Multiprocessor resource sharing Protocol

Implementation and evaluation

Sebastiano Catellani

University of Padua
Supervisor: Prof. Tullio Vardanega

9 October 2014

Overview

Workload

- $T = \{\tau_1, \ldots, \tau_n\}$
- $\tau_i = (C_i, P_i, D_i)$

A task τ_i is characterized by a utilization factor $U_i = C_i / P_i$

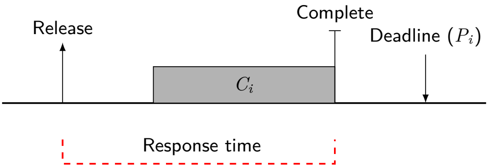


Figure: Task τ_i releases a job J_i in response to a triggering event

Resources

- **Processors**, necessary to execute
  - Assigned by a scheduling algorithm
- **Memory, lock, I/O, . . .** , provide functionality
  - Need mutual exclusion access → resource acccess protocol
  - Can be *local* or *global*

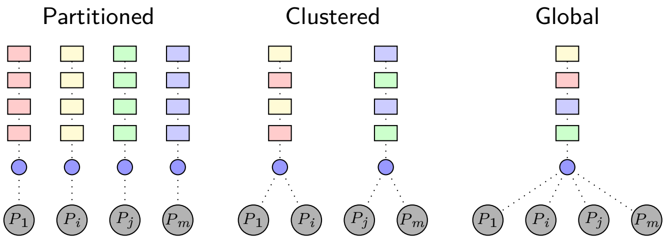
Scheduling algorithm

It determines which task has to be executed on each processor at any time

MrsP

Sebastiano Catellani

By the **number of ready tasks queues**:



and by the **priority assignment function**:

- *Fixed-Priority*
- *Job-Level Fixed-Priority*
- *Job-Level Dynamic-Priority*

Real-Time System Model
Platform

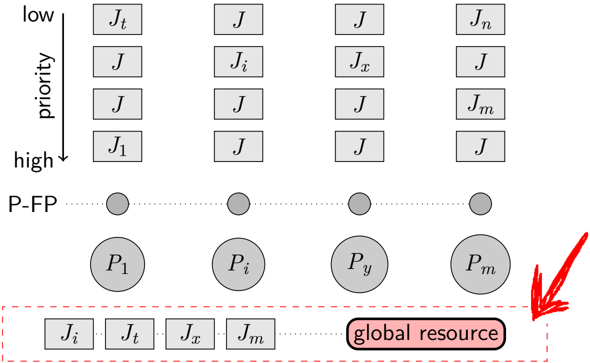
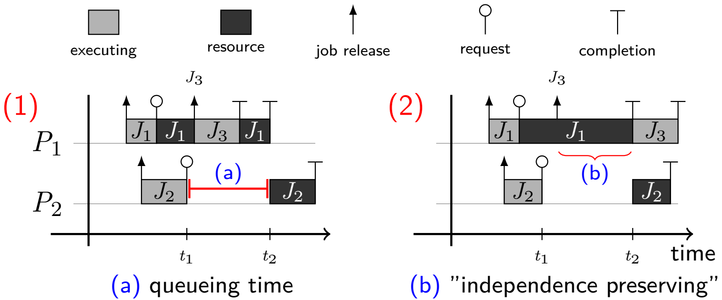


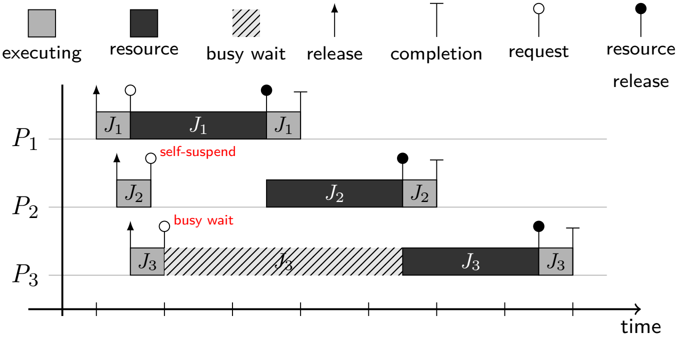
Figure: Partitioned Fixed-Priority scheduler on a platform with m processors $(P_1, \ldots, P_m)$ and a global resource



