A Comparitative Analysis of different Scheduling Algorithm for Different Real Time Operating Systems

Suresh P
Asst.Professor, Dept. of CSE
SVCE, Bengaluru
Suresh.rvce@gmail.com

Maria Navin J R
Asst.Professor, Dept. of ISE
SVCE, Bengaluru
marianavin.jr@gmail.com

Pradeep
Asst.Professor, Dept of ISE
SVCE, Bengaluru
pradeep.kr22@gmail.com

Abstract

In this paper we are comparing the scheduling algorithms in VxWorks and RTLinux. Here we are going to discuss how the tasks are handled in operating systems by comparing the two Real Time operating systems. The paper deals with mainly how the Algorithm for Scheduling works for different Real Time Operating systems where we are dealing with concurrent programming and Parallel Execution. Benefits in each operating system work differently, depending on the priority of the system. Since RTLinux is an open-source and can be compiled to implement our own scheduling algorithm and it can be a good choice to implement our own. RTLinux but have two algorithms included as loadable kernel modules which are not standard algorithms. They are based on the Rate Monotonic and Earliest Deadline First algorithms. When VxWorks was introduced it used the pre-emptive priority based scheduling algorithm and a round-robin Scheduling Algorithm.

1. Introduction

In a hard real-time system, a scheduling algorithm decides what task should be progressed next depending on its priority. Moreover, it's not only priority that can determine if a task should be progressed, also algorithms can include deadlines which must be fulfilled. RTLinux and VxWorks are two of the most used real-time operating systems on the market. One of the differences is that RTLinux is an open-source and VxWorks is not. Because of that it is hard to find detailed information about VxWorks algorithms. RTLinux scheduling use both priority based and deadline algorithms. VxWorks only use priority based scheduling. In RTLinux you have the possibility to implement own or modified scheduling algorithms which is a good benefit compared to VxWorks where you are locked to the developers own ones.

We are going to explain the different scheduling algorithms in each operating system and leave the comparison for the last section of this paper. We will try to find the different benefits between two very popular real-time operating systems.

RTLinux is a hard Real-Time operating system that runs on machines that coexists with Linux as its lowest priority execution thread. RTLinux is developed by FSMLabs Inc[1] and is available as a community supported free version as well as a commercial version from FSMLabs. The commercial version was developed out from the free version but have been extensively rewritten by FSMLabs and works on many architectures while free is mostly X86. In the free version, a lot component has been added by the Ocera Project[2]. In this report we don't focus on either the commercial or the free version, we focus on the two scheduling algorithms which is mentioned most in associate to the development name RTLinux.

VxWorks is a hard real-time system. It has been used for a lot of applications but is mostly known for spacecraft and latest the Mars Exploration Rovers[3]. It almost entirely supports the POSIX standard but it's not open source so you can't do any own changes do

[1] FSMLabs Inc. homepage: http://www.fsmlabs.com
[2] For more information visit http://www.ocera.org
[3] For more information visit http://marsrovers.jpl.nasa.gov/home/index.html

the kernel and Wind River4, the creator of VxWorks, do not offer any source code. It supports two scheduling algorithms a basic pre-emptive priority one and a round-robin algorithm. It gives the user maximal control of the code but also leave responsibility for deadlines to him.

2. Literature Survey

The Rate Monotonic Algorithm

Rate Monotonic algorithm is based on static priority. A static priority algorithm assigns all priorities at design time, and those priorities remain constant for the lifetime of the task. The system assign priorities based on the expected execution time, task with less execution time will be assigned higher priority.

- Rate Monotonic scheduler gives higher priority to tasks with smaller periods.

The following example will illustrate the Rate Monotonic algorithm, considering three tasks T1, T2 and T3 in fig1. T2 and T3 are periodic tasks, cycle time of T1 is 30ms, T2 is 40ms and T3 is 50ms. Above you can see how much time each task is assigned to the CPU. T1 will be assigned highest priority because it has the shortest cycle time, followed by T2 and T3. If we have tasks which is non-periodic it gives the lowest priority and will only be assigned CPU time when none of the periodic processes are requesting the CPU. This is because we don't know the deadline of this task, or the deadline is very far away.

