ANALYZING THE EFFECT OF GAIN TIME ON SOFT-TASK SCHEDULING POLICES IN REAL TIME SYSTEMS

A.V.Thulasi Pratiba[1] B.Chandrakala[2]

[1], PG Student, Embedded Systems, Sri Venkateswara College of Engineering

[2], Assistant Professor, Department of ECE, Sri Venkateswara College of Engineering

tulsi.pratiba@gmail.com

ABSTRACT

The paper is dealing with the effect of gain time on soft task scheduling in RTOS based application. RTOS is an operating system that supports real-time applications and embedded systems by providing logically correct result within the deadline. In Multitasking gain time is a key factor which explicit the difference between the actual time and maximum time for completion of a process. In some real time applications, delay in a particular process may lead to severe effects. In this project work , delay in a task is avoided by using an effective preemptive scheduling and giving importance to high priority interrupts even though if there is any pending low priority interrupts on semaphore. As a prototype demonstration hereby implementing in the hardware using ARM Processor for an automobile application. An object tracking system will be fixed in the vehicle with vehicle-object distance measurement facility. If an object approaches the vehicle beyond the minimum distance limit, then preemptive scheduler will assign the object interference as a high priority interrupt. As an immediate response the speed of the vehicle will be changed by using a control method PWM. The Pulse width modulating technique will be initiated automatically by the Processor without any manual braking system and a heavy alert will be given to the person to bring concentration in driving.

Keywords: RTOS, ARM Processors, PWM

INTRODUCTION

The embedded systems are sewn into our day-to-day life in various forms of visible and invisible manner via many different application areas which include consumer electronics, medical imaging, telecommunications, automotive electronics, avionics, space systems, etc. For instance, the progress in use of multi-core platforms in embedded systems has already reached our hands as a form of mobile phones and related devices with small form factor. The main purpose of a real-time system is to produce the required result within strict time constraints including computational correctness.

In other words, in the physical world the purpose is to construct a physical effect within a chosen time-frame. There are several perspectives to classify real-time systems. Depending on the system characteristics, a real-time system can be categorized as hard real-time or soft real-time by considering factors inside the system and factors outside the system. As many embedded systems are used in safety-critical applications, their correct functionality in the whole system is imperative to avoid severe consequences. It is estimated that 99% of produced microprocessors are integrated into embedded systems. Furthermore, as a result of this abrupt technological progress, a significant increment in software complexity and processing demands of real-time systems is seen.

To cope with the processing demands, silicon vendors are concentrating on using multi-core platforms for high-end real-time applications instead of incrementing processor clock speeds in uni-core platforms. By the same token, scheduling research of multi-core architectures offers a broad spectrum of significant opportunities for real-time system producers. Research of uni-core and multi-core real-time scheduling both originated back in late 1960s and early 1970s, consequential advances were made in 1980s and 1990s. Still, there is sufficient scope for research, although uni-core real-time scheduling is considered reasonably mature to be in industrial practice. On the other hand, many of well researched multi-core scheduling techniques are not mature enough to either be applicable or optimal as much as currently available uni-core real-time scheduling techniques. For this reason, reliable simulation platforms are required to augment the research of scheduling techniques for real-time embedded multi-core architectures, which is also coupled with analytical results that expect guaranteed real-time administration over the system by the most effective use of the available processing capability through employing efficient scheduling policies placed on the underlying hardware.

Scheduling Paradigm

In the field of hard real-time systems, the main goal is to achieve that none of the so-called hard tasks in the system ever fails to meet its temporal requirements, usually defined in terms of deadlines. The current practice for achieving this goal is to adopt a certain scheduling paradigm in the development of the real-time system. The paradigm imposes both a particular task model at design time and a corresponding scheduling policy at runtime, and then provides the system designer with a formal, offline feasibility analysis by which it is possible to prove whether all hard tasks will be able to meet their deadlines before the system starts running.

One of the most sound and widespread paradigms is fixed-priority pre-emptive scheduling. In this paradigm, the task model requires each hard task to have some known temporal attributes (release times, computation times, deadlines, etc.) and a fixed priority. At runtime, the system always selects the ready task with the highest priority for execution in a pre-emptive manner. Hard real-time systems may also include some other tasks without hard or strict deadlines, which are normally referred to as soft tasks. The scheduling paradigm typically considers that the execution of a soft task produces some utility value to the system if the task can be completed before some point in time (related to the task's arrival time), after which this value progressively decreases; in contrast, the utility value of a hard task instantly drops to zero after reaching its deadline.

