Web-Enabled Framework for Real-Time Scheduler Simulator (A Teaching Tool)

Yaashuwanth .C, IEEE Member
Department of Electrical and Electronics Engineering,
Anna University Chennai, Chennai 600 025
yaash_success@yahoo.co.in

Dr.R. Ramesh, Assistant Professor
Department of Electrical and Electronics Engineering,
Anna University Chennai, Chennai 600 025
rramesh@annauy\niv.edu

***Abstract*—The main objective is to develop a new Web-enabled simulation framework: to study and evaluate the performance of various uniprocessor real-time scheduling algorithms for real-time scheduling. 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. Evaluation of the performance of real-time schedulers can be done as well. Evaluation criteria are based on the mean response time, number of deadline misses, processor utilization, number of preemptions and context switches. The Web-based deployment of the simulator enables the user a platform-, machine- and software-independent utilization of the technical resource.**

Keywords- real-time, scheduler, simulator, pre emptions and context switch.

I. INTRODUCTION

A real-time computing system can be defined as a real-time application which is expected to respond to stimuli within some small upper bound on response time and any late result is as bad as a wrong one. Thus correctness of a real-time system could be stated true with logical perfection in the computational result and its timeliness. A soft real-time system is a system that has timing requirements, but occasionally missing the task deadlines have negligible effects. 'hard real-time should meet the timing requirements of system, computations must always be met or the system will fail. Determinism, guaranteed worst-case interrupt latency and guaranteed worst-case context switch times characterize real-time operating systems, the software employed in real-time systems. Given these characteristics and the relative priorities of tasks and interrupts in the system, it is possible to analyze the worst-case performance of the software and hence the real-time characteristics of the system.

II. . SCHEDULING: FROM REAL-TIME DESIGN PERSPECTIVE

The purpose of task scheduling is to organize the set of tasks ready for execution by the processor system so that performance objectives are met[3]. The order of these tasks is called a 'schedule'. For real-time embedded systems, the primary objective is to ensure that all tasks meet their deadlines. A schedule can be feasible or optimal: a feasible schedule orders tasks making them to meet all their deadlines; an optimal schedule is one which ensures that failures to meet task deadlines are minimized. The scheduler is responsible for coordinating the execution of several tasks on a processor. The scheduler may be preemptive or non-preemptive. The scheduler for hard real-time systems must coordinate resources to meet the timing constraints of the physical system which implies that the scheduler must be able to predict the execution behavior of all tasks within the system[4]. so the basic requirement of real-time systems is predictability. Unless the behavior of a real-time system is predictable, the scheduler cannot guarantee that the computation deadlines of the system will be met.

The requirement of predictability differentiates real-time systems from conventional computing environments and makes the scheduling solutions for conventional systems inappropriate for real-time systems.

The scheduling theory provides numerous schedulability tests for each scheduling policies and locking protocols used in real-time systems[6],[12],[13],[14].

These offer a way for programmers to predict, in advance, whether a multi-tasking design will meet its deadlines or not. Early work was limited to 'rate monotonic' task priorities with deadlines equal to periods, and used a notion of 'processor utilization' to assess schedulability. More recent works have extended schedulability analysis to apply to any fixed-priority scheduling policy, and to support arbitrary deadlines. These works test for schedulability by calculating the worst-case response time for each task. The schedulability analysis (evaluation criteria) of real-time task set will be discussed later in this paper.

III. NEED FOR SCHEDULE SIMULATOR

From design perspective, real-time systems can be approached from different views. As an example, engineers prefer to deal with hardware control while computer scientists prefer to deal with the system modeling. The system modeling will explain how to model task interactions and how to allocate processor time for each task. The system modeling is burdensome because there are many different scheduling policies and scheduling problem is known as a

978-0-7695-4043-6/10 $26.00 © 2010 IEEE
DOI 10.1109/ICCRD.2010.181

strongly complex one. As a consequence of complexity, most learners feel that the scheduling theory is only a collection of rules that have to be memorized [1]. Therefore, much of the attention is not paid to the fact that the most important concept is not the exact description of a rule but what kind of conditions and problems are better suited for each rule. The foresaid misunderstanding can be solved, assigning jobs that require not only the resolution of a schedule but the experimentation with the problem. Although this can be done by hand, it has limitations due to the exponential growth in the resolution time with the problem size. The use of simulation technique would thus help circumvent this issue.

