October 2013

REAL-TIME SCHEDULING POLICY IN EMBEDDED SYSTEM DOMAIN: A FRAME WORK

ASHIS KUMAR MISHRA

College of Engineering and Technology, Department of Computer Science. & Engineering, BBSR,
ashiskumar.misra@gmail.com

ZULFIKHAR AHMAD

College of Engineering and Technology, Department of Computer Science. & Engineering, BBSR,
ahmad_mca@yahoo.co.in

Follow this and additional works at: <https://www.interscience.in/ijcsi>

Part of the [Computer Engineering Commons](#), [Information Security Commons](#), and the [Systems and Communications Commons](#)

Recommended Citation

MISHRA, ASHIS KUMAR and AHMAD, ZULFIKHAR (2013) "REAL-TIME SCHEDULING POLICY IN EMBEDDED SYSTEM DOMAIN: A FRAME WORK," *International Journal of Computer Science and Informatics*: Vol. 3 : Iss. 2 , Article 10.

DOI: 10.47893/IJCSI.2013.1130

Available at: <https://www.interscience.in/ijcsi/vol3/iss2/10>

This Article is brought to you for free and open access by the Interscience Journals at Interscience Research Network. It has been accepted for inclusion in International Journal of Computer Science and Informatics by an authorized editor of Interscience Research Network. For more information, please contact sritampatnaik@gmail.com.

REAL-TIME SCHEDULING POLICY IN EMBEDDED SYSTEM DOMAIN: A FRAME WORK

YOGAMAYA MOHAPATRA¹, ASHIS KUMAR MISHRA², ZULFIKHAR AHMAD³ &
ANIL KUMAR MISHRA⁴

^{1,4}Orissa Engineering College, Department of Computer Science & Engineering, BBSR

^{2,3}College of Engineering and Technology, Department of Computer Science. & Engineering, BBSR
Email: ashiskumar.misra@gmail.com,ahmad_mca@yahoo.co.in

Abstract: Scheduling a sequence of jobs released over time when the processing time of a job is only known at its completion is a classical problem in CPU scheduling in time-sharing and real time operating system. We discuss here different scheduling techniques used in Real-Time systems. Even if there are several scheduling policies, the preemptive scheduling policies hold promising results. In this paper we have done an extensive survey on various scheduling algorithms. We are extracting the positive characteristics of each scheduling and placed it on this paper.

Key words: RTOS, round robin, edf, fcfs, sjin, deadline, rms, preemption.

1. INTRODUCTION

The purpose of a task scheduling is to organize the set of tasks ready for execution by the processor more precisely, to organize them so that performance objective is met. Thus it is essential an optimization problem. The order of arrangement of tasks are called schedule. A schedule can be a feasible or optimal: A valid schedule is called a feasible schedule, if all the tasks meet their respective time constraints in the schedule. A real-time task scheduler is called optimal, if it can feasibly schedule any task set that can be feasibly scheduled by other scheduler. Scheduling real-time tasks is an extremely important activity in real-time systems as this is the ultimate factor that governs the final temporal properties of tasks. The problem is of allocating the tasks to computation resources which may be the CPU, memory, communication channels or I/O devices. The model most often used in representing the scheduling problem reflects an allocation of processes to processors and objective of scheduling algorithm. This objective function may vary with application. For real-time systems it usual takes of the form that task must finish within stipulated deadline. Formally, we define the set of processes and processors as

follows. A set of processes $V_p = (p_1, p_2, \dots, p_n)$, are related to each other through a set of logical links E_p to form a graph $G_p = (V_p, E_p)$. A set of processors $G_q = (V_q, E_q)$. Allocating processes to processors is function $F: V_p \rightarrow V_q$. (1) Task scheduling in real-time systems can be static or dynamic. A static approach calculates schedules for tasks off-line and it requires a complete prior knowledge of task's characteristics. A dynamic approach determines schedules for tasks on the fly and allows the tasks to be dynamically invoked. (2) Real-Time tasks can be of two types: periodic and aperiodic. Periodic tasks are those which recur with a regular time interval e.g. a transducer like thermocouple to measure temperature of a process at regular intervals. Aperiodic tasks are associated with asynchronous events like occurrence of an alarm event due to some parameter of the controlled physical system going above the threshold.

