ISSN: 2278-4632
Vol-11 Issue-02 2021

REAL-TIME SCHEDULER DESIGN FOR EMBEDDED SYSTEM DOMAIN

R Subba Rao, Dr. Prakash Pathak[2]
[1](*Computer Science & Engineering, Gandhi Engineering College, BBSR)*
[2](*Computer Science & Engineering, Gandhi Engineering College,BBSR)*

Abstract: *The main objective of this paper is to develop a new scheduling algorithm for scheduling of task in Real-Time operating systems. The proposed architecture is a modified version of Round-Robin architecture which is used for scheduling of tasks in Real-Time operating systems. It is observed that the proposed architecture solves the drawbacks of simple Round-Robin architecture in Real-Time operating system by decreasing the number of context switches waiting time and response time thereby improving the system performance. This paper also explains the development of a new CLI simulation framework: to study and evaluate the performance of various uniporcessor real-time scheduling algorithm for Real-Time system. Task ID, Deadline, Priority, period, Computation time, and Phase are the input task attributes to the scheduler simulator and chronograph imitating the real-time execution of the input task set and computational statistics of the schedule are the output. The proposed framework for the scheduler simulator is mainly developed to be used as a teaching tool. The CLI deployment of the simulator enables the user a platform, machine and software-independent utilization of the technicalresource.*

Key words: RTOS, Round-Robin, EDF, FCFS, CLI, RMS, Preemption, MUF.

I. 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 stipulateddeadline.Formally,wedefinethesetof processes and processors as follows. A set of processes V_p = (p1, p2,…., pn), 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, Eq). Allocating processes to processors is function F: Vp-> 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 thethreshold.

II. 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 willbe 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(lowestperiod)isaccordedhighestpriority.RMShas been proved to be optimal static priority scheduling algorithm.

Copyright @ 2021 Authors

ISSN: 2278-4632
Vol-11 Issue-02 2021

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

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

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 is 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, if

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

Where u_i is the utilization due to the task $T_{i..}$. As n->∞, the utilization bound->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 aremet"[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. T his 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 smallestlaxity.

CPU load (also known as processor utilization factor) is defined as:

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

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 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 subtasksofvariouspriorities.Priorityinversionarises 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.

III. 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. A scheduler requires a time management function to implement the round robin architecture and requires the tick timer. The time slice is proportional to period of clock ticks. The time slice length is an critical issue in soft real time embedded application as missing of deadlines will have negligible effects in the system performance. The time slice must not be too small which results in frequent context switches and should be slightly greater than average process computationtime.

Copyright @ 2021 Authors

ISSN: 2278-4632
Vol-11 Issue-02 2021

IV. 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 bedecided.

(2) Job response Time: In addition to a timer resolution, a real-time kernel also needs to provide a short job response time.Inourdiscussion,thejobresponsetimeisdefinedto 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 followingcomponents.

110

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 implementationpart.

Proposed Algorithm to calculate the timeslice

1. Algorithm Time Slice (P,T)
2.// N=P.length represents the no. ofprocesses
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 individualprocesses.
6. Range= (max (T) +min(T))/2
7.// max (T) returns the maximum CPU burst time 8.//min[T] returns the minimum CPU bursttime
9. for i=1 to P.length
10. TS[i] = (Range*P.length)/(P[i]*T.length)
11. return TS

Proposed Architecture

Input Components: The input components are the processes and the priority. The inputs components will be allocated to the mini-processor.

Mini-Processor: The Mini-processor is a Kernel level Programming (logical Processor). It keeps track the Process-ID, Priority of each Process. It will calculate the range, time slice of each process.

Shared Memory: Shared Memory is method of Interprocess Communication (IPC) where two or more processes share a single chunk of memory to communicate.

The shared memory can also be used to set permission on memory. In this proposed model the shared memory stores all the calculated data computed by the Mini-Processor.

Main Processor: The main Processor will run all the processes that are being taken as the input. And the scheduling will take place according to the Round-Robin Algorithm.

Time Slice Calculation for Proposed Architecture:

Time slice = (R×N) / (Pr.×P)

Range = maximum CPU Burst + minimum CPU Burst / 2 Where Pr = Priority of Process

R= Range

T.Pr = Total no. of Processes in the system T.Pr = Total no. of Priority in thesystem

Copyright @ 2021 Authors

ISSN: 2278-4632
Vol-11 Issue-02 2021

Figure 4.2.1 ProposedArchitecture

Development of a simulator usingLinux:
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.

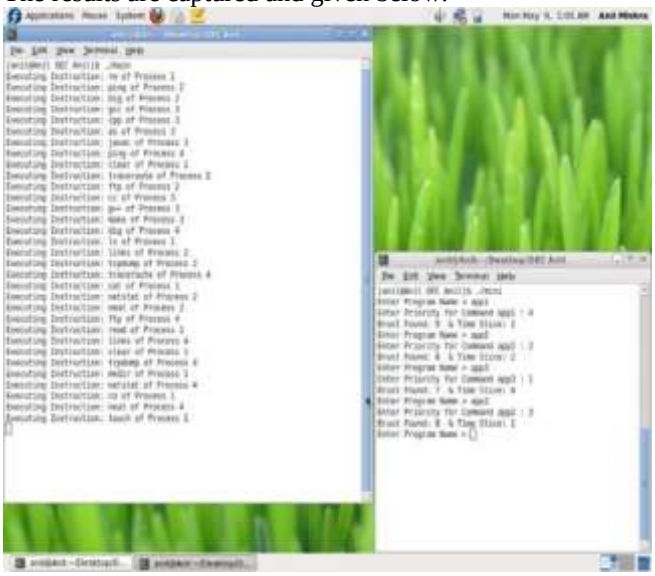
111 **V. IMPLEMENTATION**

The module-1 is designed to calculate the range and the CPU burst time of each processor. We have named it as Mini-processor. The module-2 consists of all the applications that are designed to run in the module-1. The shared memory is constructed to capture all the results produced by module-1. The module-3 is the program based on the logic of modified Round-Robin algorithm.

