International Journal of Trend in Scientific Research and Development (IJTSRD)
International Open Access Journal
ISSN No: 2456 - 6470 | www.ijtsrd.com | Volume - 2 | Issue – 1

Investigative Study on Scheduling Procedures for Real Time Operating System

Pallavi Ganeshpurkar
M.Tech Scholar, Dept. of CSE Kalinga University, Raipur, Chhattisgarh, India

ABSTRACT

Scheduling procedures are a backbone of any operating system. In this paper I analyze and conclude that the types of different scheduling procedures used in real time. As I know scheduling procedures are basically divided into two main streams: first is the uniprocessor and another one is multiprocessor.

This paper describes the uniprocessor system which are based on static rate monotonic (SRM) and static deadline monotonic (SDM). Here I studied and analyzed different SRM and SDM procedures to conclude that which algorithm or which policy is best for real time scheduling. This paper is also defined that which static algorithm take less time for the completion of different tasks whether it is SRM or SDM. I also have given short details about the dynamic approach of scheduling. Under this I have explained Earlier deadline first (EDF) and Least laxity first (LLF) in brief.

Keywords: *Static Rate Monotonic (SRM), Static Deadline monotonic (SDM), Earlier deadline first (EDF), Least laxity first (LLF)*

INTRODUCTION

The scheduling of real-time tasks is very different from general scheduling. Ordinary scheduling procedures attempt to ensure fairness among tasks, minimum progress for any individual task, and prevention of starvation and deadlock. Within computer science real-time systems are an important while often less known branch. I use Real-time systems in so many ways today more than PCs in our real life, still I are not so familiar about it when I use the devices in which they reside. Some of the devices in which real time system resides are cars, planes and entertainment system which govern the working of those devices which I do not consider that such system exist within the chosen device .Basically I can say that a real-time-system is a computer based system in which the major aspect of the system is to perform tasks on time, not finishing too early nor too late. A classic example is that of the opening of para suit; it is of great importance that the para suit must be pulled in time not too soon not too late in order to land safely while skydiving. One more example is of the air- bag in a car; it is of great importance that the bag inflates neither too soon nor too late in orders to be of aid and not be potentially harmful.

In this paper I survey several procedures developed over the last few years that are designed to schedule real-time tasks in distributed systems. The choice of algorithm can influence the behavior of a real-time system and for this reason there are many available procedures. For the different categories of real- time systems there are specialized procedures developed. With the help of this paper I will attempt to provide an overview of many of the different available real-time procedures.[1]

Before examining the actual procedures, it is helpful to establish the exact meanings of the terms real-time task and distributed system. I provide a basic definition of what a real- time task is and identify the different dimensions along which this definition may vary.

Other overviews of real-time scheduling procedures have been presented by Burns[1], Burns and

Audsley[2] and by Mohammadi and Akl[3]. Those are somewhat more in depth on some topics then this overview.

The rest of the paper is organized as follows: - In section 2 basic concepts of real-time system and scheduling are explained. Section 3 deals with the different scheduling procedures being the main section of this paper. Here I have discussed about Uniprocessor algorithm under which static and dynamic scheduling procedures are covered. The final section, 4, covers summary and conclusions.

REAL TIME SYSTEM

Real-time applications usually are executed on top of a Real- time Operating System .scheduling procedures are the rule set that defines that how I manage the real-time system in the scheduler, that is, how processor-time is allotted to the task present in any queue. The choice of algorithm depends on whether our system base is uniprocessor, multiprocessor or distributed.

A *uniprocessor* system executes only one process at a time and is capable of switching between processes, due to this reason context switching add some more time to the overall execution time when I preempt the process.

Multiprocessing is the use of two or more central processing units (CPUs) within a single computer system. A multiprocessor system will range from multi-core, essentially several uniprocessors in one processor, to several separate uniprocessors controlling the same system.

A distributed system will range from a geographically dispersed system to several processors on the same board. In a distributed system the nodes are autonomous while in a

In real-time systems processes are referred to as tasks and these have certain temporal qualities and restrictions. First of all a real-time task is a task like any other. HoIver, there is essential difference to other computation: the notion of time. Maintaining the Integrity of the Specifications

Each of the tasks will have a deadline, an execution time and a release time. In addition there are other temporal attributes that may be assigned to a task. The three mentioned are the basic ones. The release time, or ready time is when the task is made ready for execution. The deadline is when a given task must be done executing and the execution time is how long time it takes to run the given task. In addition most tasks are recurring and have a period in which it executes. Such a task is referred to as periodic. The period is the time from when a task may start until when the next instance of the same task may start and the length of the period of a task is static.

An example, shown in Figure 1, of scheduling can be made using three tasks T1, T2 and T3 with execution time and deadline of (1, 3), (4, 9) and (2, 9) respectively and periods equal to their deadlines. These tasks can be scheduled so that all tasks get to execute before the deadlines.

