ISSN (Online) : 2454-4159
International Journal of Advanced Research in Science Management and Technology
Volume 2, Issue 6, June 2016

A Comparative study of Real Time Operating System & Scheduling Algorithm

Nakul Sandhanshive[1], Prof A. B. Kanwade[2],

PG Scholar, Dept. Of VLSI & Embedded System Engg, SITS Narhe, Pune, M.S., India[1]
Assistant Professor, Dept. Of VLSI & Embedded System Engg, SITS Narhe, Pune, M.S., India[2]

ABSTRACT— Some real-time applications require a high-determination time tick keeping in mind the end goal to work appropriately. Be that as it may, supporting a high-determination time tick forces a high overhead on the system. What's more, such systems may need to change scheduling discipline every now and then to fulfil some client prerequisites, for example, Quality of Service (QoS). The element changing of the scheduling order is typically connected with deferrals amid which a few due dates may be missed. In this paper, we display a configurable hardware scheduler engineering which minimizes the processor time squandered by the scheduler and time-tick handling. The hardware scheduler is adaptable and gives three scheduling disciplines: need based, rate monotonic and soonest due date first. The scheduler in hardware likewise gives exact planning. The scheduling mode can be changed at runtime, giving backing to an extensive variety of uses on the same gadget. The hardware scheduler is given as an Intellectual Property (IP) hinder that can be altered by fashioner's contribution, to suite a specific application, by a device we have created.

KEYWORDS: configurable hardware scheduler, hardware scheduler, real-time systems, real-time operating system, scheduling algorithm.

I. INTRODUCTION

A Real-Time Operating System (RTOS) permits realtime applications to be composed and extended effortlessly. In any case, the RTOS presents overhead, which may keep some real-time systems, for example, rapid bundle changes, from working productively. Subsequently, due dates might be missed. The overhead can be lessened by moving bit administrations, for example, scheduling, time tick (an occasional hinder to monitor time amid which the scheduler settles on a choice) preparing [7], and intrude on taking care of to hardware. This will altogether enhance the reaction time and the interfere with inactivity, give exact planning, and expand the CPU usage.

An execution of a hardware scheduler more often than not can bolster one and only scheduling algorithm. Thus, the hardware can bolster a limited scope of uses, which function admirably under the same scheduling algorithm. Dissimilar to programming segments, a hardware unit is not so much adaptable but rather more hard to alter after execution. Subsequently, hardware arrangements are every now and again maintained a strategic distance from. In any case, if the hardware scheduler is configurable to bolster a few scheduling algorithms, then the hardware arrangements turn out to be more adaptable. Future installed gadgets will bolster an extensive variety of uses. The hardware scheduler may should be reconfigured at the time of use exchanging. For instance, assume the present application on a handheld gadget is running under a need based scheduling algorithm and assume that

the client presses a catch to change to another application, which functions admirably under an Earliest-Deadline-First (EDF) algorithm. With a specific end goal to bolster the new application productively, the hardware scheduler will be reconfigured from the need based mode to the EDF mode. Moreover, distinctive classes of utilizations will ave diverse quantities of errands in the system. Once the hardware scheduler is created or designed into a Field Programmable Gate Array (FPGA), the most extreme number of errands is altered. In this manner, the quantity of undertakings must be indicated for the application class before the hardware is constructed. In any case, the operations of the hardware scheduler ought to be autonomous of the quantity of undertakings. Adaptability of the hardware scheduler can be refined by executing settled cycle operations. Every operation requires a settled number of cycles. The prepared line design must be versatile. At the point when the prepared assignment is embedded to the prepared line, it must be sorted in a steady time.

Some FPGA sellers have as of late discharged reconfigurable rationale with processors, for example, PowerPC [13] and ARM [16]. With chips accessible containing both reconfigurable rationale and processor(s) together on one bite the dust, the 2 hardware scheduler can be effortlessly designed. Moreover, with a runtime bolster environment for reconfigurable systems, any scheduling algorithm or any RTOS segment actualized in hardware can be downloaded and reconfigured at runtime. This will empower the hardware answer for be as adaptable as the product arrangement; for instance, a current part, the Xilinx XC3000 is reconfigurable in 1.5 ms, and future FPGA items guarantee to be reconfigurable in a great deal less time than this [12]. We actualize a configurable hardware scheduler in the Verilog Hardware Description Language (HDL) and a RTOS in C. Our usage is versatile. We moved the product scheduler and the time tick foundation handling to the hardware. Hence, the product overhead from these administrations is disposed of.