IV. Study Of Schedulers And Simulators

A. Study of existing simulator

Real-time simulation tools speed up the decision making processes during the selection of suitable scheduling algorithm for a real-time embedded application. They also stand as teaching tool helping learners of real-time system grasp the core ideas related to system modeling quickly

There have been various simulator frameworks created for this purpose, too[2],[7],[15]. The performance analyses of the above mentioned simulator frameworks were carried out and the need for developing a new Web-based simulator framework was discovered.

The study of existing frameworks of simulation clearly reveals that each tool is better in its own way. Thus an appreciable combination of values of each of the tools is chosen and an earnest attempt has been made to make the proposed framework to be more flexible for the future users for trying different other combinations of evaluation criteria that may be of interest for different real-time resource capacities. The experimentation results obtained from the analyzed simulators were compared to form the reference data for functional verification of the tool under development...

B. Real-Time Schedulers

Unlike the conventional schedulers of the modern operating systems, which provide fairness to all the tasks/processes, the real-time schedulers are partial and work primarily on priorities/deadlines of the tasks. In such real-time systems, most of the timelines of the tasks are already known or the arrival of the tasks to the system is very much predictable Jan Blumenthal[7]. Hence most of the real-time systems implement static scheduling algorithms and are simple. But in some complex environments, dynamic scheduling is often required. Foundational description of various scheduling policies employed in uniprocessor real-time systems is presented here.

1) First-Come-First-Served: The FCFS policy is the simplest scheduling strategy to be found, as it does not invoke any task constraints. Ready-to-run tasks are organized in a list, that at the top being executed first [3]. When a task becomes ready, it is added to the end of the ready list. Thus tasks execute in the order in which they are readied - first come, first served. It is non- preemptive, each task being allowed to run to completion.

2) Simple Round-Robin: The RR policy is the pre-emptive version of the FCFS technique. Tasks are still arranged in the ready list in the order in which they become ready-to-run, but when they are set running they execute within a fixed time slice[9]. If a task is still running at the end of its time slice, it is forcibly removed from the processor. Its replacement is the task which was at the head of the ready queue and the preempted task is sent to the end of the ready queue. As the task set is executed, it gradually works its way to the top of the queue. When once more installed on the processor it resumes execution from the point of interruption.

3) Shortest-Job-First: The SJF policy is a static priority-based alternative to the FCFS scheduling strategy. It uses a single criterion in defining priority, task execution time. Within the ready list, the task with the shortest (worst-case) computation time is allocated the highest scheduling priority. As a result it is placed at the front of the ready queue, waiting for service by the processor.
However, the policy is non-preemptive, and the current executing task is always allowed to complete.

4) Least-Laxity-First (LLF): The LLF (or Earliest Deadline as Late as possible, EDL) scheduling principle uses the criterion of task 'laxity' (i.e. spare time). EDL assigns highest priority to the task having least laxity. For EDL, once a task is readied, time to deadline reduces as time elapses. The general expression defining the laxity of a task i is:

$$\text{laxity}(i,t) = \max(T_D(i) - T_c(i) - t, 0.000) \quad (1)$$

where $T_D(i)$ is its deadline, $T_c(i)$ is its computation time and t is the current run-time. (The second max function parameter is fixed to 0.000 because a laxity value must never be negative). It is obvious that laxity is a dynamic attribute; thus making LLF a dynamic scheduling policy.

5) Rate Monotonic (RM): Rate monotonic scheduling is a static-priority scheduling algorithm for periodic tasks. In RM, priorities are equal to the periods of the associated tasks. Hence, the task with the shortest period has the highest priority, and the task with the longest period has the lowest priority. Intuitively, this prioritization makes sense, since the task that has the shortest period will be the first one to be re-released. Hence, it should be the first one to complete. RM is not optimal when each task's deadline is not concurrent with the task's next release (period).