In order to work correctly, certain pre-emption must be satisfied:

$$U = \sum_{i=1}^{n} C_i/T_i \leq n(\sqrt[n]{2} - 1)$$

n is the total task number in the system. Ci stands for the longest execution time in the system and Tn is the release period assuming that the deadline is one period later.The workload U must fulfil a given statement to guarantee that no deadlines will be met. If we let n go to infinity we get a percentage of 69.3% which is the percentage of the CPU utilization time the real-time tasks cannot exceed to guarantee that no deadlines will be met for a system where we have unpredictable number of tasks.

[4] Wind River Inc. homepage: http://www.windriver.com

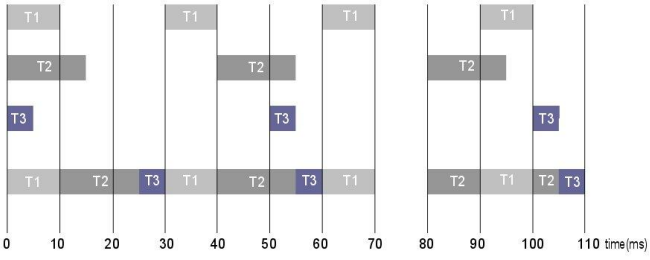

Fig1. Example of scheduling based on Rate Monotonic Algorithm

Earliest Deadline First schedule algorithm is based on dynamic priority. The dynamic priority algorithm means that the tasks are assigned priority levels during run-time. A task will be assigned higher priority if circumstances request it. The Earliest Deadline First schedule algorithm assign a priority for each task according to the deadline of the task. Tasks with earliest deadline will be assigned highest priority.

- Earliest Deadline First scheduler gives highest priority to tasks with earliest deadline

The following example will illustrate the Earliest Deadline First algorithm, considering three tasks T1, T2 and T3 in fig2.

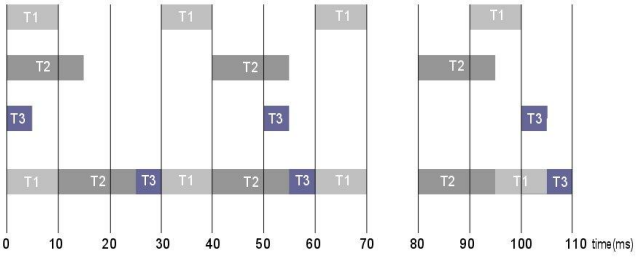

Fig2. Example of scheduling based on Earliest

T1, T2 and T3 are periodic task with deadlines when the periodic time have exceeded. For non-periodic task the deadline can be assigned as an absolute deadline. Each thread in RTLinux holds two deadline attribute. One is relative deadline and the other one is absolute deadline. The relative deadline is the deadline assigned due to the periodic time, an absolute deadline is the moment in time at which the response must be completed.

In order to schedule correctly, certain pre-emption must be satisfied:

$$U = \sum_{i=1}^{n} C_i/T_i \leq 1$$

n is the total task number in the system. Ci stands for the longest execution time in the system and Tn is the release period assuming that the deadline is one period later.

As long the CPU utilization is less than 100% the Earliest Deadline First algorithms will always work. This is only the theoretical number when dynamic priority requires more complex systems to work.

3. Scheduling in VxWorks

VxWorks is a hard real-time system. It has been used for a lot of application from Boeings new airliner to ABB's robotics. It's not open source so you can't do any changes to the code. A lot of facilities are provided through available modules from either Wind River, the creator of VxWorks or from third party companies.

A task is a keyword in VxWorks. A task is a program similar to a process or a thread. A tasks context among other things includes a program counter, the CPU registers, a stack and timers. When there is a context switch, a switch between the current running task and a task that is ready to run, all these things must be copied from memory into the CPU. A critical element in all real-time operating systems is that the context switch is fast. VxWorks includes advanced functionality do optimize the context switch by leave out some registers that may not be necessary.

A task in VxWorks can have different states. A state diagram is shown in fig 3.

