Journal of Computer Science and Applications.
ISSN 2231-1270 Volume 6, Number 2 (2014), pp. 115-123
© International Research Publication House
http://www.irphouse.com

Simulation And Comparison of Various Scheduling Algorithm For Improving The Interrupt Latency of Real –Time Kernal

[1.]Lavanya Dhanesh [2.]Dr.P.Murugesan

[1.]Research Scholar, Sathyabama University, Chennai, India.
[2.]Professor, S.A. Engineering College, Chennai, India.
Email:[1.] dhaneshforyou@gmail.com

Abstract

The main objective of the research is to improve the performance of the Real-time Interrupt Latency using Pre-emptive task Scheduling Algorithm. Interrupt Latency provides an important metric in increasing the performance of the Real Time Kernal So far the research has been investigated with respect to real-time latency reduction to improve the task switching as well the performance of the CPU. Based on the literature survey, the pre-emptive task scheduling plays an vital role in increasing the performance of the interrupt latency. A general disadvantage of the non-preemptive discipline is that it introduces additional blocking time in higher priority tasks, so reducing schedulability . If the interrupt latency is increased the task switching delay shall be increasing with respect to each task. Hence most of the research work has been focussed to reduce interrupt latency by many methods. The key area identified is, we cannot control the hardware interrupt delay but we can improve the Interrupt service as quick as possible by reducing the no of preemptions. Based on this idea, so many researches has been involved to optimize the pre-emptive scheduling scheme to reduce the real-time interrupt latency. Based on the literature survey, A Deadline Monotonic Priority Assignment technique is used to reduce the latency with respect to the deadline. Deferred pre-emption scheduling and Fixed pre-emptive scheduling algorithm’s are used to reduce the interrupt latencies. Thus we employ the Pre-emptive task scheduling algorithm which preempts and serve the task with highest priority The performance of the Interrupt Latency can be analysed using the SKYEYE simulator, a tool which gives accurate time of the Interrupt Service Routine(ISR).

Keywords: CPU, Performance, Scheduling, Interrupt service, Interrupt Latency, Preemption.

Introduction

Real Time Operating System is considered as efficient when its throughput rate is high. They are the embedded devices which are designed for the specific use and are expected to meet the specific time deadlines for completing the tasks[1]. For this, the response time of the important tasks is most important which can be achieved using the priority based task – scheduling[2,3]. This paper puts forth the idea of applying the pre-emptive based task scheduling algorithm to reduce the Real time Interrupt Latency. Better processor utilization and more flexibility is achieved with the scheduling tasks than non-preemptive scheduling. Furthermore, preemptive approaches may need less runtime support (e.g. no task ordering required). In this paper we present a low-cost algorithm, called the Preemptive Task Scheduling algorithm (PTS), which is intended for compile-time scheduling of coarse-grain problems.

Background

Preemption

Tasks waits for the processing.The Scheduler assigns priorities and the task with highest priority will be scheduled first[4,5].It preempts the current execution if a higher priority (more urgent) task or a real-time processes have higher priority arrives.

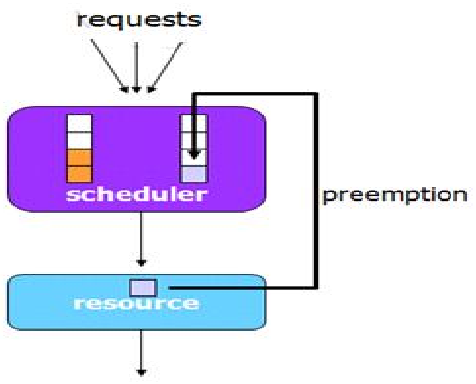


Figure 1: Preemption Process

Interrupt Latency(IL):

The Interrupt Latency is the time from when an Interrupt is triggered until the Interrupt process starts. In figure 2.2 there is an overview of the different stages that passes when an interrupt occurs.