6) Earliest Deadline First: EDF is a dynamic priority scheduling algorithm that assigns highest priority to whatever task has the nearest deadline. Formally, a task τ_i's priority at time t is given by

$$P_i = d_i(t) - t \quad (2)$$

where $d_i(t)$ is the next deadline of τ_i (at or after t). For task sets where task's period is identical to its deadline span, EDF will produce a valid schedule if and only if the processor utilization of the task set is one or less. If a task set has utilization over one, the task set has no valid schedule.

7) Deadline Monotonic: DM scheduling is a static priority scheduling algorithm for periodic tasks. DM uses

the deadline span of each task for its priority. Thus, tasks with the smallest deadline span will have highest priority, and tasks with the largest deadline span will have the lowest priority. The intuition behind DM is that the task with the smallest deadline span (not necessarily the one with the smallest period) should be the task considered most urgent and therefore the task with the highest priority.

V. IMPLEMENTATION PRINCIPLES

A. Design of the Proposed Web- Enabled Simulator:

This section describes the development of the proposed simulator framework in LabVIEW.The AURTSS (AU Real-Time Scheduler Simulator) is being developed to be used for teaching real-time scheduling as well as to test and evaluate real-time scheduling policies used in embedded real-time applications.

Laboratory Virtual Instruments Engineering Workbench (LabVIEW) was originally intended as an environment for the development and execution of software analogs for conventional laboratory instruments for the non-programmer scientist. Accordingly a graphical approach was taken both for user I/O to allow the computer to visually resemble the imitated instrument, and for programming to facilitate novice program development. LabVIEW provides two graphical environments: the front panel and the block diagram (National Instruments Corporation, Aug. 2005).

A framework for evaluation of a scheduling algorithm must satisfy characteristics such as simplicity, compatibility with the PC platform and the used operating system, 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 that looks as shown in the Figure 1. The proposed Web-enabled scheduler simulator could be operated through a Web browser through a set of click-on and data input windows

B. Real-Time Scheduler Simulator

Scheduling algorithm evaluation and analysis tool performs the task definition, task sets generation, execution of selected algorithms, execution analysis of the execution and results displaying. The performance evaluation of the real-time scheduling algorithms is carried out based on the results obtained through computational analysis. Various stages of evaluation procedure are: Identification of the tasks, Task set implementation, Selection of algorithms, Simulation Timing definition, Simulation execution and Scheduling algorithms evaluation

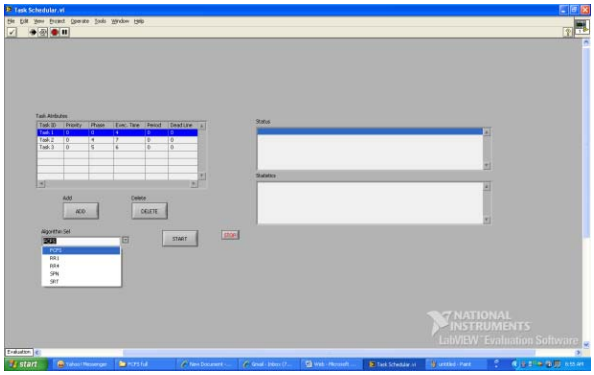

Figure 1. Task elements in the simulator

The most successful scheduling algorithm for the periodic tasks scheduling is the one that has minimal response times, minimal number of tasks with missed deadlines and maximal resource utilization in the given workload and with other parameters.

The complete task model is too complex for implementation and some of the task parameters are hence ignored. In real-time systems two characteristics of tasks are considered to be of primary interest: criticality or importance; and timing. Task importance is frequently a subjective issue, whereas timing is objective. The essential timing attributes of tasks are deadline (T_D), worst-case computation time (Tc_w) and period (T_p),

C. Elements of the simulator

Task Attributes are part of the simulator will allow the user to add Task (to the existing task set) with parameters like Task ID, Priority, Phase, Execution Time, Period, and Deadline. Among these parameters, Task ID should be unique for each input task. All other parameters are numerals. All the parameters are required to save a new task to a task set, modify the parameters of a task from the declared task set. delete task(s) from the input task set. Fig.5.1 shows the elements of simulator.