- 2 processors and 3 tasks, $prio(J_i) > prio(J_y) \iff i > y$
- J_1 and J_2 share a global resource
- (1) J_1 remains preemptable
- (2) J_1 inhibits preemption

Suspension-based or *spin-based* protocol

A job, attempting to access a busy resource, will self-suspend (J_2) or will perform busy-wait (J_3)



Burns and Wellings [4] design a multiprocessor extension of PCP/SRP [1] with the aim of adapt a schedulability analysis to the protocol

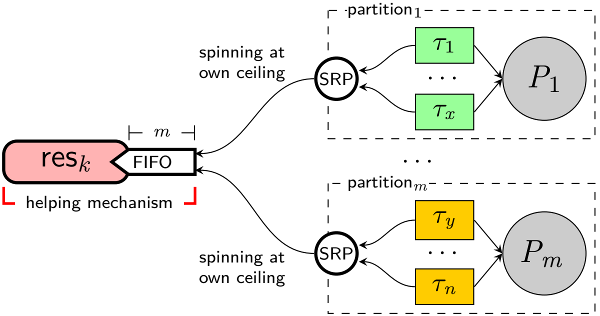
Response Time Analysis incorporating PCP/SRP

The parameter e_j reflects the contention for the resource (r):

$$e_j = |map(G(r))| \times c_j$$

$$R_i = C_i + max\{e_j, \hat{b}\} + \sum_{\tau_j \in hp(i)} \lceil \frac{R_i}{T_j} \rceil C_j$$

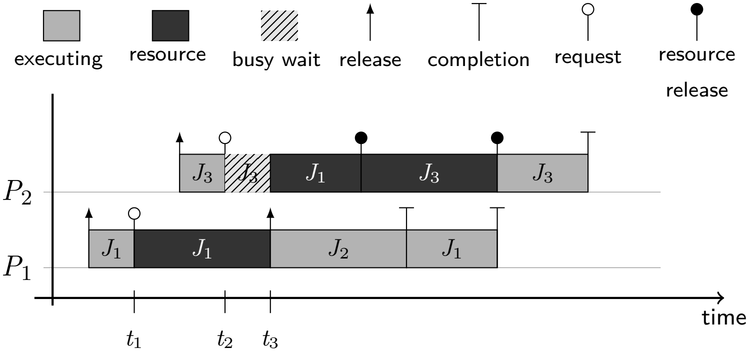
$$C_i = WCET_i + \sum_{r^j \in F(\tau_i)} n_i e_j$$



Protocol's properties

- It inherits the properties of PCP/SRP
- At most one job per processor requires the resource
- The length of the requests queue is at most $|map(G(r_j))|$
- At most e_j to gain the resource and to execute the critical section

MrsP
Runtime example



- t_1: J_1's priority is raised: it gains access to r
- t_2: J_3's priority is raised: it starts spinning
- t_3: J_2 is released and J_3 "helps" J_1

Proposed solution

Algorithm

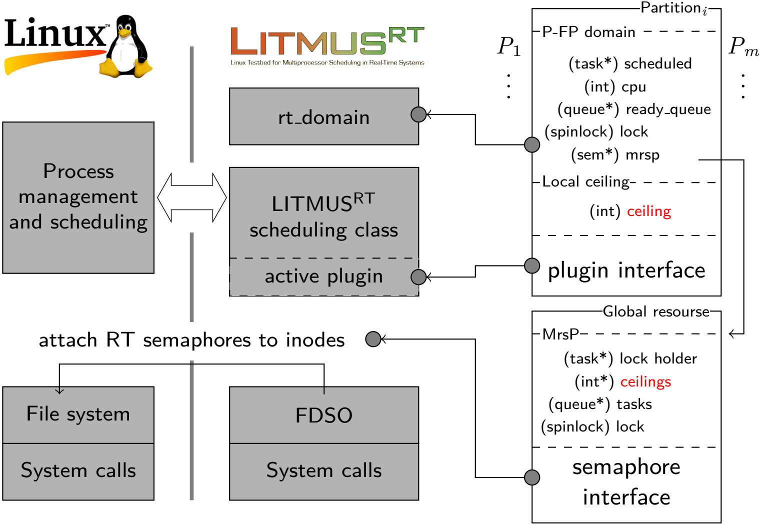
1) Each resource has a set of ceilings, one for each processor
2) An access request causes the rise of the job's priority and activates a local ceiling
3) The requests are queued and served in arrival order
4) A job executes, until resource's release, at the inherited priority
5) If preempted, the lock holder migrates to the first processor available

