Comparison of CPU scheduling in VxWorks and LynxOS

Henrik Carlgren
henca887@student.liu.se

Ranjdar Ferej
ranfe402@student.liu.se

Abstract

The main objective of a scheduler in a hard real-time system is that tasks are finished before their deadline. A secondary objective is to do this as effective as possible. This comparison will look into how the two well known hard real-time systems; VxWorks and LynxOS handle the CPU scheduling problem. This comparison will also look into reason to why some problems are solved in similar ways and why some are solved in different ways.

Introduction

The CPU scheduling problem in hard real-time systems consists of a few smaller problems, to most of the problems there is a few standardised robust solutions all with their pros and cons. With this knowledge we can assume that the solutions will be quite similar and the differences will depend on the fact that the two products to some extent have different definitions of what are the most important things in a real-time operating system. The algorithms that we describe for each of the real-time operating systems will be described as clear as possible so we can see the differences that are expected to be quite small. Also reason for these differences will be explained as clearly as possible. Another interesting side of this comparison is if there are any significant differences that can be important to take into consideration when choosing a hard real-time operating system for a project with real-time requirements.

LynxOS

The details of the algorithms presented here are based on version 4.2 of LynxOS which conceptually should be the same as version 4.0 but with minor improvements and bug fixes.

The scheduler of LynxOS is preemptive and priority based, this is that the current process is preempted as soon as a higher priority thread is ready to run. If two or more processes have the same priority there is three different ways of handling this; Round-robin, Quantum and FIFO. Round-robin is when all processes get small time-slices of the processor over and over again until all processes is completed. Quantum is very similar to round-robin the only difference is that the length of the time-slice is not fixed; it is a variable for each priority level. FIFO is the first-in-first-out principle that let the processes run until completion; the run order is determined by the time a process became ready to run. The scheduler works with a total of 512 priority levels, 256 for user tasks and 256 for kernel threads.

The LynxOS scheduler schedules both user tasks and kernel tasks together. The kernel tasks are called kernel threads. Kernel threads usually are handlers of different device drivers and their interrupts. Because interrupts have the highest priority in the system they will preempt and block any user task running until the interrupt handler has completed. So having fast interrupt-processing is very important for having a responsive real-time system. So if interrupt handlers can be reduced to spawning kernel threads and then having them scheduled as any other task this will improve system correctness. The main problem with interrupts that take long time to complete is that they might be working for a low priority user task and by working in a interrupt they will steal time from higher priority tasks. By using kernel threads the drawbacks of interrupt handling can be reduced to a minimum. This is because the processing time used on the behalf of other tasks than the highest priority task is reduced to a minimum.

Because user tasks often rely on the work of device drivers they should not block device drivers that it need, this is not having higher priority than the device driver. Still device drivers should not have too high priority so that they block user tasks that are more important than a particular device driver. LynxOS solves this problem with what is called priority tracking. Priority tracking is the method for dynamically changing the priority of kernel threads so that they have a slightly higher priority than the user task with the highest priority that is waiting for the kernel thread. This is done by having one extra priority level attached to every user task priority level and treating that extra level as higher than its parent but lower than the user priority level above the parent. In practice this is done by having 512 priorities where even priority numbers are used by user tasks and odd priority numbers are used for kernel threads. The mapping of the 256 user tasks priorities to the internal 512 priorities is done by simply multiplying the user task priority by two.

A multiprogramming environment needs synchronization mechanisms, LynxOS mainly rely on semaphores, disabling of interrupts and disabling of preemption. Semaphores are used to protect resources by only letting one process at a time work with a resource and having all other process waiting for it to complete. The process with the highest priority that is waiting on the semaphore will be the next to have access to the protected resource. This is the basics of semaphores which introduce a problem with a priority based scheduler; the problem is known as priority inversion. Priority inversion occurs when a resource is held by a low priority task when a high priority task needs the same resource, what happens is that the high priority task have to wait for all processes with higher priority than the low priority task that

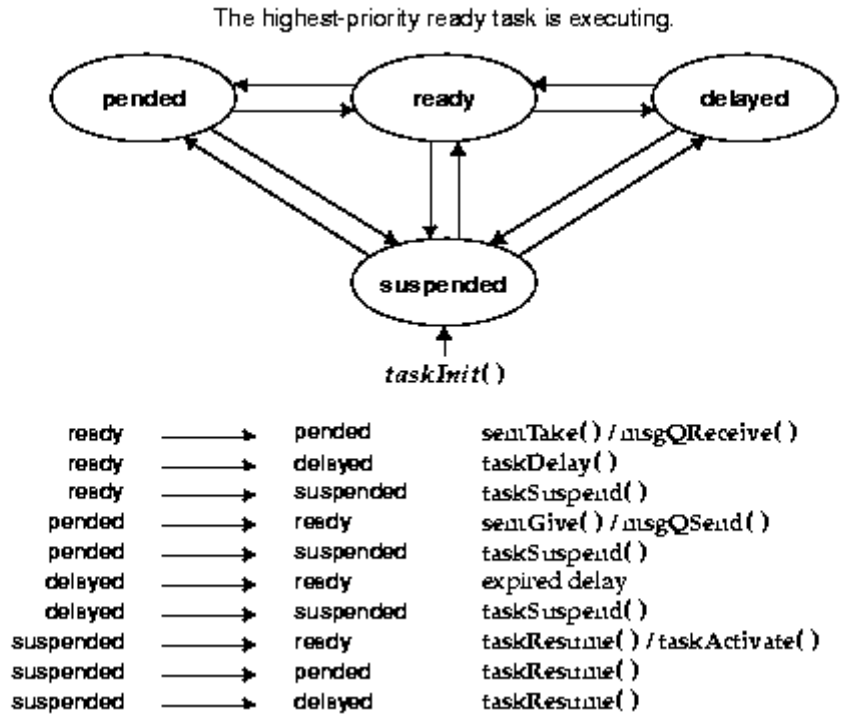
holds the resource. This will have the effect that the high priority task will have the same priority as the low priority task while the low priority task holds the resource. The solution to this problem is to give the low priority task the same priority as the high priority task until it releases the resource; this is known as priority inheritance. Priority inheritance will enforce that processes is run in the order of priority as much as possible. Also disabling of interrupts and preemption can be used to protect resources. If interrupts and preemption is disabled during the critical-section that uses a resource the access to the resource will be atomic; which will ensure a predictable usage of the resource. Even though it is safe to disable interrupts and preemption to access a critical-section this should be avoided because it might introduce unwanted delays for interrupts, this is especially true when a critical-section is takes long time to execute. This kind of blocking is even worse when a low priority process blocks the whole system; the main reason to why this is so bad is that it severely violates the prioritization.

