INTERNATIONAL JOURNAL OF RESEARCH IN COMPUTER APPLICATIONS AND ROBOTICS
ISSN 2320-7345

AN OVERVIEW AND STUDY OF VARIOUS UNIPROCESSOR REAL TIME SCHEDULING ALGORITHMS

Jaitsri Kaur[1], R.S Uppal[2]

[1]*Jaitsri Kaur student BBSBEC, Jaitsri24@gmail.com*
[2] *Prof. R.S Uppal BBSBEC, RS.uppal@bbsbec.ac.in*

Abstract

Real time systems have been an active topic of research since the late 1960. The most important factor in real time scheduling is Time. Real time system supports the applications that meet their deadlines. In real time system the correctness of the system depends **o**n their temporal aspects as well as their functional aspects. A missed deadline can create a catastrophic fault in the system when talking about a hard real time system whereas in soft real time system it can lead to certain loss .Thus there are different scheduling algorithms to deal to above said, classified as Static & Dynamic Scheduling Algorithms. Thus, In this paper an Overview of different scheduling algorithm of a uniprocessor system are illustrated.

Keywords*:* Real Time scheduling algorithm, Deadline, Utilization, Laxity.

1. Introduction

REAL-TIME systems are playing a crucial role in our society. There has been an explosive growth in the number of real-time systems being used in our daily lives and in industry production. Systems such as chemical and nuclear plant control, space missions, flight control systems, military systems, telecommunications, multimedia systems, and so on all make use of real-time technologies. The most important attribute of real-time systems is that the correctness of such systems depends on not only the computed results but also on the time at which results are produced. In other words, real-time systems
Have timing requirements that must be guaranteed. Scheduling and schedulability analysis enables these guarantees to be provided. [1]

Definition: Real– time system are defined as those systems in which the correctness of the system does not depend only on the logical results of computations but also on the time at which the results are produced [2] So correctness for real-time systems is not only to react to changes; it also depends on with what action and at what time the event has been treated. A common mistake people make is to believe that fast systems are the same as real-time systems. The goal for fast systems is that for a given set of activities; try to minimize the average response time, which is NOT the same goal as for real-time systems. There are basically two kinds of Real time systems: Hard Real time System and Real time system.

In Hard Real-Time System requires that fixed deadlines must be met otherwise disastrous situation may arise whereas in Soft Real-Time System, missing an occasional deadline is undesirable, but nevertheless tolerable. System in which performance is degraded but not destroyed by failure to meet response time constraints is called soft real time systems.[3] The Basic Objective Of real time System is to have a scheduler which schedules the tasks of the system in a way that the timing constraints are met. Thus Clearly, a real-time operating system must be able to perform integrated CPU scheduling and resource allocation so that collections of cooperating tasks can obtain the resources they need, at the right time, in order to meet timing constraints. In addition to proper scheduling algorithms, predictability requires bounded operating system primitives.

II Real time scheduling Theory:

The real-time scheduling theory is a framework which provides:
1. Algorithms to share a processor (or any resources) by a set of tasks (or any resource users) according to some timing requirements equivalent to take urgency of the tasks into account.
2. Analytical methods, called feasibility tests or schedulability tests, which allow a system designer to early analyse/"compute" the system behaviour before implementation/execution time Different kinds of real-time schedulers:

1. On-line/off-line scheduler: the scheduling is computed before or at execution time?
2. Static/dynamic priority scheduler: priorities may change at execution time?
3. Pre-emptive or non-pre-emptive scheduler: can we stop a task during its execution?
3.1. Pre-emptive schedulers are more efficient than non-pre-emptive schedulers (e.g. missed deadlines).
3.2 Non pre-emptive schedulers ease the sharing of resources.
3.3 Overhead due to context switches [4]

III.BASIC Terminology of Real time scheduling

SCHEDULING:
Basically, it's a timing plan for the tasks. One can say that a timing plan is possible if and only if every timing requirements (=deadlines) are met. Various algorithms for creating timing plans exist. A set of tasks *T* can be scheduled using some algorithm *A* if and only if *A* always generates a possible timing plan for *T*. In other words, no matter how we arrange the contents of *T*, we'll still receive a possible timing plan. The scheduling algorithm *A* is optimal if it always generates a possible timing plan if such plan exists. Being optimal does not guarantee the minimum execution time of the tasks, but it does guarantee that all deadlines are met (temporal correctivity).

