Resource Management in Real-Time Systems

Michal Sojka

Czech Technical University in Prague,
Faculty of Electrical Engineering,
Department of Control Engineering

December 22, 2016

Based on the book and accompanying materials to
Alan Burns and Andy Wellings, Real-Time Systems and Programming Languages, Addison Wesley Longmain, 2001

http://www.cs.york.ac.uk/rts/books/RTSBookThirdEdition.html#Teaching%20Aids

and slides from Giorgio Buttazzo

Outline

1 Introduction

2 Fixed Priority Protocols
- Priority Inheritance Protocol
- Priority Ceiling Protocols

3 Dynamic Priority Protocols
- Stack Resource Policy

Outline

1 Introduction

2 Fixed Priority Protocols
- Priority Inheritance Protocol
- Priority Ceiling Protocols

3 Dynamic Priority Protocols
- Stack Resource Policy

Shared resources, critical sections

Definition (Shared resource)

A resource accessed from multiple threads of execution that must be used in a *mutually exclusive manner*.
Examples: data structure in memory, device, . . .

- Access to shared resources must be "protected" by mutexes, spinlocks, disabling of interrupts/preemption, RCU, etc. $\Rightarrow$ locks.
- Locks can be **nested**.

Definition (Critical section)

Critical section is a segment of a job (piece of code) that begins with lock operation and ends with matching unlock operation.

Problem

Priority Inversion

A high priority task is blocked by a lower priority task for an **unbounded** interval of time.

Deadline Inversion

A task with short deadline is blocked by a task with longer deadline for an **unbounded** interval of time.

Real-Time computing is about determinism and unbounded blocking is not deterministic!

Terminology

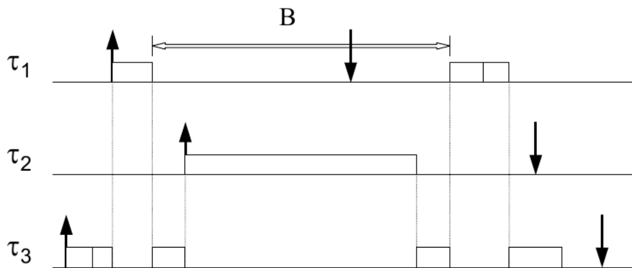
Blocking, preemption

- If a task is waiting for a lower-priority task, it is said to be blocked. Note: VxWorks calls this state "pending".
- If a task is waiting for a higher priority task, it is said to be preempted.

Priority inversion

- If a task is blocked waiting for an **unrelated** lower-priority task to complete some required computation then the priority model is, in some sense, being undermined.
- The blocked task is said to suffer priority inversion.

Conflict in concurrent access to a critical section



Solution

Introduce a **concurrency control protocol** (resource access protocol) to control the access to shared resources.

A resource access protocol, is a set of rules that govern

1. when and under what conditions each request for resource is granted and
2. how jobs requiring resources are scheduled.

Outline

1 Introduction

2 Fixed Priority Protocols
- Priority Inheritance Protocol
- Priority Ceiling Protocols

3 Dynamic Priority Protocols
- Stack Resource Policy

Fixed Priority Protocols

- Non Preemptive Protocol (NPP)
  - Cyclic scheduling, `taskLock()`

Fixed Priority Protocols

- Non Preemptive Protocol (NPP)
  - Cyclic scheduling, `taskLock()`
- Priority Inheritance Protocol (PIP)
  - Mutexes in VxWorks and Linux

Fixed Priority Protocols

- Non Preemptive Protocol (NPP)
  - Cyclic scheduling, `taskLock()`
- Priority Inheritance Protocol (PIP)
  - Mutexes in VxWorks and Linux
- Priority Ceiling Protocol (PCP)
- Immediate Priority Ceiling Protocol (IPCP)