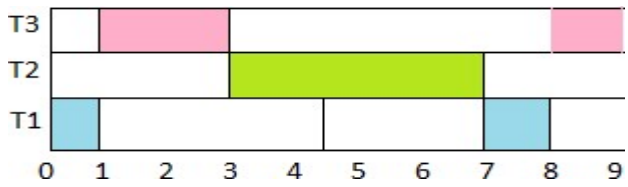


Figure 1: Scheduling of T1, T2 and T3.

The example is very simple as it does not show priorities or use preemption. There are also other properties of interest when looking at scheduling. Properties a task may use briefly explained:

- **Release/ready time:** The time a task is ready to run and just waits for the scheduler to activate it.
- **Deadline:** The time when a task must be finished executing.
- **Execution/run time:** The active computation time a tasks need to complete.

SCHEDULING PROCEDURES

A. STATIC SCHEDULING

Scheduling procedures themselves can be categorized as being static or dynamic.[4] The static scheduling procedures are those procedures which come under uniprocessors. The tasks present here have enough execution time and are ensured to fulfill the condition of deadline if possible. It calculates (or pre-determines) schedules for the system. Static approach requires prior knowledge of the process characteristics in order to process it in particular time. Certainly in safety critical systems it is reasonable to argue that no event should be unpredicted and that schedulability should be guaranteed before execution. This implies the use of a static scheduling algorithm. When all the scheduling decisions are made prior to the running of the system then it is static and offline. A table is generated which carry the scheduling decisions which are to be used during run-time.

***Rate monotonic* (RM)** is a scheduling algorithm [5,6] used in real time operating systems with static priority preemptive scheme. It is static-priority in the sense that all priorities are determined for all instances of tasks before run time. The length of the period of respective tasks determines the priority of a task. Tasks with short period times are assigned higher priority. Periodic tasks are scheduled using RM. The following are preconditions for the rate monotonic algorithm formalized by Liu and Layland.

1. Periodic tasks have constant known execution times and are ready for execution at the beginning of each period(T).
2. Deadlines(D) for tasks are at the end of each period: ($D = T$)
3. The tasks are independent, that is, there is no precedence between tasks and they do not block each other.
4. Scheduling overhead due to context switches and swapping etc. are assumed to be zero.

The ***rate monotonic*** priority assignment is *optimal* meaning that if any *static priority* scheduling algorithm can meet all the deadlines, then the *rate monotonic* algorithm can too. The Utilization For the given process is obtained by the given formula which was proposed by Lui & Layland(1973)[8] which is as:

$$U = \sum_{i=1}^{n} \frac{C_i}{T_i} \leq n(2^{1/n} - 1) \quad \text{.....(1)}$$

Where Ci=Computation Time, Ti=Release Time Period, N=No. of processes to be Scheduled.

An Example for rate monotonic is explained as follows:

TABLE 1: Process Timing

Process	Execution Time	Period
P1	3	7
P2	4	9
P3	6	10

The utilization for the given processes present in table 2 will be solved by the given formula:

The Utilization will be: 3/7 + 4/9 + 6/10 = 0.6492.
With the help of this utilization time I conclude the feasibility of the algorithm.

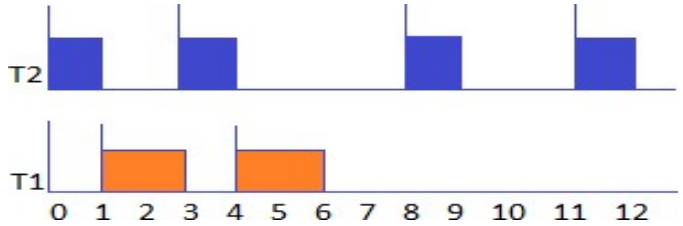
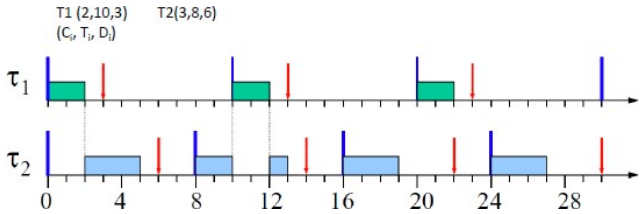


Figure 2: *Scheduling example of Rate-Monotonic*

One more example for RM is shown below which shows the two different tasks T1 and T2 are shown with their execution time $T1$ with a shorter period & therefore higher priority runs before $T2$. They then run as they are release.

Deadline Monotonic is a scheduling algorithm is an algorithm that uses fixed priority preemptive scheduling. The tasks in this procedures assigned according to the deadline of the given processes are assigned according to the given deadline. The task having the shortest deadline is assigned with the highest priority. Each task is assigned a priority inversely proportional to its relative Deadline. Deadline monotonic priority assignment is **not** optimal for fixed priority non-pre-emptive scheduling.

An Example that shows the feasibility for deadline



can be shown by the example below:

Figure 3: ***Scheduling example for deadline monotonic.***

Can I derive utilization based tests with the Given Formula:

$$U = \Sigma_{j=1}^{n} C_i / D_i \quad \text{.....(2)}$$

Here Di=Deadline of the task, Ci=Computation Time, N=no. of process to schedule.

