RTOS: Deadline and Rate Monotonic Scheduling

Prof. P.H.Dave
DDU, Nadiad;
-
Prof. H.B.Dave
eInfochips, Ahmedabad.

March, 2015

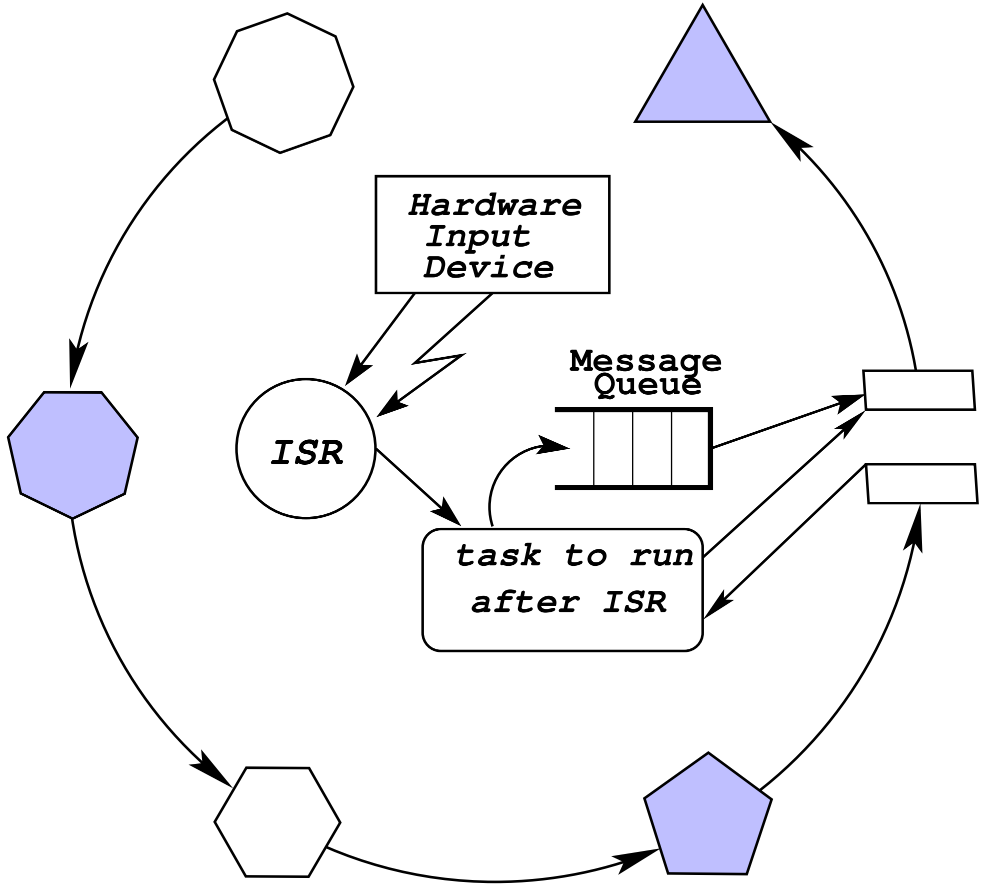
Task schedulers vs. RTOS

- Mechanisms for IPC as well as communication to/from ISRs need be provided
- The primary difference between *just* a task scheduler, and a *full-fledged* RTOS
- The specific mechanisms provided vary in the various RTOSs.
- Most real-time operating systems provide messaging via queues, semaphores and some additionally provide event flags, pipes, mutexes, and/or asynchronous signals.

Use of a message queue

- The message passing services of the Real-time Operating System guarantee the integrity of the message that is passed from task to task, from ISR to task, or from task to ISR.
- This must be used every time information is to be passed between tasks, in order to ensure reliable delivery of that information in a pre-emptible environment

A Pre-emptive Task can communicate via a RTOS Message Queue



Deadline scheduler

- How can we tell the scheduler the deadlines for our tasks to be met?
- The schedulers discussed till now have no way of dealing with that kind of information.
- We can specify only each task's priority P and then they'll do their task scheduling based on P.
- The mapping between deadlines and priorities is usually not simple.
- It is almost impossible for a software designer to be 100% sure that tasks will always meet their deadlines if a priority-based preemptive scheduler with fixed task priorities is used.

- A Deadline scheduler tries to provide execution time to the task that is most quickly approaching its deadline.
- This is usually done by the scheduler changing priorities of tasks on-the-fly as they approach their individual deadlines.

