A study and comparison of Priority Inheritance Protocol and Priority Ceiling Protocol

Vishal Prajapati
BVM Engineering College,
Anand, Gujarat, India.
vsprajapati@gmail.com

Apurva Shah
G H Patel College of Engineering and Technology,
Anand, Gujarat, India.
apurvashah@gcet.ac.in

Abstract - **Resource allocation is one of the challenging problems for real-time operating system. Priority inheritance protocol (PIP) and Priority ceiling protocol (PCP) are very popular for resource allocation in real-time operating system. Both algorithms have certain pros and cons.**

In priority inheritance protocol when any higher priority job is scheduled, it may ask for resource. If that resource is acquired by lower priority job than lower priority job inherits the priority of currently scheduled job and it will be executed.

In priority ceiling protocol at starting of the scheduling the resources are allocated to the highest priority job. When lower priority job request for resource than it will not be allocated to that job even though the resource is free.

Therefore, there are advantages and disadvantages of both the protocols. We have studied and compared both protocols in this paper.

Keywords–**Priority inheritance protocol, Priority Ceiling Protocol, Resource Allocation, Real-time operating system.**

I. INTRODUCTION

Real-time system is required to complete its work and deliver its services on the basis of time. The results of real-time systems are judged based on the time at which the results are produced in addition to the logical results of computations [3]. Therefore, real-time systems have well defined, fixed time constraints i.e. processing must be done within the defined constraints otherwise the system will fail. Real-time systems can be categorized in two basic types: Hard and Soft. In hard real-time systems, all jobs must complete execution prior to their deadline- a missed deadline constitutes a system failure.[7] Such systems are used where the consequences of missing a deadline may be serious or even disastrous. A soft real-time system is less restrictive. Jobs may continue execution beyond their deadlines at some penalty - deadlines are considered as guidelines, and the system tries to minimize the penalties associated with missing them. Such systems are used when the consequences of missing deadlines are smaller than the cost of meeting them in all possible circumstances. Cell phones and multimedia applications would both use soft real-time systems. [2]

II. PRIORITY INVERSION AND DEADLOCK

Priority inversion can occur when the execution of some jobs or portions of jobs is nonpreemptable.[4] Resource contentions among jobs can also cause priority inversion. Because resources are allocated to jobs on a nonpreemptive basis, a higher-priority job can be blocked by a lower-priority job if the jobs conflict, even when the execution of both jobs is preemptable.

Without good resource access control, the duration of a priority inversion can be unbounded. The example in Figure 1 illustrates this fact. Here, jobs $J1$ and $J3$ have the highest priority and lowest priority, respectively. At time 0, *J*3 becomes ready and executes. It acquires the resource *R* shortly afterwards and continues to execute. After *R* is allocated to *J*3, *J*1 becomes ready. It preempts *J*3 and executes until it requests resource *R* at time 3. Because the resource is in use, *J*1 becomes blocked, and a priority inversion begins. While *J*3 is holding the resource and executes, a job *J*2 with a priority higher than *J*3 but lower than *J*1 is released. Moreover, *J*2 does not require the resource *R*. This job preempts *J*3and executes to completion. Thus, *J*2 lengthens the duration of this priority inversion. In this situation, the priority inversion is said to be uncontrolled [6]. There can be an arbitrary number of jobs with priorities lower than *J*1 and higher than *J*3 released in the meantime. They can further lengthen the duration of the priority inversion. Indeed, when priority inversion is uncontrolled, a job can be blocked for an infinitely long time.

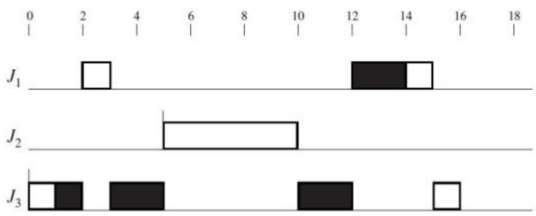


Figure 1 Priority Inversion

Nonpreemptivity of resource allocation can also lead to deadlocks. The classic example is one where there are two jobs that both require resources *X* and *Y*. The jobs are in deadlock when one of them holds *X* and requests for *Y*, while the other holds *Y* and requests for *X*. The conditions that allow this circular wait of jobs for each other (i.e., a deadlock) to occur are well-known.