2. LITERATURE REVIEW

Conventional Scheduling strategies like First come First Served (FCFS) or Round Robin cannot be used in real-time systems because they do not take into account the importance of task characteristics like

deadline. Some important scheduling strategies used in real-time systems are discussed below.

Heuristic Scheduling this policy is often called “static priority scheduling”. It proceeds from the assumptions that each task has associated a fixed (static) priority. This defines its importance for scheduling application. Tasks are connected in order of priority in the ready list, the highest priority job will be on the top. This is preemptive policy, thus at a reschedule time the running task will be preempted if a higher priority task is ready. Task importance is evaluated heuristically by application designer.. This policy is simple and easy to use and generally effective and is used in commercial real-time operating system like RMK, VRTX, VxWORKS and Venix.

Rate Monotonic Scheduling (RMS) the policy introduced by Liu and Leyland [1] considers a single task criterion. RMS is an event driven scheduling algorithm. This is a static priority algorithm and is extensively used in practical applications. The lower occurrence rate of a task, the lower priority is assigned to it. A task having highest occurrence rate (lowest period) is accorded highest priority. RMS has been proved to be optimal static priority scheduling algorithm.

Necessary conditions: A set of periodic real-time task would not be RMS schedulable unless they satisfy the following necessary condition

$$\sum_{i=1}^n e_i / p_i = \sum_{i=1}^n u_i \leq 1 \quad (1)$$

Where e_i is the worst case execution time and p_i is the period of task T_i , n is the number of tasks to be scheduled and u_i is the CPU utilization due to the task T_i . This test simply expresses the fact that the total CPU utilization due to the task T_i . This test simply expresses the fact that total CPU utilization due to all tasks in the task set should be less than 1.

Sufficient conditions: The derivation of the sufficiency conditions for RMS is an important result and was obtained by Liu and Layland in 1973. A set of n real-time periodic tasks are schedulable under RMS,

$$\sum_{i=1}^n u_i \leq n(2^{1/n} - 1) \quad (2)$$

Where u_i is the utilization due to the task T_i . As $n \rightarrow \infty$, the utilization bound $\rightarrow 0.693$. This has said led to the simple rule of thumb that says that “if the CPU utilization is less than 69%, then all deadlines are met”[1].

Earliest Dead line First (EDF) Scheduling

In this scheduling strategy, priority is defined using a single criterion, time to deadline (task deadline). A task will be assigned the highest scheduling priority if its current deadline is the earliest(nearest) and placed in the front of the ready queue. It should be clear that deadline values change during the program execution. This algorithm belongs to a class of dynamic policies. This scheme is also known as earliest deadline as soon as possible scheduling policy. There is another scheduling scheme known as Least Laxity First (LLF). When invoked an EDF Scheduler simply scans through all the tasks in the system and dispatches the one with the earliest deadline. The difference between the remaining execution time of a task and its remaining time is the laxity. The LLF scheduler dispatches the task and its remaining time to deadline is its laxity. The LLF scheduler dispatches the task with the smallest laxity. CPU load (also known as processor utilization factor) is defined as:

$$U = \sum_{i=1}^n C_i / T_i \quad (3)$$

Rate Monotonic Scheduling- a hard real-time scheduling algorithm- can guarantee time restraints only up to 70% CPU load. Beyond that it does not support dynamic systems very well. In addition to schedulable bounds that are less than 1.0, two problems exist for RMS algorithms provide no support for dynamically changing task periods or priorities and task may experience task inversion. The first problem can be resolved by considering the fixed priority scheduling of periodic task with varying task execution priorities. Specifically task may have subtasks of various priorities. Specifically tasks may have subtasks of various priorities. Priority inversion arises when a high priority task must wait