Deadline Scheduling: Information about a task

- ready time
- starting deadline : a time by which a task must begin
- completion deadline
- processing time
- resource requirements
- priority
- subtask structure: mandatory and optional subtask

Schedulability

- Let
$$P = \{p_1, p_2, ..., p_n\}$$
be a task set.
- a task p_i is said to be schedulable if it meets its deadline all the time
- P is said to be schedulable if each task in P is schedulable

Earliest Deadline Scheduling

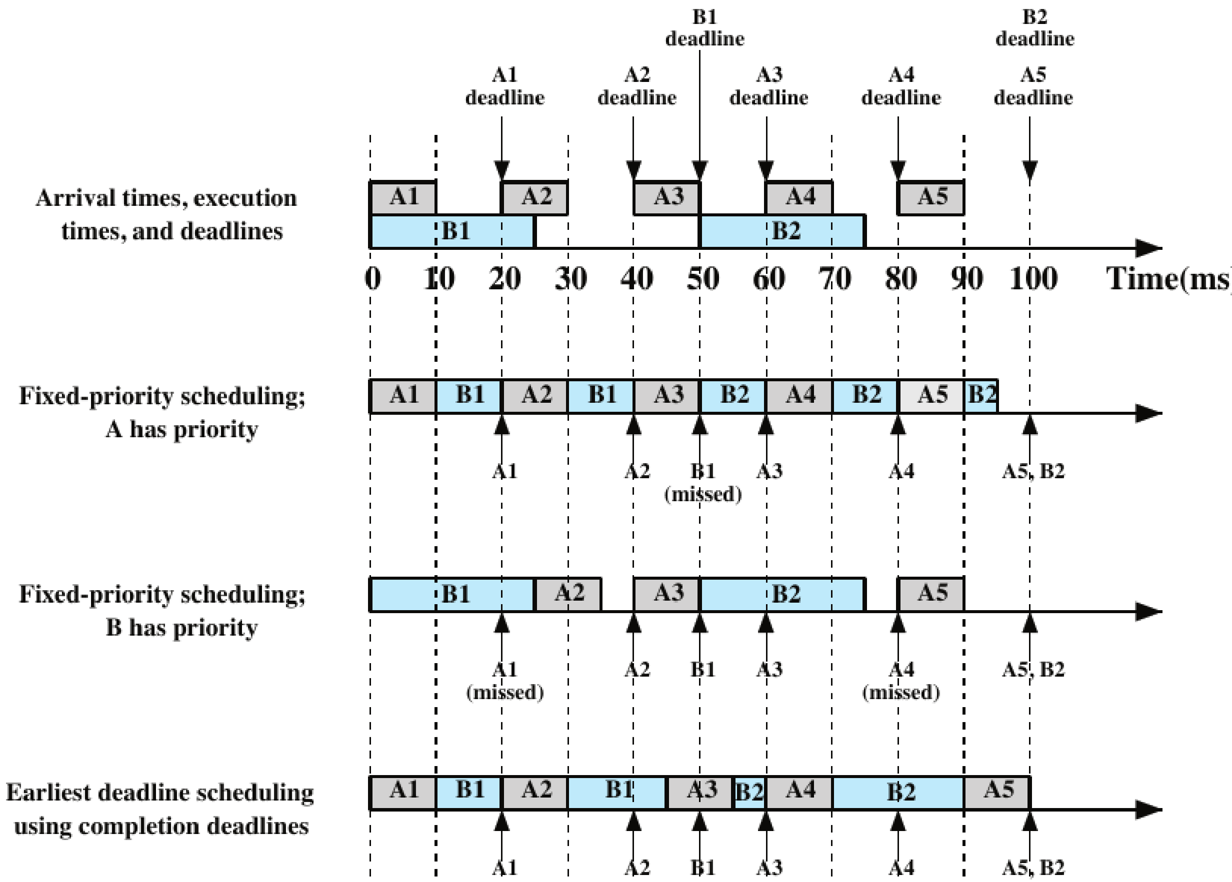
At each scheduling points, the task with the earliest deadline is selected to be run next.