Some important Definitions
Processes or tasks: *Tasks (processes) form the logical units of computation in a processor. A single application program will typically consist of many processes. Each process has a single thread of control*.
TYPES of tasks:

• **Hard and Soft Deadline Periodic Tasks.** A periodic task has a regular inter arrival time equal to its period and a deadline that coincides with the end of its current period. Periodic tasks usually have hard deadlines, but in some applications the deadlines can be soft.

• **Soft Deadline Aperiodic Tasks.** An aperiodic task is a stream of jobs arriving at irregular intervals. Soft deadline aperiodic tasks typically require a fast average response time.

• **Sporadic Tasks.** A sporadic task is an aperiodic task with a hard deadline and a *minimum* inter arrival time. Note that without a minimum inter arrival time restriction, it is impossible to guarantee that a sporadic task's deadline would always be met.

Critical Task: A task is said to be critical if the consequences of not meeting the deadline leads the system to a fatal fault, it can be catastrophic. Periodic tasks usually have deadlines, which belong to this category.

Scheduler: A scheduler provides an algorithm or policy for ordering the execution of the outstanding processes on the processor according to some pre-defined criteria. 17
Schedulers produce a schedule for a given set of processes. If a process set can be scheduled to meet given pre-conditions the process set is termed feasible.

Types of Different SCHEDULE

Valid Schedule: A valid schedule of a set of tasks is a schedule which satisfying the following properties:
- Each process can only start execution after its release time.
- All the precedence and resource usage constraints are satisfied.
- The total amount of processor time assigned to each task is equal to its maximum or actual execution time.

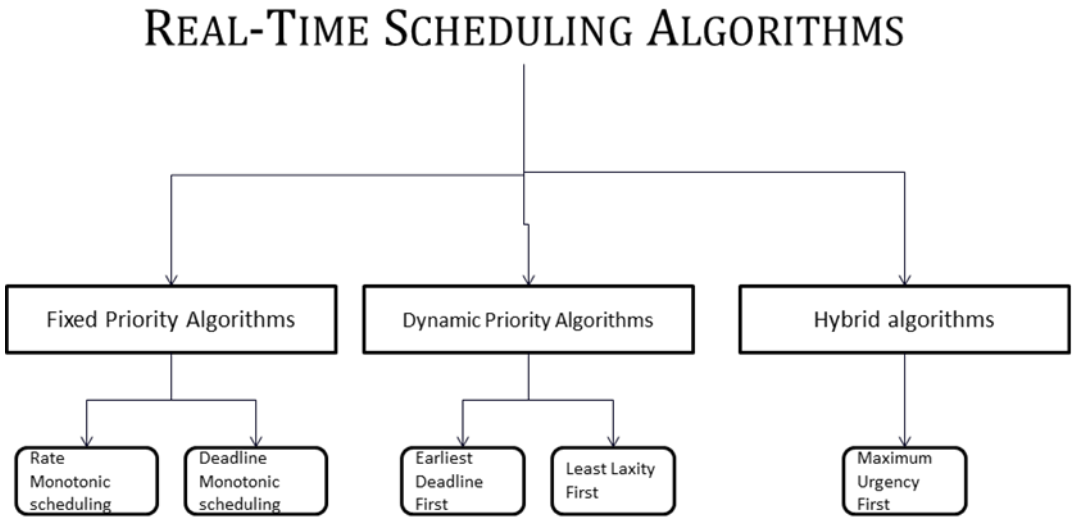
Feasible schedule: A feasible schedule of a set of tasks is a valid schedule by which every task completes by its deadline a set of tasks is schedulable according to a scheduling algorithm if the scheduler always produces a feasible schedule.