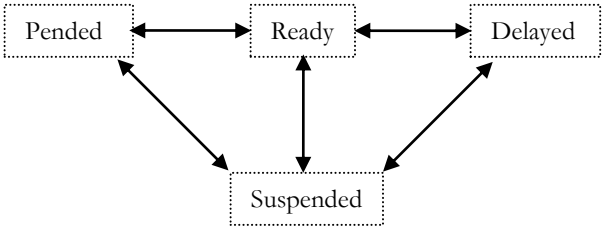

Fig3. Task state diagram

A short description of each state:

- Pended – a task is blocked for some reason
- Ready – a task is ready to run.
- Delayed – a task is in sleep for some reason.
- Suspended – a task is unavailable for execution.

When a task is created it will enter the suspended state and from there it can be activated. There also exists functionality to create and place a task in the ready state with one function call. The VxWorks task management library includes functionality to move tasks from different states. There is no state to show that a task is the task current running, like in for example Nachos. Task can be deleted from all states.

3.1 Standard pre-emptive priority based algorithm

In standard mode VxWorks uses a preemptive priority based scheduler. Pre-emptive means that the scheduler directly switches from a current running task to a task with higher priority even if the task with lower priority has not finished executing. In standard mode a task that is executing will run until it is finish even if there still exist tasks with the same priority.

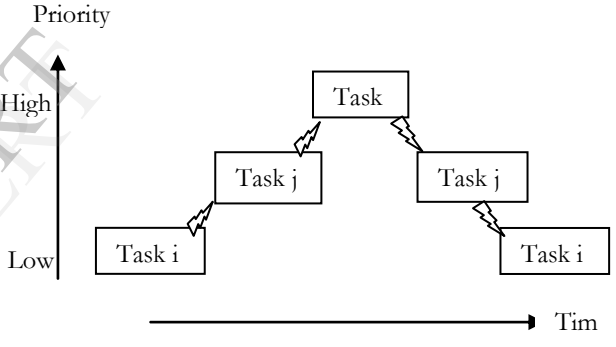

Fig4. Pre-emptive standard algorithm in VxWorks

Fig4 show the algorithm in action. It starts with a task, task i, and when it has run for a while there is a task with higher priority available, task j. The scheduler will then automatically do a context switch and replace the current executing task with the one with higher priority. After a while there is a task with even higher priority available, task k. A switch will be made and after that task k will run until termination, marked with a star in figure, then task j, will run until termination and after that task i will run until end.

There are 256 priority levels in VxWorks. The priority range is counted from 0 to 255. Where the lowest priority is 255 and highest is 0. Priority is assigned to a task when it's created but can also be changed during running using the taskPrioritySet () command. There is no limit in the number of task you

can create in VxWorks as long as you have enough memory. There is a convention that program tasks should have priority from 100-250.

3.2 Round-Robin algorithm

VxWorks can also be enabled to use a round-robin algorithm. In VxWorks that means that you assign a timeperiod to the kernel. If there are several tasks with the same priority, each task will run for the assigned timeperiod and then leave place for the next one. Priorities still exist. If there is a task with higher priority available there will always be a context switch even if a time period is set. The function for changing time period is kernelTimeSlice (int ticks) which could be found in the kernel library.

Round-robin mode does not affect the performance of a context switch, but there will be more context switches which could affect performance. Not either is more memory allocated if round-robin mode is turned on.

During round-robin algorithm in action. There are 3 different tasks, two tasks with lower priority and one with higher priority. In the beginning task i will run for the selected time period. After that there will be a context switch to task j, and task j will run until there is a task, task k, with a higher priority available. Task k is the only task with high priority so it will run until completion. After task k has finished, task j will continue to run because it has time left in its time period. The time is enough for task j to finish and after that task i finish.

A problem with this mode is that it can be hard to set a good time period. If the time period is to short it will make the scheduler run too often and require too much of the CPU. A to long period may not be good for short tasks. If there is a big difference in the execution time of the tasks the shorter tasks will be favoured with round-robin. It exist functionality in the kernel to lock the scheduler from changing tasks. Maybe it would be a desired solution if you need to compute something and don't want some other thread with higher priority to change with you. Or if the priority is equal or higher and round-robin mode is selected.

A problem will occur when you have locked the scheduler and then wait for some resource held by some other thread. VxWorks has solved this by then activate the scheduler, run some other tasks, and when the old task gets rescheduled it will deactivate the scheduler again. The functions for deactivate and active the scheduler is taskLock() and taskUnlock() which could be found in the task management library.