Fixed Priority Protocols

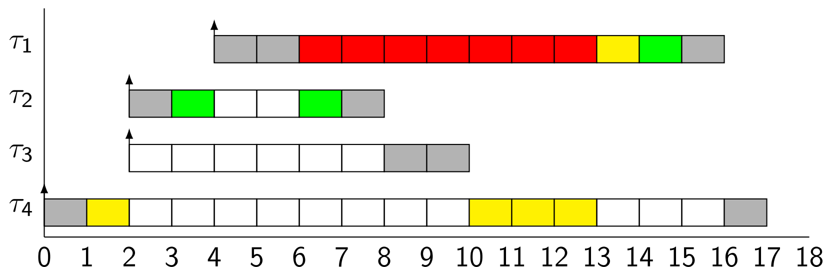
- Non Preemptive Protocol (NPP)
  - Cyclic scheduling, `taskLock()`
- Priority Inheritance Protocol (PIP)
  - Mutexes in VxWorks and Linux
- Priority Ceiling Protocol (PCP)
- Immediate Priority Ceiling Protocol (IPCP)
- Stack Resource Policy (SRP) – also for dynamic priority

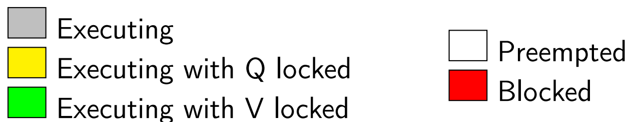
Priority Inversion

- To illustrate an extreme example of priority inversion, consider the executions of four periodic tasks $\tau_{1,\ldots,4}$ and two resources: Q and V

Task	Priority	Execution Sequence	Release Time
τ_1	4	E E Q V E	4
τ_2	3	E V V E	2
τ_3	2	E E	2
τ_4	1	E Q Q Q Q E	0

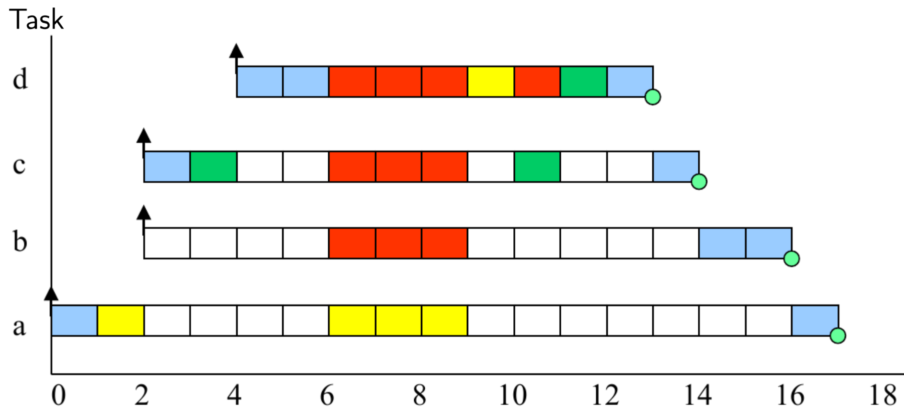
Example of Priority Inversion





Priority Inheritance

- If task p is blocking task q, then p runs with q's priority



Calculating Blocking

- If a task has m critical sections that can lead to it being blocked then the maximum number of times it can be blocked is m
- The task i has an upper bound on its blocking B_i given by:

$$B_i = \sum_{k=1}^{K} \text{canblock}(k, i) C(k),$$

where

- K is the number of critical sections in the system,
- $\text{canblock}(k, i)$ is 1 if task i can suffer blocking from critical section k, i.e. if k is executed by a task with lower priority than i, and 0 otherwise,
- $C(k)$ is the worst-case execution time of critical section k.

Blocking Term (cont.)

$$B_i = \sum_{k=1}^{K} \text{canblock}(k, i) C(k),$$

- In fact, the above formula for B_i is too pessimistic. The total number of terms in the sum is at most $v \times l$, where v is the number of resources accessed by task i and l is the number of lower priority tasks that can conflict with task i.
- Don't forget that nested blocking can create a blocking chain.

Schedulability Analysis and Blocking

Response-Time Analysis

$$\underset{i=1,\ldots,n}{\forall i} \quad R_i = C_i + B_i + \sum_{j=1}^{i-1} \left\lceil \frac{R_i}{T_j} \right\rceil C_j \leq D_i$$

Utilization-based Analysis

$$\underset{i=1,\ldots,n}{\forall i} \quad \left(\sum_{j=1}^{i} \frac{C_j}{T_j} \right) + \frac{B_i}{T_i} = U_i + \frac{B_i}{T_i} \leq U_{RM}(i)$$

Priority Ceiling Protocols

Two forms

- Original priority ceiling protocol (OPCP)
- Immediate priority ceiling protocol (IPCP)

