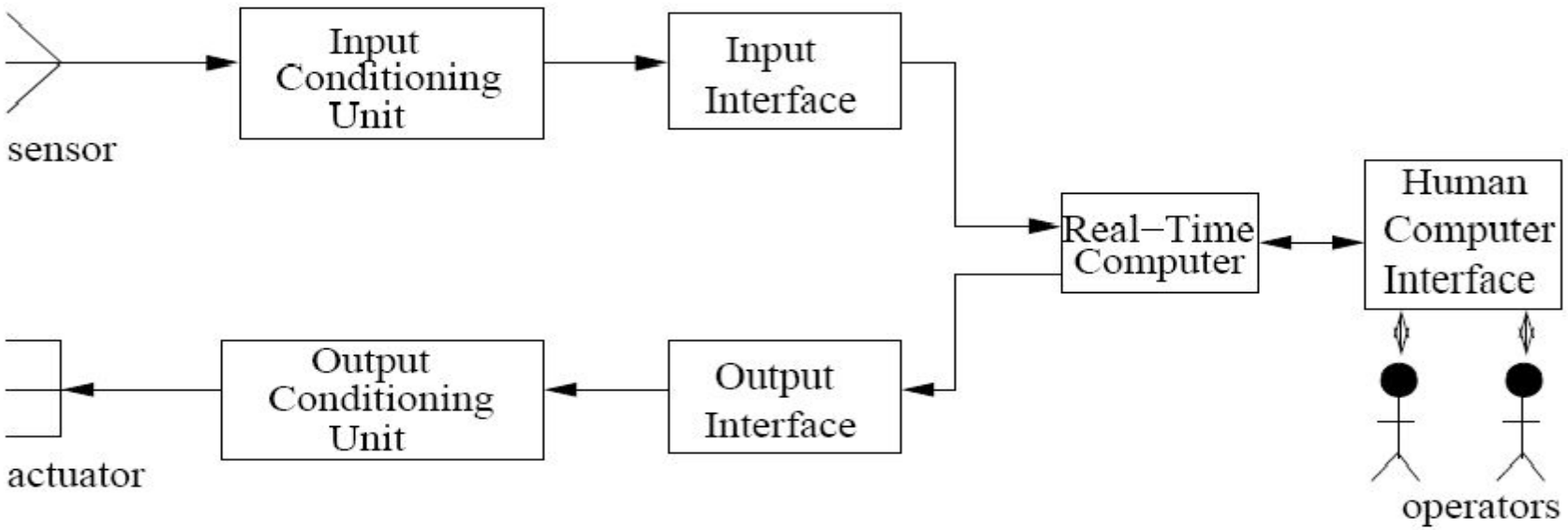
  Hard deadline: late data may be a bad data. Soft deadline: late data may be a good data.

- Hard real-time systems (e.g., Avionic control).
- Firm real-time systems (e.g., Banking).
- Soft real-time systems (e.g., Video on demand).

Applications

I. Industrial Applications(Chemical Plant Control, Automated Car Assembly Plant)
II. Medical(Robotused in recovery of displaced radioactive material)
III. Peripheral Equipment(Laser Printer, Scanner)
IV. Automotive and Transportation(Multi Point fuelinjection System)
V. Telecommunication application(Cellular Systems)
VI. Aerospace(Computer on Board an aircraft)
VII. Internet and Multimedia(Video Conferencing)
VIII. Consumer Electronics(Cell Phones)
IX. Defense applications(Missile Guidance System)
X. Miscellaneous Applications(Railway Reservation System)

Basic Model of a RTS



Characteristics of RTS

1. Time constraints(dead line associated with tasks)
2. New Correctness Criterion(not only logical correctness, time is also important)
3. Embedded(RTS are embedded in nature)
4. Safety – Criticality(failure of the system cause extensive damages)
5. Concurrency(RTS must process concurrent data)
6. Distributed and feedback structure
7. Task Criticality(is a measure of the cost of failure of a task)
8. Custom hardware
9. Reactive(RTS are often reactive means an ongoing interaction between system and environment is maintained)
10. Stability
11. Exception handling

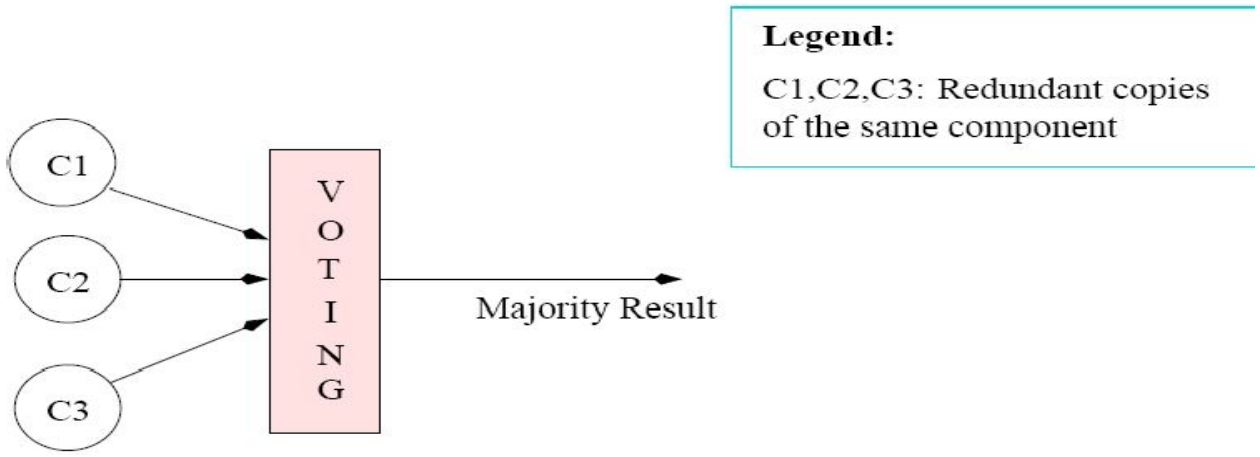
Safety and Reliability

- In Traditional system safety and reliability are independent issues
- In RTS safety and reliability are dependent issues
- **Fail safe state**: of a system is one which if entered when the system fails ,no damage would result.
- **Safety critical systems**: is one whose failure can cause severe damages

How to achieve high reliability

Steps to achieve highly reliable software

1. Error avoidance
2. Error detection and removal
3. Fault tolerance Hardware Fault Tolerance
4. Built In Self Test
5. Triple modular redundancy



Software fault tolerance

1. N-version programming
   - an adaptation of TMR(Triple modular Redundancy) Technique
   - Is not very successful in achieving fault tolerance
2. Recovery block

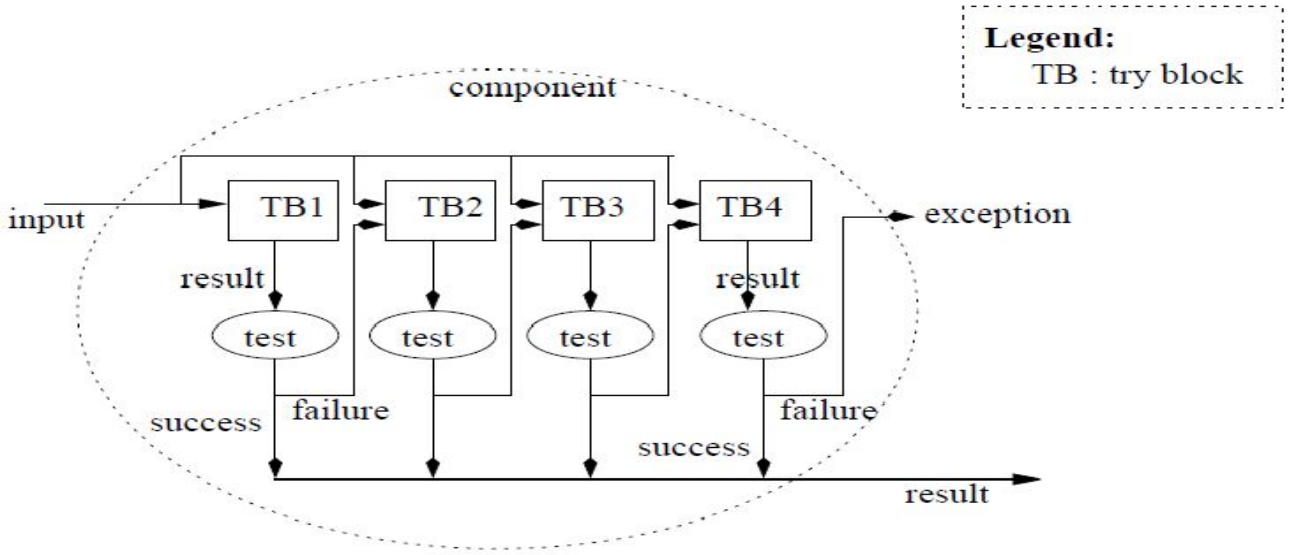
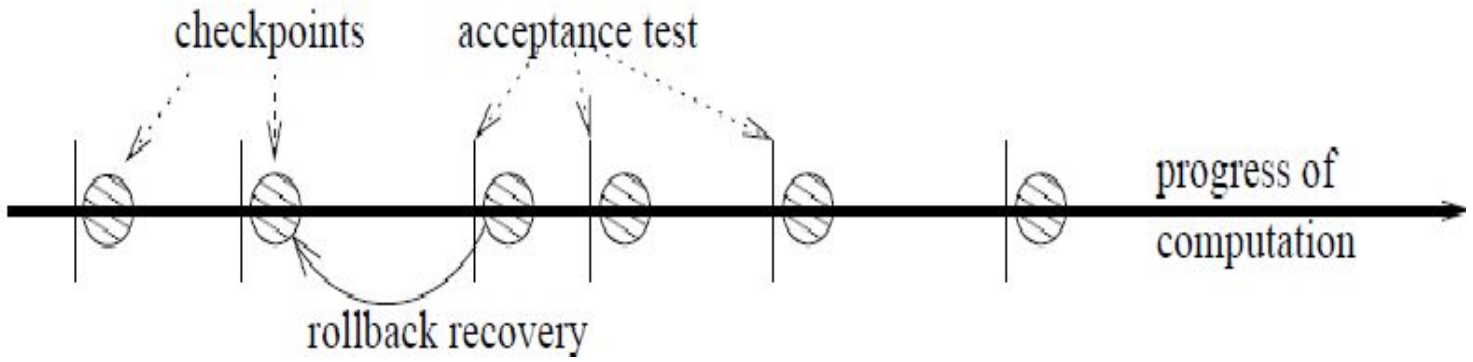


Figure 12: A Software Fault-Tolerance Scheme Using Recovery Blocks

Checkpointing and Rollback Recovery:



Types of RT Task

RTT can be classified based on the consequences of a task missing its deadline

- Hard RTT(robot)
- Firm RTT(video conferencing)
- Soft RTT(url)
- Non RTT(e-mail)

Timing constraints

- Events in a RTS:an event may be either by the system or its environment. Based on this, events can be classified into

1. Stimulus Events: are generated by the environment and act on the system
2. Response Events: are usually produced by the system in response to some stimulus andact on the events, environment

- Classification of Timing Constraints

1. Performance constraints: are imposed on the response of the system. it ensure that the system performs satisfactorily
2. Behavioral constraints: are imposed on the stimuli generated by the environment.

Each of performance and behavioral constraints can further classified into

1. Delay Constraint
2. Duration Constraint
3. Deadline constraint

Deadline Constraint:

capturesthe permissible maximum separation between e1 and e2. ie the second event must follow the first event within the permissible maximum separation time . where t(e1) and t(e2) are the time stamps of e1 and e2,d is the deadline and Δ is the actual separation in time between the occurrence of two events

$$t(e_2) - t(e_1) \leq d$$

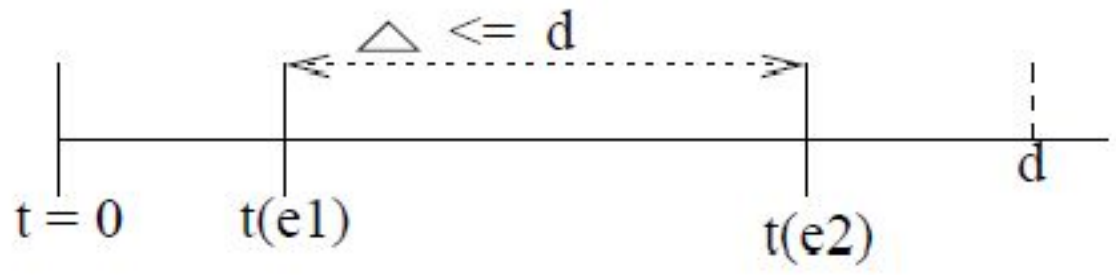


Figure 17: Deadline Constraint Between Two Events e1 and e2

Duration Constraint: Specifies the time period over which the event acts. it can be either minimum or maximum.

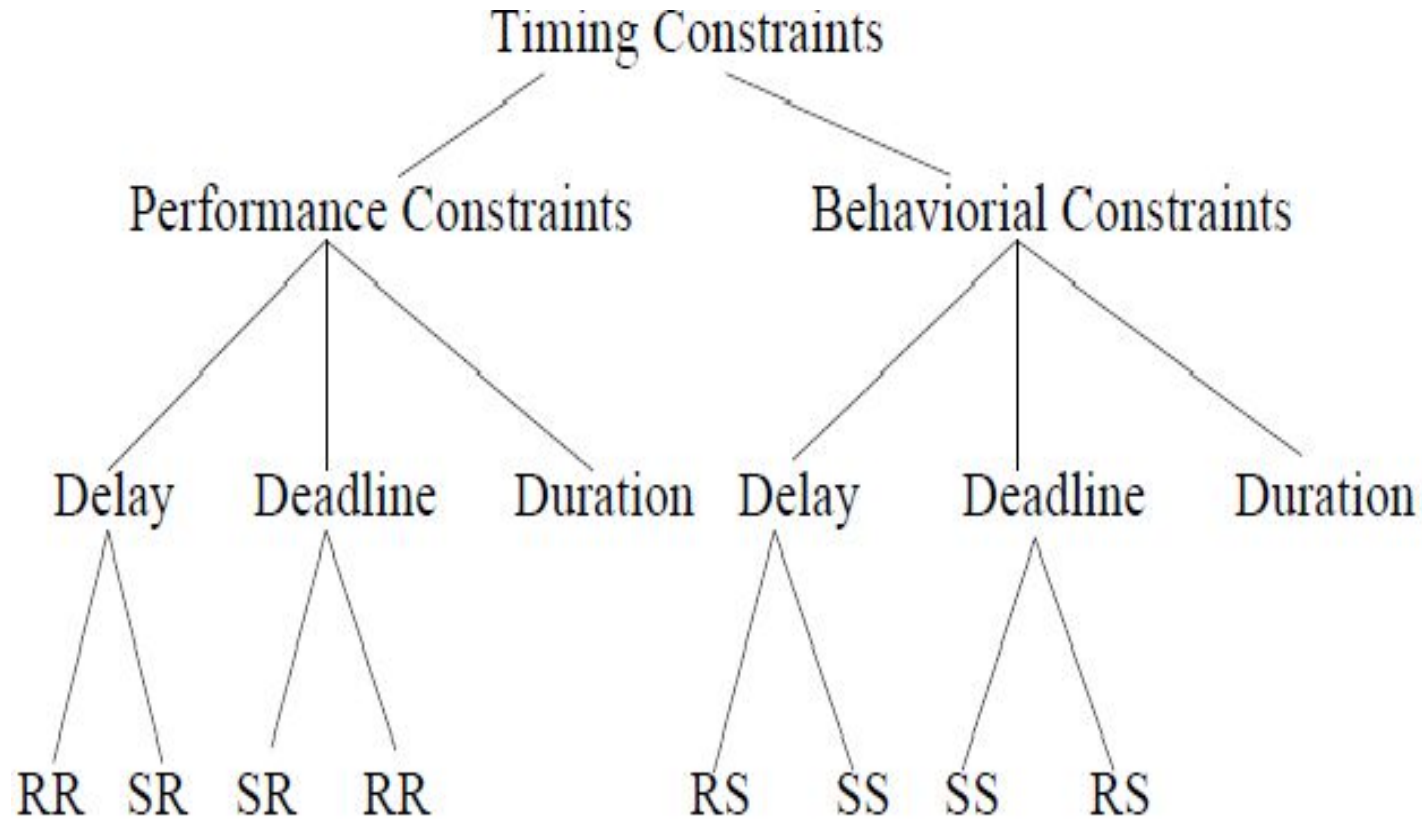


Figure 19: Classification of Timing Constraints

Modeling timing constraints

- It can serve as a formal specification of the system
- Finite State Machine
- is used to model traditional systems.
- at any point of time a system can be in any one of a state.
- State is represented by a circle.
- A state change is called state transition
- A transition from one state to another state is represented by drawing a directed arc from source to destination.
- Extended Finite State Machine
- Used to model time constraints
- It extends FSM by incorporating the action of setting a timer and the expiry event of a timer

