A Study on Fixed Priority Scheduling Based RTOS in Automotive Domain

B.Suresh *, N.P.Subiramaniyam b, A.Balamurugan b

* PG Department of Electronics and Communication Systems, AJK College of Arts and Science, Coimbatore, India
b Department of Electronics and Communication Systems, Nehru Arts and Science College, Coimbatore, India  
* e-mail id: suresb@gmail.com

Keywords: Electronic Control Unit, AUTOSAR, Fixed Priority Scheduling.

Abstract: This study deals the Embedded Systems with Real Time Operating systems in automotive domain. Multisource software running on the same Electronic Control Unit (ECU) is becoming increasingly wide spreading the automotive industry. This case is one of the main reasons that car manufacturers want to reduce the number of ECUs, which grew up above 70 for high-end cars. One major outcome of the Automotive Open System Architecture (AUTOSAR) initiative and, more specifically, its operating system (OS) is to help car manufacturers shift from the “one function per ECU” paradigm to more centralized architecture designs by providing appropriate protection mechanisms. Another crucial evolution in the automotive industry is those chips manufacturers are reaching the point where they could no longer cost effectively meet the increasing performance requirements through frequency scaling alone. This condition is one reason that Multicore ECUs are gradually introduced in the automotive domain. As seen in research incentives there is a demand for better performance with lower power consumption for fixed priority preemptive scheduling make it easy to add functions to an existing ECU. In fixed-priority scheduling, the priority of a job once assigned, may not change. A new fixed-priority algorithm for scheduling systems of periodic tasks upon identical multiprocessors is proposed. This algorithm has an achievable utilization of (m+1)/2 upon m unit-capacity processors. It is proven that this algorithm is optimal from the perspective of achievable utilization in the sense that no fixed-priority algorithm for scheduling periodic task systems upon identical multiprocessors may have an achievable utilization greater than (m+1)/2. Multiprocessor task scheduling is a fixed priority scheduling and each processor should be assigned and allocated at runtime in order to balancing the load CPU and its usage is 88%.

Introduction

The higher level of performance provided by Multicore architectures may help simplify in-vehicle architectures by executing on multiple cores that the software previously run on multiple ECUs. This possible evolution toward more centralized architectures is also an opportunity for car manufacturers to decrease the number of network connections and buses. As a result, parts of the complexity will be transferred from the electrical/electronic architecture to the hardware and software architecture of the ECUs. However, fixed priority preemptive scheduling makes it easy to add functions to an existing ECU. In practice, important architectural shifts are hindered by the carryover of ECUs and existing sub networks, which are widely used by generalist car manufacturers. As seen in research incentives there is a demand for better performance with lower power consumption. The only solution for such requirement is the shift towards multi-cores. There are many different scheduling algorithms employed in automotive domain. In this study we are adopting fixed priority preemptive scheduling. Because compared to other scheduling fixed priority preemptive scheduling based CPU has higher loading percentage.

Fixed-Priority preemptive Scheduling:

Scheduling disciplines are algorithms used for distributing resources among parties which simultaneously and asynchronously request them. The main purposes of scheduling algorithms are to minimize resource starvation and to ensure fairness amongst the parties utilizing the resources. Scheduling deals with the problem of deciding allocated request for their resources. In Fixed priority Preemptive scheduling The OS assigns a fixed priority rank.
to every process, and the scheduler arranges the processes in the ready queue in order of their priority. Higher priority processes first even then interrupted by incoming lower priority processes.

This study deals with the fixed priority preemptive scheduling of tasks in a real time systems with hard constraints is mandatory. More specifically we consider offset free systems where the offset can be chosen by the scheduling algorithm. The model of the system is defined by a task set $\Delta$ of cardinality $n$, $\Delta=\{T_1, T_2, \ldots, T_n\}$. A periodic task $T_i$ is characterized by a quadruple $(C_i, T_i, D_i, O_i)$ where each request of $T_i$, called instance, has an execution time of $C_i$, a relative deadline $D_i$. $T_i$ time units separate two consecutive instance of $T_i$ (hence $T_i$ is the period of the task). The first instance of $T_i$ occurs at time $O_i$ (the task offset in the following).

![Diagram of the runnables.](image)

**Figure. Model of the runnables.**

The system is said to be schedulable if each instance finishes before its deadline. If the system is schedulable in synchronous case it follows that this is also the case in all asynchronous situation and determine feasible offset from engine rpm. This is usually done to load balance a system effectively. After its release, an instance of a runnable has to be executed before the next instance is released (i.e., the deadline is set to the period).