Optimal schedule: An optimal schedule of a set of tasks is valid schedule of set of tasks with minimal lateness. A hard real-time scheduling algorithm is optimal if the algorithm always produces a feasible schedule for a given set of tasks.

Efficient scheduling: Efficient task schedule is the one that minimizes the total completion time, or the schedule length, of the application.

Release time (or ready time): Time at which the task is ready for processing.
Deadline: Time by which execution of the task should be completed, after the task is released **Minimum delay:** Minimum amount of time that must elapse before the execution of the task is started, after the task is released.
Maximum delay: Maximum permitted amount of time that elapses before the execution of the task is started, after the task is released.
Worst case execution time: Maximum time taken to complete the task, after the task is released. The worst case execution time is also referred to as the worst case response time.

IV.SCHEDULING ALGORITHMS



IV.I Fixed Priority Algorithm:

These are those algorithms in which the priority of the task is freezed

Assumptions/properties of fixed priority scheduling:

a>Scheduling based on fixed priority =) static and critical applications.
b>Priorities are assigned at design time (off-line).
c>Efficient and simple feasibility tests.
d>Scheduler easy to implement into real-time operating systems.

RATE MONOTONIC:

The term rate monotonic derives from a method of assigning priorities to a set of processes as a monotonic function of their rates. [6] While rate monotonic scheduling systems use rate monotonic theory for actually scheduling sets of tasks, rate monotonic analysis can be used on tasks scheduled by many different systems to reason about schedulability. We say that a task is schedulable if the sum of its pre-emption, execution, and blocking is less than its deadline.[5] A system is schedulable if all tasks meet their deadlines. Rate monotonic analysis provides a mathematical and scientific model for reasoning about schedulability. The RM algorithm is probably the most used priority assignment in real-time applications, because it is very easy to implement on top of those commercial kernels that do not support explicit timing constraints on the task set. Indeed, an RM scheduler can be implemented just by assigning each task a fixed priority level inversely proportional to its period.[7]

Assumptions

- Task switching is instantaneous.
- Tasks account for all execution time.
- Task interactions are not allowed.
- Tasks become ready to execute precisely at the beginning of their periods and relinquish the CPU only when execution is complete.
- Task deadlines are always at the start of the next period.
- Tasks with shorter periods are assigned higher priorities; the criticality of tasks is not considered.
- Task execution is always consistent with its rate monotonic priority: a lower priority task never executes when a higher priority task is ready to execute.

It is immediately obvious that some of these assumptions do not completely conform to actual systems. However, extensions to broaden these assumptions will be discussed later. The importance of these assumptions is that they allow reasoning with certainty about whether or not a set of tasks can be scheduled.[8]

Benefits

Given certain information about a particular set of tasks, under rate monotonic conditions, one can evaluate certain tests to understand whether or not those tasks can all meet their deadlines in a real time system. Because these values are known at design time and are monotonic, any analysis and scheduling can be done statically. Static scheduling is one advantage that the industry has a strong preference for in hard real-time applications. [6] This subsection will examine two schedulability tests that can be used under rate-monotonic assumptions.

Given the computation time, Ci, and period, Ti, for task i, its CPU utilization can be calculated with the following equation:

$U_i = C_i/T_i$

For rate monotonic scheduling, the processor utilization for n tasks has been shown to be the following:

$$U(n) = n(21/n - 1)$$

U(n) asymptotically converges to ln(2) or 69%, which is less efficient than some runtime schedulers such as earliest deadline, but again, there is a strong preference for static scheduling. [6] The utilization bound (UB) test allows schedulability analysis by comparing the calculated utilization for a set of tasks and comparing that total to the theoretical utilization for that number of tasks:

$$C1/T1 + \ldots + Cn/Tn <= U(n) = n(21/n - 1)$$

If this equality is satisfied, all of the tasks will always meet their deadlines. If the total utilization calculates to greater than 100%, the system will have scheduling problems. However, if the total utilization is between the utilization bound and 100%, the UB test is inconclusive and a more precise test must be used.

