Comparison of Scheduling Algorithms in Chimera and VxWorks

Ulrika Uppman
ulrup562@student.liu.se

Gustav Wikander
guswi700@student.liu.se

LiTH

2006-11-19

Abstract

This report is a comparison of the scheduling algorithms and the schedulers of two real-time operating systems (RTOS), VxWorks and Chimera. VxWorks is a commercial RTOS by Wind River and Chimera is developed at the Carnegie Mellon University.

VxWorks has priority based preemptive scheduling and round robin scheduling, both based on user set task priority levels. Chimera has a Maximum Urgency First (MUF) scheduling algorithm, which is a mix of fixed and dynamic priority scheduling algorithms. Chimera's MUF scheduler is a bit more complex and has built-in support for meeting deadlines. The Chimera scheduler with its features and its support for reconfiguring them might give a good base for schedulers in time critical systems whereas the VxWorks scheduler is simpler and faster for those small systems, but still allows you to implement any algorithm you need.

Table of contents

1 Introduction

This report is a part of the course TDDB72 in operating systems at LiU, the University of Linköping. We have been assigned the task to look deeper into real-time operating systems, and we chose to look deeper into scheduling algorithms. We have picked two different hard real-time operating systems and read more about the ways scheduling can be implemented. We have studied and compared the commercial real-time operating system VxWorks, by Wind River, with Chimera which was developed at the Carnegie Mellon University.

An important thing to consider when looking at hard real-time operating systems compared to non-real-time operating systems is that tasks are usually periodic and have a deadline, before which the task must have finished or there will be terrible consequences. To help ensure this, a scheduler with support for meeting deadlines can be quite helpful, especially if tasks don't have static execution times and deadlines. If they do, you can analyze if deadlines will be met beforehand, when building the system, and set up a static scheduling table.

At the same time you don't want the scheduler to use up too much of the system resources using too complex algorithms. Which scheduler and algorithm you want to use depends on what kind of system you are developing. Sometimes you might want to use an already existing scheduler that has all the features you need. You might want to add a few features or just fine-tune an existing scheduler. In some cases perhaps it is best to write your own scheduler from scratch, customized to fit only your exact needs.

In VxWorks executing programs are referred to as tasks. VxWorks also supports POSIX threads which are similar to VxWorks tasks [7]. In Chimera the programs execute as a sort of lightweight processes and are also referred to as tasks.

In chapter 2 and 3 we first describe the two real-time operating systems. We first introduce the operating systems and then we look at their schedulers and scheduling algorithms. In chapter 4 we will make a comparison between the two of them and state some of their differences and discuss the pros and cons between them. Finally we conclude what we came up with in chapter 5.

2 VxWorks

2.1 About VxWorks

One of the two real-time operating systems we are looking at is VxWorks, which is manufactured by Wind River [6]. VxWorks is used in many products all around the world and can be customized to fit many different situations where you would need a hard real-time operating system. VxWorks is built to have a small footprint, meaning it uses little memory and processor time to leave as much system resources as possible available for application programs. It is also scalable and has support many extra packages. This real-time operating system is used globally and to name a few you can find VxWorks in the Boeing 787 Airliner, the Mars Exploration Rovers, digital cameras, printers, industry robots and much more [6, 10].

2.2 Scheduling Algorithms

By default, VxWorks supports two different task scheduling algorithms. The first is a preemptive priority based algorithm and the other one is a priority based Round Robin algorithm. This simple scheduler helps keeping the footprint small. There are 256 different task priority levels in VxWorks, but only 150 of them should be used for application tasks and the others for system critical tasks [7].

2.2.1 The Preemptive Priority Based Algorithm

This is a commonly used algorithm which you find in many operating systems and it is the default algorithm in VxWorks.

2.2.1.1 How it Works

Preemptive priority based task scheduling let tasks with higher priority run first hand. If a task with lower priority happens to be running when a higher priority task arrives in the scheduling queue it is preempted and the higher priority task gets to run immediately after the switch of tasks. If you assign the same priority to all tasks it basically acts as a First In First Out (FIFO) algorithm, letting the tasks in queue at the same priority level run one after another [7].

2.2.1.2 When to Use it

This algorithm is quite useful when tasks truly have different priority levels and it is important for certain tasks to be completed as soon as possible. If you on the other hand use it in a system where tasks are periodic you might end up having some tasks miss their deadlines or even suffer from starvation if the system is overloaded.