- Dynamic, priority-based preemptive scheduling
- Applicable to both periodic and aperiodic tasks
- Scheduling tasks with the earliest deadline minimized the fraction of tasks that miss their deadlines

Execution Profile of Two Periodic Tasks

Process	Arrival Time	Execution Time	Ending Deadline
A(1)	0	10	20
A(2)	20	10	40
A(3)	40	10	60
A(4)	60	10	80
A(5)	80	10	100
• • •	• • •	• • •	• • •
B(1)	0	25	50
B(2)	50	25	100
• • •	• • •	• • •	• • •

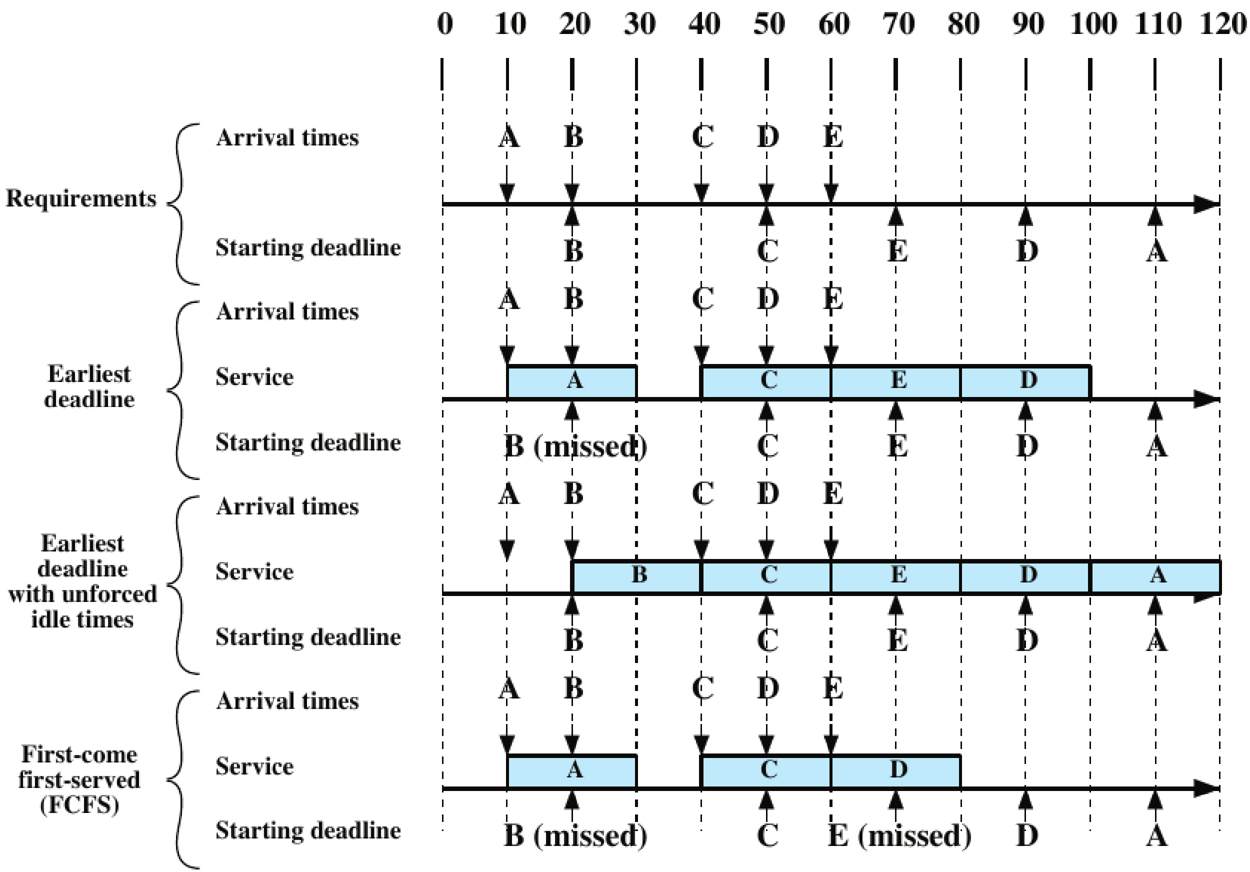
Scheduling of Periodic Real-time Tasks with Completion Deadlines



Execution Profile of Five Aperiodic 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