1) Resource Usage: Add a resource to specific task(s) for its execution (critical section) with Resource ID, Task ID, Start, and Time. Fig.5.2 shows the adding and deleting of tasks

2) Simulator Controls: The Simulator Controls part of the simulator is the main part of the simulator which provides the following functionalities to the user: Selection of Scheduling Policy ,selection of Task Synchronization Protocol, choice of preemptive or non-preemptive scheduling and run Feasibility Test to find whether the declared task set is feasibly schedulable with the desired scheduling scheme

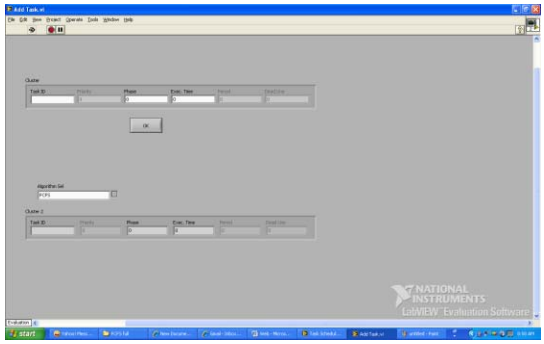

Figure 2. Adding and deleting task in the simulator

A run-time scheduler allocates processing time to the tasks in discrete quanta. Each task has a static base priority, although at run-time that task may temporarily acquire a higher active priority. At run time, each task requires an infinite number of invocations. The time at which a task invocation becomes ready to execute is its *arrival time*. The time at which a task invocation actually begins execution is its *starting time*. This may be significantly later that the arrival time if, for instance, a higher priority task was already running when the invocation arrived. The time at which a task invocation completes execution is its *finishing time*. Invocations of a task I are usually assumed to arrive regularly with a fixed period, T_i. (Sporadic, or non-periodic, tasks, arrive at irregular intervals but with a known minimum separation. For each task the programmer specifies a deadline, D_i, by which each of its invocations must finish, measure relative to the arrival time of the invocation. To support analysis, the programmer also postulates a worst-case computation time, C_i, for each invocation of task i. it is assumed that C_i includes the context switching overheads associated with scheduling the task invocation the run-time scheduler thus appears to operate instantaneously in the model. The worst-case difference between task i's arrival and finishing times is its response time, R_i. Each task normally starts to perform invocations from time 0, but this can be delayed by assuming an initial offset, O_i.

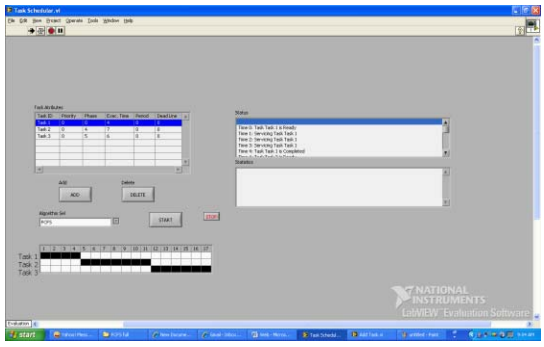

Figure 3. Output (timing diagram)

VI. Significance Of Web-Based Deployment

There are vital reasons behind the implementation of the simulator as a Web-enabled framework and Web-based deployment as listed out here. They are ease in deployment and enhancement of functionality. Anytime, anywhere access to users – any computer with Web connectivity can be used for learning and teaching. Easy access to users over the Internet since no extra hardware or software is required to access the application

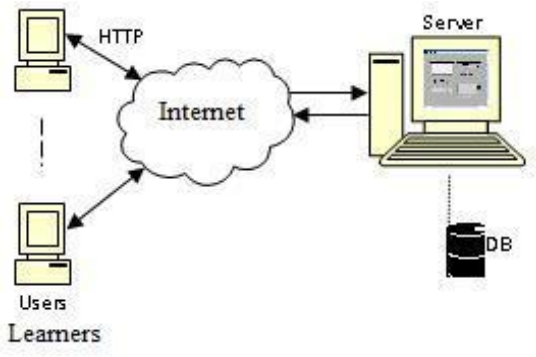

Figure 4. Deployment of the simulator through Web

Advantages of proposed simulator: Platform independent access and use of the simulator for learning, User-friendly interface that requires minimal training / re-training, users will be able to access only the latest implementation of the simulator with no ambiguity of versions of the application, ease of maintenance from a Programming / maintenance group perspective .and run Simulation to view the chronogram (timing diagram) and understand the way the tasks are scheduled in real-time using the selected scheduling policy.

