EE5903 RTS
Chapter 6
Real-Time Process Synchronization

Bharadwaj Veeravalli

elebv@nus.edu.sg

Acknowledgments: Parts of the material are from the book, OS Concepts by Silberschatz, Galvin and Gagne, 9th Edition

RTS Resource Sharing Protocols

- In traditional OS, one way of handling CS execution is via Semaphores;

- This may not suit well in the case of RTS!

Primarily this is due to two distinct fundamental issues in RTS way of dealing scheduling. These are:

- Priority Inversion
- Unbounded Priority Inversion

RTS Resource sharing – Priority Inversion

- When a task executes in CS, it should not be preempted from using its resource(s) (*not the CPU preemption! Preemption is only for CPU.*)
- This means a low priority task may proceed with computations while a higher priority task need to wait;
- Consequently, a higher priority task may miss its deadline! This is violating our first principles of scheduling in RTS!

RTS Resource sharing – Priority Inversion

- Let us suppose a resource needs to be shared by a task in an *exclusive mode*;
- This means no other task can ask for this resource;
- Thus if a low priority task is in its CS execution and owns a resource under exclusive mode, then any higher priority task that needs the same resource needs to wait/blocked!

We observe that a priority inversion happens here!

RTS Resource sharing – Unbounded Priority Inversion

- Consider the same situation as earlier - a low priority task (T_L) is in its CS execution now and owns a resource under exclusive mode, then any higher priority task (T_H) that needs the same resource is in a blocked state for that resource.

- **Question**: What if some other set of tasks {T_i, i=1,2,…k}, that do not need the resources used by T_L, have priorities such that *pri*(T_L) < *pri*(T_i) < *pri*(T_H), for all i, arrive for processing?

In this case, T_i will all preempt T_L first for processing and hence, T_H has to wait for all T_i and T_L to finish before it can start processing!

Main issue here is we do not know how many such intermediate priority tasks would arrive! – referred to as *Unbounded priority inversion!*

RTS Resource sharing – Unbounded Priority Inversion

- Mars pathfinder (July 1997) – highly cited and celebrated example for this case!
- *What happened?* Mars Pathfinder (MP) had a *Sojourner Rover* which when started roaming on the Martian surface started to do its work – sending pictures back to earth;
- Very soon started experiencing "*total system resets*"! (initial investigation reported superficially – S/w bugs, unable to receive commands, etc) while the main problem is at a real-time kernel!

RTS Resource sharing – Unbounded Priority Inversion

- The RTS kernel used in MP is *VxWorks* (supplied by Wind River Systems Ltd.,) in which RMA scheduling of threads were undertaken;
- In the MP we have an information bus linking to a shared memory which is used for passing and sharing values among different components in the spacecraft;
- *How did they resolve?*

VxWorks allows trace mode simulations to carry out on a replica vehicle back on earth. This means lots of trace simulations can be conducted for precise conditions under which these resets can be triggered!

RTS Resource sharing – Solution to Priority Inversion

- Finally they could find:

VxWorks has a mutex object which accepts a boolean variable to indicate if Priority Inheritance Protocol (PIP) should be used or not! This variable was initialized to OFF, which caused the main problem.

This switching OFF of the mutex variable also resulted very long duration waiting times for high priority tasks which eventually missed their deadlines - *an unbounded priority inversion scenario!*

Note: Th<u>e longest duration for which a simple priority inversion can occur is bounded by the duration for which a lower priority task uses a shared resource it is holding in the exclusive mode</u>!

Solution to Unbounded Priority Inversion – Priority Inheritance Protocol

- A simple priority inversion can be tolerated by –

Limiting the time for which a task executes its CS and this can be achieved in programming; Priorities can be considered explicitly in the programming to limit the access to resources for a limited time;

Thus, handling simple PI problem is not a critical problem.

- Challenge lies in handling Unbounded PI problem!

Solution to Unbounded Priority Inversion – Priority Inheritance Protocol

- Recall the unbounded priority inversion scenario discussed earlier;

  Result – In the worst case scenario a high priority task can miss its deadline!

- A simple solution is proposed to handle this situation is as follows: Priority Inheritance Protocol

- We know that a task that is in its CS cannot be preempted for a resource when it is using a resource in exclusive mode.

Solution to Unbounded Priority Inversion – Priority Inheritance Protocol

- Main Idea: Allow the task to complete as early as possible!

How do we do this?

-- Raise its priority, so that the low priority tasks are not able to preempt it and hence it will finish as early as possible!!

- So no intermediate tasks can preempt! *But by how much?*
- [Solution: Sha & Rajkumar(199X)] Raise it to the level at least to the highest priority level of the task that is currently waiting for the resource!

Solution to Unbounded Priority Inversion – Priority Inheritance Protocol

- Thus this task "inherits" the priority of the highest task (temporarily). Once a task releases the resource, it gets back its original priority, if it is not holding any other critical resources needed by another process.