Key features

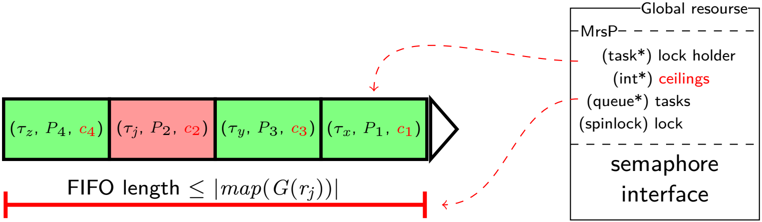
- Points 2 and 4 make MrsP independence-preserving
- Point 5 guarantees a limited waiting and blocking time

Implementation

Data structures



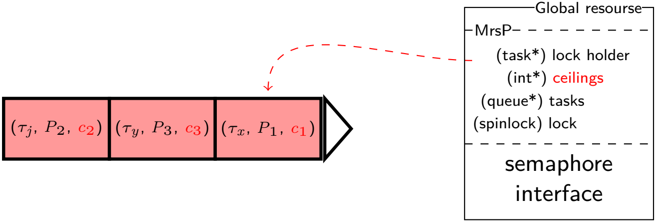
Focused on managing the access requests queue



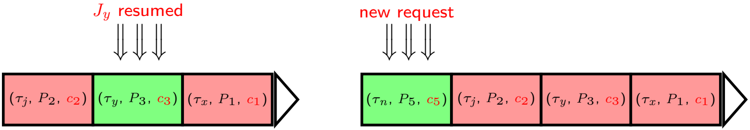
If preempted, the lock holder (J_x)