When using two or more semaphores there is a risk for so called dead locks, this occurs when two process want to use the same semaphores and does it in different order. What can happen is that process one holds resource A and process two holds resource B, while both gets to a state where they are waiting for the resource held by the other process. This situation can be solved in two ways either by some dead lock detection and recovery method or requiring that semaphores always is accessed in the same order when multiple semaphores is needed. LynxOS rely on that semaphores always are accessed in the same order in all processes. So there is a risk for dead locks with this approach, but it can only happen if the software running is badly written, but if software is well written (dead lock free at least) everything should work well. The upside of not having dead lock detection and recovery is that there will be no processing overhead and the system should still be dead lock free as long as software is written correctly.

VxWorks

VxWorks is a commercial real time OS and are the most popular RTOS in the world for embedded systems. It is used widely in many different embedded systems like in automobiles, switches and routers and also in two rovers exploring Mars since 2004. The OS was created at Wind River in the early 1980s and are now 20 years later in use in an estimate of more than 350 million devices around the world.

VxWorks is centered around the Wind microkernel. The microkernel has a very little footprint. In VxWorks are processes and threads both refered as *tasks*. The task model of VxWorks consists of four states: READY, PEND, DELAY and SUSPEND.



Figur 1 Task model states of VxWorks [4].

The VxWorks microkernel uses priority based scheduling with two types of scheduling models with 256 priority levels: preemptive and nonpreemptive round-robin (RR) scheduling. Priority 0 is highest and 255 is the lowest. When a task with a higher priority is ready to run the current task running is preempted. The lower priority tasks context is saved and the kernel loads the context of the new task. In preemptive priority based scheduling the FCFS rule is used when tasks with the same priority want to use the CPU while in RR ready tasks with the same priority share the CPU equally.

RR solves the problem with several tasks of same priority wanting to get hold of the CPU. The Round Robin method uses time-slicing to equally share the CPU among tasks with the same priority level. Each task belongs to a group of tasks with the same priority and each of the groups has a defined time slice for CPU-allocation. When a task has consumed its time slice the CPU is relinquished to another ready task in the priority group and the old task moves to the tail of the group queue. This ensures that all tasks of the same priority are allowed to run before a task gets another time slice. When a task is pre-empted by a higher

priority task while it is running in its time slice interval the time count is saved before the context switch so it later can be restored to allow the low priority task to consume the remaining of its time slice.

The scheduler of VxWorks can be explicitly disabled and enabled by the programmer if desired. Interrupt- and preempt locks, and semaphores with priority inheritance are used for synchronization. Semaphores are the fastest method for synchronization and VxWorks provides not only the Wind Semaphores but also the POSIX semaphores for portability reasons.

Similarities

The algorithms and solutions used in both operating systems that are similar will be listed here together with some kind of explanation to why both systems use the same solution.

Both systems has chosen to use a preempting priority-based scheduler, also the available policies for tasks with the same priority is almost the same the only difference is that LynxOS has the Quantum policy which is very similar to round-robin that both systems support, so the difference is neglect able. The reason to why both systems solve this problem in basically the same way is because this solution gives the best CPU utilization in most cases compared to other known robust scheduling solutions.

Differences

The only major difference between the two systems scheduling methods seems to be dynamic time-slicing in LynxOS which VxWorks don't use.

Conclusion

VxWorks and LynxOS have very similar scheduling algorithms, the differences are few and where they differ the effects of the differences are in most cases small to neglect able. This isn't that surprising because both products is supposed to be very high quality hard real-time operating systems and to achieve top performance and quality it looks like it isn't much room for differences in this type of CPU scheduling. This can be view as there isn't any clear reason to choose one or another of these two operating systems when building a hard real-time operating system when considering the scheduling of the operating system. But there might of course be a clear choice if taking other aspects in to consideration that this report doesn't take into consideration.

References

[1] LynuxWorks, “LynxOS User’s Guide”, v4.0,
http://www.lynuxworks.com/support/lynxos/docs/0453-02-los4_ug.pdf

[2] LynuxWorks, “Writing Device Drivers for LynxOS”, v4.2,
http://www.lynuxworks.com/support/lynxos/docs/lynxos4.2/0732-00-los42_writing_device_drivers.pdf

[3] Operating System Concepts, Abraham Silberschatz, Peter Baer Galvin, Greg Gagne, 7:th ed, 2005 John Wiley & Sons.

[4] VxWorks Programmers guide,
http://www.slac.stanford.edu/exp/glast/flight/sw/vxdocs/vxworks/guide/c-basic.html#100061

[5] Wind River, http://www.windriver.com