II. RELATED STUDY

A few past papers manage scheduling algorithms executed in hardware. The vast majority of them are in the field of parcel scheduling in real-time systems [1], [2], [8]. Scheduling in such systems depends on needs. Subsequently, a key perspective is to execute need lines. Numerous hardware structures for the lines have been proposed: twofold tree comparators, FIFO lines in addition to a need encoder, and a systolic cluster need line [1]. The greater part of the hardware proposed addresses the execution of one and only scheduling algorithm (e.g., Earliest Deadline First) [8]. In the field of real-time preparing, there have been couple of proposition of hardware executions. In the spring portion venture [3], [10], a coprocessor was worked to upgrade the multiprocessing scheduling [9]. This coprocessor could perform plausibility checks and compute a complete attainable timetable. FASTHARD [4] and FASTCHART [5] are two ways to deal with execute a hardware piece for single or multiprocessor systems. The FASTCHART approach utilized an uncommon reason CPU to execute the scheduling algorithm running in parallel to the primary CPU. In FASTHARD, the creator actualized custom hardware in a FPGA to play out the functionalities of the need scheduler [11].

The past exploration on the hardware execution of real-time schedulers concentrated just on actualizing stand out scheduling algorithm, in this manner making them wasteful and not appropriate for systems where the required scheduling discipline changes amid runtime. We, then again, present a

ISSN (Online) : 2454-4159
International Journal of Advanced Research in Science Management and Technology
Volume 2, Issue 6, June 2016

configurable scheduler that backings three scheduling disciplines. The scheduler can change starting with one scheduling train then onto the next on the fly amid runtime to adjust to changes in the system. Our hardware scheduler was intended to bolster various scheduling disciplines utilizing least territory overhead.

The executed scheduling disciplines have the same hardware segments and utilize the greatest measure of regular rationale and least measure of multiplexers to choose a scheduling discipline. Our usage is altogether not quite the same as having three free hardware schedulers running in parallel.

III. SCHEDULING ALGORITHMS

A. Earliest deadline first (EDF)

Minimum time to go is a dynamic scheduling algorithm utilized as a part of real-time operating systems to place forms in a need line. At whatever point a scheduling occasion happens (undertaking completes, new assignment discharged, and so forth.) the line will be hunt down the procedure nearest to its due date. This procedure is the by be planned for execution.

EDF is an ideal scheduling algorithm on pre-emptive uniprocessors, in the accompanying sense: if a gathering of autonomous employments, each described by a landing time, an execution prerequisite and a due date, can be planned (by any algorithm) in a way that guarantees all the occupations complete by their due date, the EDF will plan this accumulation of employments so they all complete by their due date.

With scheduling occasional procedures that have due dates equivalent to their periods, EDF has a use bound of 100%..

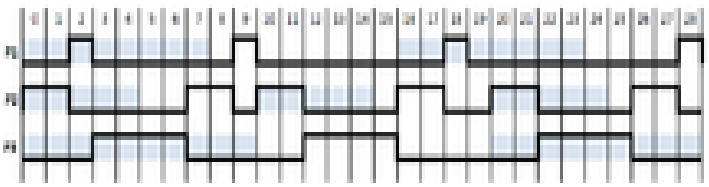
Consider 3 periodic processes scheduled on a preemptive uniprocessor. The execution times and periods are as shown in the following table:

Process Timing Data

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

In this example, the units of time may be considered to be schedulable time slices. The deadlines are that each periodic process must complete within its period.

[1] Timing diagram



Timing Diagram showing part of one possible schedule for the example.

ISSN (Online) : 2454-4159
International Journal of Advanced Research in Science Management and Technology
Volume 2, Issue 6, June 2016

In the planning chart, the sections speak to time cuts with time expanding to one side, and the procedures all begin their periods at time cut 0. The planning chart's exchanging blue and white shading demonstrates every procedure's periods, with due dates at the shading changes.

The principal procedure booked by EDF is P2, on the grounds that its period is most limited, and thusly it has the soonest due date. In like manner, when P2 finishes, P1 is planned, trailed by P3.