1. inherits the ceiling $(c_3 + 1)$
2. migrates to P_3
3. preempts J_y

The job will be re-queued in the *ready_queue*

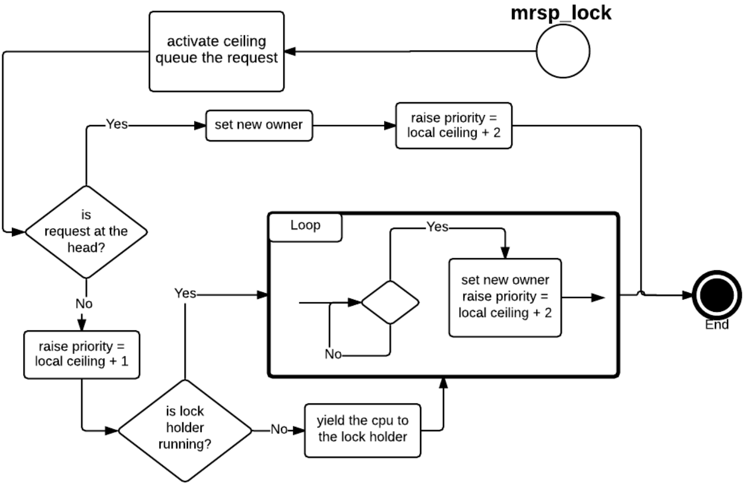


The algorithm catches the operations that

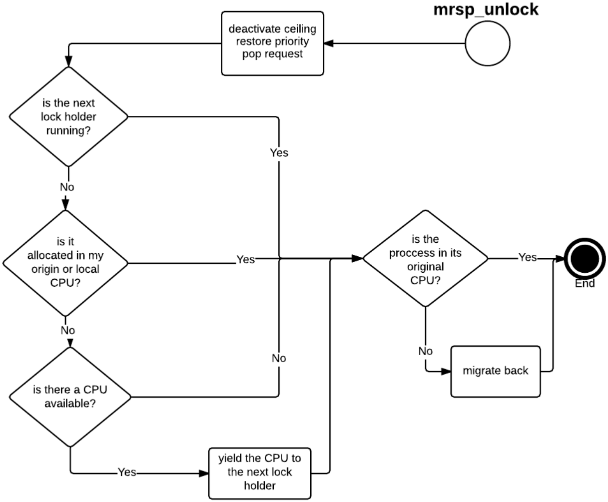


(a) a processor becomes available

(b) add a new request to the queue

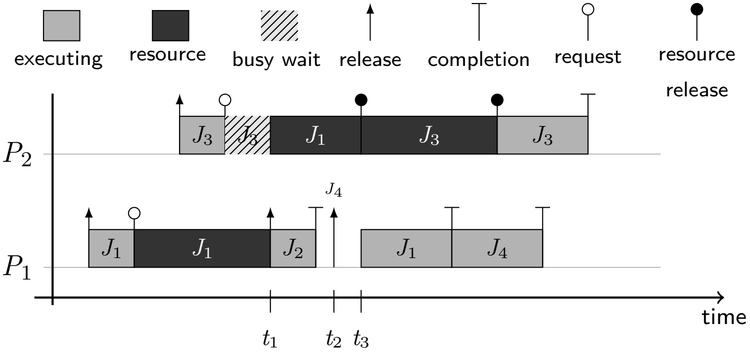

mrsp_lock
activate ceiling
queue the request
Yes
set new owner
raise priority =
local ceiling + 2
is
request at the
head?
No
raise priority =
local ceiling + 1
Yes
Loop
Yes
set new owner
raise priority =
local ceiling + 2
No
End
is lock
holder
running?
No
yield the cpu to
the lock holder

Implementation
Primitive: mrsp_unlock



Implementation

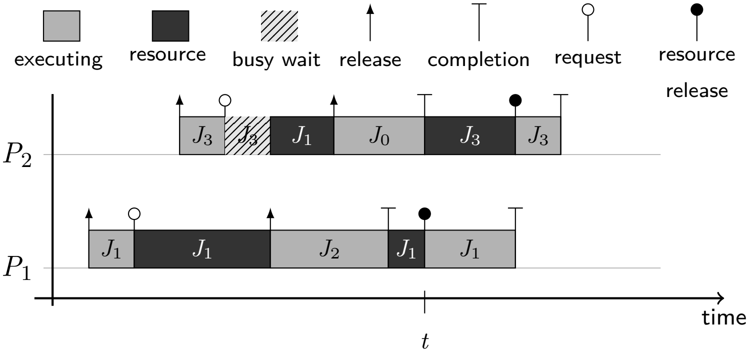
Primitive: pfp_schedule and finish_switch - 1



- t_1: J_1 is marked for migration
- t_2: J_4's priority is lower than the local ceiling
- t_3: default migration mechanism

Implementation

Primitive: pfp_schedule and finish_switch - 2



- t: J_2 completes and P_1 returns available

Experiments

Overview

Experiment #1: Comparison among protocols

MrsP outperfmors protocols based on simple ceiling or non preemption

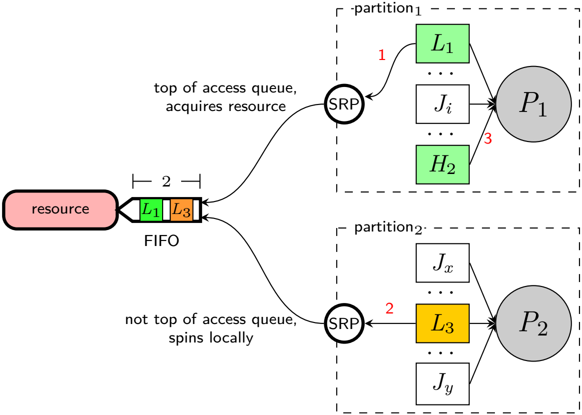
Experiment #2: Sampling of the overheads

