Volume 4, Issue 6, June 2014 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

# IPIP: An Improved Priority Inheritance Protocol for Real-Time Systems

**Apurva Shah***
Faculty of Technology,
The M S University of Baroda,
Vadodara, Gujarat, India

**Vishal Prajapati**
Department of IT,
G H Patel College of Engg and Tech,
V V Nagar, Gujarat, India

**Prem Balani**
Department of IT
G H Patel College of Engg and Tech,
V V Nagar, Gujarat, India

***Abstract—*** ***Resource allocation is one of the challenging problems for real time operating system. Priority inheritance protocol (PIP) and Priority ceiling protocol 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 and asks for resource, it increases the number of context switches. Because of this overhead, CPU efficiency is decreased.***
***Our proposed algorithm will decrease the overhead and therefore overall efficiency of the CPU increases considerably.***

***Keywords—*** ***Priority inheritance protocol, Resource allocation, Real-time operating system, Scheduling, Resource control***

## I. INTRODUCTION

Real-time systems are 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. 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 are examples of soft real-time systems [2].

## II. 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:

- Predictable or non-predictable
- Blocking or non-blocking
- Runtime non-blocking or pre-runtime non-blocking
- Preemptive blocking or non-preemptive blocking

### *A. Priority Inheritance Protocol*

The priority inheritance protocol (PIP) [1] assumes that:

- Static priorities are assigned to processes
- Resources are accessed in a mutually exclusive manner
- Resource accesses are properly nested
- A preemptive priority driven scheduler is used (where the highest priority runnable process is given the processor)
- The resources that a process accesses can be determined pre-runtime.

### *B. Rules of the Basic Priority-Inheritance Protocol*

- *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.

 

- *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.

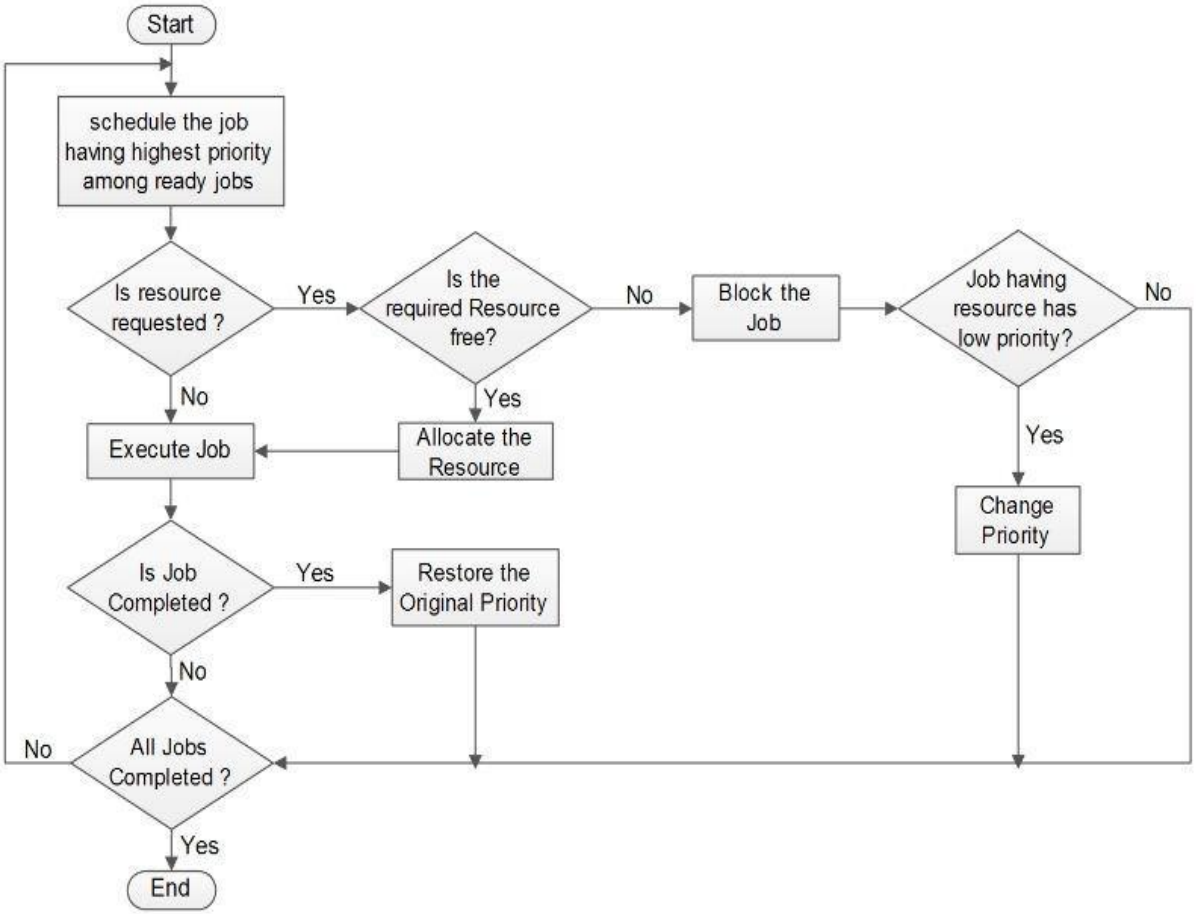


Figure 1.Flowchart for priority inheritance protocol

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

*A. Improved Priority Inheritance Protocol*

*As we have seen the numbers of context switches are more while scheduling the jobs. More context switches will occupy more CPU time so we have proposed Improved Priority Inheritance Protocol which will lead to less number of context switching.*