Scheduling of Aperiodic Real-time Tasks with Starting Deadlines



Scheduling parameters in RTOS

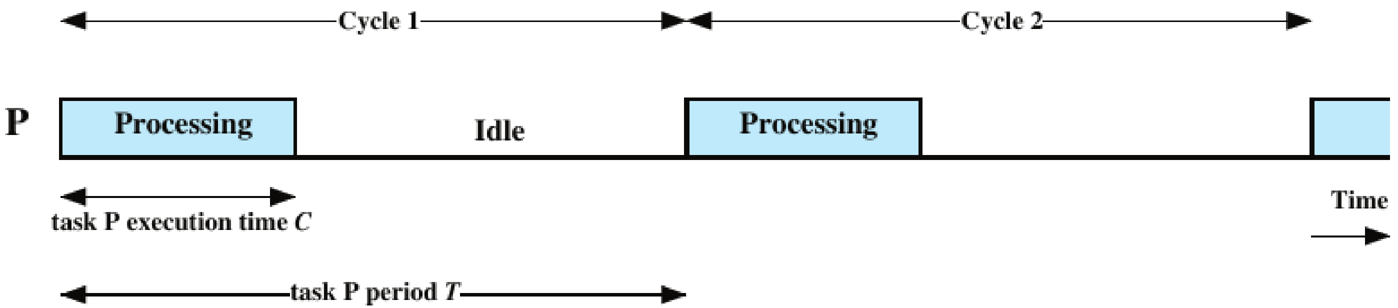
We have to know the basic scheduling parameters. A task i is characterized by:

- $\boldsymbol{c_i}$: computation time
- $\boldsymbol{s_i}$: start time
- $\boldsymbol{d_i}$: deadline (relative to start time)
- $\boldsymbol{p_i}$: period or minimum separation, i.e., Periodic vs. Aperiodic task
- **Laxity**: $l_i = d_i - c_i$ amount of time margin (*Laxity*) before Task must begin execution
- **Utilization factor**: $U = \sum_{i=1}^{n} \frac{c_i}{p_i}$, where n = number of tasks.
- **Schedulability Test**: $U \leq n(2^{1/n} - 1)$, also called Feasibility test, which means meeting timing constrains and resource requirements.

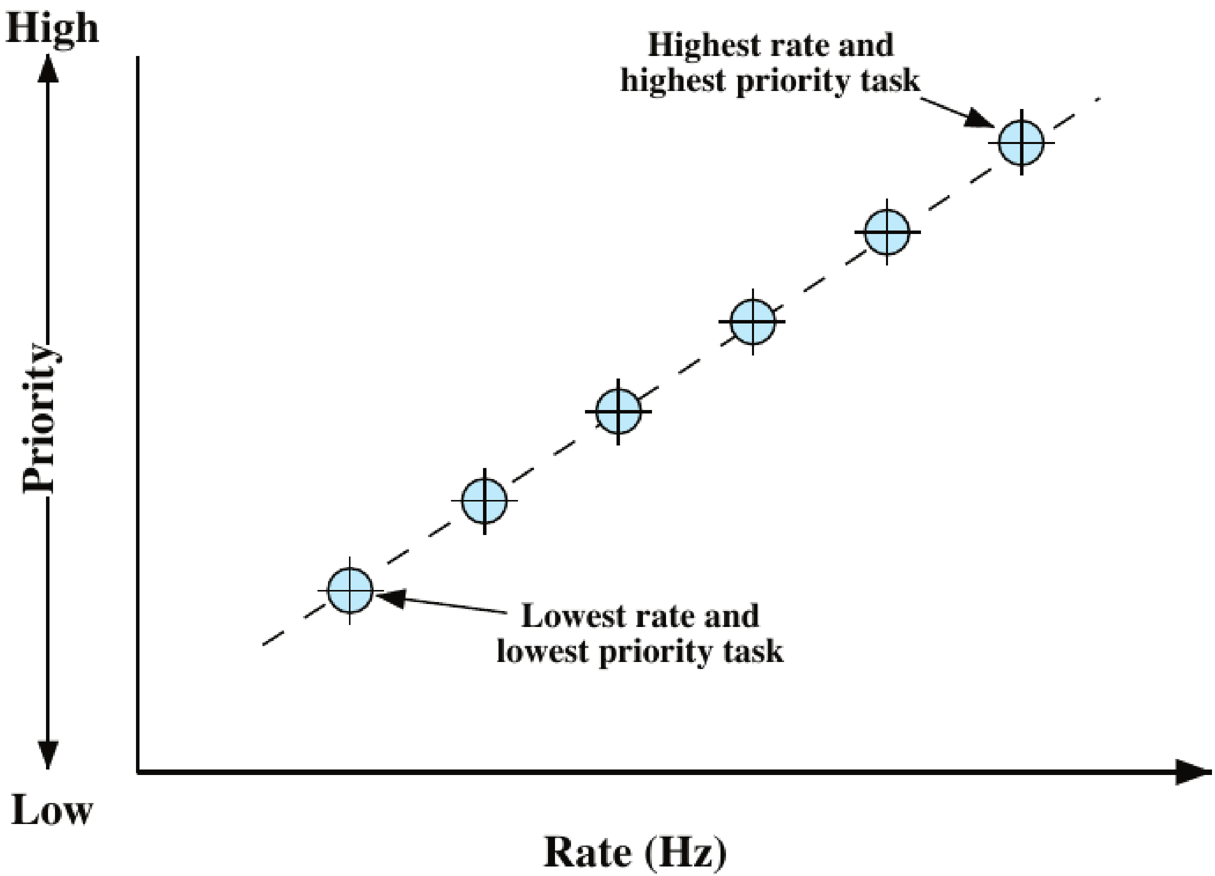
Rate Monotonic Scheduling (RM)