Aim and Scope

The main results of this paper show that, in general, the fact that hard tasks consume less execution time than their estimated WCETs (which in turn produces the availability of gain time) negatively affects the performance benefit of using any of the policies under study with respect to scheduling soft tasks in background. These are also true even for those policies that are specifically designed to efficiently reclaim and use gain time.

In nearly all cases, the performance benefit is significantly reduced as the amount of gain time increases in the system. Under some conditions, this performance benefit is so small, or even negative, that the use of a specific scheduling policy for soft tasks becomes questionable. The final purpose of this work is for it to be used as a guide to determine which scheduling policies for soft tasks are more appropriate depending on the running conditions of the system and, specifically, the amount of gain time that is available at runtime.

As a prototype demonstration we are implementing in the hardware using ARM Processor for an automobile application. An object tracking system will be fixed in the vehicle with vehicle-object distance measurement facility. If an object approaches the vehicle beyond the minimum distance limit, then preemptive scheduler will assign the object interference as a high priority interrupt. As an immediate response the speed of the vehicle will be changed by using a control method PWM. The Pulse width modulating technique will be initiated automatically by the Processor without any manual braking system and a heavy alert will be given to the person to bring concentration in driving.

LITERATURE SURVEY

Contemporary RTOS

A real-time operating system (RTOS) supports applications that must meet deadlines in addition to providing logically correct results. The paper reviews pre-requisites for an RTOS to be POSIX and discusses memory management and scheduling in RTOS. We survey the prominent commercial and research RTOSs and outline steps in system implementation with an RTOS. We select a popular commercial RTOS within each category of real-time application and discuss its real-time features. A comparison of the commercial RTOSs is also presented. We conclude by discussing the results of the survey and suggest future research directions in the field of RTOS.A real-time system is one whose correctness involves both the logical correctness of outputs and their timeliness.

It must satisfy response-time constraints or risk severe consequences including failure. Real-time systems are classified as hard, firm or soft systems. In hard real-time systems, failure to meet response-time constraints leads to system failure. Firm real-time systems have hard deadlines, but where a certain low probability of missing a deadline can be tolerated. Systems in which performance is degraded but not destroyed by failure to meet response time constraints are called soft real-time systems. An embedded system is a specialized real-time computer system that is part of a larger system. In the past, it was designed for specialized applications, but reconfigurable and programmable embedded systems are becoming popular. Some examples of embedded systems are: the microprocessor system used to control the fuel/air mixture in the carburetor of automobiles, software embedded in airplanes, missiles, industrial machines, microwave ovens, dryers, vending machines, medical equipment, and cameras. We observe that the choice of an operating system is important in designing a real-time system.

Designing a real-time system involves choice of a proper language, task partitioning and merging, and assigning priorities to manage response times. Language synchronization primitives such as Schedule, Signal and Wait simplify translation of design to code and also offer portability. Depending upon scheduling objectives, parallelism and communication may be balanced. Merging highly cohesive parallel tasks for sequential execution may reduce overheads of context switches and inter-task communications. The designer must determine critical tasks and assign them high priorities. However, care must be taken to avoid starvation, which occurs when higher priority tasks are always ready to run, resulting in insufficient processor time for lower priority tasks.

WCET Problem – Overview

The determination of upper bounds on execution times, commonly called Worst-Case Execution Times (WCETs), is a necessary step in the development and validation process for hard real-time systems. This problem is hard if the underlying processor architecture has components such as caches, pipelines, branch prediction, and other speculative components. The article describes different approaches to the problem and surveys several commercially available tools and research prototypes.

Hard real-time systems need to satisfy stringent timing constraints, which are derived from the systems they control. In general, upper bounds on the execution times are needed to show the satisfaction of these constraints. Unfortunately, it is not possible in general to obtain upper bounds on execution times for programs. Otherwise, one could solve the halting problem. However, real-time systems only use a restricted form of programming, which guarantees that programs always terminate; recursion is not allowed or explicitly bounded as are the iteration counts of loops.