*Rules of the Improved Priority-Inheritance Protocol*

*1. Scheduling Rule: Ready jobs are scheduled on the processor pre-emptively in a priority driven manner*

- *If highest priority job do not require resource in future than schedule it.*
- *If the higher priority job requires resource in future and those resources are not acquired by other jobs than schedule it.*
- If the higher priority job requires resource in future and any of those resource is acquired by any other job than block this job and change the priority of job having resource.

2. *Allocation Rule*: When a job *J* requests a resource *R* at time *t*, than allocate the resource to that job.

3. *Improved priority-Inheritance Rule*: When the requesting job *J* going to be scheduled 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*.

## III. A CASE STUDY OF A RESOURCE ALLOCATION INSTANCE

In order to compare the performance of different algorithms, a simulator program has been developed in the C language [10]. The required task set is written in a separate file indicating different parameters as shown in the Table 1.

PIP and IPIP protocols were implemented and results are shown in figure 2 and figure 3. We can observe that number of context switches are 13 in case of PIP. The same is decreased to 11 with IPIP during same environment. Thus, we achieved 15.39% reduced context switches with this protocol in the above conditions.

 

TABLE I
PARAMETERS OF JOBS

| Job | $r_i$ | $e_i$ | $\pi_i$ | 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 : 3 Black : 1 |
| $J_5$ | 0 | 6 | 5 | Black : 4 |

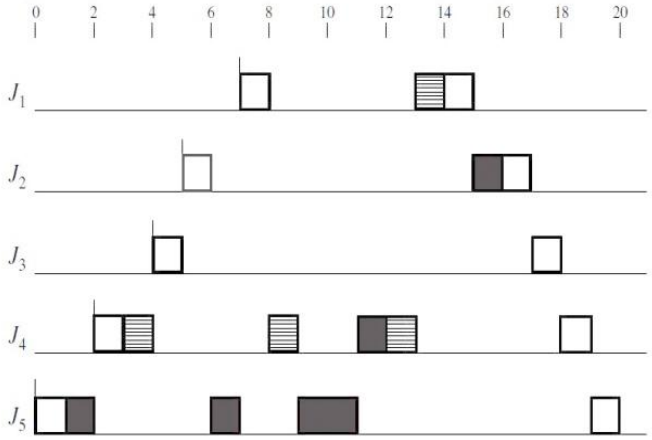


*Figure 2. Schedule with PIP*

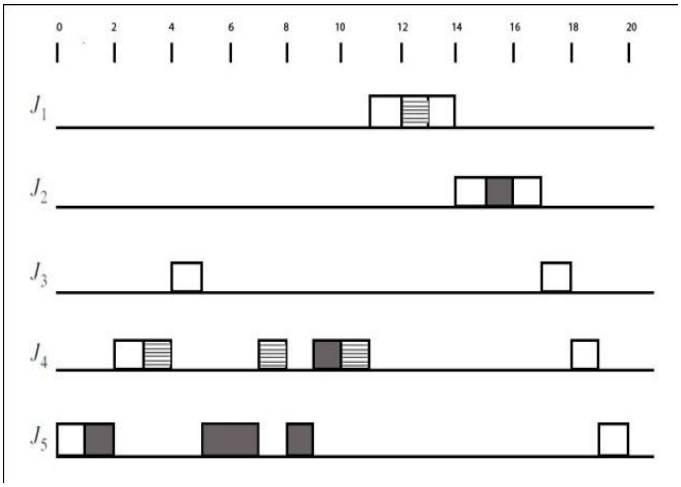


*Figure 3. Schedule under IPIP*

TABLE III
COMPARISON OF PIP AND IPIP PROTOCOLS

| Sr. No | PIP | IPIP | Reduction |
|---|---|---|---|
| 1 | 7 | 7 | 0.00 |
| 2 | 13 | 11 | 15.38 |
| 3 | 5 | 5 | 0.00 |
| 4 | 8 | 5 | 37.50 |

 