SS Constraints

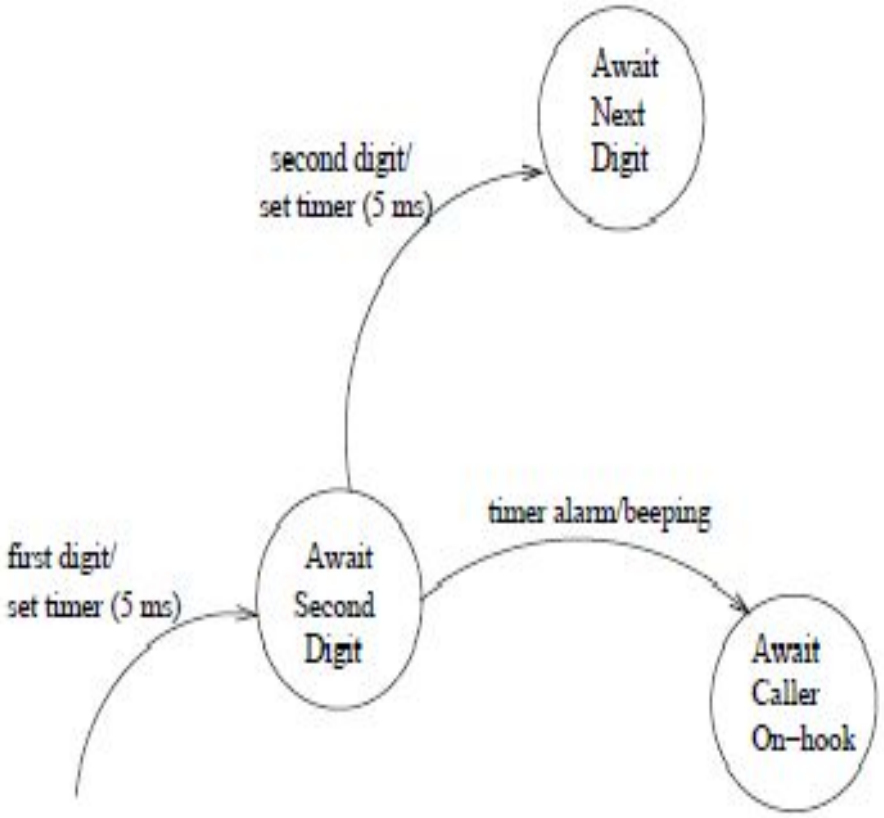


Figure 21: Model of an SS Type of Deadline Constraint

RS Constraints

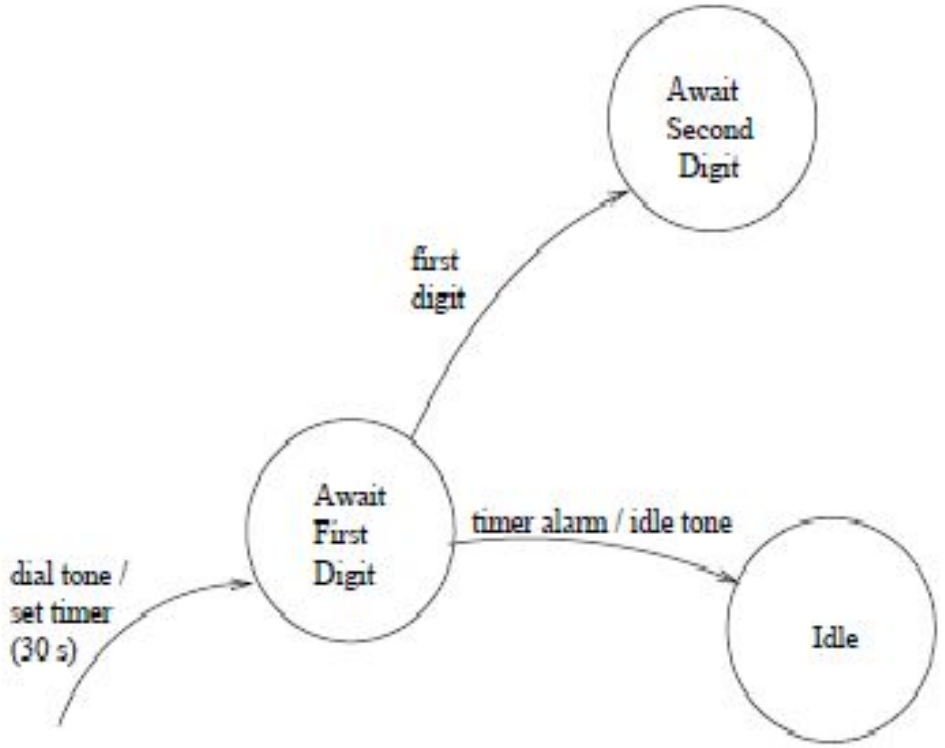


Figure 22: Model of an RS Type of Deadline Constraint

SR Constraints

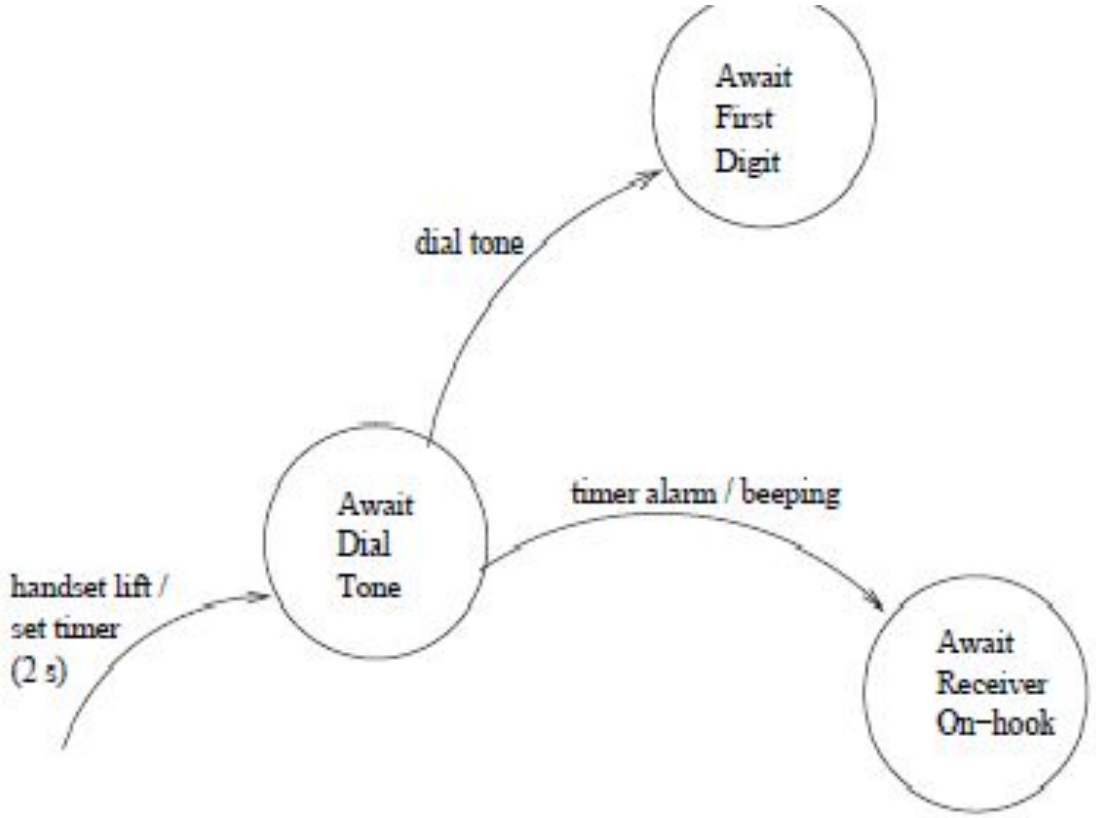


Figure 23: Model of an SR Type of Deadline Constraint

RR Constraints

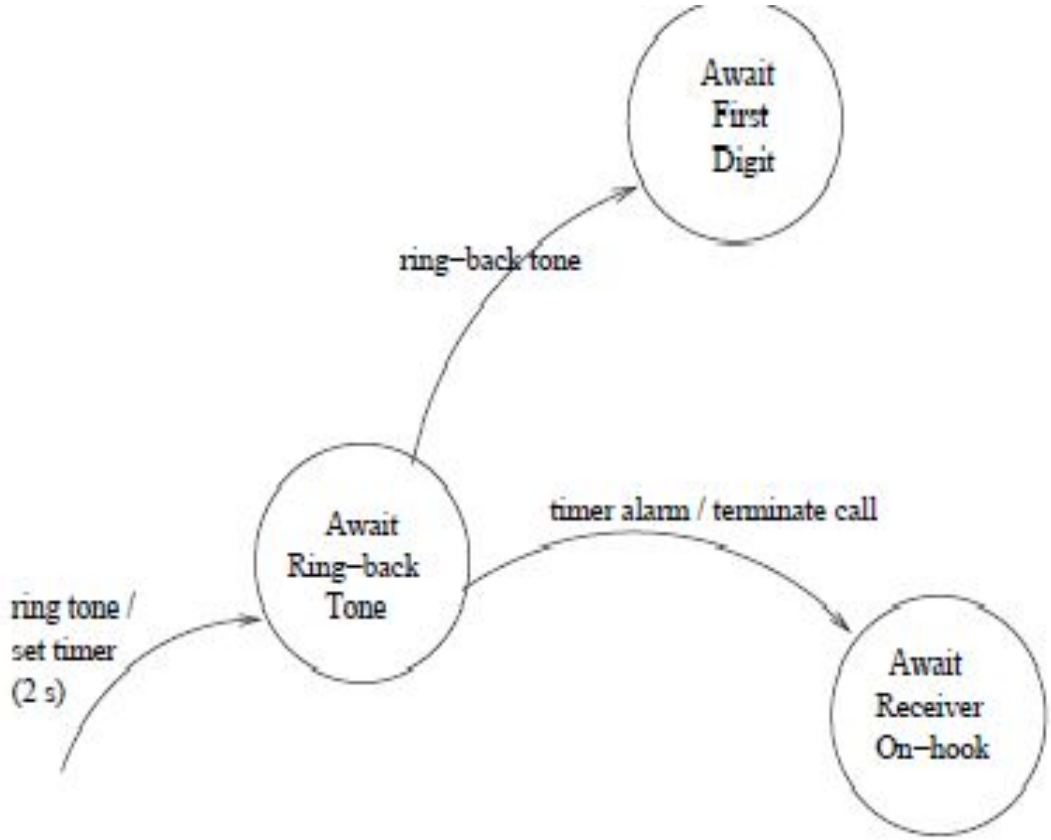


Figure 24: Model of an RR Type of Deadline Constraint

RTT Scheduling

- Task instance: task is generated when some specificevent occurs
- Relative deadline and Absolute deadline

Absolute deadline of a task is the absolute time value by which the results from the task are expected

Relative deadline is the time interval between the start of the task and the instance at v

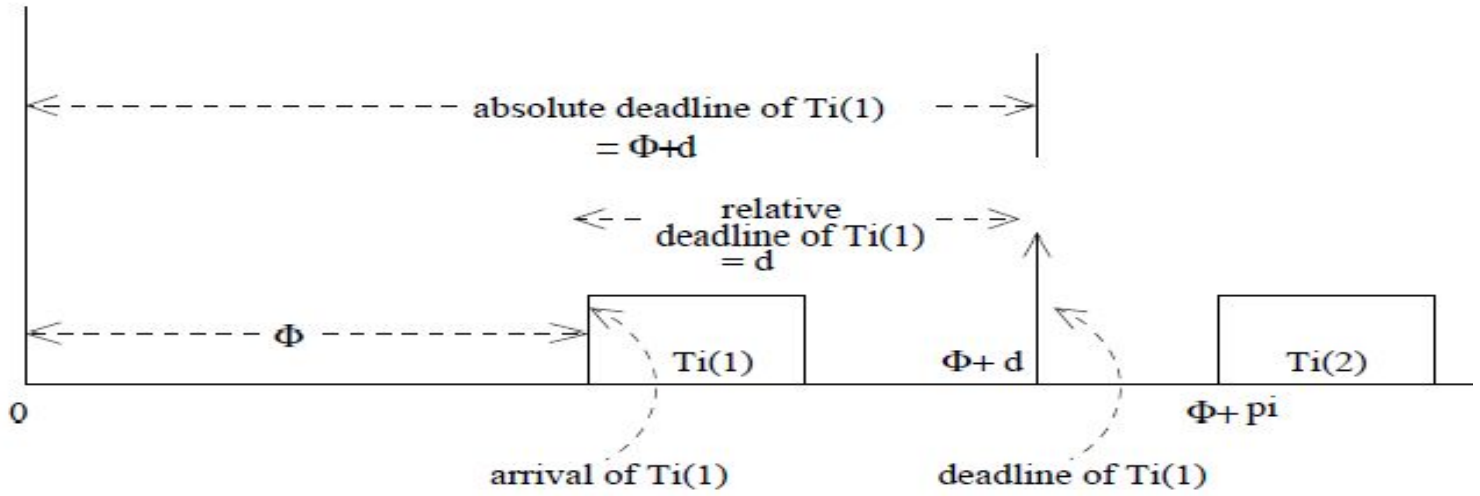


Figure 1: Relative and Absolute Deadlines of a Task

- Response time
- Task Precedence: a task is said to precede another task ,if the first task must complete before the second task can stat
- Data sharing

Types of Real-Time Tasks

- **Periodic tasks**
  - Time-driven.
  - repeats after a certain fixed time interval
  - represented by (ϕ_i, p_i, e_i, d_i)

  E.g.: monitoring temperature of a patient in an ICU.
- **Aperiodic tasks**
  - Event-driven.
  - arise at random instants
  - E.g.: Task activated upon detecting change in patient's condition.
- **Sporadic Tasks**
  - Recurs at random instants
  - Represented by $T_i = (e_i, g_i, d_i)$
  - E.g: emergency message sending

p_i : task period a_i : arrival time r_i : ready time d_i : deadline

g_i :minimum separation between the consecutive instances of a task

e_i : worst case execution time.

RTT scheduling basic concepts

Valid Schedule. A valid schedule for a set of tasks is one where at most one task is assigned to a processor at a time, no task is scheduled before its arrival time, and the precedence and resource constraints of all tasks are satisfied.

Feasible Schedule. A valid schedule is called a feasible schedule, only if all tasks meet their respective time constraints in the schedule.

- Proficient scheduler
- Optimal scheduler
- Scheduling points
- Preemptive scheduler
- Utilization

Classification of RTT Scheduling algorithms

1. Clock Driven:
   - Table-driven
   - Cyclic
2. Event Driven:
   - Simple priority-based
   - Rate Monotonic Analysis (RMA)
   - Earliest Deadline First (EDF)
3. Hybrid:
   - Round-robin

The clock-driven schedulers are those in which the scheduling points are determined by the interrupts received from a clock. In the event-driven ones, the scheduling points are defined by certain events which precludes clock interrupts. The hybrid ones use both clock interrupts as well as event occurrences to define their scheduling points.

- Another classification is based on task acceptance test
    1. Planning Based
    2. Best Effort

- Another classification is based on the target platform on which the tasks are to be run
    1. Uniprocessor
    2. Multiprocessor
    3. Distributed

Clock driven Scheduling

- Scheduling points are determined by timer interrupts.
- Also Called off line scheduler because it fix the schedule before the system starts to run.

1. Table Driven Scheduling

- Precompute which task would run when and store this schedule in a table at the time the system is designed or configured
- Application programmer can set his own schedule
- Schedule table used to store the schedule
- Major cycle

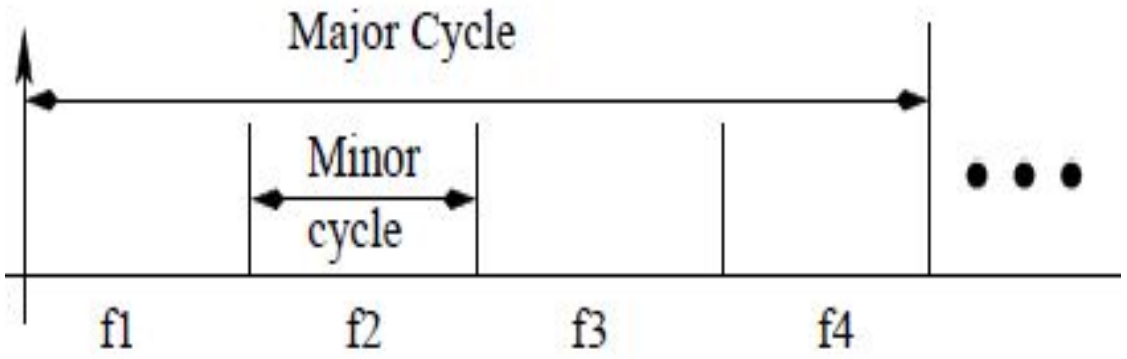
Task	Start Time in milli Seconds
T_1	0
T_2	3
T_3	10
T_4	12
T_5	17

Example of a Table-Driven Schedule

> A major cycle of a set of tasks is an interval of time on the time line such that in each major cycle, the different tasks recur identically.