EXISTING SYSTEM

The existing system dealing with the effect of gain time on soft task scheduling in RTOS based application. RTOS is an operating system that supports real-time applications and embedded systems by providing logically correct result within the deadline. In Multitasking gain time is a key factor which explicit the difference between the actual time and maximum time for completion of a process. In some real time applications, delay in a particular process may lead to severe effects. In this project work , delay in a task is avoided by using an effective preemptive scheduling and giving importance to high priority interrupts even though if there is any pending low priority interrupts on semaphore.

- Manual Control
- More gain time
- No quicker response to the interrupt
- No distance measurement

The paper has presented the results of an empirical study on the most relevant scheduling policies for soft tasks in fixed-priority, pre-emptive real-time systems. In particular, the goal of the study was to characterize the effect of gain time on the behavior of these scheduling policies. The existence of gain time, which is defined as the difference between the WCET of a hard task and its actual execution time, is typical in many real-time systems for two main reasons: first, because the WCET overestimation is still a common practice in the design of many real-time systems in order to ensure the safety of the schedulability analysis, and second, because even if WCETs are accurately calculated, the typical case for tasks is to consume only a fraction of their WCETs at runtime. Traditionally, gain time has been regarded as a design problem for hard tasks (when related to WCET overestimation), but also as an opportunity for soft tasks, which can use this spare time in order to improve their response times. Indeed, some scheduling policies for soft tasks have included specific extensions to make an effective use of this gain time.

PROPOSED SYSTEM

RTOS is an operating system that supports real-time applications and embedded systems by providing logically correct result within the deadline. In these, dealing with the effect of gain time on soft task scheduling in RTOS based application. In Multitasking, gain time is a key factor which explicit the difference between the actual time and maximum time for completion of a process. In some real time applications, delay in a particular process may lead to severe effects. In these, delay in a task execution is avoided by using an effective pre-emptive scheduling and giving importance to high priority interrupts even though if there is any pending low priority interrupts on semaphore.

- Effective Preemptive scheduling technique
- Immediate response to the priority interrupts
- Automatically speed change
- Vehicle to Object distance measurement
- Intimation to the person inside the vehicle

As a prototype demonstration, we are implementing the hardware using ARM Processor for an automobile application. An object tracking system will be fixed in the vehicle with vehicle-object distance measurement facility. If an object approaches the vehicle beyond the minimum distance limit, then pre-emptive scheduler will assign the object interference as a high priority interrupt. As an immediate response, the speed and direction of the vehicle will be changed by using a control method PWM. The Pulse width modulating technique will be initiated automatically by the Processor without any manual braking system and a heavy alert will be given to the person to bring concentration in driving.

The proposals greatly reduce the manpower, save time and operate efficiently without human interference. This project puts forth the first step in achieving the desired target. An embedded system is a combination of software and hardware to perform a dedicated task. Some of the main devices used in embedded products are Microprocessors and Microcontrollers. Microprocessors are commonly referred to as general purpose processors as they simply accept the inputs, process it and give the output. In contrast, a microcontroller not only accepts the data as inputs but also manipulates it, interfaces the data with various devices, controls the data and thus finally gives the result. Embedded technology is now in its prime and the wealth of knowledge available is mind-blowing. Embedded technology plays a major role in integrating the various functions associated with it. This needs to tie up the various sources of the Department in a closed loop system

SOFT REAL-TIME TASKS

Consider a set of n soft real-time tasks. There exists one processor and only one task can be executed on the processor at any given time. Except for the processor, there are no other shared resources to be taken into account. The tasks are pre-emptive, independent and aperiodic. For each task τi , we assume that ri , ei , Gi , and di ,which are respectively the release time, execution time, penalty factor and deadline of the task, are known. A slot is the smallest time unit. The objective is to minimize Pn i=1 G (τi). Therefore, we can formally express the objective function as follows. Let us define xi,t = 0, 1 if the processor is assigned to task τi at time slot t 0 otherwise Our goal is to minimize the objective function Xn i=1 (ri + ξiei − di) +Gi , subject to the following conditions Xn i=1 xi,t = 1, which means only one processor is working at any given time t, and X∞ t=1 xi,t = ei , meaning that the total time slots assigned to any given task i over time is equal to its execution time.

As mentioned earlier, the problem defined in this section is known to be NP-hard. Thus, the known algorithms for obtaining an optimal schedule require time that grows exponentially with the number of tasks. Assume that the tasks are prioritized by a function for the optimal algorithm. Since the problem is NP-hard (non-deterministic polynomial-time hardness), it is not known if there is any polynomial time function for prioritizing for any optimal algorithm of the problem. Despite that, the behavior of any optimal scheduling algorithm, when the optimal order of priorities is provided.