2.2.2 Round Robin Algorithm

To help solving the problem with starving periodic tasks you can enable time slicing among tasks using the Round Robin algorithm [7].

2.2.2.1 How it Works

The algorithm simply gives tasks with the same priority level an equal time quantum to run. You can say that the tasks take turns in using the processor. A time quantum usually is somewhere around 10 to 100 ms. After each time quantum there is a task switch and a new task gets to run for an equal amount of time. Remember that in VxWorks the priority levels still comes first hand and higher priority tasks will preempt lower priority tasks [7].

2.2.2.2 When to Use it

As long as tasks are of equal size, meaning they need an equal amount of processor time, this algorithm is fair. In the case where some tasks are small and some are large the time they each get to run is no longer fair. The small tasks then only get what they need, which probably is less than one time quantum, while the large tasks end up having several time quanta. Simply put, large tasks are favored over the small ones. But the real problem that you solve by enabling time slicing is that small tasks don't get locked out by large tasks of the same priority level [7].

2.3 VxWorks Scheduler

The VxWorks real-time operating system is a commercial product and the source code is not available, which makes it hard to replace the actual default scheduler. If you would want to use other scheduling algorithms than the built-in ones you would need to implement your own scheduler as an application task. The whole point from the start with the built-in scheduler is that it is fast. You can rely on the fact that the task switching is fast and always uses the same amount of time when you need to consider overhead times for a hard real-time environment [6].

We found two sources that had benchmark numbers for task switching. The overhead time for a task switch is somewhere around 14 μs [8, 9]. The small overhead for switching tasks can be of importance if you use the Round Robin algorithm with small time quanta.

You can use VxWorks for both periodic and non-periodic tasks. The scheduler itself hasn't got that much to offer when it comes to ensuring deadlines. In a hard real-time environment you often consider static periodic tasks and static deadlines when implementing the applications and do the math yourself to ensure deadlines are met. If you would want some support for meeting deadlines from the system you would need to implement that in an application task.

To implement other scheduling algorithms you could make a task that acts as a scheduler and give it highest priority among your application tasks. Let the task set all other tasks priorities in such a way that it follows the scheduling algorithm you want to implement. Say you want to use the Earliest Deadline First (EDF) algorithm you would simply set the task with the closest deadline to the highest priority. EDF has a scheduling bound of 100%, not counting the overhead of switching tasks and the time used by your scheduler to dynamically change priorities to follow the algorithm [11].

3 Chimera

3.1 About Chimera

Chimera is a multiprocessor real-time operating system (RTOS) developed by The Advanced Manipulators Laboratory, at Carnegie Mellon University. It is mostly used in sensor-based systems, like robotics [2], so this is an operating system that can handle hard real-time environments. Chimera was built in the early 1990's, and the project now seems to be discontinued.

3.2 Chimera Scheduler and Scheduling Algorithms

The Chimera kernel supports both static and dynamic scheduling, and by default it uses a mixed priority scheduling algorithm called Maximum Urgency First (MUF) [2]. The user can also customize parts of the default scheduler, or add new schedulers in the system [4].

3.2.1 Time Consumption

In Chimera, a context switch takes 66 μs and a rescheduling operation in the dynamic scheduler takes 26 μs on a MC680x0 processor [4]. Every time a new task arrives in the system, the rescheduling operation will be run.

3.2.2 The Maximum Urgency First Algorithm - MUF

The Maximum Urgency First scheduling algorithm is a kind of mixed priority scheduling; it uses both fixed and dynamic scheduling. The urgency priority is based on two fixed priorities and one dynamic priority.

The schedulable bound of the algorithm is 100%. This means that in the worst-case scenario, the CPU will make all deadlines when its utilization reaches 100%. In other words, we won't miss any deadlines unless the CPU is overloaded. This is to be compared with for example a fixed priority Rate Monotonic (RM) scheduling algorithm which has a schedulable bound of 69% in worst case (and about 88% in average) [2]. In the MUF scheduling algorithm, it is also possible to set the priorities so that one will have control over which tasks will fail in the first place if the CPU gets overloaded.

3.2.2.1 Criticality

The first fixed priority is the criticality. This is set to either high or low and it tells if this particular task is critical to finish within its deadline. The scheduler will make sure that all tasks set to critical will make their deadline.

3.2.2.2 User Priority

The other fixed priority is the user priority. This gives a chance for the user to set which tasks are more important not to fail.