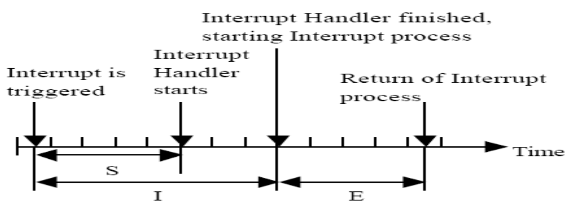

Interrupt Handler finished, starting Interrupt process
Interrupt Handler starts
Interrupt is triggered
Return of Interrupt process
Time
S
I
E

Figure 2: Interrupt Latency in OSE

S	=	Time to start the Interrupt Handler
I	=	Interrupt Latency
E	=	Interrupt process execution time

Scheduling theory

We present an overview of real-time scheduling in this section. A real-time OS kernel (scheduler) switches concurrently running tasks based on a scheduling scheme. A task is composed of a series of jobs, thus, it is the scheduling scheme that decides which job of the task is to run next and for how long[6,7]. Therefore,the scheduler plays a crucial role in real-time systems.

The main goal of the scheduler is to meet predefined deadlines and, at the same time, to increase processor utilization. There are three types of tasks:

- Periodic task releases each job regularly with the task period Ti.
- Aperiodic task releases irregularly.
- Sporadic task is an aperiodic task characterized by a minimum inter arrival time between consecutive jobs of the task.

A real-time scheduling algorithm ensures that each job is finished before its deadline. In hard real-time systems, a scheduling method has to guarantee that each job finishes before its deadline. In soft real-time systems, the system is still operable even if some jobs occasionally miss their deadlines. In addition to other definitions earlier, the processor utilization factor is defined as

***U* = ∑ *i*=1 *Ci / Ti* in order** to measure the processor utilization of CPU resource, where *Ci* is the worst-case execution time (WCET) and *n* is the number of tasks in process. The preemptive scheduler can stop a running task during its execution (preempt the task) to give CPU resources to another ready task. This feature is used for CPU time sharing between ready tasks with the same priority. It is also used in case of an interrupt which may wake up a task waiting for a signal or for some data[8,9]. The woken task should have a higher priority than the current running task to be allocated CPU time directly.

Context Switch Overhead

The context switch overhead is defined as the amount of time taken by the operating system to switch from one task to another, without any other task or interrupt subroutine being executed in between them . In Free RTOS, the procedure of context switch is visible in the code of tick timer interrupt, which consists of two parts excluding the context saving and restoring. Increase the tick count and check if any task that is blocked for a finite period, requires its removal from a blocked list and placing on a ready list[10] . Set the pointer to the current Task Control Block (TCB) to the TCB of the highest priority task that is ready to run.

Real – Time Task Scheduling

In general, real-time systems are required to meet time constraints. Most of them adopt either an in-house or commercial real-time operating system (OS). One of the important functions of the OS is to schedule concurrently running tasks in order to avoid missing of deadline. A real-time task scheduler plays an essential role in switching tasks to share the processor resource, and also in ensuring that all tasks to be finished before deadline. A real-time scheduler dispatches multiple tasks to be run in a CPU to meet deadlines[11]. From now on, we will interchangeably use a real-time scheduler, an OS task scheduler, and a task scheduler to indicate the task scheduler mechanism of a given real-time system. In order to do the scheduling analysis and design for real time systems, it is important to obtain related scheduling parameters, such as a task execution time and a task period. It is difficult, however, to obtain precise scheduling parameters for three reasons:

- Computing systems consist of many resource queues, i.e., ready queues, event queues, semaphore queues, and message queues and the time measurements of activities in these interrelated queues are not trivial.
- Modern CPU makers use more and more advanced technologies to improve performance, such as an instruction and a data cache, and an instruction pipeline[12]. This makes it troublesome to estimate scheduling parameters.
- It is a recent trend that even a simple real-time system is composed of network-connected distributed nodes. Network-connected systems inherit uncertainty that causes difficulties in measuring the scheduling parameters[13]. The high degree of uncertainty challenges to the accurate measurement of the scheduling parameters. This leads to scheduling errors that cause performance degradation (deadline missing) and inefficient processor utilization in real-time systems.