On a single processor

- A high-priority task can be blocked at most once during its execution by lower-priority tasks
- Deadlocks are prevented
- Transitive blocking is prevented
- Mutual exclusive access to resources is ensured (by the protocol itself)

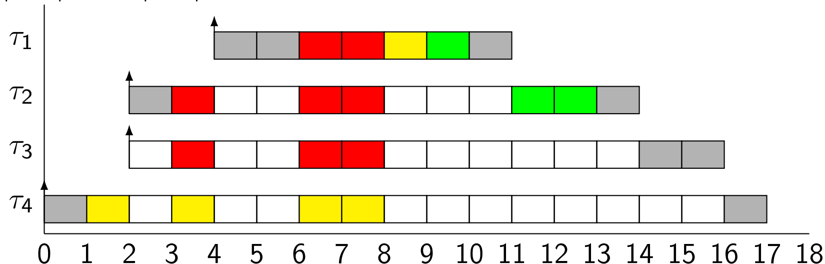
Original priority ceiling protocol (OPCP)

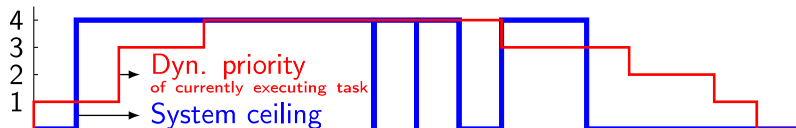
1. Each task has a static default priority assigned (perhaps by the deadline monotonic scheme)
2. A task has a dynamic priority that is the maximum of its own static priority and any it inherits due to it blocking higher-priority tasks.
3. Each resource k has a static ceiling value $\lceil k \rceil$ defined as the maximum priority of the tasks that use it
4. A task can only lock a resource if its dynamic priority is higher than the system ceiling i.e. the ceiling of any currently locked resource (excluding any that it has already locked itself)

Blocking term calculation

$$B_i = \max_{k=1}^{K} \text{usage}(k, i) C(k)$$

OPCP example



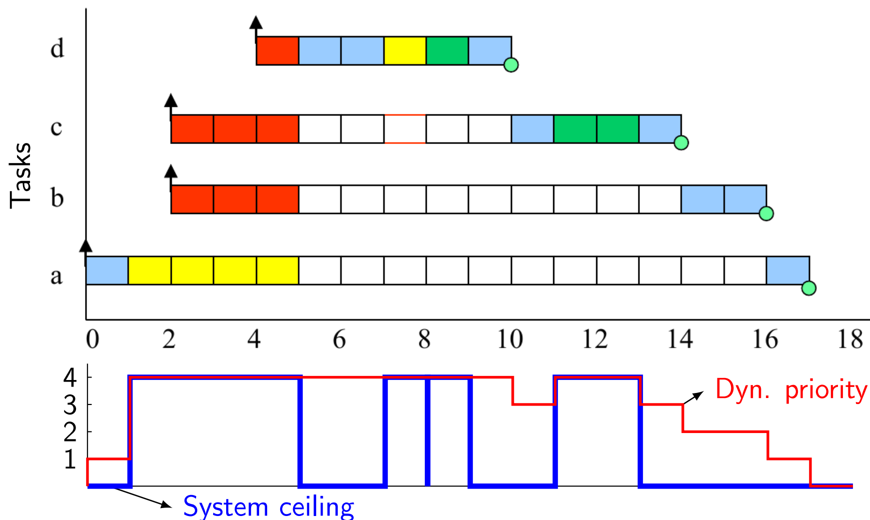


Immediate priority ceiling protocol (IPCP)

- Each task has a static default priority assigned (perhaps by the deadline monotonic scheme).
- Each resource k has a static ceiling value $\lceil k \rceil$ defined as the maximum priority of the tasks that use it.
- A task has a dynamic priority that is the maximum of its own static priority and the ceiling values of any resources it has locked
- As a consequence, a task will only suffer a block at the very beginning of its execution
- Once the task starts actually executing, all the resources it needs must be free; if they were not, then some task would have an equal or higher priority and the task's execution would be postponed

IPCP example

$\lceil Q \rceil = 4, \lceil V \rceil = 4$



OPCP versus IPCP

Although the worst-case behavior of the two ceiling schemes is identical (from the scheduling view point), there are some points of difference:

- IPCP is easier to implement than the OPCP as blocking relationships need not to be monitored
- IPCP leads to less context switches as blocking is prior to first execution
- IPCP requires more priority movements as this happens with all resource usage
- OPCP changes priority only if an actual block has occurred