DRAWBACK OF RM

One drawback of the RM algorithm is that task priorities are defined by their periods. Sometimes, we must change the task priorities to ensure that all critical tasks get completed.
In the RM scheduling we cannot guarantee the schedulability of the tasks. However, in the average case, they are all RM-schedulable. The problem is how to arrange matters so that all the critical tasks meet their deadlines under the RM algorithm even in the worst case, while the noncritical tasks, such as Ti, meet their deadlines in many other cases.

DEADLINE MONOTONIC:

We begin by observing that the processes we wish to schedule are characterised by the following relationship:

Computation time ≤ deadline ≤ period

Leung et al [8] have defined a priority assignment scheme that caters for processes with the above relationship. This is termed inverse-deadline or deadline monotonic priority assignment. Deadline monotonic priority ordering is similar in concept to rate monotonic priority ordering. Priorities assigned to processes are inversely proportional to the length of the deadline [8]. Thus, the process with the shortest deadline is assigned the highest priority, the longest deadline process the lowest priority. This priority ordering defaults to a rate monotonic ordering when period =deadline. Deadline monotonic priority assignment is an optimal static priority scheme for processes that share a critical instant. This is stated as Theorem 2.4 in [8].
"The inverse-deadline priority assignment is an optimal priority assignment for one processor." To generate a schedulability constraint for deadline monotonic scheduling the behaviour of processes released at a critical instant is fundamental: if all processes are proved to meet their deadlines during executions beginning at a critical instant these processes will always meet their deadlines [9, 8].

DMS: Schedulability analysis
 UB test (sufficient):
Σ Ci/Di <= n*(21/n-1) implies schedulable by DMS

IV.II Dynamic priority algorithm:

Dynamic priority scheduling consists of scheduling algorithm in which the priorities are calculated during the execution of the system. The goal of dynamic priority scheduling is to adapt to dynamically changing progress and form an optimal scheduling in self-sustained manner Examples are EDF & LLF.
EARLIEST DEADLINE FIRST

Assumptions and properties:

- Dynamic priority scheduler =) suitable for dynamic real-time systems.
- Is able to schedule both aperiodic and periodic tasks.
- Optimal scheduler: can reach 100 % of the cpu usage.
- But difficult to implement into a real-time operating system.
- Becomes unpredictable if the processor is over-loaded
- =)not suitable for hard-critical real-time systems

EDF Scheduling is proposed by Liu and Leyland [6]. This Scheduling Algorithm is based on uniprocessor dynamic-priority pre-emptive scheme and optimal among all dynamic priority scheduling algorithm. This Algorithm schedules the task with the earliest deadline first. The algorithm of EDF scheduling is following as;
Step 1: Set up all tasks' start time, end time, remaining time and deadline.
Step 2: If system is idle, then add task to schedule and go to step 4. Otherwise go to step 3.
Step 3: If new task's deadline is earlier than processing task's deadline, then update processing
Task's remaining time and exchange tasks. Otherwise update new task's start time.
Step 4: If all the tasks have not been scheduled, then go to step 2. Otherwise stop [11]

How does it work:

1. **Compute task priorities (called "dynamic deadlines")** =) Di (t) is the priority/dynamic deadline
Of task i at time t:
Aperiodic task: Di (t) = Di + Si.

Periodic task: Di (t) = k + Di, where k is the task release time before t.

2. **Select the task:** at any time, runs the ready task which has the shortest dynamic deadline.

Schedulability Test A task set of periodic tasks, with relative deadlines equal to periods, the task set is schedulable by EDF if and only if

- In fact, if U > 1 no algorithm can successfully schedule the task set.
- If U ≤ 1, then the task set is schedulable by EDF (and maybe by other n algorithms).

In EDF necessary condition should be satisfied, sufficient condition may or may not be satisfied.