Existing Scheduling Algorithms

Earliest Deadline First (EDF)

It is a Preemptive scheduling based on dynamic task priorities.Task with closest deadline has highest priority. stream priorities vary with time.Dispatcher selects the highest priority task.

Assumptions

- Requests for all tasks with deadlines are
- Periodic
- The deadline of a task is equal to the end on its period (starting of next)
- Independent tasks (no precedence)
- Run-time for each task is known and constant
- Context switches can be ignored

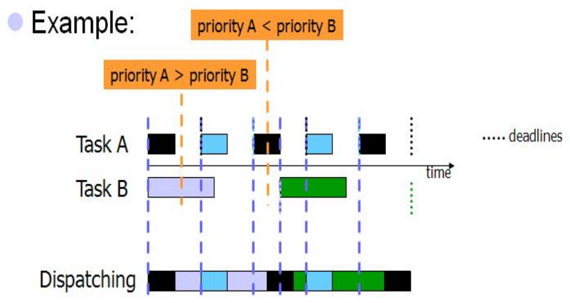


Figure 3: EDF Priority order

Rate Monotonic (RM) Scheduling :

It is a classic algorithm for hard real-time systems with one CPU. Pre-emptive scheduling based on static task priorities.

Assumptions

- Requests for all tasks with deadlines are periodic
- The deadline of a task is equal to the end on its period (starting of next)
- Independent tasks (no precedence)
- Run-time for each task is known and constant
- Context switches can be ignored any non-periodic task has no deadline

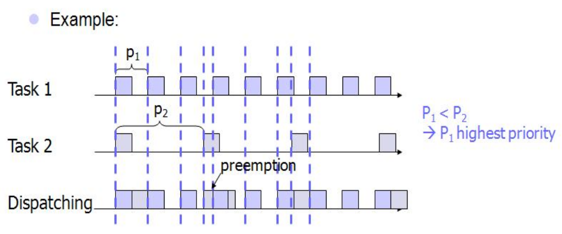


Figure 4: RM Task scheduling

Deferred Preemptions Scheduling (Dps)

According to this method, each task *τi* defines a maximum interval of time *qi* in which it can execute non-preemptively[14]. Depending on the specific implementation, the non-preemptive interval can start after the invocation of a system call inserted at the beginning of a non-preemptive region (floating model), or can be triggered by the arrival of a higher priority task (activation-triggered model).

Round Robin (RR) Scheduling
Works well for short jobs, typically used in timesharing systems. High overhead due to frequent context switches. Increases average waiting time, especially if CPU bursts are the same length and need more than one time quantum[15]. Because of high waiting times, deadlines are rarely met in a pure Round Robin system.

Pre - Emptive Task Scheduling (Pts)
Above all algorithms the *Preemptive Task Scheduling Algorithm (PTS)* is the only algorithm designed for preemptive distributed memory systems . PTS can not fully utilize the potential of a preemptive environment. It gives emphasis on load balancing and keeping the scheduling cost low. PTS schedules the tasks in the order of latest possible task time on the processor which has the least load at each iteration. The processor load is given by the number of tasks simultaneously running on that processor. In preemptive system the a task's executing time on its assigned processor depends not only on task's size, but also by the processor load. As the processor load keep on changing with time, it is not possible to calculate a task's execution finish time ahead of time. So, execution finish time of a task have to be calculated on the run. Now we briefly describe the operations of the PTS algorithm. At first PTS computes the latest possible start time of the tasks, and it is used as the priority. Then PTS schedules one task at a time. At each iteration the ready task with the highest latest possible start time is selected. Then the task execution simulation is updated by stopping the tasks that would finish before current (to be scheduled) task. So, processor loads are updated, than the current task is scheduled. The key concept present in any operating system which allows the system to support multitasking, multiprocessing,etc. is Task Scheduling [1]. Task Scheduling is the core which refers to the way the different processes are allowed to share the common CPU. Scheduler and dispatcher are the softwares which help to carry out this assignment [2].In this type of algorithms the CPU access is taken away by the operating system kernel.In this scheduling if a first task is unable to execute for any reasons (if a page fault occurs) then the Kernal will stop executing the first task and the second task can begin its execution.If the fault occurs in one system it will not affect the entire application. OS schedules such that higher priority task when ready preempts the lower priority by blocking. Hence it solves the problem of large Worst case latency for the higher priority task