Note that IPCP is called Priority Protect Protocol in POSIX and Priority Ceiling Emulation in Real-Time Java

Outline

1 Introduction

2 Fixed Priority Protocols
- Priority Inheritance Protocol
- Priority Ceiling Protocols

3 Dynamic Priority Protocols
- Stack Resource Policy

Dynamic Priority Protocols

- Dynamic Priority Inheritance (DPI)
- Dynamic Priority Ceiling (DPC)
- Dynamic Deadline Modification (DDM)
- Stack Resource Policy (SRP)

Stack Resource Policy

[Baker 1990]

- Generalization of PCP
- Works both with fixed and dynamic priorities
- Limits blocking to the beginning of the job
- Prevents deadlock
- Supports multi-unit resources
- Allows stack sharing (stack is considered as a "special resource")
- Is easy to implement

Stack Resource Policy

For each resource k

- Maximum units: N_k
- Available units: n_k

For each task τ_i the system keeps:

- its resource requirements: $\mu_i(R_k) \in \mathbb{Z}$
- a priority p_i: RM: $p_i \propto 1/T_i$ EDF: $p_i \propto 1/d_i$
- a static preemption level: $\pi_i \propto 1/D_i$

Stack Resource Policy

- Resource ceiling:

$$\mathcal{C}_k(n_k) = \max_j \{\pi_j : n_k < \mu_j(R_k)\}$$

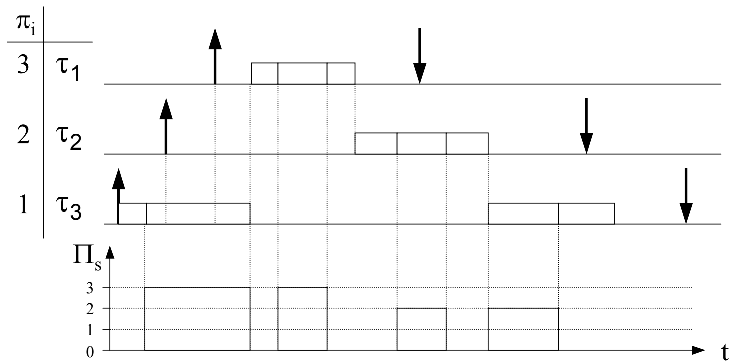
- System ceiling:

$$\Pi_s = \max_k \{\mathcal{C}_k(n_k)\}$$

- SRP Rule:

A job cannot preempt until p_i is the highest and $\pi_i > \Pi_s$.

Example

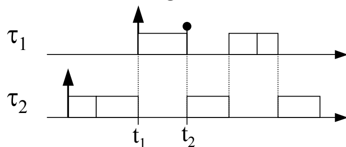


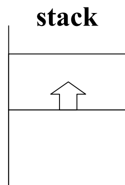
SRP: Notes

- Blocking always occurs at preemption time
- A task never blocks on a wait/lock primitive
  (semaphore queues are not needed)
- Semaphores are still needed to update the system ceiling
- Early blocking allows stack sharing

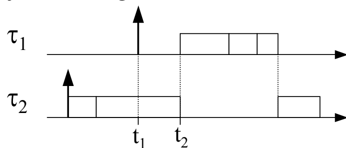
SRP: Stack sharing

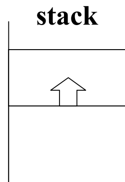
Classical blocking





Early blocking





Stack sharing

- If tasks can be grouped in M subsets with the same preemption level, then tasks within a group cannot preempt each other.
- Then the stack size is the sum of the stack memory needed by M tasks.

Example

If we have 100 tasks with 10 preemption levels and each task requires 10 kB of stack, then

$$\text{Stack size} = \begin{cases} \text{1 MB without SRP} \\ \text{100 KB under SRP} \end{cases}$$

Schedulability of EDF with SRP

- n tasks
- Tasks τ_i ordered by increasing relative deadlines D_i
- B_i is the execution time of the longest critical section of any task τ_k such that $D_i \leq D_k$ or zero it there is no such τ_k

$$\underset{k=1,\ldots,n}{\forall k} \left(\sum_{i=1}^{k} \frac{C_i}{D_i} \right) + \frac{B_k}{D_k} \leq 1$$