for a lower priority task to execute, typically due to other resources being used by executing tasks. i.e. tasks waiting on critical selection.[3]This implies applications have to state their run-time requirements beforehand – how often they must be called in a second, which maximum response time is acceptable etc. All this information must be provided by the application programmer. On the other hand, with the earliest deadline first (EDF) and minimum-laxity-first (MLF) dynamic scheduling algorithm, a transient overload in the system may cause a critical task to fail, which is certainly undesirable. *The maximum-urgency-first* (MUF) combines the advantages of RM, EDF and MLF[3]. Like EDF and MLF, MUF has a schedulable bound of 100% for the critical state. And like RMS, a critical set can be defined that is guaranteed to meet all its deadlines. The MUF algorithm also allows the scheduler to detect forms of deadline failure handler routines for tasks, which fail to need their deadlines. In this perspective the present work was undertaken-to design an efficient algorithm for scheduling soft real-time tasks in a real-time embedded system. And run the algorithm on a simulated embedded environment.

3. PROBLEM DEFINITION

The main aim is to study the policy mechanisms of different real time schedulers in embedded domain, evaluation of performance mechanism to arrive at a common solution. The main problem is the improvements in RR (ROUND ROBIN) algorithm. And how it will be suitable for real time embedded system domain.

4. METHODOLOGY

Since an embedded real-time system is not available to test the working of the scheduler. The embedded real-time environment is simulated using RED-HAT

LINUX platform by using c and REDHAT LINUX.

For this we have to depend three major functionalities of the LINUX kernel 1)System Timer (2) Job response time. (3)Kernel preemption.

(1)System Timer: In time-sharing systems, an operating system uses a periodic timer to divide the CPU time among all the jobs. By selecting a proper timer frequency to define the time slice, OS may achieve a good balance between the job responsiveness and context switching over head. Depending on the system architecture, the period of the timer will be decided. (2)Job response Time: In addition to a timer resolution, a real-time kernel also needs to provide a short job response time. In our discussion, the job response time is defined to the interval between an occurrence (e.g. device signal, periodic job arrival etc.) and the start time of a job execution in response to the event (e.g. interrupt service, periodic job response etc.). It has been referred to as the task dispatch latency. In general, the job response time includes the following components. Interrupt dispatch time (IDT): When an interrupt occurs, a system must save all registers and other system execution status before calling the interrupt service routine to handle it.

Interrupt service time (IST): The time used by the interrupt service routine to retrieve information from the hardware device or to gather information from the system.

Kernel preemption time: The time to preempt the current user job. If the job is running on user mode, KPT is zero since the preemption may happen

immediately. If the user is running on the kernel mode, KPT is the time before it exits the kernel mode. Scheduling delay time (SDT): The time used by the scheduler to select the next user job in response to interrupt. Context switching time (CST): The time used to save registers and status of current job, and also reset registers and the status of next job. (3) Kernel preemption: To reduce the job response time, we must also improve the kernel preemption to reduce the KPT. Otherwise a low priority job can block another higher priority job/task for a long time staying in the kernel mode. Two different approaches are possible to preempt a job running on kernel mode. The first is the full preemption model and the other is the cooperative preemption model. We will discuss it later in the implementation part.

4.1 Proposed Algorithm to calculate the time slice

1. Algorithm Time Slice (P, T)
2. // N=P.length represents the no. of processes
3. // P[1..N] is the array containing the priority of N no. of processes.
4. // T[1..N] is the array containing the CPU burst time of N no. of processes
5. // TS [1..N] is the array that will contain the time slice for individual processes.
6. $\text{Range} = (\max(T) + \min(T)) / 2$
7. // $\max(T)$ returns the maximum CPU burst time
8. // $\min[T]$ returns the minimum CPU burst time
9. for $i=1$ to P.length
10. $\text{TS}[i] = (\text{Range} * \text{P.length}) / (\text{P}[i] * \text{T.length})$
11. return TS

Figure 4.1.1

The case study that provides the clear idea about the proposed algorithm and it also shows how the time slice calculated for individual processes.