VII. Conclusion And Future Work

This paper discussed the various existing real-time scheduling algorithms at the beginning. Various existing Real-Time Scheduling Simulation frameworks and their features were studied. The Web-enabled framework proposed in this paper gives the developer the possibility to evaluate the schedulability of the real time application. Numerous benefits were quoted in support of the Web-based deployment technique employed. The framework which is proposed can be used as an invaluable teaching tool. Further, the GUI of the framework will allow for easy comparison of the framework of existing scheduling policies and also simulate the behavior and verify the suitability of custom defined schedulers for real-time applications. In addition, the real-time scheduler co-processor hardware can help the learners have a closer view of scheduling tasks in real-time hardware.

Future work includes implementation of multiprocessor and aperiodic real-time schedulers in the simulator for exploring the full spectrum of real-time scheduling theory and development of co-processor architecture to complete the teaching tool.

REFERENCES

[1] Aleardo Manacero Jr., Marcelo B. Miola, Viviane A. Nabuco, "Teaching real-time with a scheduler simulator," in 31st ASEE/IEEE Frontiers in Education Conference, Oct.2001, pp. T4D 15-19, Reno, NV.

[2] Arnoldo Diaz, Ruben Batista and Oskardie Castro, Realtss: "A real time scheduling simulator," in 4th International Conference on Electrical and Electronics Engineering (ICEEE), Sep.2007, pp. 165-168, Mexico City, Mexico.

[3] Barbara Korousic-Seljak, "Task scheduling policies for real-time systems, in Microprocessors and Microsystems," Vol.18 No. 9. pp.501-511, Nov.1994.

[4] Chi-scheng shih, Lui Sha and Jane Liu, "Scheduling Tasks with Variable Deadlines," IEEE, pp. 120-122.

[5] J.E. Cooling, P. Tweedale, "Task scheduler co-processor for hard real-time systems, in Microprocessors and Microsystems," Vol. 20, pp. 553-566, 1997.

[6] Enrico Bini and Giorgio C. Buttazzo, "Schedulability Analysis of Periodic Fixed Priority Systems," in IEEE Transactions on Computers, Vol. 53, No. 11, pp. 1462-1473.

[7] Jan Blumenthal, Frank Golatowski, Jens Hildebrandt, Dirk Timmermann, "Framework for Validation, Test and Analysis of Real-Time Scheduling Algorithms and Scheduler Implementations," in Proceedings of the 13th IEEE International workshop on Rapid System Prototyping (RSP'02), 2002.

[8] Krishna, C. M., and Shin, K. G., Real-Time Systems, MIT Press and McGraw-Hill Company, 1997.

[9] Krithi Ramamritham and John A. Stankovic., "Scheduling Algorithms and Operating Systems Support for Real-Time Systems," in Proceedings of the IEEE,Vol. 82, No. 1, pp. 55-67, Jan.1994.

[10] Liu, C.L., and James W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment," in Journal of the ACM, Vol. 20, No.1, pp. 46-61, Jan.1973.

[11] National Instruments Corporation, LabVIEW™ Fundamentals, Aug. 2005.

[12] Reinder J. Bril and Pieter J.L. Cuijpers, "Analysis of hierarchical fixed-priority pre-emptive scheduling revisited," in Technische Universiteit Eindhoven (TU/e) CS-Report, Dec. 2006, pp. 06-36.

[13] Sha, L., Abdelzaher, T, Arzen, K., Cervin, A., Baker, T.P., Burns, A., Buttazzo, G., Caccamo, M., Lehoczky, J., and Mok, "AReal time scheduling theory: A historical perspective," in Real-Time Systems, 28(2): , Nov. 2004, pp. 46-61.

[14] Sha, L., R. Rajkumar, and J.P. Lehoczky, Priority Inheritance Protocols: "An Approach to Real-Time Synchronization," in IEEE Transactions on Computers, Sep. 1990, pp. 1175-1185.

[15] F. Singhoff, J. Legrand, L. Nana, L. Marce, Cheddar: "A Flexible Real Time Scheduling Framework," in ACM SIGADA'2004 International conference proceedings, Atlanta, Giorgia, USA. Nov. 2004.