2. Cyclic scheduler

- Very popular and extensively used in industry
- Simple , efficient and easy to program
- Repeats a precomputed schedule, precomputed schedule needs to be stored only for one Major cycle.
- Major cycle is divided into one or more minor cycles,and is called frame



- Scheduling points occur at frame boundaries

- Each task is designed to run in one or more frames.
- Assignment of tasks to frame is stored in a schedule table

Task Number	Frame Number
T3	F1
T1	F2
T3	F3
T4	F2

- Frame size should satisfy the following constraints

1. Minimum context switching
2. Minimization of table size
3. Satisfaction of task deadline

Generalized task scheduler

- Initially a schedule(assignment of task to frames) for periodic task is prepared, sporadic and aperiodic task are scheduled in the slack times that may available in the frame

An efficient implementation of this scheme is that the slack times are stored in a table and during acceptance test this table is used to check the schedulability of the arriving tasks.

Another popular alternative is that the aperiodic and sporadic tasks are accepted without any acceptance test, and best effort is made to meet their respective deadlines.

Hybrid scheduling

Time sliced Round Robin scheduler

- Preemptive scheduling method
- Ready tasks are held in circular queue
- Time slice

EVENT DRIVEN SCHEDULER

- Overcomes the shortcomings of other types(wastage of frame size)
- Can handle aperiodic and sporadic tasks more proficiently
- Less efficient and deploy complex scheduling algorithms.
- Less suitable for embedded applications.
- Scheduling points are defined by task completion and task arrivals events.
- Are normally preemptive

Foreground and background scheduler

- Simplest priority driven scheduler.
- RTT in an application is run as foreground tasks.
- Sporadic , aperiodic and non RT are run as background tasks.
- Among foreground tasks at every scheduling point the highest priority task is taken up for scheduling.
- A background task can run when none of the foreground is ready

Earliest deadline first(EDF)

- A dynamic priority scheduling algorithm
- At every scheduling point task having the shortest deadline is taken up for scheduling.
- A task set is schedulable under EDF,if and only if it satisfies the condition that the total processor utilization due to the task set is less than 1
- Is an optimal uniprocessor scheduling algorithm means if a set of tasks is unschedulable under EDF,then no other scheduling algorithm can feasibly schedule this task set.
- A variant is Minimum Laxity First(MLF)
- Every scheduling point a laxity value is computed for every task and task having minimum laxity is executed first.
- Laxity measures the execution time

- **Implementation of EDF**

1. Maintain all tasks that are ready for execution in a queue, at every preemption point task having shortest deadline scheduled. its inefficient
2. Maintain all ready tasks in a priority queue
3. A FIFO queue maintained for tasks, based on relative deadline and absolute deadline

- Shortcomings

Transient Overload Problem: Transient overload denotes the overload of a system for a very short time.

Resource Sharing Problem:

Efficient Implementation Problem

Rate monotonic algorithm

- Important Event driven optimal Static priority algorithm used in practical applications.
- Assigns priority to tasks based on their rate of occurrence.
- Priority of a task is directly proportional to its rate

priority of any task T_i is computed as: $priority = \frac{k}{p_i}$, where p_i is the period of the task T_i and k is a constant.
priority of a task increases linearly with the arrival rate of the task and inversely with its period.

- Schedulability test

1. Necessary condition: total cpu utilization due to all the task in the task set should be less than 1
2. Sufficient condition:

$$\sum_{i=1}^{n} u_i \leq n(2^{\frac{1}{n}} - 1)$$

where u_i is the utilization due to task T_i.

- **Advantage RMA**

1. RMA Transient overload handling: means when a lower priority task doesn't complete within its planned completion time ,cant make any higher priority task to miss its deadline .

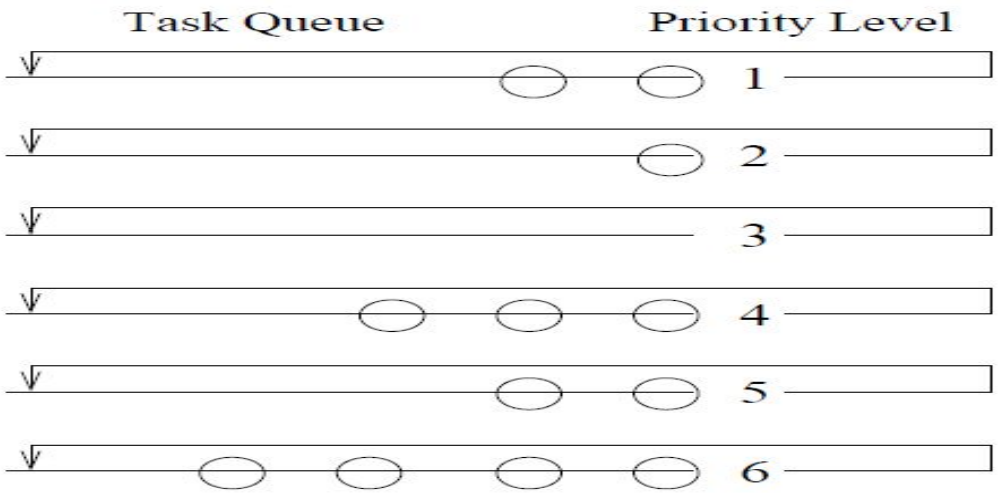


Figure 15: Multi-Level Feedback Queue

- Disadvantages

1. Difficult to support aperiodic and sporadic task.
2. Is not optimal when task periods and deadlines are differ

Deadline monotonic algorithm

- A variant of RMA and assigns priorities to tasks based on their deadlines, rather than assigning priorities based on task periods .
- Assigns high priority to tasks with shorter deadline
- Issues in using RMA in practical situations
    1. Handling critical tasks with long transformation technique)
    2. Handling aperiodic and sporadic tasks
    - Aperiodic server
    1. deferrable server(tickets are replenished at regular intervals ,independent of actual ticket usage)
    2. sporadic server(replenishment time depends on exact ticket usage time, guarantees a minimum separation between two instances of a task)

3. Coping with limited priority levels(assigning priority to task)

a. Uniform scheme(all the tasks in the application are uniformly divided among the available priority levels, if its not possible then more tasks should be made to share the lower priority levels)

b. Arithmetic scheme(the number of tasks assigned to different priority levels form an arithmetic progression. Let N be the number of tasks then

   $N=r+2r+3r+..nr$,where n be the total number of priority levels)

c. Geometric scheme(the number of tasks assigned to different priority levels form a geometric progression

   $N=r+kr^2+kr^3+..kr^n$

d. Logarithmic scheme(shorter period tasks should be allotted distinct priority levels and many lower priority tasks on the other hand can be clubbed together at the same priority levels without causing any problem to the schedulability of HP Task)

Resource Sharing Among RTT

- Until now, we have assumed that tasks are independent.
- We now remove this restriction.
- We first consider how to adapt the analysis discussed previously when tasks access shared resources.
- Later, in our discussion of distributed systems, we will consider tasks that have precedence constraints.

Resource Access Control Protocols

- We now consider several protocols for allocating resources that control priority inversions and/or deadlocks.
- From now on, the term “critical section” is taken to mean “outermost critical section” unless specified otherwise.

Non preemptive Critical Section Protocol

The simplest protocol: just execute each critical section nonpreemptively. If tasks are indexed by priority (or relative deadline in the case of EDF), then task Ti has a blocking term equal to maxi+1 ≤ k ≤ n ck , where ck is the execution cost of the longest critical section of Tk

Resource Sharing Among RTT

- Serially reusable resource
- Non preemptable resource

PRIORITY INVERSION

- Simple priority inversion
- Unbounded priority inversion

Unbounded priority inversion

Unbounded priority inversion occurs when a higher priority task waits for a lower priority task to release a resource it needs, and in the meanwhile intermediate priority tasks preempt the lower priority task from CPU usage repeatedly; as a result, the lower priority task can not complete its usage of the critical resource and the higher priority task waits indefinitely for its required resource to be released.

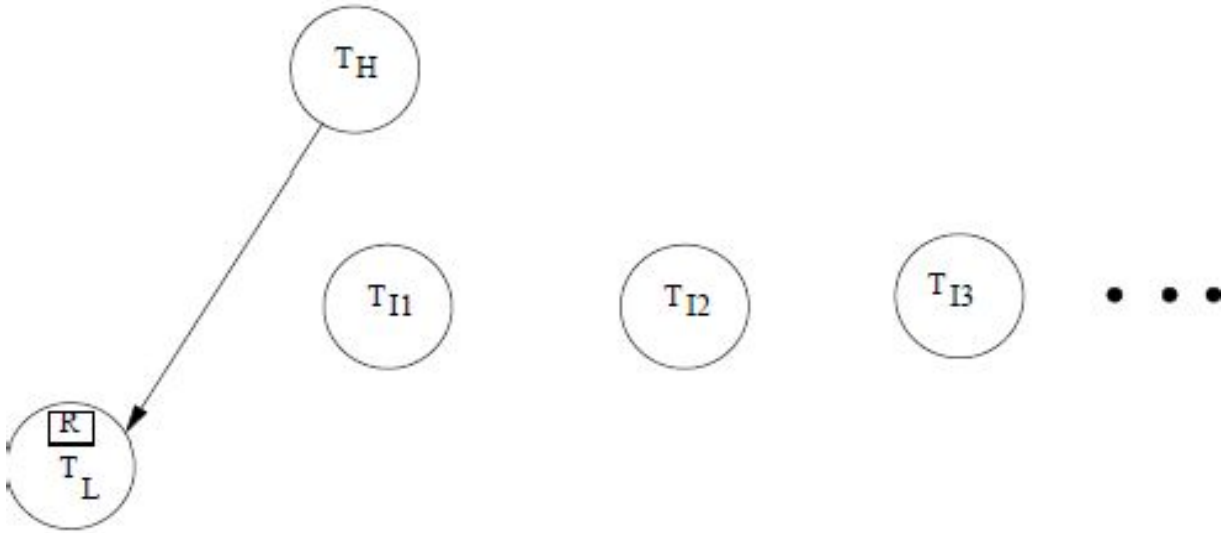


Figure 1: Unbounded Priority Inversion

Priority inheritance protocol (pip)

- Priority Inheritance Protocol (PIP) is a technique which is used for sharing critical resources among different tasks. This allows the sharing of critical resources among different without the occurrence of unbounded priority inversions.

- Is a simple technique to share CR among tasks without incurring unbounded priority inversions.

- The essence of this protocol is that whenever a task suffers priority inversion, the priority of the lower priority task holding the resource is raised through a priority inheritance mechanism. it enables it to complete its usage of the CR as early as possible without having to suffer preemption from intermediate tasks.

- When several tasks waiting for a resource ,the task holding the resource inherits the highest priority of all tasks waiting for the resource.(if this priority is greater than its own priority)

Basic Concept of PIP :

- The basic concept of PIP is that when a task goes through priority inversion, the priority of the lower priority task which has the critical resource is increased by the priority inheritance mechanism.
- It allows this task to use the critical resource as early as possible without going through the preemption.
- It avoids the unbounded priority inversion.

Working of PIP :

- When several tasks are waiting for the same critical resource, the task which is currently holding this critical resource is given the highest priority among all the tasks which are waiting for the same critical resource.
- Now after the lower priority task having the critical resource is given the highest priority then the intermediate priority tasks can not preempt this task. This helps in avoiding the unbounded priority inversion.
- When the task which is given the highest priority among all tasks, finishes the job and releases the critical resource then it gets back to its original priority value (which may be less or equal).
- If a task is holding multiple critical resources then after releasing one critical resource it can not go back to it original priority value. In this case it inherits the highest priority among all tasks waiting for the same critical resource.

If the critical resource is free then

allocate the resource

If the critical resource is held by higher priority task then

wait for the resource

If the critical resource is held by lower priority task

{

lower priority task is provided the highest priority

other tasks wait for the resource

}

Advantages of PIP :

Priority Inheritance protocol has the following advantages:

- It allows the different priority tasks to share the critical resources.
- The most prominent advantage with Priority Inheritance Protocol is that it avoids the unbounded priority inversion.

Disadvantages of PIP :

Priority Inheritance Protocol has two major problems which may occur:

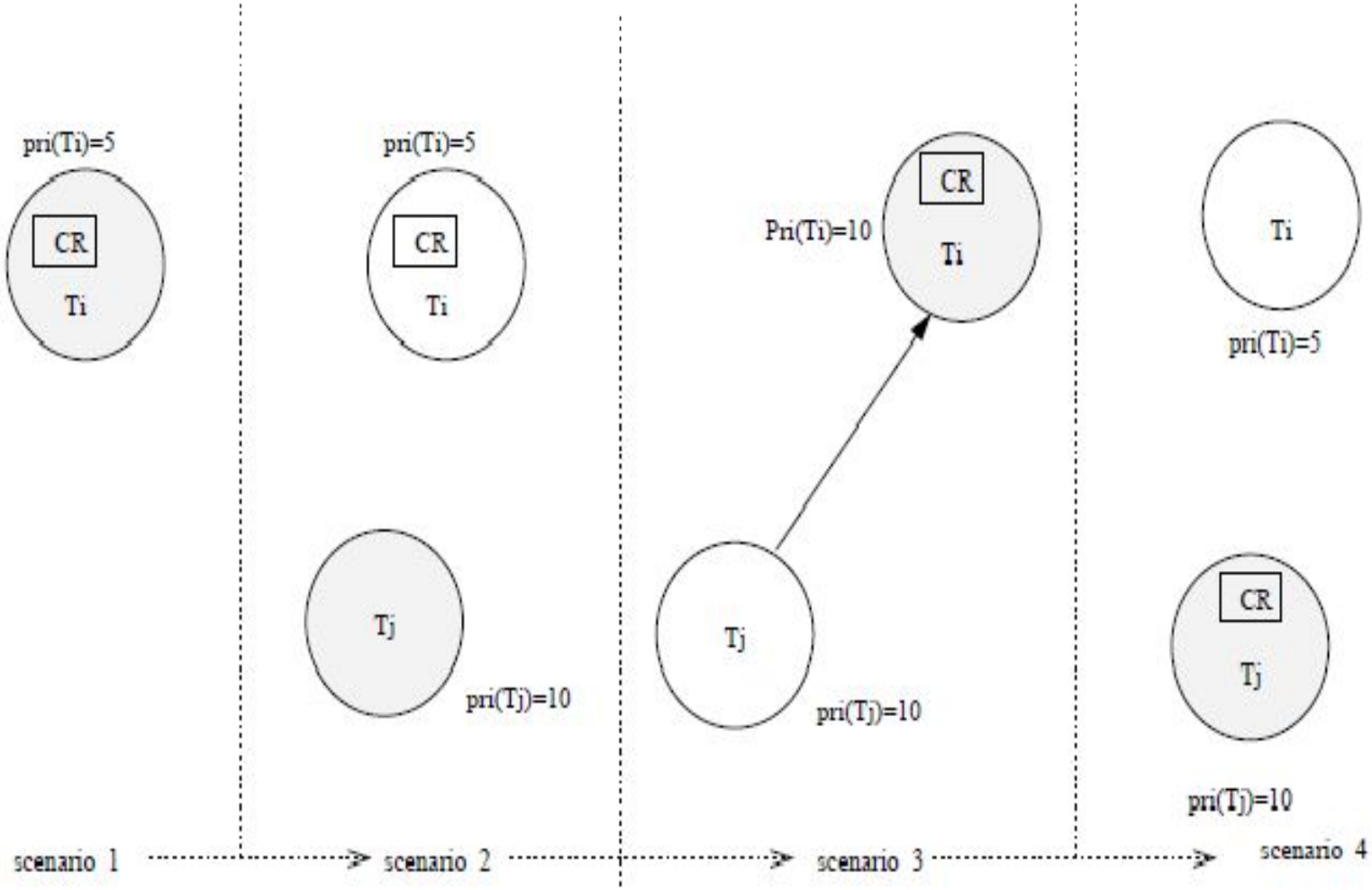


Figure 3: Snapshots Showing Working of Priority Inheritance Protocol

Deadlock –

- There is possibility of deadlock in the priority inheritance protocol.

  For example, there are two tasks T_1 and T_2. Suppose T_1 has the higher priority than T_2. T_2 starts running first and holds the critical resource CR_2.
- After that, T_1 arrives and preempts T_2. T_1 holds critical resource CR_1 and also tries to hold CR_2 which is held by T_2. Now T_1 blocks and T_2 inherits the priority of T_1 according to PIP. T_2 starts execution and now T_2 tries to hold CR_1 which is held by T_1.
- Thus, both T_1 and T_2 are deadlocked.