Experiments andResults

Assumptions: The environment where all the experiments performed is a single processor environment and all the processes are independent. The processes (applications) are designed and the time slice is calculated along with the burst time of the process according to the proposed algorithm. All the parameters like no of processes and priority are known. The burst time and time slice will be calculated. All the processes are CPUbound.

Experimental Framework: Our experiment consists of several input output parameters. The input consists of the Processes or Applications which is designed. The output parameter consists of time slice or time quantum.
Experiments Performed: To evaluate performance of our proposed algorithm. Four applications have been written and the Mini-Processor will calculate the burst time, time slice of each application. The results are captured and shown.
The results are captured and given below:

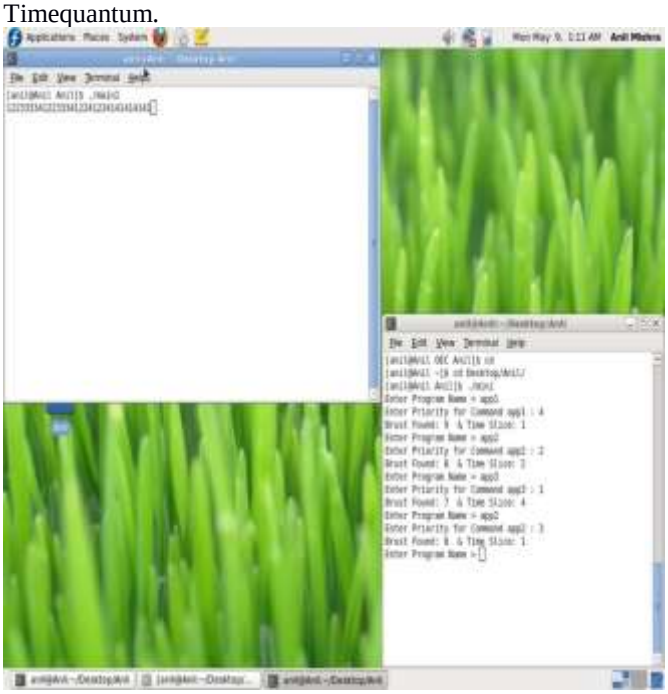


(1) Time Slice Calculation ofMini-Processor

Copyright @ 2021 Authors

ISSN: 2278-4632
Vol-11 Issue-02 2021

In this above results "Experimental Result1" represents the role of a Mini-Processor along with the modified RR is shown. It shows how the Mini-Processor calculates the CPU burst for each application along with its Time slice or Timequantum.



(2) Simulation for RRAlgorithm

The "Experiment Result2" represents the sequence in which the RR Scheduling will takeplace.

Comparison

Comparison with the earlier work has performed. Yaashuwant.C and Dr.R.Ramesh designed an architecture and algorithm for scheduling tasks in Real-Time operating systems. They have provided a web enabled framework. There exist three major differences from the earlier work. The First difference is the processor; in this algorithm the logical mini-Processor is proposed. But in the paper [3] a physical processor was proposed. The second difference is the shared memory where the results are stored each time the programs runs it removes the old data from the shared memory and inserts the new data. In the earlier work the shared memory concept was not invoked. And the third major difference is, the simulator that was designed we have to enter the input manually the output will come according to the formula the user will select. So, the implementation is not specific for Round-Robin scheduling. But the thing that is proposed by us is also implemented the same thing. Previously in the earlier work it was implemented in a web platform. But here it is implemented in Linux Platform with the accuracy. In the earlier work the Real-Time scheduler Co-Processor hardware gives closer view of scheduling.

VI. 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. From the above, comparisons and the test results our newly proposed architecture along with performs better. Then we arrive at a common solution to simulate parametric scheduling policy for real-time embedded system domain. It is also concluded that the proposed architecture is superior as it has less waiting and response time, usually less preemption and context switching therefore reducing the overhead and saving of memory space. Future work can be done on this architecture modification and algorithm for hard real-time systems where hard deadline system requires partial output to prevent the catastrophiceffect.

REFERENCES

[1] M.V. Panduranga Rao, K.C. Shet, R.Balkrshna, K.Roopa (2008) "Development of Scheduler for Real Time Embedded System Domain", 22nd International conference on Advance Information Networking and ApplicationWorkshops.

Copyright @ 2021 Authors

ISSN: 2278-4632
Vol-11 Issue-02 2021

[2] Arnoldo Diaz, Ruben Batista and Oskardie Castro (ICEEE 2007) "Realtss: a real-time scheduling simulator", 2007 4th International Conference on Electrical and ElectronicsEngineering.
[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 andInstruments.
[4] C.Yaashuwant, Dr.R.Ramesh (IJCSIS 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 EmbeddedSystems", IEEE/ASMETRANCTIONS ON MECHATRONICS, VPL.13, NO.1, FEBRUARY2008.
[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, AUGUST2009.
[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] [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] [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.
[11] [11] C.Yaashuwant, Dr.R. Ramesh (IJCSIS2010) "Design of Real-Time Scheduling Simulator and Development of Modified Round Robin Architecture", International Journal of Computer Science and Information Security vol. 10, No.3, 2010.

Copyright @ 2021 Authors