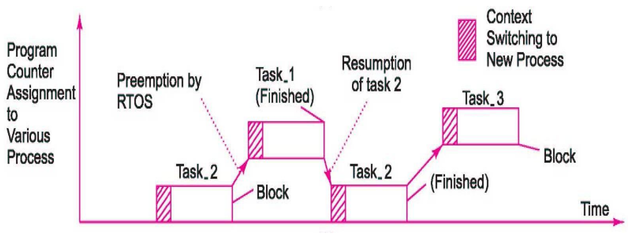


Figure 5: Program counter assignments on the a scheduler call to preempt task 2. when *priority* of task_1 > task_2 > task_3

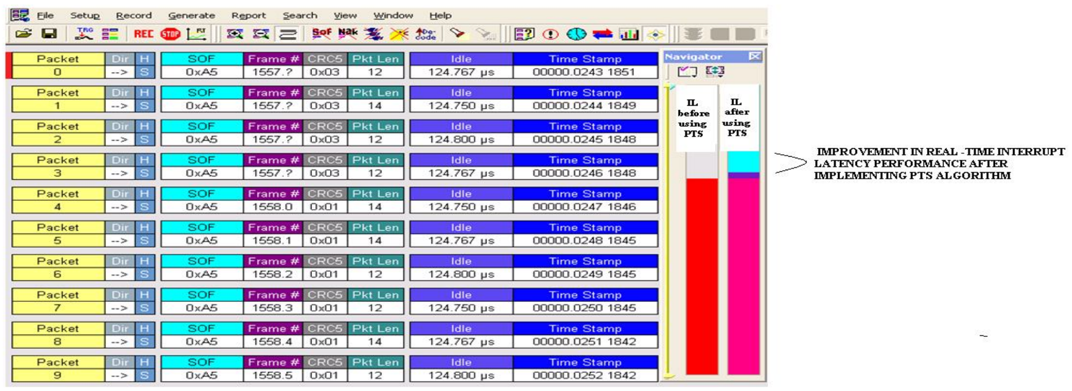
Pre- Emptive Task Scheduling (Pts) Algorithm

```
PTS ()
BEGIN
For all tasks compute bottom levels.
WHILE NOT all tasks scheduled DO
t Task with the highest bottom level.
MATt's last message arrival time.
Stop the tasks finishing before MAT
and update their successors.
ST t's start time.
pLeast loaded processor.
Start task t on p at ST.
ENDWHILE
END
```

Figure 6: The PTS algorithm

Experimental Results

Here we employ Lecroy USB Tracer software and SKYEYE simulator, a tool which gives accurate time of the Interrupt Service Routine which in turn is used to calculate the Interrupt latency. From the Interrupt Latency performance it is noted that when PTS algorithm if employed it gives good improvement in the performance of the Interrrupt Latency when compared with the other algorithms. The simulation output also gives the details of the number packets, synchronization of data, Packet length, Idle state and Time stamp which is useful to judge the performance improvement in the Interrupt latency.