● Chain Blocking –

- When a task goes through priority inversion each time it needs a resource then this process is called chain blocking.
- For example, there are two tasks T_1 and T_2. Suppose T_1 has the higher priority than T_2. T_2 holds the critical resource CR_1 and CR_2. T_1 arrives and requests for CR_1. T_2 undergoes the priority inversion according to PIP.
- Now, T_1 request CR_2, again T_2 goes for priority inversion according to PIP. Hence, multiple priority inversion to hold the critical resource leads to chain blocking

PIP- Two important problems

1. Deadlock

   consider following sequence of actions by two tasks T_1 and T_2,which need access to two shared CR_1 and CR_2

   T_1 :Lock CR_1,Lock CR_2,Unlock CR_2, Unlock CR_1

   T_2: Lock CR_2,Lock CR_1,Unlock CR_1, Unlock CR_2

i. T_1 has higher priority than T_2
ii. T_2 starts running first and locks CR_2
iii. T_1 arrives ,preempts T_2 and starts executing
iv. T_1 locks CR_1 and then tries to lock CR_2 which is being held by T_2
v. T_1 blocks and T_2 inherits T_1 's priority according to the PIP
vi. T_2 resumes its execution and after sometime needs to lock the resource CR_1 being held by T_1
vii. T_1 and T_2 are both deadlocked

- **Chain Blocking**

A task is said to undergo chain blocking ,if each time it needs a resource ,it undergoes priority inversion

1. T_1 needs several resources ,and high priority than T_2
2. T_2 holding CR_1 and CR_2 and T_1 arrives and requests to lock CR_1.it undergoes priority inversion and causes T_2 to inherit its priority
3. As soon as T_2 release CR_1 its priority reduces to its original priority and T_1 is able to lock CR_1
4. After executing for some time T_1 requests to lock CR_2 .This time it again undergoes priority inversion since T_2 is holding CR_2. T_1 waits until T_2 release CR_2

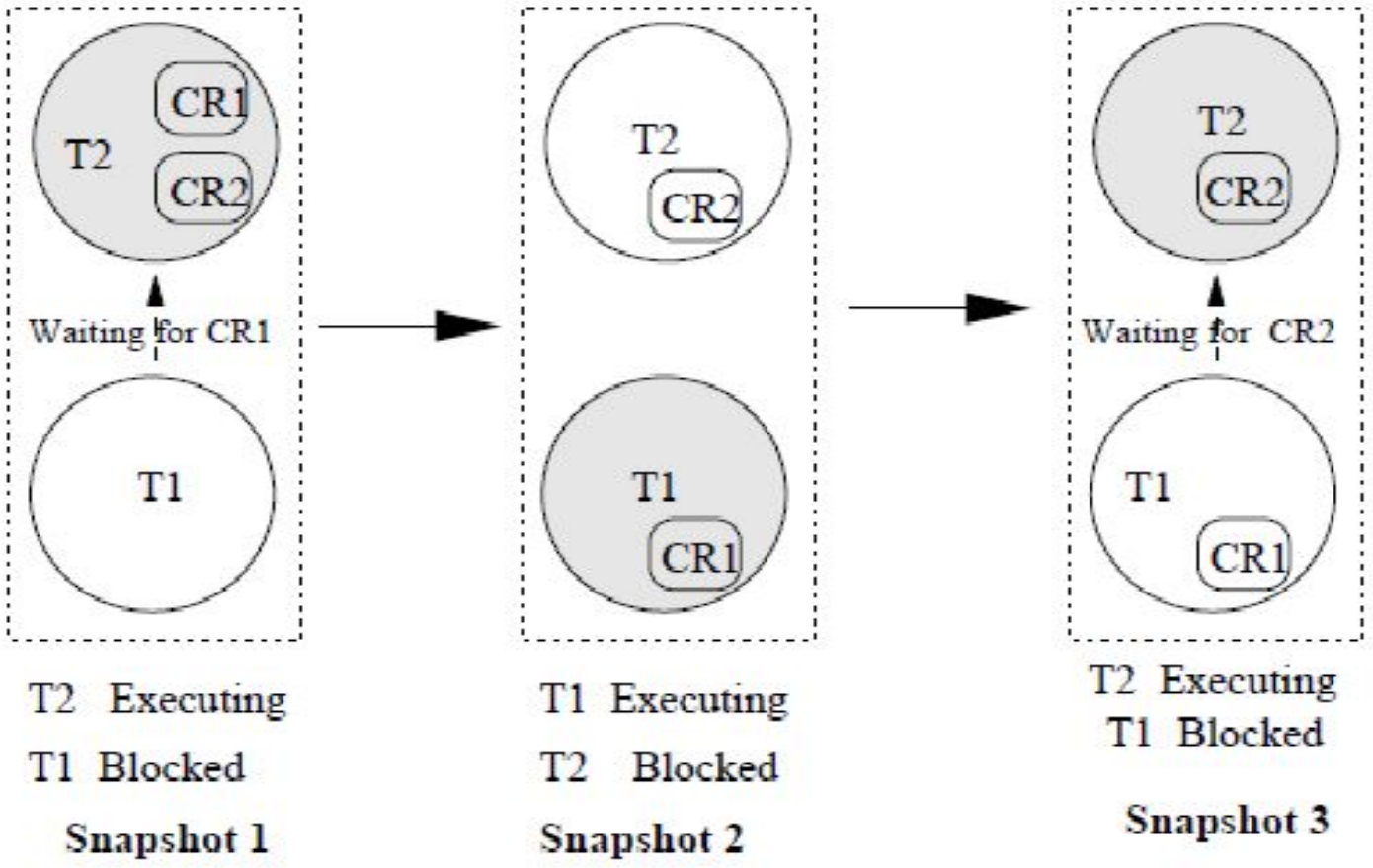


Figure 5: Chain Blocking in Priority Inheritance Protocol

Highest Locker Protocol (HLP)

- **Highest Locker Protocol (HLP)** is a critical resource sharing protocol which is an extension of Priority Inheritance Protocol which was introduced to overcome the limitations of Priority Inheritance Protocol.
- In this critical resource sharing protocol, every critical resource is assigned a ceiling priority value.
- This value is the maximum of priorities of all those tasks which may request to hold this critical resource.

Basic Concept of HLP :

- The basic concept of Highest Locker Protocol is based on the ceiling priority value.
- When a task holds a critical resource its priority is changed to the ceiling priority vale of the critical resource.
- If a task holds multiple critical resources, then maximum of all ceiling priorities values is assigned as priority of the task.

Working of HLP :

- Resources required by each task is found before the compile time.
- Initially a ceiling priority value is assigned to each critical resource.
- Ceiling priority value of a critical resource is calculated as the maximum of priorities of all those tasks which may request to hold this critical resource.
- When a task holds a critical resource, corresponding ceiling priority value is assigned as priority to the task.
- Task acquiring multiple critical resources is assigned maximum of all ceiling priority value.
- Further the execution is done on the basis of allotted priorities.

Features of HLP :

- When HLP is used for resource sharing, once the task holds the required critical resource then it is not blocked any further.
- Before a task can hold one resource, all the resources that may be required by this task should be free.
- It prevents tasks from going into deadlock or chain blocking.

Advantages of HLP :

Following are the advantages of Highest Locker Protocol:

- It is useful into critical resource sharing by several tasks.
- It avoids the unbounded priority inversion among tasks.
- It overcomes the limitations of priority inheritance protocol.
- It prevents from deadlock as a task hols one resource, all other required resources by this task must be free.
- A task can not go into chain blocking using Highest Locker Protocol.

Disadvantages of HLP :

The major disadvantage of Highest Locker Protocol is **Inheritance related Priority Inversion**.

Inheritance related Priority Inversion occurs when the priority value of low priority task acquiring a critical resource is assigned the highest priority using ceiling rule then the intermediate priority tasks that do not need the resource cannot execute and undergo Inheritance related Priority Conversion.

Highest Locker Protocol (HLP):	Priority Ceiling Protocol (PCP) :
• Highest Locker Protocol (HLP) is a critical resource sharing protocol which is an extension of Priority Inheritance Protocol (PIP) which was introduced to overcome the limitations of Priority Inheritance Protocol (PIP). • In this critical resource sharing protocol, every critical resource is assigned a ceiling priority value. • This value is the maximum of priorities of all those tasks which may request to hold this critical resource. • When a task holds a critical resource its priority is changed to the ceiling priority vale of the critical resource. If a task holds multiple critical resources, then maximum of all ceiling priorities values is assigned as priority of the task.	• Priority Ceiling Protocol (PCP) is an extension of Priority Inheritance Protocol (PIP) and Highest Locker Protocol (HLP). • It solves the problem of unbounded priority inversion of Priority Inheritance Protocol, deadlock and chain blocking of Highest Locker Protocol and also minimizes the inheritance-related inversion which was an also a limitation of Highest Locker Protocol. • It is not a greedy approach like Priority Inheritance Protocol. In PCP, it is possible that a task may be denied for access although the resources is free.

Priority Ceiling Protocol

- **Priority Ceiling Protocol** is a job task synchronization protocol in a real-time system that is better than Priority inheritance protocol in many ways. Real-Time Systems are multitasking systems that involve the use of semaphore variables, signals, and events for job synchronization.
- In **Priority ceiling protocol** an assumption is made that all the jobs in the system have a fixed priority. It does not fall into a deadlock state.
- The chained blocking problem of the Priority Inheritance Protocol is resolved in the Priority Ceiling Protocol.

The basic properties of Priority Ceiling Protocols are:

1. Each of the resources in the system is assigned a priority ceiling.
2. The assigned priority ceiling is determined by the highest priority among all the jobs which may acquire the resource.
3. It makes use of more than one resource or semaphore variable, thus eliminating chain blocking.
4. A job is assigned a lock on a resource if no other job has acquired lock on that resource.
5. A job J, can acquire a lock only if the job's priority is strictly greater than the priority ceilings of all the locks held by other jobs.
6. If a high priority job has been blocked by a resource, then the job holding that resource gets the priority of the high priority task.
7. Once the resource is released, the priority is reset back to the original.
8. In the worst case, the highest priority job J_1 can be blocked by T lower priority tasks in the system when J_1 has to access T semaphores to finish its execution.

Highest locker protocol(hlp)

- Extension of PIP
- Every CR is assigned a ceiling priority value
- Ceiling priority of a CR is defined as the maximum of the priorities of all tasks which may request to use this resource.
- when a task acquires a resource its priority is set equal to the ceiling priority of all its locked resources, if the task holds multiple resources then it inherits the highest ceiling priority of all its locked resources.
- Ceiling priority of a resource R_i be $Ceil(R_i)$ priority of a task T_j is $pri(T_j)$

$$Ceil(R_i) = \max(\{ pri(T_j) / T_j \text{ needs } R_i \})$$

Theorem

When HLP is used for resource sharing once a task gets a resource required by it. It is not blocked any further.

Corollary 1.

under HLP before a task can acquire one resource ,all the resources that might be required by it must be free.

Corollary 2.

a task can't undergo chain blocking in HLP

- Shortcomings

1. Inheritance related inversion

when the priority value of a low priority task holding a resource is raised to a high value by the ceiling rule, the intermediate priority tasks not needing the resource cannot execute and are said to undergo inheritance related inversion

Priority ceiling protocol(pcp)

- Minimizing inheritance related inversions
- A resource may not be granted to a requesting task even if the resource is free
- Associates a ceiling value with every resource, that is the maximum of the priority values of all tasks that might use the resource
- an OS variable called CSC(Current System Ceiling) is used to keep track of the maximum ceiling value of all the resources that are in use at any instant of time

  CSC=max({Ceil(CR_i)/CR_i is currently in use})
- CSC is initialized to 0(lower priority than the least priority task in the system)

- Resource sharing among tasks under PCP is regulated using two rules

Resource Grant Rule: consist of two clauses. these two clauses are applied when a task requests to block a resource

1. **Resource request clause:**
   (a) If a task T_i is holding a resource whose ceiling priority equals CSC, then the task is granted access to the resource.
   (b) Otherwise, T_i will not be granted CR_j, unless its priority is greater than CSC (i.e. $pri(T_i) > CSC$). In both (a) and (b) above, if T_i is granted access to the resource CR_j, and if $CSC < Ceil(CR_j)$, then CSC is set to $Ceil(CR_j)$

2. **Inheritance clause:** When a task is prevented from locking a resource by failing to meet the resource grant clause, it blocks and the task holding the resource inherits the priority of the blocked task if the priority of the task holding the resource is less than that of the blocked task.

Resource release rule:

If a task releases a critical resource it was holding and if the ceiling priority of this resource equals CSC, then CSC is made equal to the maximum of the ceiling value of all other resources in use; else CSC remains unchanged. The task releasing the resource either gets back its original priority or the highest priority of all tasks waiting for any resources which it might still be holding, whichever is higher.

HIGHEST LOCKER PROTOCOL

- It is a critical resource sharing protocol which is an extension of PIP.
- It overcomes the limitations of PIP.
- It requires moderate support from the operating system.
- It is the least efficient among all resource sharing protocols.
- It maximizes the inheritance-related inversions.
- It solves the problem of unbounded priority inversion, deadlock and chain blocking.
- In HLP, highest priority task can not be denied access if resource is free.
- It is rarely used in real-life applications.

PRIORITY CEILING PROTOCOL

- It is a critical resource sharing protocol which is an extension of PIP and HLP.
- It overcomes the limitations of PIP and HLP.
- While it requires maximum support from the operating system.
- While it is the most efficient one among all resource sharing protocols.
- While it is able to minimize the inheritance-related inversions.
- While it solves the problem of unbounded priority inversion, deadlock, chain blocking and inheritance-related inversions.
- While in PCP, highest priority task can be denied access although resource is free if priority value is less than CSC (Current System Ceiling).
- While it is used in large applications.

Issues in using a resource sharing protocol

- Using PCP in dynamic priority systems
- Comparison of resource sharing protocols

1. PIP

- Simple and effectively overcomes the unbounded priority inversion problem
- Tasks may suffer from chain blocking and deadlock
- Requires minimal support from OS

2. HLP

- Requires moderate support from the OS
- Solves chain blocking and deadlock
- Can make the intermediate priority tasks undergo large inheritance related inversion and can cause tasks to miss their deadlines

3. PCP

- Free from deadlock and chain blocking
- Priority of a task is not changed until a higher priority task requests the resource
- Suffers much lower inheritance related inversions than HLP

Task Dependency

- **Task Dependency** is a relationship in which a **task** or milestone relies on other **tasks** to be performed (completely or partially) before it can be performed.
- A logical relationship can be a **dependency** between project **tasks** or between **tasks** and milestones.

Handling task dependencies

- Develop a satisfactory schedule for a set of tasks
- Table Driven Algorithm: determines feasible schedule for a set of periodic real time tasks whose dependencies are given
- EDF and RMA based schedulers: precedence constraints among tasks can be handled in both EDF and RMA through the modification to the algorithm
- Don’t enable a task until all its predecessors complete execution
- Check the tasks waiting to be enabled (on account of its predecessors completing their execution) after every task completes

Scheduling Real time Tasks in Multiprocessor and distributed systems

- Multiprocessor task allocation
- Utilization balancing algorithm
  - Maintains the tasks in increasing order of their utilizations
  - It removes task one by one from the head of the queue and allocates them to the least utilized processor time.
- Next fit algorithm for RMA
  - Classifies the tasks into few classes based on the utilization of the task.
  - One or more processors are assigned to each class of tasks
- Bin packing algorithm for RMA Tasks are assigned to processors such that utilization at any processor does not exceed 1

Dynamic allocation of tasks

- Focused addressing and bidding
  - Every processor maintains two tables

a) Status Table :contains information about the tasks which it has committed to run, including information about the execution time and period of the tasks.

b) system load table: contains latest load information of all other processors of the system

- When a task arise at a processor ,it first checks whether the task can be processed locally itself .if it can be processed ,it updates its status table .if not it looks for a processor to which it can offload the task(consults its system load table ,sends out request for bid(RFB) ,focused processor).
- Every processor on receiving a broadcast from a processor about the load position updates the system load table
- High communication overhead

- Buddy algorithm
  - Tries to overcome the high communication overhead of the focused addressing and bidding algorithm
  - Processor can be in any of two states

a) Under loaded: utilization is less than some threshold value

b) Overloaded: utilization is greater than the threshold value

- Processor broadcasts only to a buddy set(set of processors) when the status of a processor changes

- Fault tolerant scheduling of tasks
  - Can be achieved by scheduling additional copies in addition to the primary copy of a task

Fault tolerant scheduling of tasks.

- **Fault-tolerant scheduling** algorithms provide the system with a means to continue meeting deadlines until this can be completed;
- their role is therefore to provide the system with some breathing room before it makes any longer-term adjustments to the **task** assignment and **scheduling** that may be appropriate.

Fault Types

There are three types of faults:

- Permanent,
- Intermittent,
- Transient.

- A permanent fault does not die away with time, but remains until it is repaired as the affected unit is replaced.
- This is an intermittent fault cycle between the fault–active and fault benign states.
- A transient fault dies away after some time.

Fault Detection

Fault detection can be done either online or offline.

Online detection goes on in parallel with normal system operation.

Offline detection consists of running diagnostic tests

Fault tolerant Deadline Scheduling

A Backup Overloading Scheduling Algorithm The following steps form the procedure used to implement the backup overloading algorithm

1) **Arriving task**

A task has four properties when it arrives, arrival time (ai), Ready time (ri), Deadline – (di) and worst case computation time (ci) represented as Ti = (ai, ri, di, ci)

2) **EDF schedulability**

Check if all the tasks can be scheduled successfully using the earliest deadline first algorithm. If the schedulability test fails, then reject the set of tasks saying that they are not schedulable.