- Static fixed priority scheduler based on Task Periods
  - Assigns priorities to tasks on the basis of their periods
  - Highest-priority task is the one with the shortest period
- Immediately pre-empts any running task with a higher priority task
- Negligible context-switching time
- Periodic tasks
- No precedence constraints

Periodic Task Timing Diagram

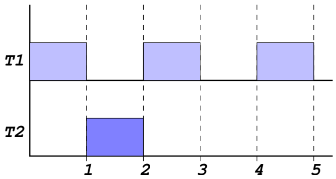


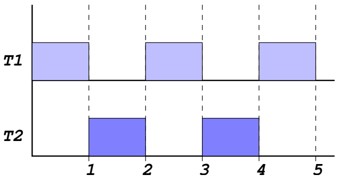
A Task Set with RMS



Value of the RMS Upper Bound

n	$n(2^{1/n} - 1)$
1	1.0
2	0.828
3	0.779
4	0.756
5	0.743
6	0.734
•	•
•	•
•	•
∞	$\ln 2 \approx 0.693$





Example of Rate Monotonic scheduling feasibility. Two tasks T1 and T2: T1 has higher priority; $p_1 = 2$, $p_2 = 5$, $c_1 = 1$, $c_2 = 1$. We can increase $c_2\ to\ 2$ and still be able to schedule, RHS figure.

Advantages and Disadvantages of RM scheduling

- Easy to implement and most widely used
- Low system overhead
- Optimal among other static priorities algorithms
- Requires static prioritization before run-time, which may not be proper for a particular embedding system
- Static prioritization itself can be difficult since it is not certain what task may be more critical at a given time

The Least Upper Bound of U: for all task sets whose U is below this bound, there exists a fixed priority assignment which is feasible.

$$LEB(U) = n(2^{1/n} - 1),\ where\ n\ is\ number\ of\ periodic\ tasks$$

Then

$$c_1/p_1 + c_2/p_2 + ...c_n/p_n \leq n(2^{1/n} - 1)$$

Priority Inversion

- Can occur in any priority-based preemptive scheduling scheme
- Occurs when circumstances within the system force a higher priority task P1 to wait for a lower priority task P2, e.g. P2 locks a resource, P1 tries to lock it and gets blocked.
- If P2 finishes with the resource soon, it will release the lock and then P1 can be scheduled, hopefully quick enough not to violate time limit.