TABLE IIII (Continued)

| | | | |
|---|---|---|---|
| 5 | 6 | 6 | 0.00 |
| 6 | 9 | 8 | 11.11 |
| 7 | 8 | 6 | 25.00 |
| 8 | 13 | 7 | 46.15 |
| 9 | 6 | 6 | 0.00 |
| 10 | 7 | 6 | 14.29 |
| 11 | 7 | 7 | 0.00 |
| 12 | 6 | 6 | 0.00 |
| 13 | 8 | 7 | 12.50 |
| 14 | 9 | 7 | 22.22 |
| 15 | 8 | 8 | 0.00 |
| 16 | 9 | 7 | 22.22 |
| 17 | 8 | 7 | 12.50 |
| 18 | 11 | 10 | 9.09 |
| 19 | 6 | 6 | 0.00 |
| 20 | 7 | 7 | 0.00 |
| 21 | 7 | 7 | 0.00 |
| 22 | 7 | 7 | 0.00 |
| 23 | 8 | 6 | 25.00 |
| 24 | 11 | 7 | 36.36 |
| 25 | 9 | 7 | 22.22 |
| 26 | 12 | 8 | 33.33 |
| 27 | 11 | 8 | 27.27 |
| 28 | 6 | 6 | 0.00 |
| 29 | 6 | 6 | 0.00 |
| 30 | 9 | 7 | 22.22 |
| 31 | 10 | 7 | 30.00 |
| 32 | 8 | 8 | 0.00 |
| 33 | 7 | 5 | 28.57 |
| 34 | 10 | 8 | 20.00 |
| 35 | 10 | 9 | 10.00 |
| 36 | 7 | 5 | 28.57 |
| 37 | 5 | 5 | 0.00 |
| 38 | 6 | 6 | 0.00 |
| 39 | 12 | 7 | 41.67 |
| 40 | 9 | 9 | 0.00 |
| 41 | 7 | 7 | 0.00 |
| 42 | 5 | 5 | 0.00 |
| 43 | 7 | 6 | 14.29 |
| 44 | 5 | 5 | 0.00 |
| 45 | 10 | 8 | 20.00 |
| 46 | 9 | 7 | 22.22 |
| 47 | 11 | 9 | 18.18 |
| 48 | 9 | 7 | 22.22 |
| 49 | 7 | 7 | 0.00 |
| 50 | 7 | 7 | 0.00 |

 

## IV. RESULTS

The simulation has been done for more than 50 different randomly generated task sets and results are generated with PIP and IPIP protocols. Table II shows the results of the same comparing PIP and IPIP protocols in terms of context switches. Results show that the average reduction in context switches is 13%. Maximum reduction in context switches is 46.15%. For some of the cases we got 0.00% results means that there is no reduction in the context switches.

We also observed the following for different cases:

- Case 1:Context switches are in between 5 to 7:
  We got the average reduction 3.73% which is negligible. Because of 5 jobs ideally it will take minimum 4 context switch. We also have the resource allocation so in between 5 to7 context switches we have got 3.73% reduction.
- Case 2: Context switches are in between 8 to 10
  Average reduction is 17.73 when context switches are in between 8 to 10 in priority inheritance protocol.
- Case 3: Context switches are in between 11 to 13
  Average reduction is 28.43 when context switches are in between 11 to 13 in priority inheritance protocol.

## V. CONCLUSION

We can conclude following from the results achieved during simulation.

- We can say that IPIP performs well in case of context switches are in between 11 to 13 for five jobs.
- IPIP gives almost same performance as PIP in case of context switches in between 5 to 7 for five jobs.
- This algorithm is more recommendable for complex real-time systems where number of jobs and number of tasks are more comparatively. The algorithm is more effective when number of context switches required heavily.

## REFERENCES

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

[2] K.Kotecha and A Shah, Adaptive Scheduling Algorithm for real-time operating system, *In proceedings of IEEE Congress on Evolutionary Computation (CEC 2008)*, HongKong, pp. 2109-2112, June 2008.

[3] K. Ramamritham and J. Stankovik, Scheduling Algorithms and Operating Support for Real-Time Systems, *Proc. of the IEEE*, vol 82(1), pp. 55-67, 1994.

[4] J. A. Stankovic, K. Ramamritham and W. Zhao, Preemptive Scheduling Under Time andResource Constraints, *IEEE Trans Computers*, vol. 38(8), pp. 949-960, Aug 1987.

[5] A. Damm, J. Reisinger, W. Schwabl and H. Kopetz, The Real-Time Operating System MARS, *ACM Operating Systems Review Special Issue*, pp. 141-157, 1989.

[6] Jane W. S. Liu, *Real-Time Systems*, Pearson, 2011

[7] D Locke, Priority Inheritance: The Real Story, July 2002

[8] P Zunjare and B.Sahoo, Evaluating Robustness of Resource Allocation in Uniprocessor Real Time System, *International Journal of Computer Applications*, vol 40(3), pp. 975-987, Feb. 2012.

[9] R. Mall, *Handling Resource sharing and dependencies among real time tasks, Module 3.*

[10] A. Shah, K. Kotecha and D. Shah, Dynamic scheduling for Real-Time distributed systems, International Journal of Intelligent Computing and Cybernetics, vol. 3 (2), pp. 279-292, 2010.

 