3) **Searching for timeslot**

When task Ti arrives, check each processor to find if the primary copy (Pri) of the task can be scheduled between ri and di. Say it is scheduled on processor Pi.

4) Try overloading

Try to overload the backup copy (Bki) on an existing backup slot on any processor other than Pi. Note: The backups of 2 primary tasks that are scheduled on the same processor must not overlap. If the processor fails, it will not be possible to schedule the two backups simultaneously since they are on the same time slot (overloaded).

5) EDF Algorithm

If there is no existing backup slot that can be overloaded, then schedule the backup on the latest possible free slot depending upon the dead line of the task. The task with the earliest deadline is scheduled first.

6) De-Allocation of backups

If a schedule has been found for both the primary and backup copy for a task, commit the task, otherwise reject it. If the primary copy executes successfully, the corresponding backup copy is deallocated.

7) Backup execution

If there is a permanent or transient fault in the processor, the processor crashes and then all the backups of the tasks that were running on this system are executed on different processors

Clock in distributed system?

- A logical **clock** is a mechanism for capturing chronological and causal relationships in a **distributed system**.
- Often, **distributed systems** may have no physically synchronous global **clock**.

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
- In **serial communication, clock synchronization** can refer to clock recovery which achieves frequency synchronization, as opposed to full phase synchronization. Such clock synchronization is used in synchronization in telecommunications and automatic baud rate detection.

Centralized clock synchronization

- **Centralized Clock Synchronization** is an internal **clock synchronization** approach where **clocks** of system are **synchronized** with one of **clock** of system.
- **Clock synchronization** is carried out to make all **clocks** in network agree on the same value.

centralized clock synchronization, one of clocks is appointed as master clock. Other clock of the system are called slaves and these clocks are kept is synchronization with master clock.

1. **Master Clock :**

   It is one of clocks of system which is designated as master clock. All remaining clocks are synchronized with this clock. Master clock is also known as *time server*. It is single in number.

2. **Slave Clocks :**

   Remaining clocks of systems after designated master clock are known as slave clocks. These clocks are synchronized with master clock of system. These are various in number in system.

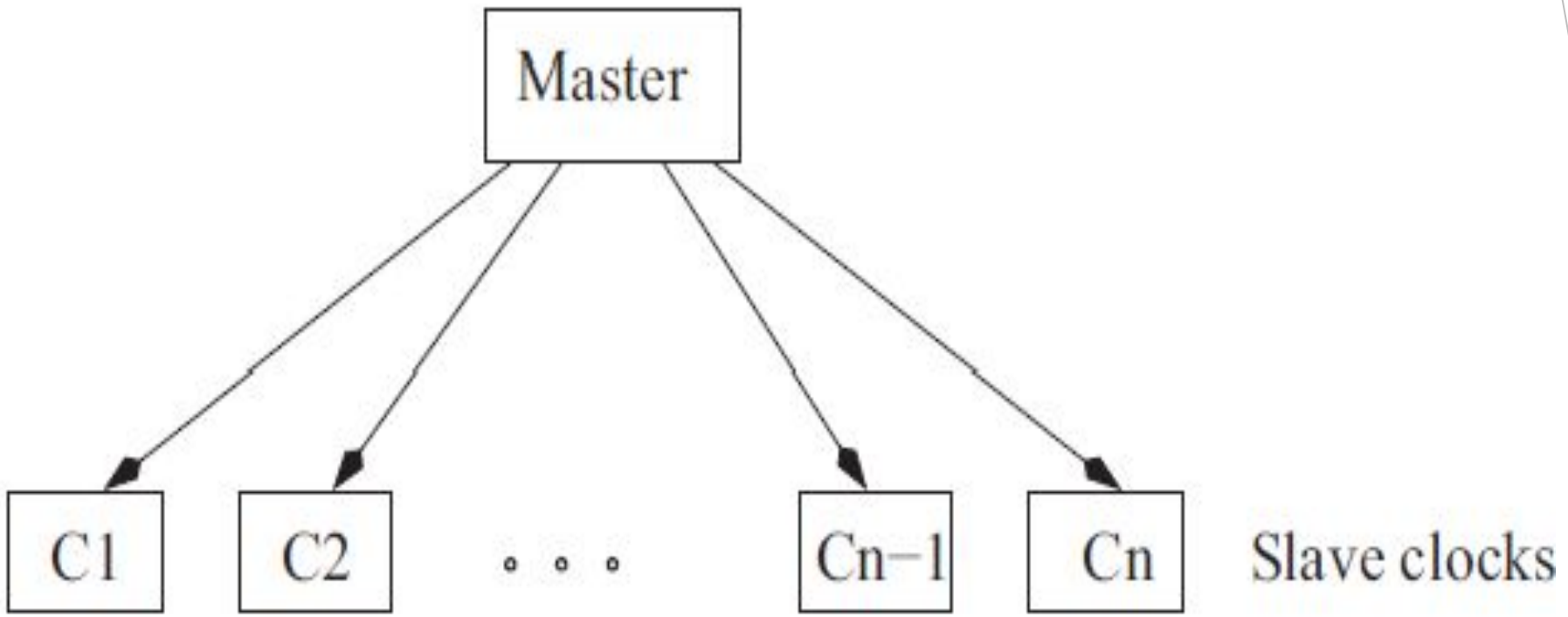


Figure 1: Centralized synchronization system

Working :

- Master clock sends its time to all other clocks (slave clocks) for synchronization.
- Server broadcasts its time after each ‘t’ time interval.
- Slave clocks receive time from master clock and set their time accordingly.
- Time interval ‘t’ is chosen quite carefully.

1.

Clocks in DRTS

- Clocks in a system are useful for two main purposes

1. Determine time out
2. Time stamping

- Clock synchronization(external and internal)

1. Centralized clock synchronization

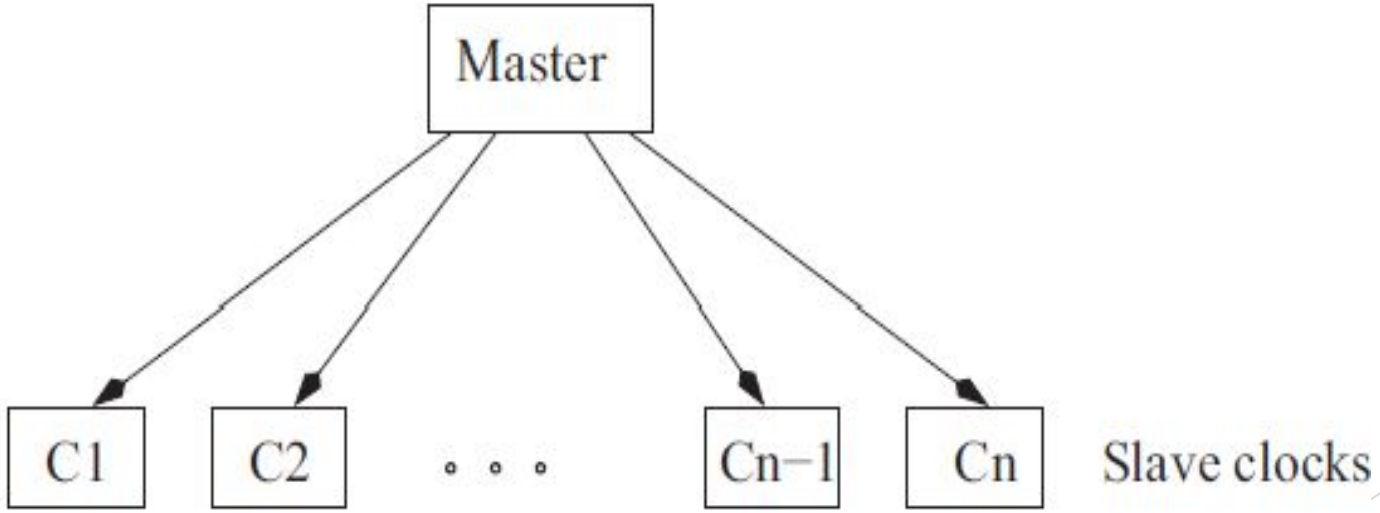


Figure 1: Centralized synchronization system

2. Distributed clock synchronization

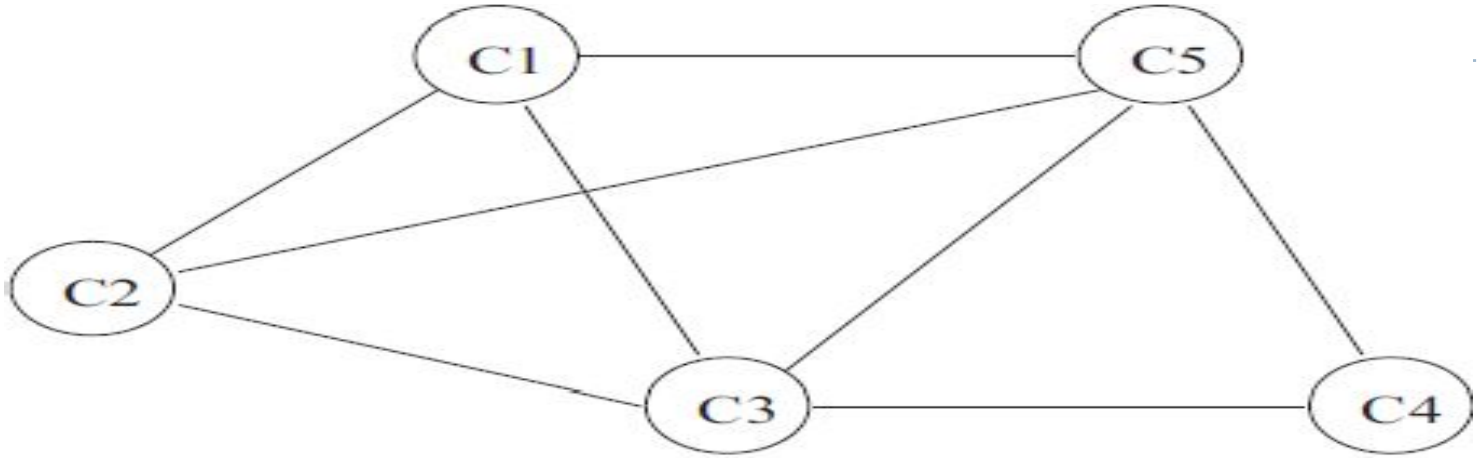


Figure 2: Distributed Clock Synchronization

- Byzantine clock
- Byzantine clock is two faced clock, it can transmit different values to different clocks at the same time.

Fault-tolerant scheduling

- **Fault-tolerant scheduling** algorithms provide the system with a means to continue meeting deadlines until this can be completed;
- Their role is therefore to provide the system with some breathing room before it makes any longer-term adjustments to the **task** assignment and **scheduling** that may be appropriate.

Fault tolerance:

- Fault tolerance aims at guaranteeing the services delivered by the system despite the presence or appearance of faults.
- Fault tolerance approaches are divided into two classes:
  - **Compensation techniques** for which the structural redundancy of the system masks the fault presence, and,
  - **Error detection and recovery techniques**, that is, detection and then resumption of the execution either from a safe state or after the operational structure modification (reconfiguration).Error recovery techniques are split into two sub-classes: Backward recovery aiming at resuming execution in a previously reached safe state and Forward recovery aiming at resuming execution in a new safe state.

Fault removal: Fault removal aims at detecting and eliminating existing faults. Fault removal is older than those on fault prevention. Fault removal techniques are often considered at the end of the model definition, particularly when an operational model of the system is complete.

Fault Evasion: Means to estimate the present number, the future incidence, and the likely consequences of faults.

Scheduling Algorithms are used for fault tolerance as well as fault avoidance which may be classified as

1. First Come First Serve
2. Shortest Job First
3. Preemptive
4. Non-Preemptive
5. Round-Robin Technique

Basics of Fault-Tolerant Scheduling

The general method of responding to a failure is as follows:

- **Transient Failures:** If the system is designed only to withstand transients that go away quickly, reexecution of the failed task or of a shorter, more basic, version of that task is carried out. The scheduling problem reduces to ensuring that there is always enough time to carry out such an execution before the deadline

- **Software Failure:** Here, the failure is that of the software, not of the processor. Software diversity is used: backup software which is different from the failed software is invoked. Again, we have to make sure, in preparing for software faults, that there is enough time for the backup version to meet the original task deadline.

- **Permanent Failure:** Backup versions of the tasks assigned to the failed processor must be invoked.

  The steps are: –

  - Provide each task with a backup copy.
  - Place the backups in the schedule, either prior to operation for offline scheduling or before guaranteeing the task for online scheduling.
  - If a processor fails, activate one backup for each of the tasks that have been affected.

Failure Type	Task Type	Offline/Online	Scheduling Algorithm
Transient	Periodic	Offline	Rate Monotonic
Transient	Aperiodic	Online	Earliest Deadline First
Software Only	Periodic	Offline	Any priority scheme
Permanent	Periods Equal	Offline	Longest exec time first
Permanent	Periodic	Offline	Rate Monotonic
Permanent	Aperiodic; Non-Preemptive	Online	Any scheme
Permanent	Aperiodic with Cost Functions	Offline	Any scheme

Commercial Real-time Operating Systems – An Introduction

- Introduction
- LynxOS
- QNX/Neutrino
- VRTX
- VxWorks
- Spring Kernel

- Commercial RTOS s different from traditional OS – gives more predictability
- Used in the following areas such as:
  - Embedded Systems or Industrial Control Systems
  - Parallel and Distributed Systems
  - E.g. LynxOS, VxWorks, pSoS, QNX , bluecat
- Traditionally these systems can be classified into a Uniprocessor, Multiprocessor or Distributed Real-Time OS

Features of RTOS

- Clock and timer support: clock and timer services with adequate resolution:
- Clock resolution denotes the time granularity provided by the system clock of a computer. Thus the resolution of a system clock corresponds to the duration of time that elapses between two successive clocks ticks.
- Real time priority levels: static priority levels
- Fast task preemption: is the time duration for which a higher priority task waits before it is allowed to execute

- Predictable and fast interrupt latency:

  interrupt latency is the time delay between the occurrence of an interrupt and the running of the corresponding ISR
- Support for resource sharing among RTT
- Requirements on memory management
- Support for asynchronous I/O: non blocking I/O
- Additional requirements for embedded RTOS:cost,size,power consumption

UNIX as a RTOS

- Popular and general purpose OS for mainframes
- Two most important problems that a real time programmer faces while using unix for real time applications are
    1. Non preemptive unix kernel
    2. Dynamically changing priorities of tasks

Non preemptive kernel

- Means all interrupts are disabled when any OS routine runs
- A process running in kernel mode can't be preempted by other processes. Unix system preempt processes running in the user mode.
- Consequence of this is that even when a low priority process makes a system call ,the high priority processes would have to wait until the system call by the low priority process completes
- For RT applications this causes a priority inversion
- Kernel routine starts to execute ,all interrupts are disabled

.interrupts are enabled only after the OS routine completes

Dynamic Priority Levels

- Unix uses round robin scheduling of tasks with multilevel feedback
- Scheduler arranges tasks in multilevel queues
- At every preempting point the scheduler scans the multilevel queue from the top(highest priority) and selects the first task of the non empty queue
- Each task is allowed to run for a fixed time quantum at a time , unix normally uses one second time slice.

- The kernel preempts a process that doesn't complete within its assigned time quantum ,recomputed its priority and inserts it back into one of the priority queues depending on the recomputed priority value of the task.
- The basic philosophy of Unix operating System is that the interactive tasks are made to assume higher priority levels and are processed a the earliest . This gives the interactive users good response time.

$Pri(T_i,j) = Base(T_i)+CPU(T_i,j)+nice(T_i)$

$Pri(T_i,j)$=priority of the task T_i at the end of j^{th} time slice

Base(Ti)=base priority is established at process creation, based on the specific type of process being created.