III. RESOURCE CONTROL TECHNIQUES

Many resource control techniques have been proposed for real-time systems. These vary from techniques for use with priority preemptive scheduling algorithms, for example the collection of techniques derived from *priority inheritance* [1], to those that rely upon scheduling resources along with processes in a rigid manner pre-runtime [4,5]. All these techniques are summarized in following series of criteria:

(a) Predictable or non-predictable

(b) Blocking or non-blocking

(c) Runtime non-blocking or pre-runtime non-blocking

(d) Preemptive blocking or non-preemptive blocking

A. Priority Inheritance Protocol

The priority inheritance protocol (PIP) [8] assumes that:

(a) Static priorities are assigned to processes

(b) Resources are accessed in a mutually exclusive manner

(c) Resource accesses are properly nested

(d) A preemptive priority driven scheduler is used (where the highest priority runnable process is given the processor)

(e) The resources that a process accesses can be determined pre-runtime.

1) Rules of the Basic Priority Inheritance Protocol

1. *Scheduling Rule*: Ready jobs are scheduled on the processor preemptively in a priority

driven manner according to their current priorities. At its release time *t*, the current priority $\pi(t)$ of every job *J* is equal to its assigned priority. The job remains at this priority except under the condition stated in rule 3.

2. *Allocation Rule*: When a job *J* requests a resource *R* at time *t*,

(a) If *R* is free, *R* is allocated to *J* until *J* releases the resource, and

(b) If *R* is not free, the request is denied and *J* is blocked.

3. *Priority-Inheritance Rule*: When the requesting job *J* becomes blocked, the job *Jl* which blocks *J* inherits the current priority $\pi(t)$ of *J* . The job *Jl* executes at its inherited priority $\pi(t)$ until it releases *R*; at that time, the priority of *Jl* returns to its priority $\pi l\ (t')$ at the time *t'* when it acquires the resource *R*

B. Priority Ceiling Protocol

The priority ceiling protocol (PCP) is one instance of a class of priority inheritance protocols [6]. The motivation of the PCP is to address the deadlock and chaining problems of the priority inheritance protocol. This is achieved by ensuring that a strict ordering of critical region execution is maintained. The same assumptions are made about processes and resources as in priority inheritance.

1) Rules of the Priority Ceiling Protocol

1. *Scheduling Rule*:

(a) At its release time *t*, the current priority $\pi(t)$ of every job *J* is equal to its assigned priority. The job remains at this priority except under the condition stated in rule 3.

(b) Every ready job *J* is scheduled preemptively and in a priority-driven manner at its current priority $\pi(t)$.

2. *Allocation Rule*: Whenever a job *J* requests a resource *R* at time *t*, one of the following two conditions occurs:

(a) *R* is held by another job. *J* 's request fails and *J* becomes blocked.

(b) *R* is free.

(i) If *J*'s priority $\pi(t)$ is higher than the current priority ceiling *(t)*, *R* is allocated to *J* .

(ii) If *J*'s priority $\pi(t)$ is not higher than the ceiling $\prod(t)$ of the system, *R* is allocated to *J* only if *J* is the job holding the resource(s) whose priority ceiling is equal to $\prod(t)$; otherwise, *J*'s request is denied, and *J* becomes blocked.

3. *Priority-Inheritance Rule*: When *J* becomes blocked, the job *Jl* which blocks *J* inherits the current priority $\pi(t)$ of *J*. *Jl* executes at its inherited priority until the time when it releases every resource whose priority ceiling is equal to or higher than $\pi(t)$; at that time, the priority of *Jl* returns to its priority $\pi l\ (t')$ at the time *t'* when it was granted the resource(s).

The PCP can be summarized as:

- A priority ceiling is assigned to each resource equal to the highest priority of all processes that could lock it.
- A resource is allocated if the priority of the requesting process is strictly greater than the ceilings of all currently held resources. If the resource is not allocated, the requesting process becomes blocked upon that resource.

A process executes at its assigned priority unless it blocks a higher priority process at which time it inherits the priority of the blocked process for the duration of the current critical region (as in priority inheritance protocol).