Knowing a number of properties of any optimal schedule for the problem will lead us to designing heuristic algorithms which, in some properties, have the same behavior as the optimal schedule. We provide a set of heuristic algorithms that are based on the properties proved here. The heuristic algorithms differ in the way that the task priorities are assigned. Also, it is desired to find an upper bound for the objective function which, unlike the optimal algorithm, would be computationally feasible. In this work, we derive a tight upper bound for the optimal solution.

PROPOSED BLOCK DIAGRAM

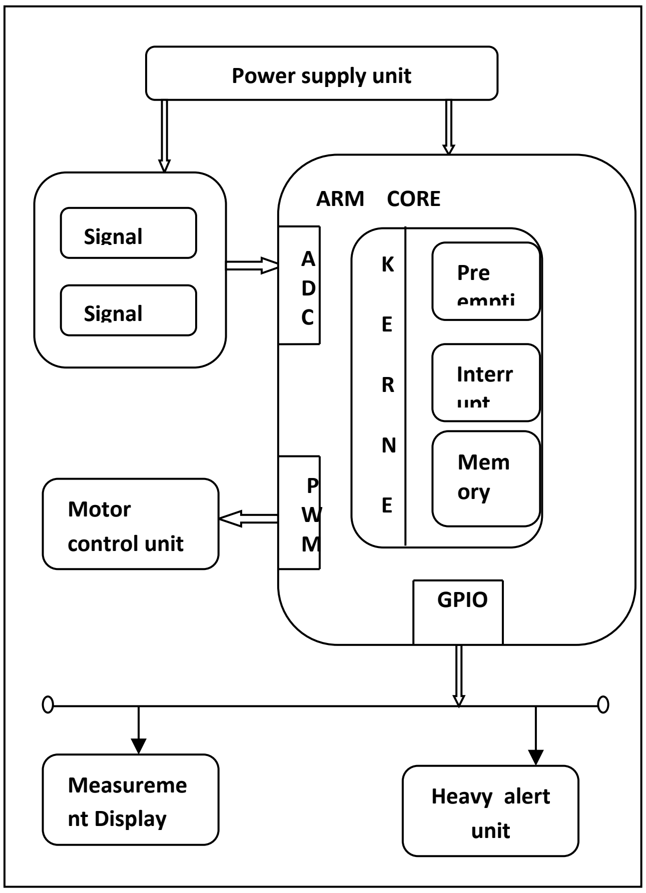

Fig 1: Block Diagram

These versatile devices are useful for driving a wide range of loads including solenoids, relays DC motors, LED displays filament lamps, thermal print heads and high power buffers. The ULN2001A/2002A/2003A and 2004A are supplied in 16 pin plastic DIP packages with a copper lead frame to reduce thermal resistance. They are available also in small outline package

RESULTS

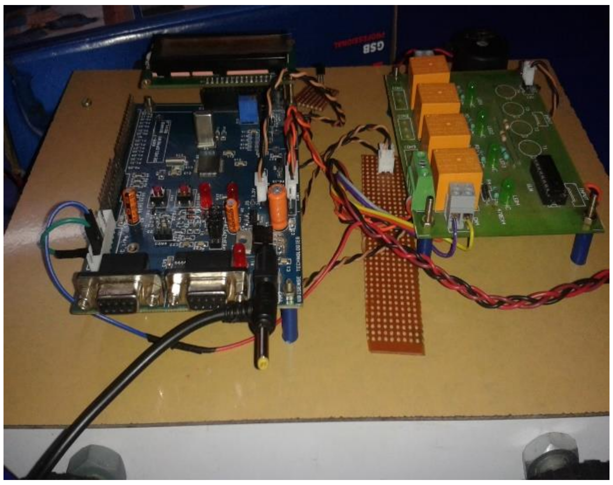

Fig 2 : Automobile Robot with ARM Processor

FUTURE WORK

The system is implemented using the pre-emptive scheduling policy to reduce the gain time and handle the tasks based on priority. However, by using the scheduling policies we can accomplish the tasks efficiently but destinations cannot be determined. We can obtain an efficient system by using a GPS system along with the present; we can locate the destinations accurately.