$CPU(T_i,j)$=weighted history of CPU utilization of the task Ti at the end of j^{th} time slice

Nice(Ti)=

$U(T_i,j)$=is the utilization of the task T_i

Other deficiencies of UNIX

- Insufficient device driver support
- Lack of RT file services
- File blocks are allocated as and when they are requested by an application.
- no guarantee is given thatdisk space would be available when a task writes a block to a file
- Traditional file writing policies result in slow writes
- Blocks of the same file may not be contiguously located on the disk
- Inadequate timer services support
- Real time timer support is insufficientfor many hard RTA
- Clock resolution is 10ms is too bad for hard RTA

Unix based RTOS

Different approaches that have been undertaken to make unix suitable for RT applications

- Host target approach

1. Host target OS are popularly being deployed in embedded applications
2. RT application development is done on a host machine which is either traditional unix or windows system supporting the program development environment, compilers, editors, library, cross compilers , debuggers etc
3. RT application is developed on the host and is then cross compiled to generate code for the target processor .the developed application is downloaded onto target board that is to be embedded in a RTS via a serial port or a TCP/IP connection
4. ROM resident small RT kernel is used in the target board and once the program works successfully it is fused in the ROM and becomes ready to deployed in applications
5. Ex:VxWorks,PSOS,VRTX

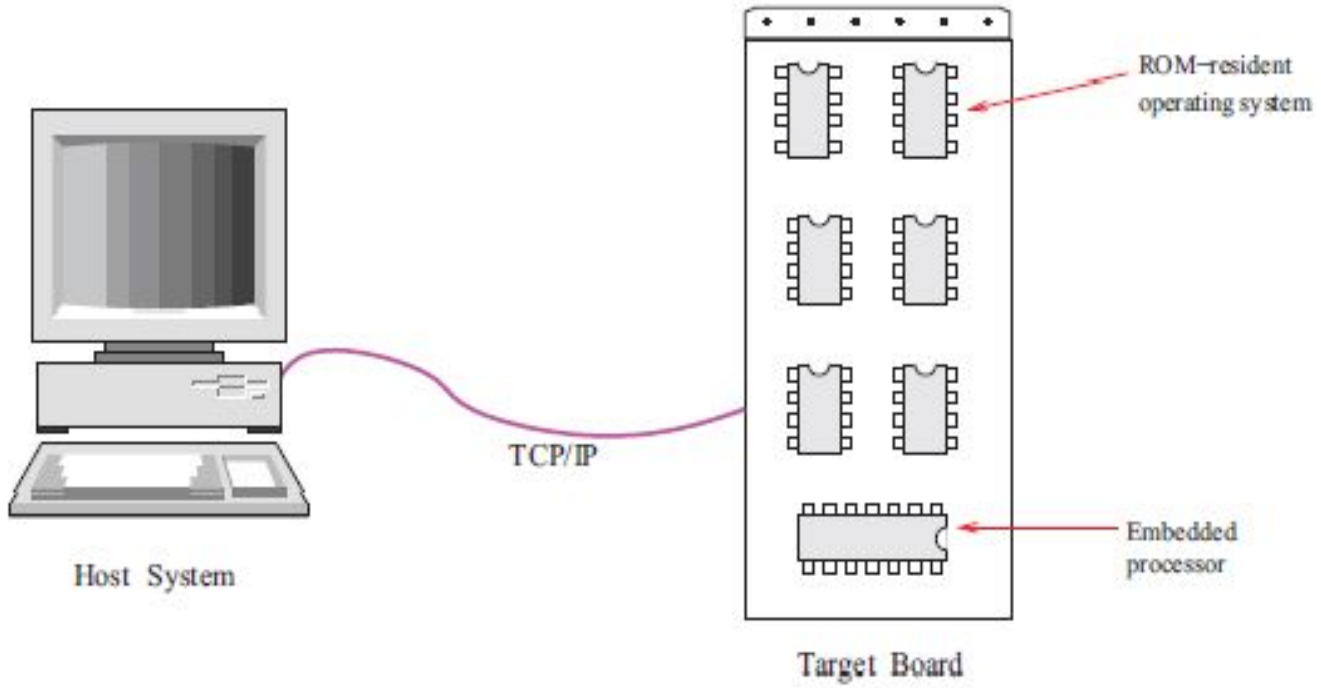


Figure 5: Schematic Representation of a Host-Target System

It needs cross compiler and cross debugger

- Extensions to the traditional unix kernel for RT Applications

I. by adding some RT capabilities(RT timer support, RT task scheduler) over the kernel.

- Preemption Point approach

1. Preemption points in the execution of a system routine are the instants at which the kernel data structure is consistent
2. In this point kernel can safely be preempted to make way for any waiting higher priority RTT to run without corrupting any kernel Data structures
3. The execution of a system call reaches a preemption point the kernel checks to see whether any higher priority tasks have become ready .if there is at least one ,it preempts the processing of the kernel routine and dispatches the waiting highest priority task immediately
4. Ex: HP UX, Windows CE

Self host systems

1. A RTA is developed on the same OS on which the RTA would finally run
2. Once the application runs satisfactorily on the host, it is fused on a ROM or flash memory on the target board along with a possibly stripped down version of the OS.
3. While deploying the application the OS modules that are not essential during task execution are excluded to minimize the size of the OS
4. Based on micro kernel architecture-only the core functionalities such as interrupt handling and process management are implemented as kernel routines. All other functionalities such as memory management, file management, device management etc are implemented as add on modules which operate in the user mode

Non Preemptive kernel

- It is necessary to use locks at appropriate places in the kernel code to overcome the problem. Two types of locks are used in fully preemptive unix systems

1. Kernel level locks
   - Similar to traditional lock
   - Inefficient due to context switch overhead
2. Spin lock

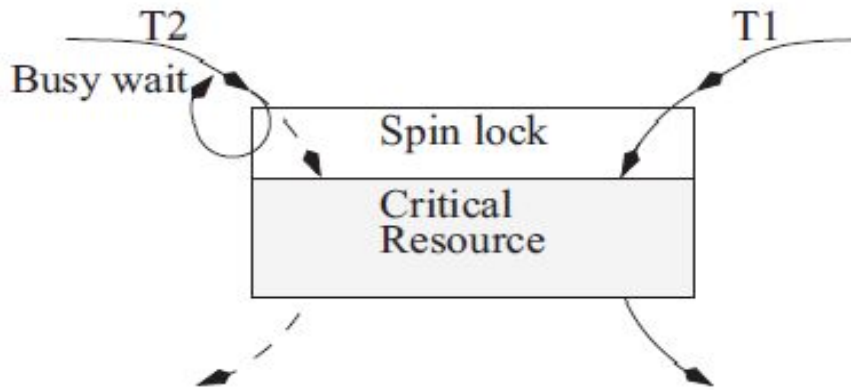
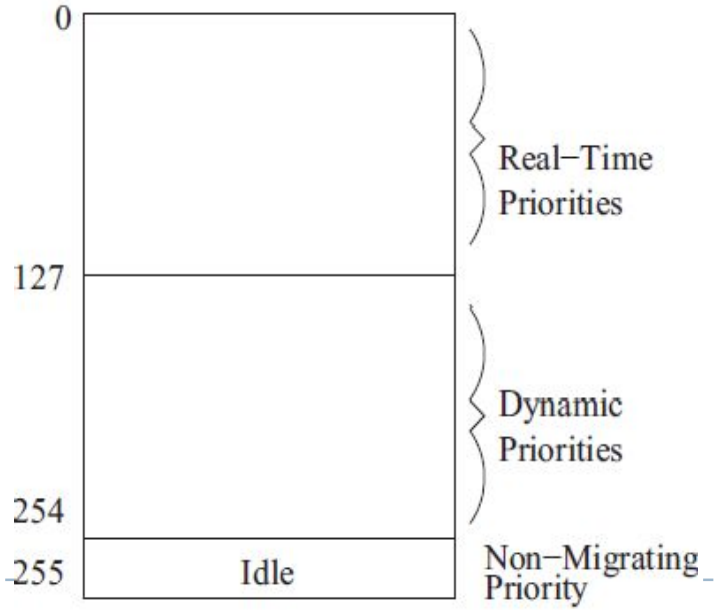


Figure 6: Operation of a Spin Lock

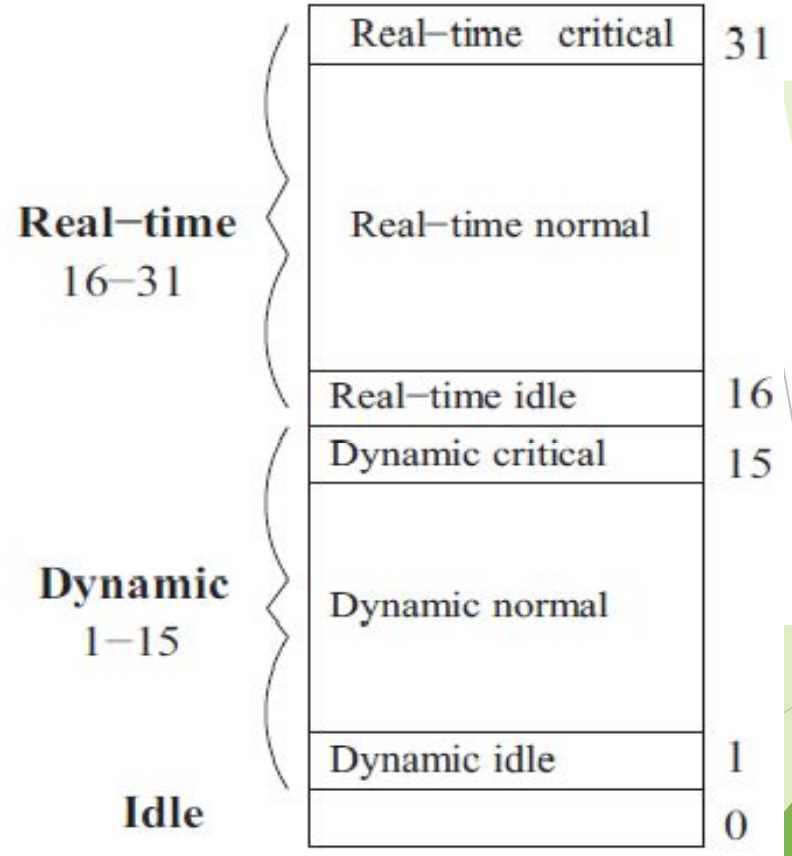
RT priorities

- Unix based RTS support dynamic ,RT and idle priorities
- Idle-lowest priority level, idle task run this level, static and are not recomputed periodically
- Dynamic-recomputed periodically,
- RT-static priorities and are not recomputed during run time, hard RTT operate at these levels



Windows as a RTOS

- Important features of windows NT
- Timer and clock resolutions are sufficiently fine for most RTA
- Support for multithreading
- Availability of RT priority levels.
- Support 32 priority levels.



Shortcomings of windows NT

- Interruptprocessing-priority level of interrupts is higher than user level threads.
- ISR(interrupt service routine):very critical processing is performed.
- DPC(deferred procedure call):for low priority process.
- Support for resource sharing protocols-NT doesn't provide any support to RTT to share CR among themselves, simplest approach is by careful priority settings while acquiring and releasing locks, another possibility is to implement PCP

POSIX(portable operating system interface)

- Open software

- Open system-are based on open standards and are not copyrighted, allows users to intermix h/w , s/w and networking solutions from different vendors, interoperability(systems from multiple vendors can exchange information among each other) and portability
- Reduces the cost of development, increase the availability of add- on software packages ,enhances ease of programming and facilitates easy integration of separately developed modules
- POSIX stands for Portable Operating System Interface, and is an IEEE standard designed to facilitate application portability.
- POSIX is an attempt by a consortium of vendors to create a single standard version of UNIX.
- If they are successful, it will make it easier to port applications between hardware platforms.

- POSIX is an evolving group of standards, each of which covers different aspects of the operating systems.
- Open software Standards
  1. Open source-provides portability at the source code level
  2. Open object-provides portability of unlinked object modules across different platforms
  3. Open binary-provide complete s/w portability across h/w platforms based on a common binary language structure

POSIX.1	System Interface (basic reference standard)[a,b]
POSIX.2	Shell and Utilities[a]
POSIX.3	Methods for Testing Conformance to POSIX[a]
POSIX.4	Real-time Extensions
POSIX.4a	Threads Extensions
POSIX.4b	Additional Real-time Extensions
POSIX.6	Security Extensions
POSIX.7	System Administration
POSIX.8	Transparent File Access
POSIX.12	Protocol Independent Network Interfaces
POSIX.15	Batch Queuing Extensions
POSIX.17	Directory Services

Overview of posix

- POSIX standard defines only interfaces to OS services and the semantics of these services, but doesn't specify how exactly the services are to be implemented
- Specifies the system calls that an OS needs to support
- Important parts of POSIX

1. POSIX 1: system interfaces and system call parameters
2. POSIX 2: shells and utilities
3. POSIX 3: test methods for verifying conformance to POSIX
4. POSIX 4: RT extensions

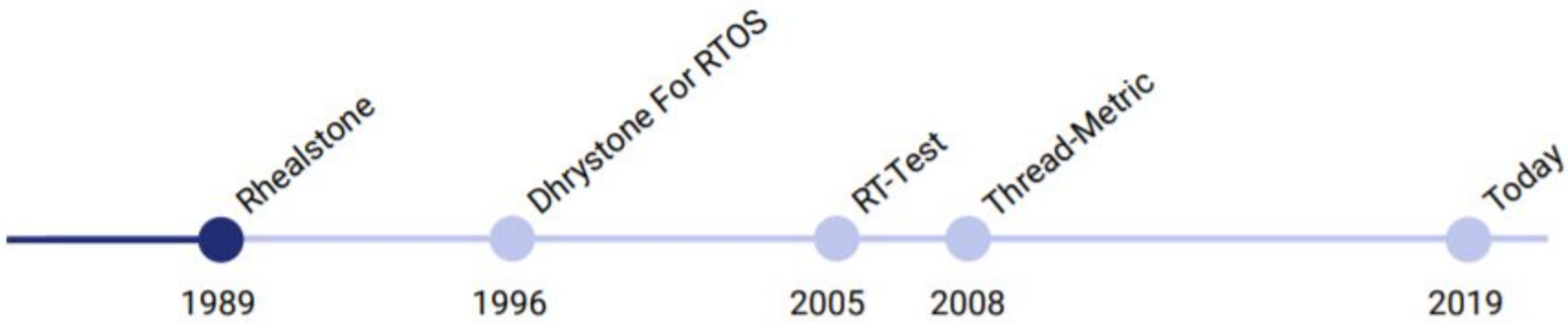
RT POSIX Standard

- Main requirements of POSIX-RT are
  - **Execution scheduling**: support for real time priorities
  - **Performance requirements on system calls**: worst case execution times required for most RTOS services has been specified by POSIX -RT
  - **Priority levels**: should be at least 32
  - **Timers**: periodic and one shot timers
  - **RT files**: RTF system should be support
  - **Memory locking**
  - **Multithreading support**

Benchmarking real time systems

- This **benchmark** defines a set of metrics to measure the performance of a **real time** operating **system**.
- It's then proposed how the results are going to be weighted together to form a single value which represents the **RTOS** performance.
- Why benchmarking?
  - Embedded systems have limited resources.
  - Overhead of the chosen RTOS can impact your design.

Previous RTOS Benchmarks



- The Rhealstone benchmark is comprised of 6 test scenarios. [1]

1. Task switching time
2. Task preemption time
3. Interrupt latency time
4. Semaphore shuffling time
5. Deadlock breaking time
6. Intertask message latency

$$RhealStone\ Performance\ Number = \frac{t_1+t_2+t_3+t_4+t_5+t_6}{6}$$

A Modern Benchmark Proposal

- Covers the most common RTOS services
  - Semaphores
  - Mutex
  - Event Flags
  - Message Queues
  - Cooperative Scheduling
  - Preemptive Scheduling
- Offers a reference implementation written with portability in mind.
  - Executing on a new RTOS only requires writing a thin porting layer.
- Produces accurate timing results.
- Results are either:
  - Printed on the standard output
  - Computed in Trace Compass and synchronized with an OS trace.

How Scenarios Are Built

- Examples: Cooperative scheduling and Semaphore scenarios.
  - Get precise measurements rather than averages.