One disadvantage of the PCP is its pessimism in terms of blocking times. The only circumstances that a high priority process can be blocked for the entire duration of the critical region of a lower priority process is when locks a resource required by(or required by an even higher priority process) and performs no execution before requires a resource. Effectively, the lower priority job must lock the resource momentarily before higher priority job becomes runnable. This is clearly pessimistic.[8]

IV. SCHEDULING UNDER PIP AND PCP

1) Parameters of Jobs

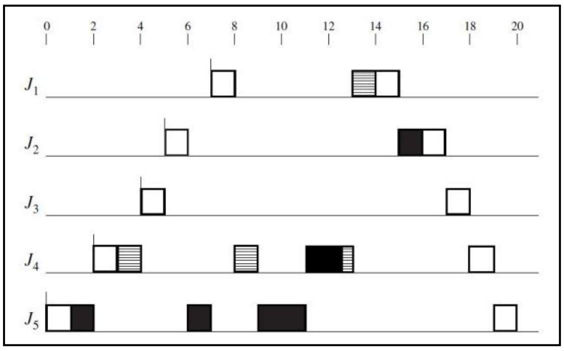
Job	r_i	e_i	π_i	Critical Sections
J_1	7	3	1	[*Shaded*; 1]
J_2	5	3	2	[*Black*; 1]
J_3	4	2	3	
J_4	2	6	4	[*Shaded*; 4 [*Black*; 1.5]]
J_5	0	6	5	[*Black*; 4]

r_i = arrival time

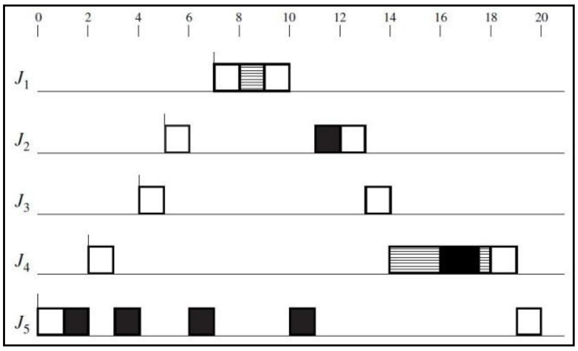
e_i = execution time

Π_i = Priority of job

2) Schedule under PIP



3) Schedule under PCP



V. COMPARISON OF PIP AND PCP

Priority Inheritance	**Priority Ceiling**
It is Greedy.[3]	It is not Greedy.[3]
PIP lets the requesting job have a resource whenever the resource is free.[2]	In PCP, a job may be denied its requested resource even when the resource is free at that time.[2]
Poor worst case behavior when there are nested locks.[1]	Good worst case priority inversion control. Handles nested locks well.[1]
Extra context switches are avoided. Medium Priority task are not preempted unnecessarily. Leads to excellent average performance.[1]	Pay cost of changing priority twice regardless of whether there is any contention for the lock or not. Resulting in higher overhead and many unnecessary context switches.[1]
Effective Lock is seldom part of a nested set and when average performance is relevant in addition to worst case performance.[1]	**Effective** When task contending for lock is known. - When there may be nested locks &worst case behavior is only in concern.[1]

VI. CONCLUSION

Comparison of PIP and PCP is done in this paper. PIP is suffering from priority inversion and dead lock issues while PCP is solving that problem at certain level. Number of context switches will be up and down in both protocols depends upon the input. So here we compared the protocols in terms of priority inversion, deadlock, resource acquiring techniques etc.

REFERENCES

[1] Doug Locke, "Priority Inheritance: The Real Story", July, 2002

[2] Pratibha Zunjare , Bibhudatta Sahoo, "Evaluating Robustness of Resource Allocation in Uniprocessor Real-time System", *International Journal of Computer Applications (0975 – 8887) Volume 40– No.3, February 2012*

[3] Prof. Rajib Mall, CSE, IIT Kharagpur, Module -3, "Handling Resource sharing and dependencies among real-time tasks." nptel.iitm.ac.in/courses/Webcourse-contents/IIT Kharagpur/Real time system/pdf/module3.pdf

[4] Jane W. S. Liu, "Real-Time Systems", Pearson, 2011

[5] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard real-time environment," *Journal of the ACM*,vol. 20, no. 1, pp. 46-61, Jan. 1973.

[6]Sha, L., R. Rajkumar, and J. P. Lehoczky, "Priority inheritance protocols: An approach to real-time synchronization," *IEEE Transactions on Computers*, vol. 39, 1990

[7] A M Shah, "Dynamic Scheduling for Real-Time Operating Systems", *Ph. D. Thesis, Information Technology Department, Sardar Patel University, India,2010*

[8] Neil C. Audsley, "Resource control for hard real-time systems: a review", *University of York, August 1991*