B. DYNAMIC SCHEDULING:

Dynamic Scheduling procedures are those procedures in which the priorities to the processes are given to them at the time of execution of the task. The main

moto behind this is to adapt to dynamically changing progress of processes and form an optimal configuration for tasks in a self-sustained manner. Earlier Deadline First (EDF)[8] is a dynamic priority driven scheduling algorithm in which the priorities of tasks is based on the given deadline uses the simple system model given earlier. Like the RM algorithm, a preemptive priority-based scheduling scheme is assumed. The algorithm used for scheduling is: the process with the (current) closest deadline is assigned the highest priority in the system and therefore executes. The schedulability constant is given by:

$$\sum_{i=1}^{n} \frac{C_i}{P_i} \leq 1 \qquad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots(3)$$

For example let us Consider 3 periodic processes scheduled on a preemptive uniprocessor. The execution times and periods are as shown in the following table:

TABLE 2: Process Timing.

Process	Execution Time	Period
P1	1	8
P2	2	5
P3	4	10

Least laxity first (LLF) a process is defined as the deadline minus remaining computation time. With the least laxity approach[9], the schedulability constraint is again given by equation above(figure 3). It assigns priority based on the laxity. The smaller the laxity value of a task is, the sooner it needs to be executed. Its most common use is in embedded system. When two or more tasks have same or approximate laxity values, LLF scheduling algorithm leads to frequent switches among tasks, causes extra overhead in a system, and therefore, restricts its application.

This scheduling algorithm first selects those processes that have the smallest "slack time". Slack time is defined as the temporal difference between the deadline, the ready time and the run time.
More formally, the *slack time* for a process is defined as: $(d-t)-c'$

where d is the process deadline, t is the real time since the cycle start, and c' is the remaining computation time.

Multiprocessor systems are the future as I see it now, but finding procedures that takes full advantage of these systems is an arduous task in which much effort has been and is being made by researchers. Future work could be to focus on these new procedures being produced as Ill as dynamic based server procedures.

CONCLUSION

As I have studied and analyzed the various procedures based on static scheduling I discuss that rate monotonic and deadline monotonic scheduling are two procedures which are used for real time task system which are periodic. In this paper I discuss the feasibility decision for the given real time tasks when the system is scheduled using rate monotonic and deadline monotonic scheduling. The complexity of both the procedures depends on the number of tasks and the maximum periods given or on the deadlines of the given processes. The time complexity for the particular algorithm depends on the number of task. I come to a conclusion that the rate monotonic is more feasible as compared to the deadline monotonic algorithm as the priorities for rate monotonic are based on the process timing and for the deadline monotonic it is based on the deadline of each process which is preempted if higher priority task comes in between.

ACKNOWLEDGMENT

I am using this opportunity to express my gratitude to everyone who supported me in research analysis for the given topic that is Investigative study on scheduling procedures for real time operating system I express my thank to Department of Computer science & Engineering.

I also express my warm thanks to project Incharge/guide Miss Shipra Rathore Asst. Professor Department Of Computer Science & Engineering, Kalinga University, Raipur (C.G.) India for the guidance, inspiration and constructive suggestions that helpful me in the preparation and execution of this project.

REFERENCES

1. Burns A.,"Scheduling hard real-time systems: a review"Software Engineering Journal, May 1991.
2. Burns A. and Audsley N., "REAL-TIME SYSTEM SCHEDULING" Predicatably Dependable Computer Systems, Volume 2, Chapter 2, Part II. or Department of Computer Science, University of York, UK.
3. Mohammadi A. and Akl S. G., "Scheduling Procedures for Real-Time Systems", Technical Report No. 2005-499, School of Computing, Queen's University Kingston, Ontario Canada K7L 3N6, July 15, 2005.
4. S.Cheng,J.A.Stankovicand K. Ramamritham, ''SchedulingProcedures forHard RealTime Systems: ABrief Survey'',pp.150-173 inHard Real-Time Systems: Tutorial, ed.
5. Liu C.L. and Layland J.W., "Scheduling Procedures for Multiprogramming in a Hard-Real-Time Environment" Journal of the Association for Computing Machinery vol. 20, no. 1, pp. 46-61., year 1973.
6. Strosnider J. K., Lehoczky J. P. and Sha L., "The Deferrable Sernnver Algorithm for Enhanced Aperiodic Responsiveness in Hard Real-Time Environments", IEEE Transactions on Computers, vol. 44, no. 1, January 1995.
7. Leung J. Y.-T., Whitehead J., "On the complexity of ixed priority scheduling of periodic, real-time tasks", Performance Evaluation, vol. 2, issue 4, pages 237-250, December 1982..
8. C.L. Liu and J.W.Layland, ''Scheduling Proceduresfor Multiprogramming in a Hard.
9. A.K. Mok and M.L. Dertouzos,''Multiprocessor Scheduling in a Hard Real- Time Environment'', inProc. 7th TexasConf.Comput .Syst. (November 1978).
10. Liu, C. L.; Layland, J. (1973), *Scheduling procedures for multiprogramming in a hard real-time environment*, *Journal of the ACM* **20** (1):46– 61, doi:10.1145/321738.3217