Figure 1. Cooperative scheduling scenario

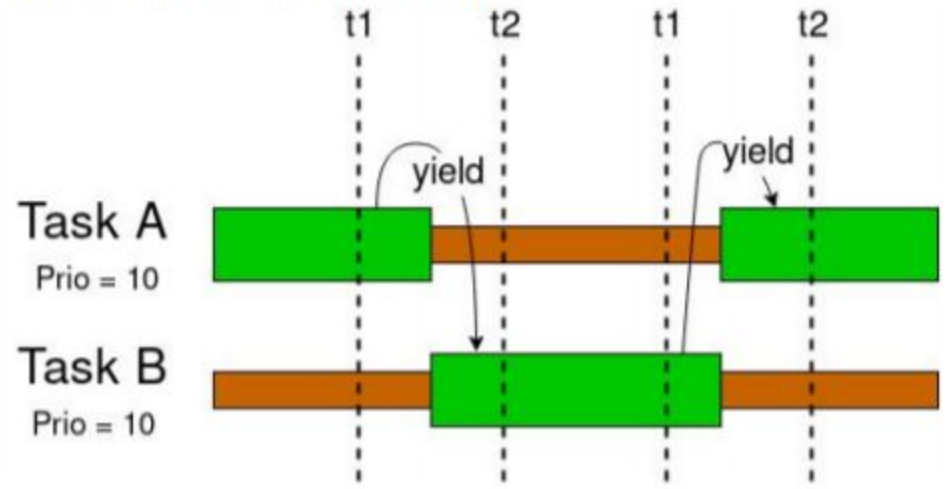
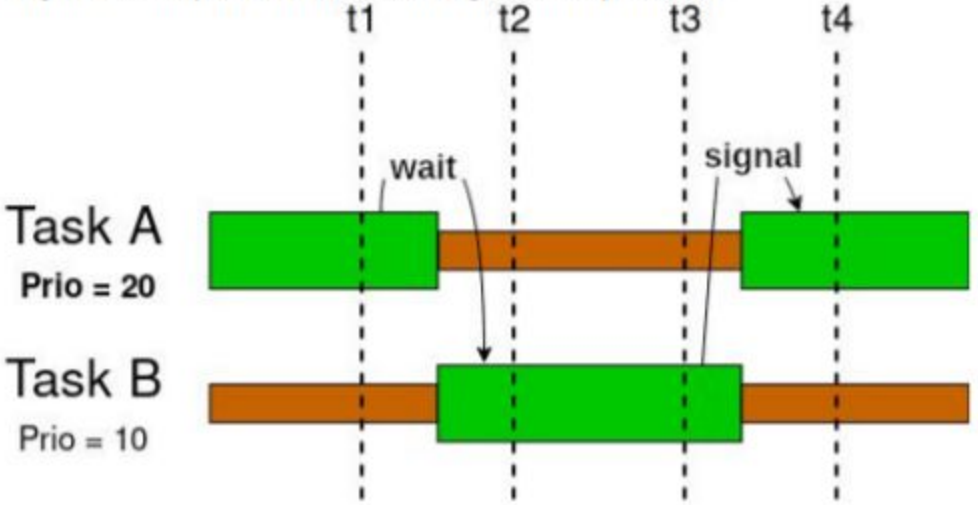


Figure 2. Semaphore wait-block and signal-wakeup scenario



Semaphore

- Measure the overhead of this service in different scenarios.
  - Signal with empty wait queue. (Signal)
  - Wait on available semaphore. (Wait)
  - Signal causing context switch. (Signal-unblock)
  - Wait on semaphore causing context switch. (Wait-block)

Diagram 1. Average Cycles for Semaphore

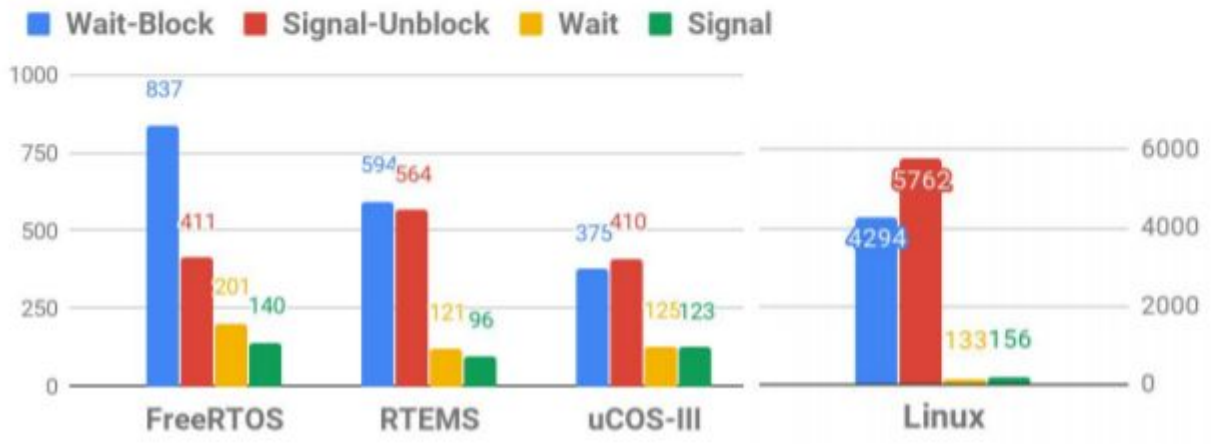
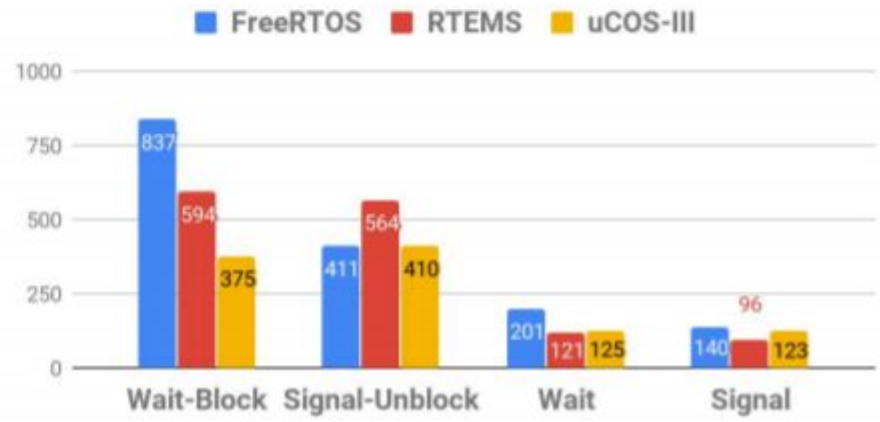


Diagram 2. Average Cycles for Semaphore



Benchmark Setup

- Benchmark executed on a 32 bit RPI 2B+ (MPCore Cortex A-7 900MHz).
  - FreeRTOS v10.1
  - uCOS-III
  - RTEMS v4.11
  - Linux 4.14.98-v7+
- L1 & L2 cache enabled, 1 to 1 virtual to physical address mapping (except Linux).
- 1kHz tick rate (1 ms period).
- Measurements are obtained through the Cycle Count Register.

Understanding the Results

FreeRTOS takes twice as much cycles to take and block on a semaphore than to signal and switch to a new task.

- Other OS do not exhibit the same behavior.

- Tracing the execution can provide insigher uCOS-III maximum time is 4.5x higher than the average time.
- FreeRTOS maximum time is 2x higher than the average time.
- uCOS-III schedules a Tick Task in the tick interrupt handler.

Benchmarking real time systems

- MIPS(MillionInstructions Per Second) and FLOPS(FLoating Point Operations Per Second)
- Rhealstone Metric

6 parameters of RTS are considered

1. Task Switching Time (t_{ts})
2. Task Preemption Time(t_{tp})
3. Interrupt Latency Time (t_{il})
4. Semaphore shuffling time(t_{ss})
5. Unbounded Priority Inversion Time(t_{up})
6. Datagram Throughput time(t_{dt})

The Rhealstone metric is computed as a weighted average of the identified parameters:

$$Rhealstone\ \ Metric\ \ = \ \ a_1 * t_{ts} + a_2 * t_{tp} + a_3 * t_{il} + a_4 * t_{ss} + a_5 * t_{up} + a_6 * t_{dt}$$

Where, a1, ..., a6 are empirically determined constants.

1. Task Switching Time (t_{ts}):

It is defined as the time it takes for one context switch among equal priority tasks

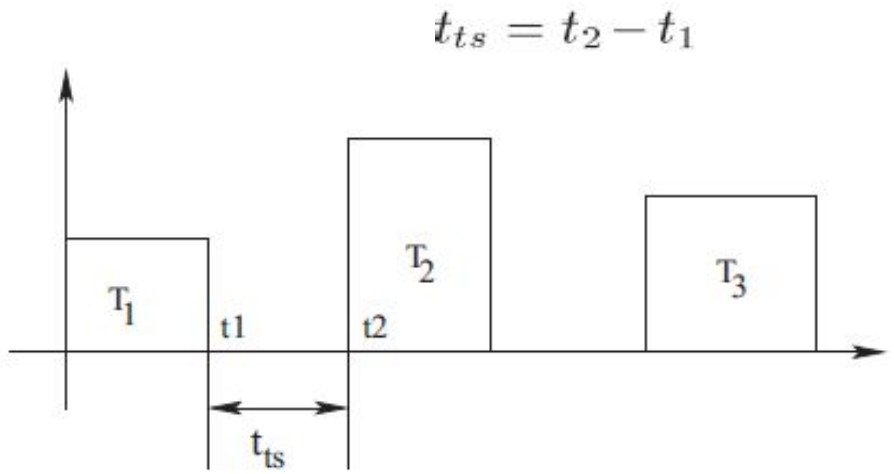


Figure 13: Task Switching Time Among Equal Priority Tasks

2. Task Preemption Time(t_{tp}):

Defined as the time it takes to start execution of a higher priority task ,after the condition enabling the task occurs .consists of the following 3 components

I. Task switching time
II. Time to recognize the event enabling the higher priority.
III. Time to dispatch

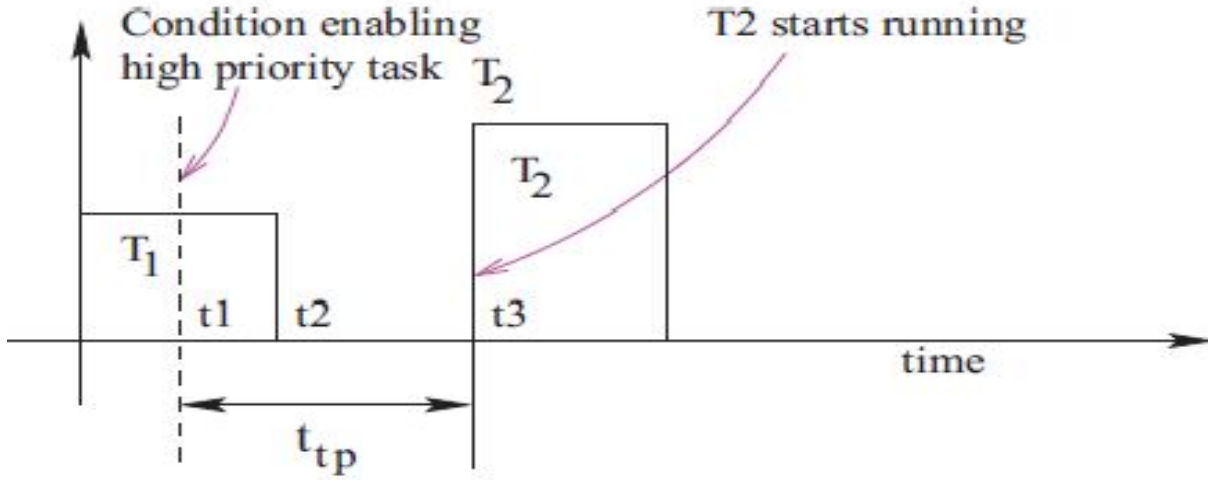


Figure 14: Task Preemption Time

3. Interrupt Latency Time (t_{il}):

Consists of the following components

1. Hardware Delay in CPU recogonizing the interrupt.
2. Time to complete the current instruction.
3. Time to save the context of the currently running task.
4. Start the ISR

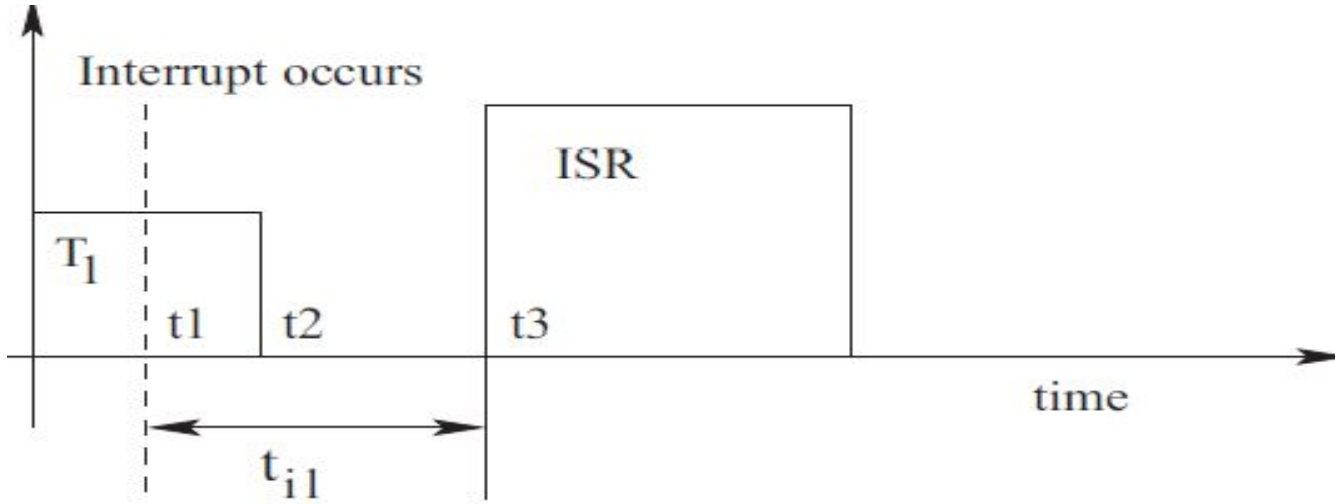


Figure 15: Interrupt Latency Time

4. Semaphore shuffling time(t_{ss}):

It is defined as the time elapses between a lower priority task releasing a semaphore and a higher priority task to start running

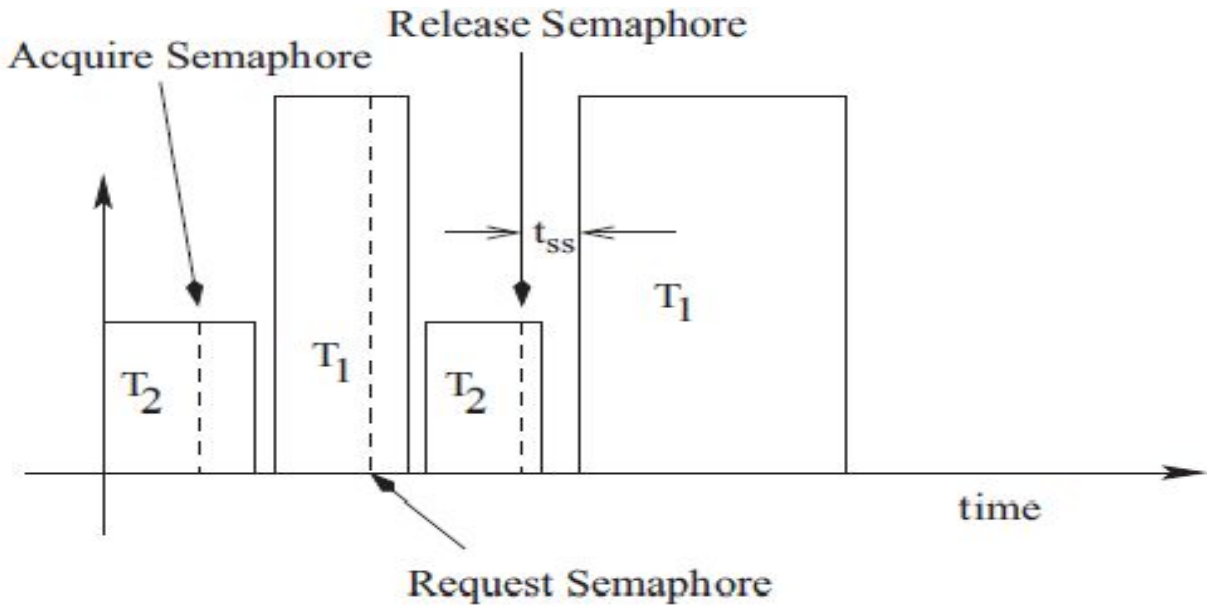


Figure 16: Semaphore Shuffling Time

5. Unbounded Priority Inversion Time(t_{up}):

$t_{up}=t_1+t_2$

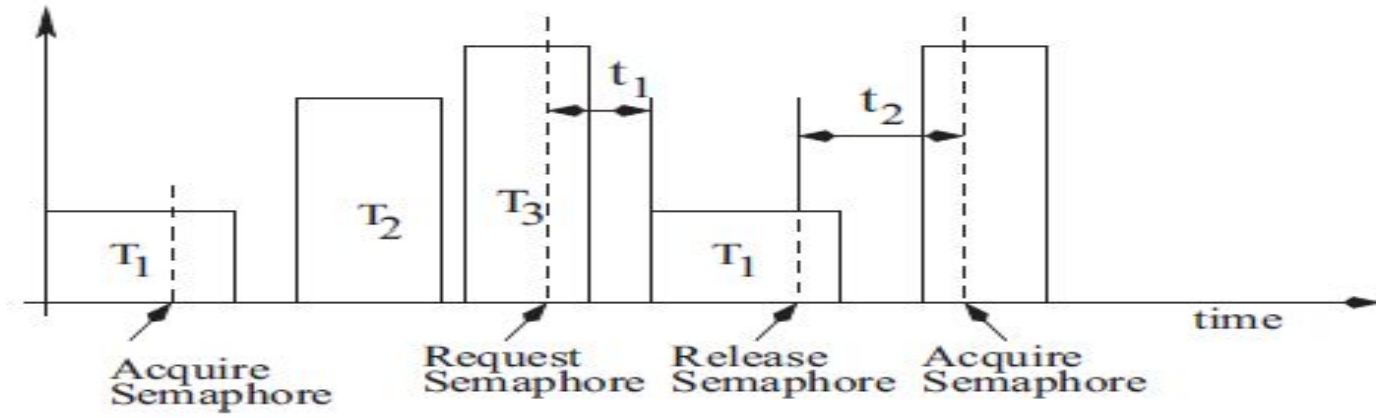


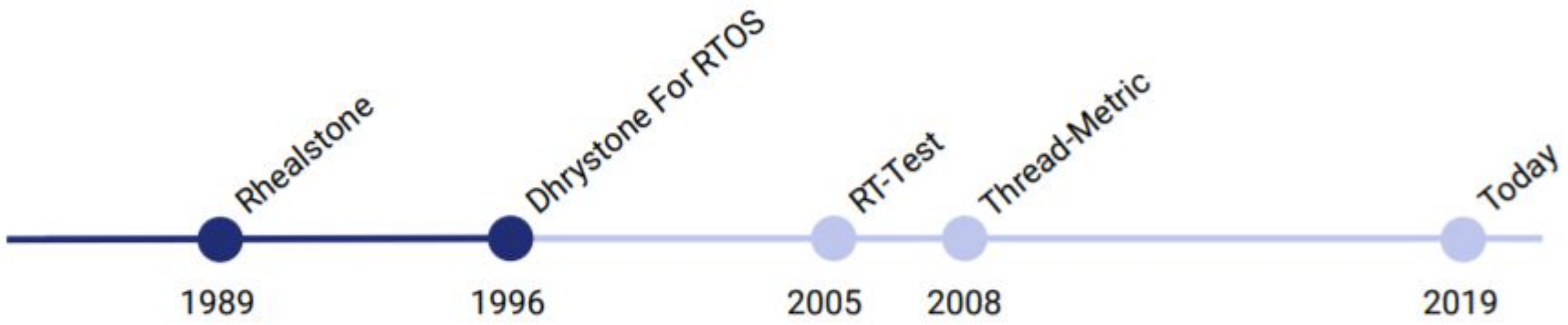
Figure 17: Unbounded Priority Inversion Time