Input Component for Processor

Process Id	CPU Burst time	Priority
1	25	2
2	5	3
3	15	1
4	8	5
5	10	4

Table 4.1.2 .Calculation of Time slice for proposed Algorithm

P.NO	CPU Burst	Pr	R	T.Process	T.Pr	Time Slice
1	25	2	15	5	5	8
2	5	3	15	5	5	5
3	15	1	15	5	5	15
4	8	5	15	5	5	3
5	10	4	15	5	5	4

Where Pr = Priority of process

R = Range

T.Pr = total no. of processes in the system

T.Pr = Total no. of priority in the system

Time slice = $(R \times N) / (Pr \times P)$

DEVELOPMENT OF A SIMULATOR USING LINUX:

This section describes the development of a proposed simulator in Linux environment. A framework for

evaluation of scheduling algorithm must satisfy characteristics such as simplicity, compatibility with pc platform usage of the standard operating system functions, accuracy of results, ease of use etc. Majority of these requests are aimed for use in the visual user interface. Scheduling algorithm evaluation and analysis tool performs the task definition, task sets generation, execution of selected algorithms, execution analysis of the execution and the results are displayed.

CONCLUSION

The proposed Linux framework gives the developer the possibility to evaluate the schedulability of real-time application. The GUI of the framework will allow for easy comparisons of the framework of existing scheduling policies and also simulate the behavior and verify the suitability of custom defined schedulers for real-time applications. The scheduler co-processor hardware can help the learner have a closer view of the scheduling tasks in real-time hardware.

REFERENCES

- [1] M.V. Panduranga Rao, K.C. Shet, R.Balkrishna, K.Roop (2008) "Development of Scheduler for Real Time Embedded System Domain", 22nd International conference on Advance Information Networking and Application Workshops.
- [2] Arnoldo Diaz, Ruben Batista and Oskardie Castro (ICEEE 2007) "Realts: a real-time scheduling simulator", 2007 4th International Conference on Electrical and Electronics Engineering.
- [3] Jwen Dong, Yang Zhang (ICMI 2009) "A modified Rate-Monotonic Algorithm for scheduling of tasks with Different Importance in Embedded System", The Ninth International Conference on Electronic Measurement and Instruments.
- [4] C.Yaashuwant, Dr.R.Ramesh (IJSIS 2009) "A New Scheduling Algorithms, International Journal of Computer Science and Information Security vol. 6, No.2, 2009.
- [5] Insop Song, Sehjeong Kim, Fakhreddine Karray "A Real-Time Scheduler Design for a class of Embedded Systems", IEEE/ASMETRANCTIONS ON MECHATRONICS, VPL.13,NO.1, FEBRUARY 2008.

- [6] M.V. Panduranga Rao, K.C. Shet "A Research in Real Time Scheduling Policy for Embedded System Domain", CLEI ELECTRONIC JOURNAL, VOL12, NUMBER 2, PAPER 4, AUGUST 2009.
- [7] I.L. Hellerstein, Y.Diao, S.Parekh, and D.M. Tilbury, "Feedback control of computing system", New York: IEEE press/Wiley/Interscience, 2004.
- [8] L.Sha, T.Abdelzaher, K.Erek Arzen, A.Cervin, T.Baker, A. Burns, G.Buttazzo, M. Caccamo, J.Lehoczky, and A.K Mok, "Real time scheduling theory", A historical perspective ", Real-time Syst. Vol-28, pp-101-155,2004.
- [9] X.Liu, and S.Goddard, "Supporting dynamic QOS in Linux", in proc. 10th IEEE Real-Time Embedded Technology Appl. Symp.(RTAS 2004), Toronto, Canada, 2008, pp. 246-254.
- [10] A.Goel, J.Walpole, and M.Shor, "Real-rate Scheduling", in proc. 10th IEEE Real-Time Embedded Technology Application Symp. (RTAS 2004), Toronto, Canada, pp, 434-441.