At time cut 5, both P2 and P3 have the same due date, expecting to finish before time cut 10, so EDF may plan possibly one.

B. Multilevel Feedback Queue

A multilevel feedback queue is a scheduling algorithm. Solaris 2.6 Time-Sharing (TS) scheduler implements this algorithm.[1] The Mac OS X and Microsoft Windows schedulers can both be regarded as examples of the broader class of multilevel feedback queue schedulers.[2] This scheduling algorithm is intended to meet the following design requirements for multimode systems:

1. Give preference to short jobs.
2. Give preference to I/O bound processes.
3. Separate processes into categories based on their need for the processor.

Scheduling parameters

In general, a multilevel feedback queue scheduler is defined by the following parameters

1. The number of queues.
2. The scheduling algorithm for each queue which can be different from FIFO.
3. The method used to determine when to promote a process to a higher priority queue.
4. The method used to determine when to demote a process to a lower priority queue.
5. The method used to determine which queue a process will enter when that process needs service.

C. Least slack time (LST)

Least slack time (LST) scheduling is a scheduling algorithm. It assigns priority based on the slack time of a process. Slack time is the amount of time left after a job if the job was started now. This algorithm is also known as least laxity first. It's most common use is in embedded systems, especially those with multiple processors. It imposes the simple constraint that each process on each available processor possesses the same run time, and that individual processes do not have an affinity to a certain processor. This is what lends it a suitability to embedded systems.

1. Slack time

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.

2. Applications

In real time scheduling algorithms for periodic jobs, an acceptance test is needed before accepting a sporadic job with a hard deadline. One of the simplest acceptance tests for a sporadic job is calculating the amount of slack time between the release time and deadline of the job.

3. Suitability

ISSN (Online) : 2454-4159
International Journal of Advanced Research in Science Management and Technology
Volume 2, Issue 6, June 2016

LST scheduling is most helpful in systems containing primarily aperiodic undertakings, on the grounds that no earlier suppositions are made on the occasions' rate of event. The primary shortcoming of LST is that it doesn't look ahead, and works just on the present system state. Along these lines, amid a brief over-burden of system assets, LST can be problematic. It will likewise be problematic when utilized with uninterruptible procedures. In any case, as most punctual due date to start with, and dissimilar to rate monotonic scheduling, this algorithm can be utilized for processor usage up to 100%.

IV. RESULT COMPARISIONS

Maximum Clock frequency of Hardware Scheduler

Scheduler Configuration	Clock period	Clock Frequency
Earliest Deadline first	4.016	231.908
Multi-Level Feedback Queue	5.550	177.098
Least Slack time	5.890	169.281

Processor utilization performance:

Sr. No	Utilization through software EDF	Utilization through Hardware MLFQ	Utilization through Hardware LST
1	**65.67%**	**66.31%**	**70.12%**

V. CONCLUSION

In this paper, we have taken a gander at the different methods offered as answers for the issue of diminishing the CPU Scheduling forced by the RTOS. We had studied Scheduler Configurations like Earliest Deadline first, Multi-Level Feedback Queue, Least Slack times. We have examined their clock frequencies & Clock periods.

As shown in above results, we had examined every possibility of utilisation through software of these Processors. After comparing we came on to the point that Utilization through Hardware LST is better than, Utilization through software EDF, Utilization through Hardware MLFQ.

REFERENCES

[1] Z Jin, M Sindhwani and T Srikanthan, 2004, RTOS Acceleration on Soft-core Processors Using Instruction Set Customization, In-ternational Conference on Field Programmable Technology (FPT 2004), Australia.

[2] Texas Instruments, 2002, TMS320VC5470 Fixed-Point Digital Signal Processor Data Manual.

[3] Inneon Technologies, 2001, TC1775 User's Manual System Units. see: http://www.in neon.com/tricore/

[4] Xilinx, 2002, Virtex-II Pro Platform FPGAs see: http://www.xilinx.com/

[5] Balough, C, 2000, Picking Winners in the Configurable System-on-Chip Space see: http://www.techonline.com/

[6] Cooling J. and Tweedale P, 1997, Task scheduler co-processor for hard real-time systems Microprocessors and Microsystems, 20 (1997), pp. 553{566.

[7] Ramakrishnan N, 2002, H37/01 { Towards an Independent On-Chip RTOS Manager. Honors Year Project Report. Nanyang Technological University (2002).