**EXPERIMENTATIONS:**

The increasing complexity of automotive electronic systems has a dramatic effect on the throughput requirements and peripheral integration of automotive microcontrollers. So in this experiment we combined the runnable sequences in the same CPU to find the Response time and we use Parallelism to enhance the multiple cores on the single machine. In this study analysis was made with the help of Least crowded algorithm.

**Response Time:**

Computing the length of time taken by a system to respond to an instruction. Prioritization is useful for browsing tasks, and tasks that use a lot of processor time. Input/Output bound tasks can take the required amount of CPU, and move on to the next read/write wait. CPU-intensive tasks take higher priority over the less intensive tasks. Prioritization can be implemented in all CICS® systems. It is more important in a high-activity system than in a low-activity system. With careful priority selection, it can improve overall throughput and response time. Prioritization can minimize resource usage of certain resource-bound transactions. Prioritization increases the response time for lower-priority tasks, and can distort the regulating effects of MXT and the MAXACTIVE attribute of the transaction class definition. Priorities do not affect the order of servicing terminal input messages and, therefore, the time they wait to be attached to the transaction manager. Because prioritization is determined in three sets of definitions (terminal, transaction, and operator), it can be a time-consuming process to track many transactions in a system. CICS prioritization is not interrupt-driven as is the case with operating system prioritization, but determines the position on a ready queue. This means that, after a task is given control of the processor, the task does not relinquish that control until it issues a CICS command that calls the CICS dispatcher.

Use small priority values and differences and concentrate on transaction priority. Give priority to control operator tasks rather than the person, or at least to the control operator’s signon ID rather than to a specific physical terminal (the control operator may move around). Consider for high priority a task that uses large resources. However, the effects of this on the overall system need careful monitoring to ensure that loading a large transaction of this type does not lock out other transactions. Also consider for high priority those transactions that cause enqueues to system resources, thus locking out other transactions. As a result, these can process quickly and then release resources.

![Diagram of the Response time Parallelism](image)

**Fig. Model of the Response time Parallelism**

Multicore ECUs exploit parallelism to enhance performance, an understanding of the key type of parallelism is important to analyzing performance. Parallelism is a complex topic, but a basic understanding of three types of parallelism
is sufficient. Instruction –level parallelism, thread –level parallelism , data –level parallelism are all employed in various multicore CPU architectures, and have different impacts on performance that must be understood to conduct through performance analysis. The parallel processing on multiple cores on a single machine or on multiple machines that have access to a common directory. If you have multiple function calls that are independent of each other, and you can reformulate your code as,

for k = 1:numel (parameterCell)
resultCell[k] = myfun(parameterCell [k]);
end
then, replacing the loop by
resultCell = startmulticoremaster(@myfun, parameterCell);
allows you to evaluate your loop in parallel.

By testing the functions on two dual core machines It produces the great gain when parallelizing processing between the processors of dual core machine or between the two machines using a single processor in each. However, there is no significant gain when trying to use both processors on both machines. Is that because the gain of using more processors is being reduced by increased load of data recording when we add more processors/machines.

Result:
The CPU load in percentage of Least crowded algorithm =10. (WCET= 98µs)
The CPU load in percentage of fixed priority preemptive scheduling = 88. (WCET= 89µs)

[WCET= Worst Case Execution Time]

CONCLUSION:
Multicore chips allow for greater increases in computing power in contrast to a single CPU continually made to run faster. For this we integrate more number of functions on a single core makes efficient distribution of runnables on CPU over run time. If the number of ECUs is reduced then we obtain many more technical and economical advantage abroad a modern trendy cars. The system builds a high performance Multicore ECU which has the power and ability to execute number of operations at a time without any contradiction between the synchronized and Nonsynchronized tasks as for as the offset concerns. The set of runnable sequencing algorithms proposed in this system aims at uniformizing the load over time and thus increases the maximum workload schedulable on the CPU.

Experimentations on realistic case studies have confirmed that the algorithms are versatile and efficient in terms of CPU usage optimization.

References:
[1]. “A new fixed priority assignment algorithm for global multiprocessor scheduling” tech report sep 2012,ISSN 1652-926x by Risat Mahmud bathan,Jan jonson.


[5]. “Scheduling algorithms and operating systems support for real-time systems “This paper appears in:Proceedings of the IEEE,Jan 1994 by Ramamritham, K.Dept. of Comput. Sci., MassachusettsUniv.,Amherst,MAVolume:82 ,Is sue:1