Unbounded Priority Inversion

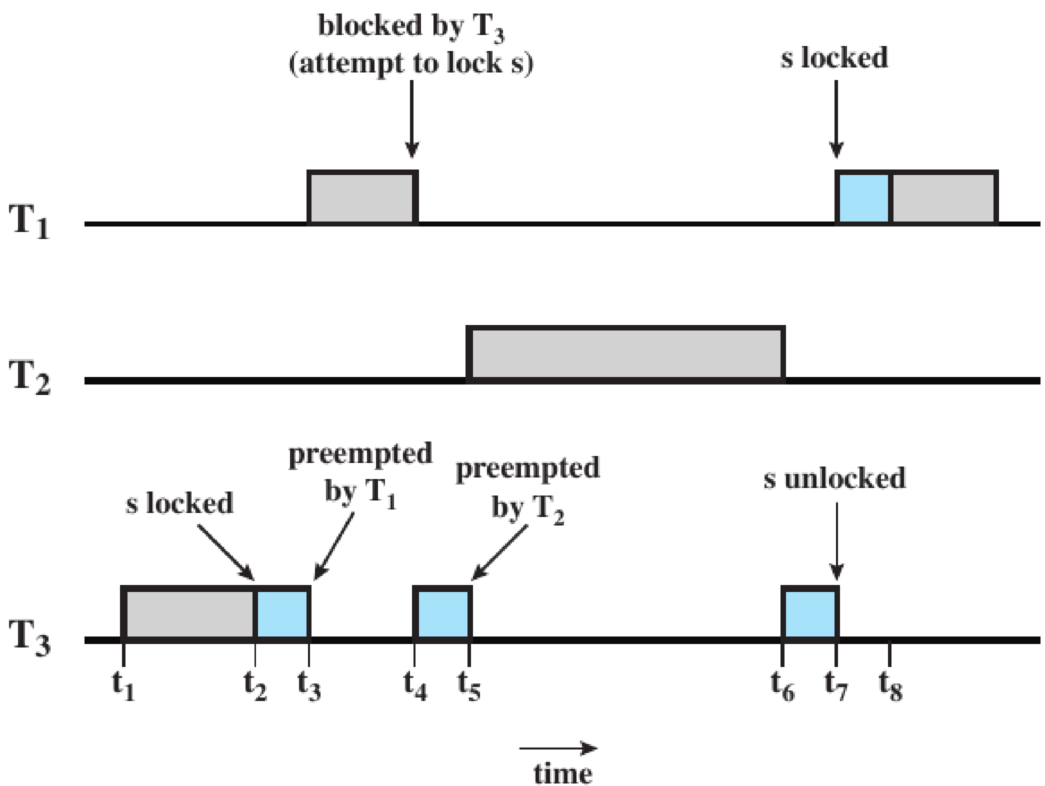
Duration of a priority inversion depends on unpredictable actions of other unrelated tasks. For example, rover Mars robot, tasks in priority order:

1. **T1**: Periodically checks the health of spacecraft and software; it initializes a Watch-Dog timer - if it runs down, complete system reset and reloading of software, testing and reboot is done (24 hrs job)

2. **T2**: Processes image data

3. **T3**: Performs occasional test on the equipment status; shares a data structure, protected by a binary semaphore *s*.

Unbounded priority inversion



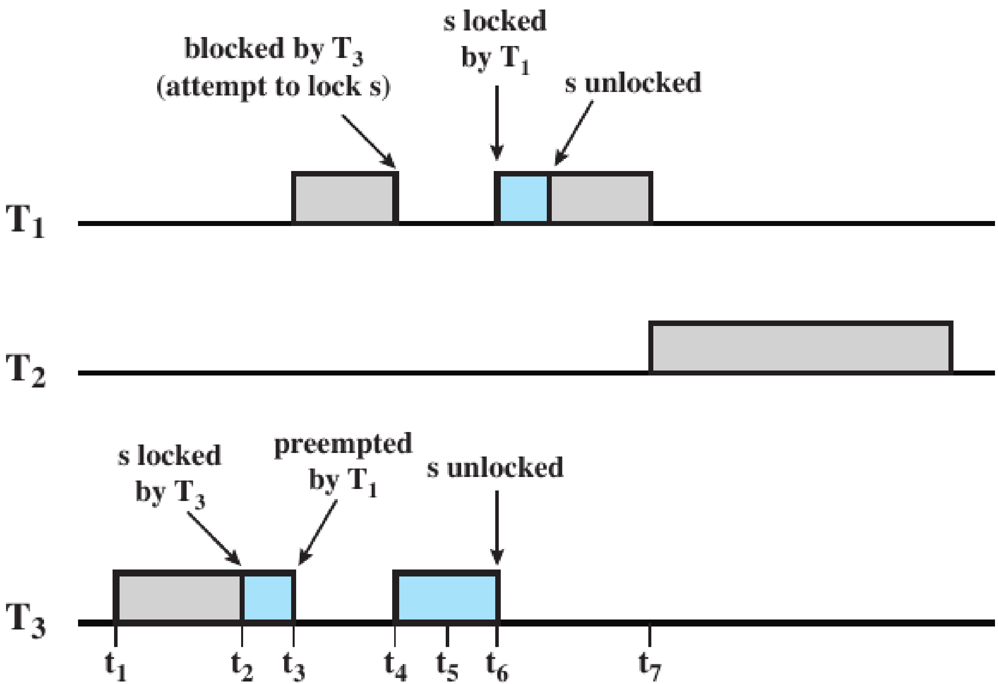
(a) Unbounded priority inversion

Priority Inheritance

A possible approach is:
Priority Inheritance:

- Lower-priority task inherits the priority of any higher priority task pending on a resource they share
- This priority change occurs as soon as the higher priority task blocks on the resource
- The priority is restored when the lower priority task releases the resource

Use of priority inheritance



(b) Use of priority inheritance