MrsP brings benefits at reasonable costs

Experiment #3: Absence of global resources

The protocol doesn't interfere with the scheduler

The experiment observes the response times of L_1, H_2 and L_3 while varying the critical section length and the WCET of H_2



Experiments

Comparison among protocols - 2

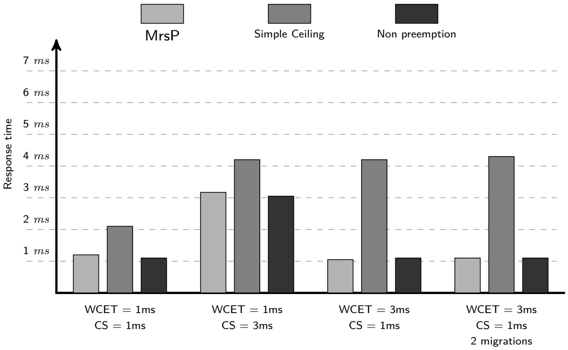


Figure: Response time of L_1

Experiments

Comparison among protocols - 3

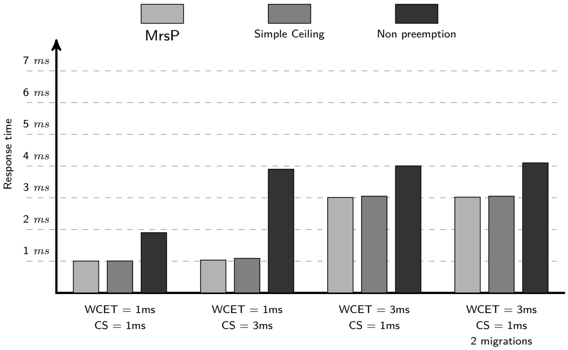


Figure: Response time of H_2

Experiments
Comparison among protocols - 4

Sebastiano Catellani

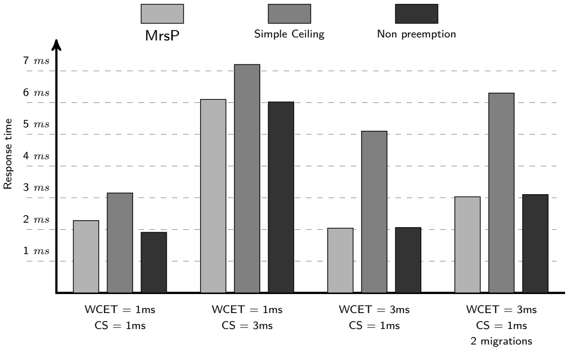
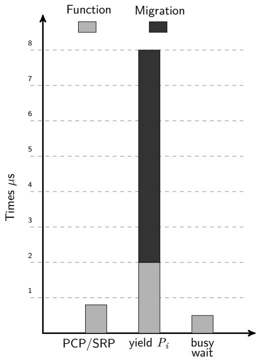