Packet	Dir	H	SOF	Frame #	CRC5	Pkt Len	Idle	Time Stamp
0	-->	S	0xA5	1557.?	0x03	12	124.767 µs	00000.0243 1851
1	-->	S	0xA5	1557.?	0x03	14	124.750 µs	00000.0244 1849
2	-->	S	0xA5	1557.?	0x03	12	124.800 µs	00000.0245 1848
3	-->	S	0xA5	1557.?	0x03	12	124.767 µs	00000.0246 1848
4	-->	S	0xA5	1558.0	0x01	14	124.750 µs	00000.0247 1846
5	-->	S	0xA5	1558.1	0x01	14	124.767 µs	00000.0248 1845
6	-->	S	0xA5	1558.2	0x01	12	124.800 µs	00000.0249 1845
7	-->	S	0xA5	1558.3	0x01	12	124.750 µs	00000.0250 1845
8	-->	S	0xA5	1558.4	0x01	14	124.767 µs	00000.0251 1842
9	-->	S	0xA5	1558.5	0x01	12	124.800 µs	00000.0252 1842

Figure 7: Simulation Output

Conclusion

In our research we implemented the Preemptive Task Scheduling (PTS) algorithm to improve the performance of the Real-Time Interrupt Latency . When compared with other scheduling algorithms it is noted that the PTS algorithm gives a better performance in the Interrupt Latency of the Real-Time system. Thus by reducing the interrupt latency we can improve the performance of the Real –Time interrupt Latency .

References

[1] I. Ahmad and Y.-K. Kwok. A new approach to scheduling parallel programs using task duplication. pages 47–55. ICCP, 1994.

[2] Y.-C. Chung and S. Ranka. Application and performance analysis of a compile-time optimization approach for list scheduling algorithms on distributed-memory multiprocessors. pages 512–521. Supercomputing, 1992.

[3] E. G. Coffman Jr. and P. J. Denning. *Operating Systems Theory*. Prentice Hall, 1973.

[4] A. Gonzlez-Escribano, A. J. C. van Gemund, and V. Cardeoso-Payo. Performance trade-offs in series-parallel programming models. pages 183–189. Eighth International Workshop on Compilers for Parallel Computers (CPC'00), Aussois, 2000.

[5] R. L. Graham. Bounds on multiprocessing timing anomalies. *SIAM J. on Applied Mathematics*, pages 416–429, 1969.

[6] J.-J. Hwang, Y.-C. Chow, F. D. Anger, and C.-Y. Lee. Scheduling precedence graphs in systems with interprocessor communication times. *SIAM J. on Computing*, 18(2):244–257, 1989.

[7] M. Kafil and I. Ahmad. Optimal task assignment in heterogeneous computing systems. Heterogeneous Computing Workshop, 1997.

[8] A. A. Khan, C. McCreary, and M. S. Jones. A comparison of multiprocessor scheduling heuristics. volume 2, pages 243–250. ICPP, 1994.

[9] B. Kruatrachue and T. G. Lewis. Grain size determination for parallel processing. *IEEE Software*, pages 23–32, Jan 1988.

[10] Y.-K. Kwok and I. Ahmad. Benchmarking the task graph scheduling algorithms. IPPS/SPDP, 1998.

[11] C.-Y. Lee, J.-J. Hwang, Y.-C. Chow, and F. D. Anger. Multiprocessor scheduling with interprocssor communication delays. *Operations Research Letters*, 7:141–147, June 1988.

[12] J.-C. Liou and M. A. Palis. A comparison of general approaches to multiprocessor scheduling. pages 152–156. Int'l Parallel Processing Symp, 1997.

[13] M. Maheswaran and H. J. Siegel. A dynamic matching and scheduling algorithm for heterogeneous computing systems. Heterogeneous Computing Workshop, 1998.

[14] F. Mueller. A library implementation of posix threads under unix. Winter, 1993.

[15] G.-L. Park, B. Shirazi, J. Marquis, and H. Choo. Decisive path scheduling: A new list scheduling method. Int'l Conf. on Parallel Processing, 1997.