3.2.2.3 Minimum Laxity Priority

The dynamic priority is a reverse version of *laxity*. Now what is laxity then? Let's look up the expression for laxity:

laxity = deadline_time – current_time – CPU_time_still_needed [2]

This will mean that if a task has laxity T_l then we won't need to worry about the task missing its deadline in another T_l time units.

In the MUF algorithm we use the inverse relation to laxity as a priority. This means that if a task has laxity 0 (meaning that it might miss its deadline if we won't let it execute now) will have a higher priority than a task with laxity 1.

3.2.2.4 Assigning Fixed Priorities

The fixed priorities will only be assigned to each task once. First we will sort the tasks with the highest frequent tasks first (or the ones with the shortest period). The first N tasks with a total of less than the schedulable bound (that is 100% in MUF) of CPU utilization will then be set with high criticality [2]. These tasks will not fail. All other tasks will get low criticality. If wanted, the user priority will be set now as well.

3.2.2.5 Actions by the MUF Scheduler in Run-Time

For every cycle, the scheduler will choose which task to execute. The scheduler will look at the priorities one after another to decide this.

1. Choose the task with highest criticality first. If there is more than one to choose here, go to step 2.
2. Choose the task with the lowest laxity. If there is more than one to choose here, go to step 3.
3. Choose the task with the highest user priority. If there is more than one to choose here, go to step 4.
4. Use "first-come-first-serve" to choose which task to run.

In practice, all of these priorities may be represented as a single value, *urgency*. The urgency will be a value consisting of the criticality, the minimum laxity priority and the user priority. The criticality will be set as most significant part of the value, the minimum laxity priority will be set in the middle, and user priority will get the least significant part of the value.

3.2.3 Fault Tolerance

If the CPU is overloaded, we will still have tasks that miss their deadlines even though we have a schedulable bound of 100%. The MUF scheduling algorithm will make sure that no critical tasks will fail their deadlines. However, if the worst case scenario for the estimated execution time of a task is not calculated right, critical tasks may risk failing too.

When a deadline of a task is missed, a failure handler can be run. The failure handler will let the user define what to do with the task that failed to meet its deadline. For example one may want to restart the task, send system notifications, shut down parts of the system or save data.

3.2.4 Modifications on the MUF Scheduling Algorithms

3.2.4.1 MUF can be Used as RM, EDF or MLF

Actually, the title of this chapter is a little misleading, because MUF can without any modifications on the actual algorithm, be turned in to both Rate Monotonic (RM), Earliest Deadline First (EDF) or Minimum Laxity First (MLF) algorithms. This can be done because of that the urgency priority essentially consists of more or less modified versions of these algorithms. For example; if the worst-case execution time is set to 0, the urgency of the task will become the same as if it only depended on the deadline. Hence, we got an EDF algorithm instead of a real MUF algorithm.

3.2.4.2 Tasks that are not Periodic

The algorithm described above uses the task period (or the frequency of the task) in the system to compute its priority. Non periodic tasks however are possible if one use some kind of algorithm similar to the *sporadic server* [2], without modifying the actual MUF algorithm.

3.2.5 Using the Scheduler in Chimera

When to create a new task, the user calls the *spawn* function. The spawn function may be called from any tasks, but there is no parent-child concept. Spawn lets the user give criticality and user priority as function arguments. Other argument like the period of the task can also be given. There is also a function, *spawnd*, which calls spawn with the default priorities for a task. The spawn function places the task on the ready list and then it might be scheduled to run at the next context switch, depending on whether it has the highest priority or not.

There are also a bunch of different primitive functions that among other things allows the user to set task deadlines, to clear task deadlines and to reset the tasks user priorities and criticality.

Before a task runs it will have to set a start time, a deadline time and a worst-case execution time so that the task scheduler can calculate the tasks priority. The time variables that are relative to the present time and therefore changes with time, like for example the deadline time and the time needed to execute the task, have predefined user macros for getting the current value.

The user can also use the *ctxt_switch* system call that forces a rescheduling operation, and thereby a swich to a new task may occur.

4 Comparison

4.1 Differences

VxWorks and Chimera are both used for hard real-time systems, but still they are very different. We will of course, as the report title says, only compare the differences in the scheduling they use.

To start with, we have different scheduling algorithms in the both systems. VxWorks uses a fixed priority scheduling as default, while Chimera on the other hand, as default uses a mix of both fixed and dynamic priority scheduling.