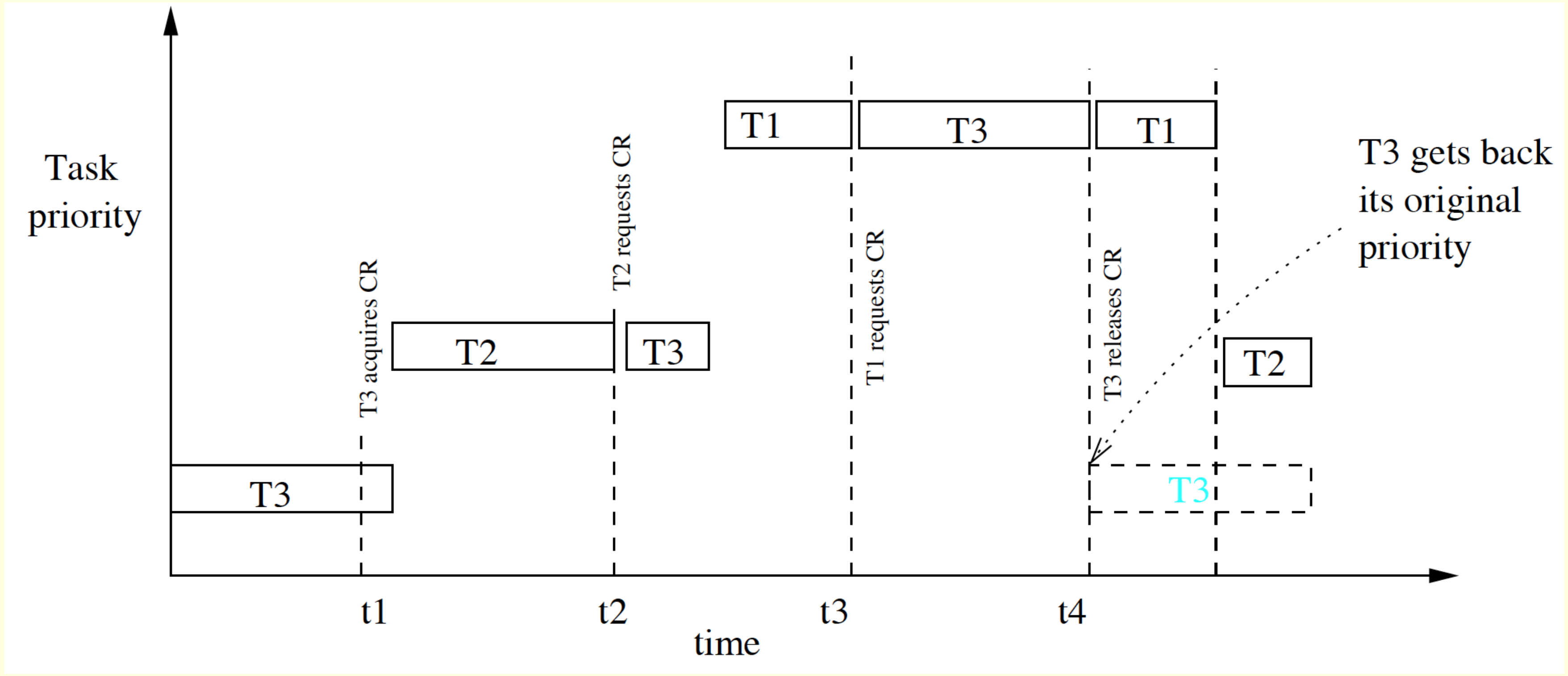
Pseudocode: T: current task

```
if (Required_resource == 1): //   1: Free;      0: not available
    Assign_resource(T);
if (Required_resource is held by a Hi-pri task):      // Res not available & It is
                                                      held by a higher priority task
    wait_for_resource(); // Not a PIP situation either;
if (Required_resource is held by a Low-pri task):     // Res not available & it is
                                                      held by a lower priority task
    wait_for_resource();
    pri(Tl) = max{pri(Tk), k=1,2,,,n}               // Pri inheritance
// where max{} is done over all higher priority tasks waiting for the resource.
```

Note: Depending on the OS, small numerical values may imply higher priority and vice versa!

Solution to Unbounded Priority Inversion – Priority Inheritance Protocol

Example



Priority Inversions under Priority Inheritance Protocol (PIP)

Note that, under PIP, a lowest priority task retains its inherited priority until it holds a resource needed for a currently waiting higher priority task.

Solution to Unbounded Priority Inversion – Priority Inheritance Protocol

- **Q 6.9:** Using semaphores, what is the max duration for which a task may undergo priority inversion?

(a) Longest duration for which a Hi-pri task uses a shared resource

(b) Longest duration for which a Low-pri task uses a shared resource

(c) Sum of the durations for which different lower-pri tasks may use the shared resource

(d) Greater than the longest duration for which the low-pri task uses a shared resource

Drawbacks in Priority Inheritance Protocol

Thus it is clear that PIP can let RT tasks share critical resources without letting them incur unbounded priority inversions which costs delays!

However, two main drawbacks exist:

(i) Deadlocks – Two or more tasks compete for shared resources and arrives at a *deadlock*;

(ii) Chain blocking – Every time a task needs a resource it is blocked and waits; In addition, every time when it needs a resource held by a low-priority task, the priority of low-priority task will undergo priority inversion.

Drawbacks in Priority Inheritance Protocol

Question 6.10(Deadlock): Consider T1, T2, both needing two resources R1 and R2. Let Pri(T1) > Pri(T2). Let T2 start first in the following sequence of events. Demonstrate and realize a deadlock triggered using PIP.

T1: Lock R1, Lock R2

T2: Lock R2, Lock R1

Question 6.11 (Chain blocking): Assume T2 needs several resources, say R1 and R2, and starts first. Consider a task T1 which also arrives for processing and looking for resource R1 first an then after sometime it requests for R2. Let Pri(T1) > Pri(T2). Demonstrate a chain blocking triggered using PIP.

Thank you!