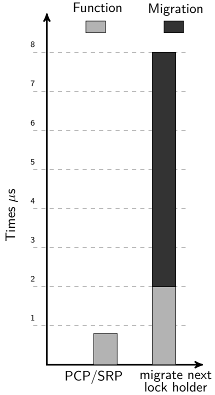
Figure: Response time of L_3

Experiments

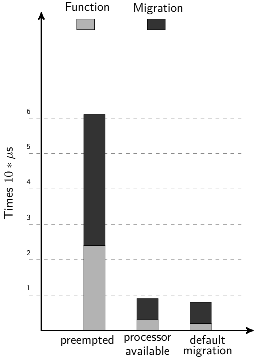
Sampling of the overheads



(a) mrsp_lock



(b) mrsp_unlock



(c) finish_switch

The collected data show the same number of deadline miss

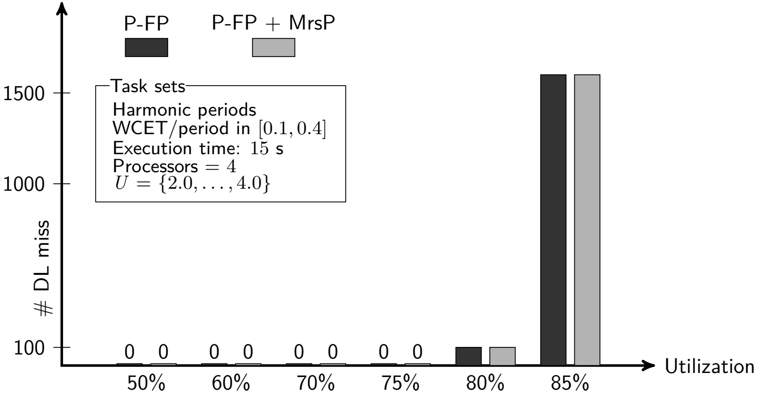
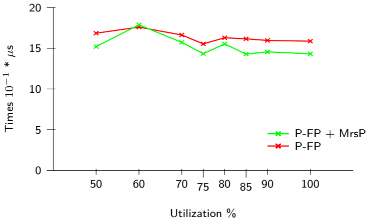
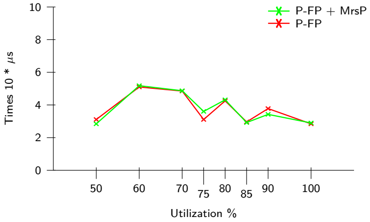


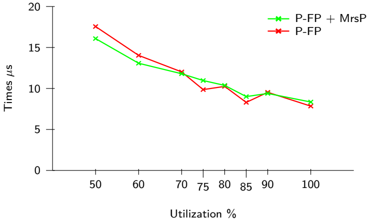
Figure: Number of *deadline miss*



(a) pfp_schedule: Min



(b) pfp_schedule: Max



(c) pfp_schedule: Average

The experiments underline how the system suffers in case of migration

A possible solution

1. Divide the resources into "long" and "short" (FMLP, [2])
2. Migration should only be caused when its benefits exceeds its cost

Supporting nested resources

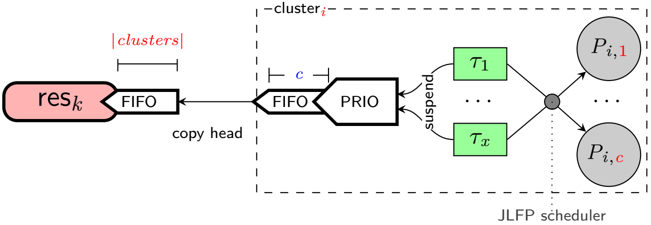
- Allow a nested request from r_i to r_j only if $i < j$
- Groups lock (FMLP)
- A k-lock system (RNLP, [5])

Conclusion

Future work - 2

Compare MrsP and the *O(m) Independence-preserving Protocol*(OMIP, [3])

- Independence preserving and limited waiting time
- Allows migrations
- Design for cluster scheduler
- Suspension-based



References

T. P. Baker.
Stack-based scheduling for realtime processes.
Real-Time Syst., 3(1):67–99, April 1991.

Aaron Block, Hennadiy Leontyev, Bjorn B. Brandenburg, and James H. Anderson.
A flexible real-time locking protocol for multiprocessors.
In *Proceedings of the 13th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, RTCSA '07, pages 47–56, Washington, DC, USA, 2007. IEEE Computer Society.

B.B. Brandenburg.
A fully preemptive multiprocessor semaphore protocol for latency-sensitive real-time applications.
In *Real-Time Systems (ECRTS), 2013 25th Euromicro Conference on*, pages 292–302, July 2013.

A. Burns and A. J. Wellings.
A schedulability compatible multiprocessor resource sharing protocol – mrsp.
In *Proceedings of the 2013 25th Euromicro Conference on Real-Time Systems*, ECRTS '13, pages 282–291, Washington, DC, USA, 2013. IEEE Computer Society.

Bryan C. Ward and James H. Anderson.
Supporting nested locking in multiprocessor real-time systems.
In *ECRTS*, pages 223–232, 2012.