It is computed as the time it takes for the OS to recognize priority inversion(t_1) and run the task holding the resource and start T_2 after T_1 completes(t_2)

6. Datagram Throughput time(t_{dt})

Indicates the number of kilobytes of data that can be transferred between two tasks without using shared memory or pointers

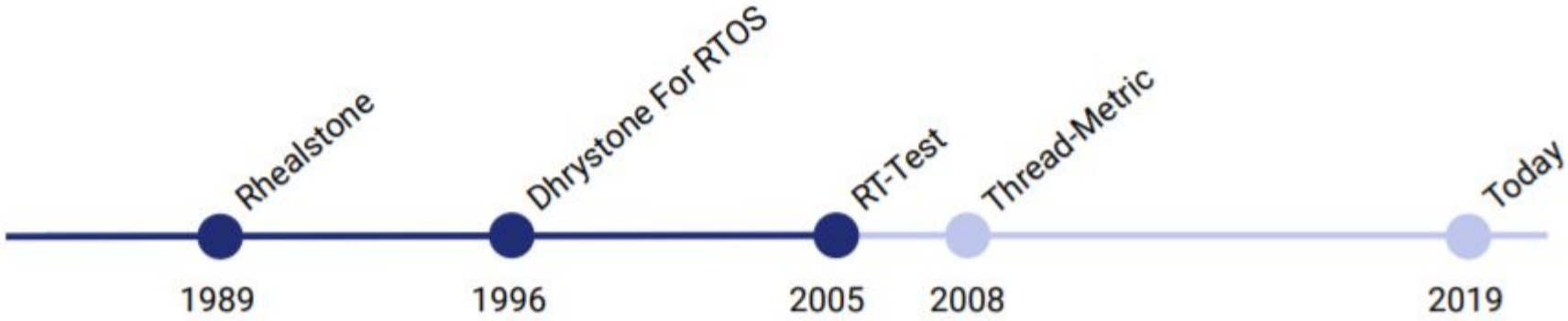
Previous RTOS Benchmarks



- This benchmark uses a Dhrystone loop as the workload and make uses of RTOS services in 6 scenarios to estimate their overhead. [2]
- Results are expressed in *Dhrystone per seconds*.

1. Round Robin
2. Task Priority Preemption
3. Semaphore
4. Memory alloc/dealloc
5. Interrupt Latency
6. Message passing

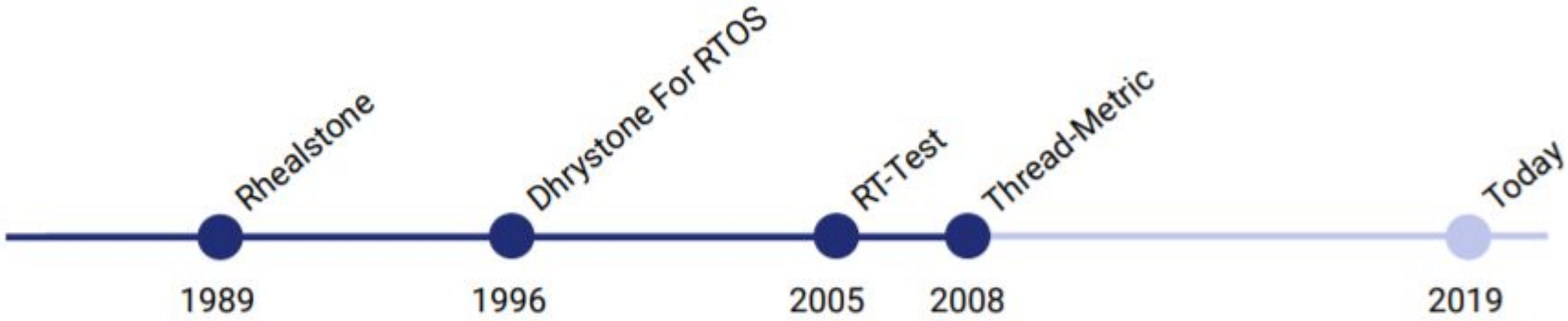
Previous RTOS Benchmarks



- RT-Test aims measures real time performance of a Linux system. [3]
- *Cyclictest* is its most known test to estimate system latency.
- Results are time measurements.

1. Message Queue latency
2. Semaphore latency
3. Mutex latency
4. Signal Latency
5. Signal round trip
6. Cyclic test

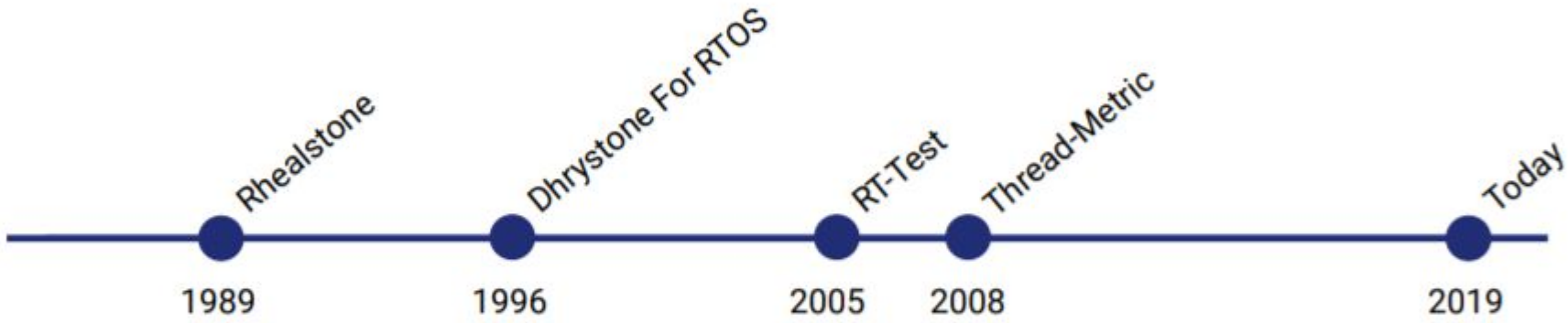
Previous RTOS Benchmarks



- Published by Express Logic, the developers of ThreadX. [4]
- Express Logic offers an easily portable reference implementation.
- Results are computed using a loop counter.

1. Cooperative context switch
2. Preemptive context switch
3. Interrupt processing
4. Message passing
5. Semaphore processing
6. Memory alloc/dealloc

Previous RTOS Benchmarks



- The previous benchmarks are either:
  - Tedious to port to a new RTOS.
  - Inaccurate.
- We propose a new benchmark that is more accurate and easy to port.

Conclusion

- Benchmarking helps to understand the behavior of your RTOS.
  - Unexpected average execution time.
  - Unexpected worst case time.
- The port for the RTOS might have room for improvement.
  - Inlining entry/exits to critical sections.
- Knowing the available configuration options is the key to getting the best performance possible.
  - Background tasks priority.
- Benchmarking helps you make an informed design choice.

Real-time database

- A **real-time database** is a database system which uses real-time processing to handle workloads whose state is constantly changing.
- Real-time databases are traditional databases that use an extension to give the additional power to yield reliable responses
- This differs from traditional databases containing persistent data, mostly unaffected by time.
- Real-time databases are useful for accounting, banking, law, medical records, multi-media, process control, reservation systems, and scientific data analysis
- Real-time processing means that a transaction is processed fast enough for the result to come back and be acted on right away

- Real-time databases are useful for accounting, banking, law, medical records, multi-media, process control, reservation systems, and scientific data analysis.
- They use timing constraints that represent a certain range of values for which the data are valid. This range is called temporal validity.
- A conventional database cannot work under these circumstances because the inconsistencies between the real world objects and the data that represents them are too severe for simple modifications.
- An effective system needs to be able to handle time-sensitive queries, return only temporally valid data, and support priority scheduling.
- To enter the data in the records, often a sensor or an input device monitors the state of the physical system and updates the database with new information to reflect the physical system more accurately.

- When designing a real-time database system, one should consider how to represent valid time, how facts are associated with real-time system.
- Also, consider how to represent attribute values in the database so that process transactions and data consistency have no violations.
- In real-time databases, deadlines are formed and different kinds of systems respond to data that does not meet its deadline in different ways.
- In a real-time system, each transaction uses a timestamp to schedule the transactions
- A priority mapper unit assigns a level of importance to each transaction upon its arrival in the database system that is dependent on how the system views times and other priorities.
- The timestamp method relies on the arrival time in the system.

Real-time databases are useful for

- accounting,
- banking,
- law,
- medical records,
- multi-media,
- process control,
- reservation systems,
- and scientific data analysis

Examples of database applications

- Amazon
- CNN
- eBay
- Facebook
- Fandango
- Filemaker (Mac OS)
- Microsoft Access
- Oracle relational database
- SAP (Systems, Applications & Products in Data Processing)
- Ticketmaster
- Wikipedia
- Yelp
- YouTube
- Google
- My SQL

Responses:

Below is a description of these responses:

Hard deadline

- If not meeting deadlines creates problems, a hard deadline is best.
- It is periodic, meaning that it enters the database on a regular rhythmic pattern.
- An example is data gathered by a sensor.
- These are often used in life critical systems

Firm deadline

- Firm deadlines appear to be similar to hard deadlines yet they differ from hard deadlines because firm deadlines measure how important it is to complete the transaction at some point after the transaction arrives.
- Sometimes completing a transaction after its deadline has expired may be harmful or not helpful, and both the firm and hard deadlines consider this.
- An example of a firm deadline is an autopilot system.

Soft deadline

- If meeting time constrains is desirable but missing deadlines do not cause serious damage, a soft deadline may be best.
- It operates on an aperiodic or irregular schedule.
- In fact, the arrival of each time for each task is unknown.
- An example is an operator switchboard for a telephone.

- Hard deadline processes abort transactions that have passed the deadline, improving the system by cleaning out clutter that needs to be processed.
- Processes can clear out not only the transactions with expired deadlines but also transactions with the longest deadlines, assuming that once they reach the processor they would be obsolete.
- This means other transactions should be of higher priority.

- The goal of scheduling periods and deadlines is to update transactions guaranteed to complete before their deadline in such a way that the workload is minimal.
- With large real-time databases, buffering functions can help improve performance tremendously.
- A buffer is part of the database that is stored in main memory to reduce transaction response time.
- In order to reduce disk input and output transactions, a certain number of buffers should be allocated

Temporal database

A **temporal database** stores data relating to time instances.

It offers temporal data types and stores information relating to past, present and future time.

Temporal databases could be uni-temporal, bi-temporal or tri-temporal.

More specifically the temporal aspects usually include valid time, transaction time or decision time.

- **Valid time** is the time period during which a fact is true in the real world.
- **Transaction time** is the time at which a fact was recorded in the database.
- **Decision time** is the time at which the decision was made about the fact.

Uni-Temporal

A uni-temporal database has one axis of time, either the validity range or the system time range

Bi-Temporal

A bi-temporal database has two axis of time:

- valid time
- transaction time or decision time

Tri-Temporal

A tri-temporal database has three axes of time:

- valid time
- transaction time
- decision time

- This approach introduces additional complexities.
- Temporal databases are in contrast to current databases (not to be confused with currently available databases), which store only facts which are believed to be true at the current time.

Features

Temporal databases support managing and accessing temporal data by providing one or more of the following features:

- A time period datatype, including the ability to represent time periods with no end (infinity or forever)
- The ability to define valid and transaction time period attributes and bitemporal relations
- System-maintained transaction time
- Temporal primary keys, including non-overlapping period constraints

- Temporal constraints, including non-overlapping uniqueness and referential integrity
- Update and deletion of temporal records with automatic splitting and coalescing of time periods
- Temporal queries at current time, time points in the past or future, or over durations
- Predicates for querying time periods, often based on Allen’s interval relations

Temporal Consistency

Temporal consistency of data has the following two main requirements

Absolute Validity :

This is the notion of consistency between the environment and its reflection in the database given by the data collected by the system about the environment

Relative Consistency

This is the notion of consistency among the data are used to derive new data

Concurrency control in real-time databases

- The **concurrency control** of transactions in a **real-time database** must satisfy not only the consistency constraints of the **database** but also the timing constraints of individual transactions

- The correctness of transaction processing depends not only on maintaining consistency constraints and producing correct results but also on the time at which a transaction is completed.

- Transactions must be scheduled in such a way that they can be completed before their corresponding deadlines expire.

- Concurrency control schemes normally ensures non interference among transactions by restricting concurrent transactions to be serializable.

- A database is called serializable ,If the database operations carried out by them is equivalent to some seral execution of these transactions

The main categories of concurrency control mechanisms are:

- **Optimistic** - Delay the checking of whether a transaction meets the isolation and other integrity rules until its end, without blocking any of its operations , and then abort a transaction to prevent the violation, if the desired rules are to be violated upon its commit. An aborted transaction is immediately restarted and re-executed, which incurs an obvious overhead. If not too many transactions are aborted, then being optimistic is usually a good strategy.

- **Pessimistic** - Block an operation of a transaction, if it may cause violation of the rules, until the possibility of violation disappears. Blocking operations is typically involved with performance reduction.

- **Semi-optimistic** - Block operations in some situations, if they may cause violation of some rules, and do not block in other situations while delaying rules checking to transaction's end, as done with optimistic.

For example,

Both the update and query operations on the tracking data for a missile must be processed within a given deadline: otherwise, the information provided could be of little value

Commercial real-time databases

- A **commercial database** is a **database** developed and maintained by a **commercial** entity that is generally made available to customers and potential customers
- A commercial real time database need to avoid using anything that can introduce unpredictable latency.
- It need to avoid using I/O operations , message passing or garbage collection .
- **Real-time databases** are useful for accounting, banking, law, medical records, multi-media, process control, reservation systems, and scientific data analysis.

Some of the Commercial Databases are :

- Aerospike DBS
- ArangoDB
- eXtremeDB
- Ehcache
- GigaSpaces
- InfinityDB
- MonetDB
- solidDB etc.

Real-time Communication

- **Real-time communications** (RTC) is a term used to refer to any live telecommunications that occur without transmission delays.
- RTC is nearly **instant** with minimal latency. RTC data and messages are not stored between transmission and reception.
- RTC is generally a peer-to-peer, rather than broadcasting or multicasting, transmission.
- Examples of RTC include
- The Internet, land lines, mobile/cell phones, instant messaging (IM), Internet relay chat, video conferencing, teleconferencing and robotic telepresence. Emails, bulletin boards and blogs are not RTC channels but occur in time-shifting mode, where there is a significant delay between data transmission and reception.

- RTC features were first introduced in Windows XP and included Microsoft Office Communicator, MSN Messenger, Windows Messenger, real-time voice and video and IM

  .

- Microsoft operating systems and software applications include RTC platforms comprised of RTC-enabled component sets.

- In RTC, there is always a direct path between the source and the destination. Although the link might contain several intermediate nodes, the data goes from source to destination without being stored in between them.

Real-time communications can take place in half-duplex or full-duplex modes:

- **Half-duplex RTC.** Data transmission can happen in both directions on a single carrier or circuit but not at the same time.
- **Full-duplex RTC.** Data transmission can occur in both directions simultaneously on a single carrier or circuit.

Real-time communications examples

Real-time communications tools and applications are many and varied, ranging from old-school telephony to cloud communications services.

They include the following:

- fixed-line telephony
- mobile telephony
- voice over IP (VoIP)
- teleconferencing
- video calling
- video conferencing
- presence
- file sharing
- screen sharing
- automatic, live meeting transcription
- team messaging (real-time or near-real-time)
- one-to-one IM (real-time or near-real-time)
- live customer chat (real-time or near-real-time)
- robotic telepresence
- two-way or multiway amateur radio

THANK YOU