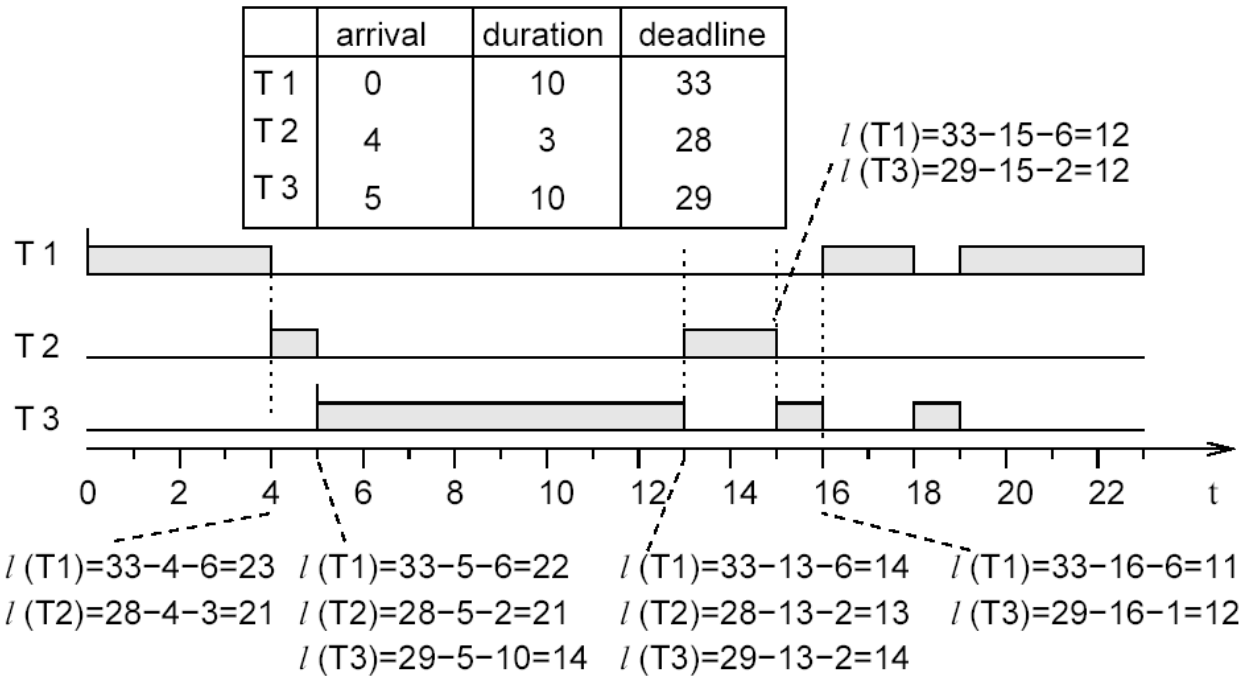
Least Laxity First Scheduling

Introduction

Laxity First (LLF) : David B. Stewart and Pradeep K. Khosla proposed this algorithm[12]. The laxity of a process is defined as the deadline minus remaining computation time. With the least laxity approach. The process which has the least laxity is assigned the highest priority in the system and is therefore executed. Whilst a process is executing it can be pre-empted by another whose laxity has decreased to below that of the running process. An executing process has constant laxity. A problem arises with this scheme when two processes have similar laxities. One process will run for a short while and then get pre-empted by the other and vice versa. Thus, many context switches occur in the lifetime of the processes. This can result in "thrashing", a term used in operating systems to indicate that the processor is spending more time performing context switches than useful work. The least laxity heuristic is optimal in the same way as the earliest deadline heuristic when the cost of context switching is ignored

Example: Consider three tasks under LLF Algorithm.