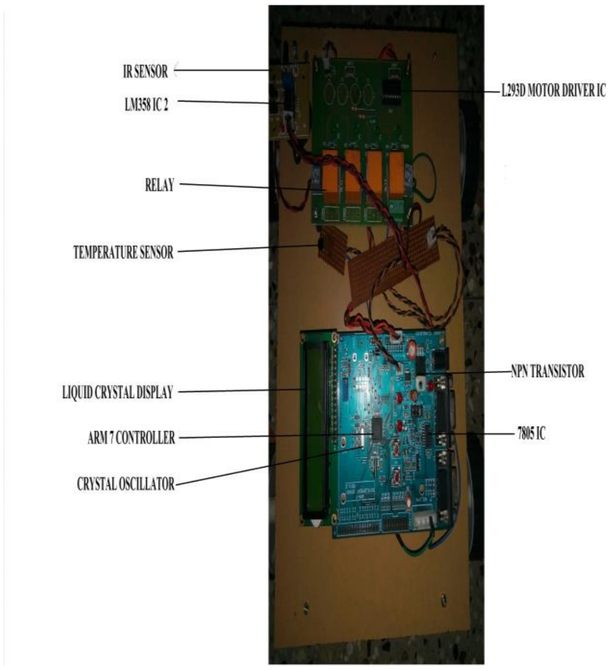

Fig3: Automobile Robot with ARM Processor

CONCLUSION

At present the vehicles are being controlled by humans which might lead to accidents sometimes, so we have introduced a system which will work in both autonomous and manual mode by using pre-emptive scheduling which divides the tasks based on priority and functions the high prioritized tasks. The vehicle can detect the obstacles automatically using the ultra-sonic sensors and proceeds in the obstacle free direction accordingly. So,we can conclude that man can make mistakes but machines cannot. Non pre-emptive scheduling executes the tasks in cyclic order so only one task can run at a time later it moves to next one even though it is a higher priority task. Hence, pre-emptive scheduling is chosen.

REFERENCES

[1] N.C. Audsley, R.I. Davis, A. Burns, and A.J. Wellings, “Appro-priate Mechanisms for the Support of Optional Processing in Hard Real-Time Systems,” Proc. IEEE 11th Workshop Real-Time Operating Systems and Software, pp. 23-27, 1994.

[2] L. Sha, B. Sprunt, and J. Lehozky, “Aperiodic Task Scheduling for Hard Real-Time Systems,” The J. Real-Time Systems, vol. 1, no. 1, pp. 27-60, 1989.

[3] J.M. Banus, A. Arenas, and J. Labarta, “An Efficient Scheme to Allocate Soft-Aperiodic Tasks in Multiprocessor Hard Real-Time Systems,” Proc. Int’l Conf. Parallel and Distributed Processing Techniques and Applications, vol. 2, pp. 809-815, 2002.

[4] J.M. Banus, A. Arenas, and J. Labarta, “Dual Priority Algorithm to Schedule Real-Time Tasks in a Shared Memory Multiprocessor,”

Proc. Int’l Parallel and Distributed Processing Symp., 2003.

[5] G. Bernat and A. Burns, “New Results on Fixed Priority Aperiodic Servers,” Proc. IEEE 20th Real-Time Systems Symp., pp. 68-78, 1999.

[6] E. Bini and G.C. Buttazzo, “Measuring the Performance of Schedulability Tests,” Real-Time Systems, vol. 30, pp. 129-154, 2005.

[7] L.A. Bu´rdalo, A. Espinosa, A. Garcı´a-Fornes, and A. Terrasa, “Framework for the Development and Evaluation of New Scheduling Policies in RT-Linux,” Proc. Workshop Operating Systems Platforms for Embedded Real-Time Applications, pp. 42-51, 2006.

[8] L.A. Bu´rdalo, A. Espinosa, A. Terrasa, and A. Garcı´a-Fornes, “Experimental Results of Aperiodic Fixed-Priority Preemptive Policies in RT-Linux,” Proc. Workshop Operating Systems Platforms for Embedded Real-Time Applications, pp. 10-19, 2007.

[9] J.M. Calandrino, D.P. Baumberger, T. Li, S. Hahn, and J.H. Anderson, “Soft Real-Time Scheduling on Performance Asym-metric Multicore Platforms,” Proc. IEEE Real-Time Technology and Applications Symp., pp. 101-112, 2007.