VxWorks wants to be minimal and does not put so much effort in the scheduling algorithm by default. For example, the context switch in Chimera takes more time than it does in VxWorks. VxWorks could perform a context switch in somewhere around 14 μs, compared to Chimera

that needs about 66 μs to perform a context switch. These numbers were gathered on different processors and might not be truly comparable. [4, 8, 9]

The scheduler in Chimera works only with periodic tasks by default.

In both real-time operating systems you can make your own scheduler if you for some reason would no want to use the default one. Although in Chimera you have a built in support for replacing or changing the default scheduler. In VxWorks you will have to make a user task that somehow handles the scheduling.

4.2 Advantages and Disadvantages

VxWorks has the advantages of being a small and fast real-time operating system, but it does not provide the user with as many features as Chimera does by default. The scheduler algorithm in Chimera has both user-set priorities and dynamically calculated priorities based on CPU-time and deadlines for the tasks. It also provides the user with a failure handler so that the user can choose what action to be taken if tasks miss deadlines when the CPU is overloaded. This makes Chimera slower on context switches and rescheduling operations than VxWorks.

5 Conclusions

There are a few main differences between the two real-time operating systems compared in this report. VxWorks has priority based preemptive scheduling and Round Robin scheduling, both based on user set task priority levels. Chimera has a Maximum Urgency First scheduling algorithm, which is a mix of fixed and dynamic priority scheduling algorithms based on both user-set priorities and dynamic priorities based on deadline and CPU usage time.

In the sum total we can say that the Chimera scheduler is a bit more complex and has built-in support for critical deadlines and a handler for missed deadlines, whereas the VxWorks scheduler is simpler and faster but still allows you to implement other scheduling algorithms if you would want to do so.

Which real-time operating system you would want to use depends on what you need it for. VxWorks gives you the basics which at times surely is enough, if you need a fast and small system. But sometimes you probably need an active scheduler which reschedules tasks during runtime. In that case, when you need to ensure deadlines are met and you need to control which tasks that will pass their deadlines and what to do when some tasks don’t, you might want Chimera’s built-in support. You can easily continue building on the Chimera scheduler whereas in VxWorks you would have to implement it all from scratch yourself, something which takes time and costs money.

References

[1] D. B. Stewart and P. K. Khosla, *Chimera 3.1, The Real-Time Operating System for Reconfigurable Sensor-Based Control Systems*, November 22, 1996, http://www.ece.umd.edu/serts/bib/manuals/Chimera_book.ps.gz

[2] D. B. Stewart and P. K. Khosla, *Real-Time Scheduling of Sensor-Based Control Systems*, May 1991, http://www.ri.cmu.edu/pub_files/pub1/stewart_d_b_1991_1/stewart_d_b_1991_1.pdf

[3] D. B. Stewart, *The Robotics Institute, Chimera*, http://www.ri.cmu.edu/projects/project_140.html

[4] D. B. Stewart and P. K. Khosla, *Chimera Real-Time Operating System*, http://www.cs.cmu.edu/afs/cs.cmu.edu/project/chimera/www/chimera/chimera.html

[5] D. B. Stewart, D. E. Schmitz, and P. K. Khosla, *Implementing Real-Time Robotic Systems using CHIMERA II*, May 1990, http://www.ri.cmu.edu/pub_files/pub2/stewart_david_b_1990_1/stewart_david_b_1990_1.pdf

[6] Wind River, VxWorks, Accessed November 2006, www.windriver.com

[7] VxWorks Programmer's Guide 5.5, 2002. In HTML: http://www.slac.stanford.edu/exp/glast/flight/sw/vxdocs/vxworks/guide/index.html or PDF: http://www-cdfonline.fnal.gov/daq/commercial/vxworks_programmers_guide5.5.pdf

[8] VxWorks FAQ, Question 13, Last modified May 16, 1994, http://www.faqs.org/faqs/vxworks-faq/part1/

[9] Naren Bala, *Real-time Operating Systems (RTOS) modeling*, Accessed November 2006, http://mesl.ucsd.edu/gupta/Teaching/cse237a-s04/ProjectPresentations/VxWorksReportv5.doc

[10] VxWorks on Wikipedia, Last modified 15 November 2006, http://en.wikipedia.org/wiki/Vxworks

[11] Silberschatz, Galvin & Gagne, *Operating System Concepts*, seventh edition, 2005. ISBN 0-471-69466-5