	arrival	duration	deadline
T 1	0	10	33
T 2	4	3	28
T 3	5	10	29



IV.III Hybrid Algorithm

Maximum Urgency First

Maximum Urgency First (MUF) scheduling algorithm resolves the problem of unpredictability of the system during transient overload that is when CPU load factor exceeds 100%. This algorithm is urgency based scheduling algorithm. It is a mixed priority scheduling algorithm and employs both fixed as well as dynamic priority for efficient scheduling of tasks. With this algorithm, each task is given an urgency which is defined as a combination of two fixed priorities (criticality and user priority) and a dynamic priority that is inversely proportional to the laxity. The critical priority is set to 1 if tasks are present in the critical set and the CPU load factor for these tasks is less than 100%.
Critical priority > dynamic priority > user priority
The MUF algorithm assigns priorities in two phases. Phase one is concerned with the assignment of static priorities to tasks. Static priorities are assigned once and do not change after the system . the MUF scheduling algorithm as mentioned in V.Salmani et.al paper is as follows
In phase 1, fixed priorities are defined and the scheduler sorts the task in increasing order of their periods. First N tasks having CPU utilization<100% are taken in critical tasks and the remaining tasks are considered in non-critical task set. Every task is given a optimal user priority that depends entirely on the user.
In phase 2, dynamic priorities are set.. If there is only 1 critical task the task is executed. If more than 1 critical task is there, the task with the minimum laxity is picked up for execution. If there are more than 1 tasks with the same laxity then the task with the highest user priority is considered and scheduled. [13]

V. Conclusion.

In this Article a review of Different scheduling algorithms for a real time system is done which specifies the working of these algorithms and their usage with the schedulability analysis. Real time operating Systems have been emerging in the field of research. A lot of work has been done in the field of real time scheduling algorithms so as to how to enhance the C.P.U utilisation and many different new algorithms have been developed to meet the utilisation of tasks within the cpu efficiently

VI. References:

[1] Fengxiang Zhang, Schedulability Analysis for Real-Time Systems with EDF Scheduling IEEE TRANSACTIONS ON COMPUTERS, VOL. 58, NO. XX, XX 2009.

[2].Jane W.S. Liu, *Real-Time Systems* , Pearson Education, India,pp. 121 & 26, 2001.

[3] M.Kaladevi, A Comparative Study of Scheduling Algorithms for Real Time Task, *International Journal of Advances in Science and Technology, Vol. 1, No. 4, 2010*

[4] University of Brest

[5] Obenza, Ray, and Mendal, Geoff. *Guaranteeing Real Time Performance Using RMA*, The Embedded Systems Conference, San Jose, CA, 1998.

[6] Sha, Lui, Klein, Mark H., and Goodenough, John B. *Rate Monotonic Analysis*, Technical Report CMU/SEI-91-TR-6 ESD-91-TR-6, March 1991

[7] GIORGIO C. BUTTAZZO, Rate Monotonic vs. EDF: Judgment Day, _c 2005 Springer Science + Business Media, Inc. Manufactured in The Netherlands

[8] Leu82a. J. Y. T. Leung and J. Whitehead, ‘‘On the Complexity of Fixed-Priority Scheduling of Periodic, Real-Time Tasks’’, Performance Evaluation (Netherlands) 2(4), pp. 237-250 (December 1982)
.
[9] Liu73a. C. L. Liu and J. W. Layland, ‘‘Scheduling Algorithms for Multiprogramming in a Hard Real-TimeEnvironment’’, Journal of the ACM 20(1), pp. 40-61 (1973).

[10] Yoo, Myungryun and M. Gen, “Study on Scheduling for Real- time Task by Hybrid Multiobjective Genetic Algorithm” , Thesis,2006.

[11] Swati Pandit, **Survey of Real Time Scheduling Algorithms ,** *IOSR Journal of Computer Engineering (IOSR-JCE)e-ISSN: 2278-0661, p- ISSN: 2278-8727 Volume 13, Issue 2 (Jul. - Aug. 2013), PP 44-51.*

[12] David B. Stewart, Pradeep Khosla, “Real-Time Scheduling of Sensor-Based Control Systems”, 1991

[13] K. K. Sahu, Sourav Mishra, Satya S. Sahoo, Upgraded Maximum Urgency First Algorithm For Real Time Systems, INTERNATIONAL JOURNAL OF SCIENTIFIC & TECHNOLOGY RESEARCH VOLUME 2, ISSUE 1, JANUARY 2013.