VxWorks also include functionality for priority inheritance. It means that if a high priority task is waiting for a resource held by a lower priority task, the low priority task will increase its priority to the highest task waiting for its resource. When the task has finished executing its critical section it will change its priority to where it was before.

If you use several priorities you can get problem with starvation. That means that a task with low priority never gets a chance to run. The development suite for VxWorks has the ability to check for starvation and other problems such like deadlocks and priority inversion.

3.3 Not Round-Robin vs. Round-Robin

According to the manual you will not get slower context switches or allocate more memory if you have round-robin mode selected. Therefore it may not be a bad solution if you want to split up the work load fairly among many tasks with the same priority.

It is not possible to enable round-robin for some priority levels and disable it for others. Either you use it or not.

4. Comparison between RTLinux and VxWorks Scheduling

The table below shows a short summary over scheduling differences in RTLinux and VxWorks.

	RTLinux	**VxWorks**
Standard algorithm	No standard algorithm(EDF/ Rate monotonic)	Preemptive priority based / Round-Robin
Possible to implement own scheduling algorithms	Yes	No
Preemptive based algorithms	Yes	Yes
Dynamic priority	Yes	No
Static priority	Yes	Yes
Deadline architecture	Yes	No

Priority inheritance	No	Yes

4. Conclusions

If you have a periodic system where tasks can be predicted RTLinux is maybe the best choice. On the other hand if you have a dynamic system where tasks can start at random and you don't in advance know when a task start VxWorks is maybe a better choice due to its priority algorithm.

If you have a system based on deadlines maybe RTLinux is a better choice when it support the deadline scheduling architecture. VxWorks does not so it's up to the user to control that deadlines are met.

Due to RTLinux is open-source software you have the possibility to implement your own scheduling algorithms or download someone that is available. VxWorks is not open-source software so you have to use the available ones.

Priority inheritance is supported in VxWorks but not in RTLinux.

5. References

[1] Silberschatz, Galvin & Gagne, "Operating System Concepts 7th edition" 2005 ISBN: 0-471-69466-5

[2] Wind River Systems Inc, "VxWorks API reference", *http://www.slac.stanford.edu/exp/glast/flight/sw/vxdocs/VxWorks_API_Reference.htm*,2006-11-17

[3] Wind River Systems Inc, "Tornado API Reference", http://www.slac.stanford.edu/exp/glast/flight/sw/vxdocs/Tornado_API_Reference.htm 2006-11-17

[4] Wind River Systems Inc, "Real-Time Processes (RTPs) for VxWorks 6.0 http://www .windriver.com/whitepapers/rtps_for_vxworks6_wp.pdf 2006-11-17

[5] Wind River Systems Inc, "Memory Allocation in VxWorks6.0"http://www.windriver.com/whitepapersvxworks_memory_allocation_wp.pdf2006-11-17

[6] "RTLinux-3.1 tgz package and RTLinuxFree documentation" http -11-172006

[7] FSMLabs Inc, "*Technical White pages*" http://www.fsmlabs.com/literature.html 2006-11-17

[8] Patricia Balbastre & Ismael Ripoll, "Integrated Dynamic Priority Scheduler for *RTLinux*"http://rtportal.upv.es /apps/edf-sched/rtlinux-edf-sched-1.0/doc/edf-sched.pdf, 06-11-17

[9] Kevin Churnetski, "*Real-time scheduling algorithms, taskvisualization*"http://scholar.google.com/url?sa=U&q=ttp:/ /www.cs.rit.edu:8080/ms/static/swm/2003/2/kmc6820/writeup.pdf,2006-11-17

[10] She Kairui, Bai Shuwei, Zhou Qingguo, Nicholas Mc Guire, Li Lian, "Analyzing RTLinux/GPL Source Code for Education"http://dslab.lzu.edu.cn/docs/Publications/Analyzing_RTLinuxGPL_Source_Code_for_Education.pdf,06-11-17

[11] David Stewart and Michael Barr, "*Introduction to Rate Monotonic Scheduling*" http://www.netrino.com/Publications/Glossary/RMA.html 2006-11-17.