Memory Management in VxWorks compared to RTLinux

Linköping University
Linköping
2006-11-17

Michael Pettersson, D3D
Markus Svensson, D3D

Abstract

This is a literature study of memory management in VxWorks and RTLinux. We compare the two systems memory management with a focus in memory protection, addressing translation and memory allocation. The report also describes memory management in general, in real-time operating systems.

Both systems are hard real-time operating systems but there are some big differences in memory management.

VxWorks supports virtual memory while RTLinux uses a strict linear memory translation. This makes RTLinux more predictable but at the same time it lacks support of memory protection. A mistake by a programmer can bring the whole system down.

RTLinux only supports static memory allocation while VxWorks supports dynamic memory allocation. This also makes RTLinux more predictable but it may be problems in the future with this approach as programming languages like C/C++ and Java are getting more widely used in real-time system development.

VxWorks 6 support user-mode and kernel-modes while all applications in RTLinux and earlier versions of VxWorks run in kernel-mode.

Table of contents

1 A brief introduction to real-time systems

The reference for the whole chapter, if nothing else is stated, is (Silberschatz, 2005).

A Real-time system is a special kind of computer system often seen in an embedded environment.

What's even more special about Real-time systems is that not just the result of a calculation is important, but also the time it takes to deliver it. But it's not necessarily to have a very quick and fast system. What's important is that you can predict the time it takes to compute a calculation and be sure that it'll be computed before a specified deadline.

Some characteristics of real-time operating systems are:

- Built for single purpose, e.g. playing mp3-files, controlling a fax machine, etc…
- Small size. As they are often seen in embedded devices, the physical space may be limited. This means that a real-time system often lack the CPU power and the amount of memory you can find in a standard desktop computer.
- Inexpensively mass-produced.
- Meeting time requirements

There are three different kinds of real-time systems.

In a hard real-time system a miss of a deadline can have catastrophic consequences. Examples of hard real-time systems are: pacemakers, antilock brake systems and weapon systems.

Soft real-time systems are less restrictive. Here deadlines are important but the system will still work correctly if deadlines are occasionally missed. Examples of soft real-time systems are mp3-players and cameras.

Firm real-time systems are actually just like soft real-time systems but there is no benefit from a computation if the deadline is missed.

Many real-time systems have a battery as the only energy source, and that's why we also must consider the energy consumption of a system. To reduce the energy consumption of a system we may consider removing the memory management unit, reduce performance of the CPU or reduce the amount of memory.

So, there is a need of a small amount of memory, both considering the energy perspective above and a constrained physical space. This leads to certain requirements on the operating systems running. It needs a small footprint (memory needed to run the operating system). The lack of a memory management unit, the need of small footprints and the demand of finishing a calculation before a specified deadline are all reasons, among others, why we cannot use a general purpose operating system as Windows or Linux for a real-time system.

1.1 Memory Management in real-time operating systems

As this report is considered to investigate memory management in different real-time operating systems, we need to know a little about what is so special about memory management in an RTOS (real-time operating system) environment.

As we can see in the chapter above the physical space and power consumption constraints may do that we cannot use a memory management unit (and therefore, cannot implement a full virtual memory environment). How do we do address translations?

There are three main approaches for address translations.

The first is to use real-addressing mode. Here we don't have any virtual memory techniques at all, just stating that the logical address is the same as the physical. Of course we don't need any memory management unit for this approach and it's fast, no time is spent for address translations, but it has some disadvantages. The programmer needs to know where the program is located in the memory and there is no memory protection between different processes. Despite these disadvantages, this is a common implementation in embedded systems with hard real-time constraints.

The second approach is to use a relocation register and just state that the physical address is equal to the sum of the logical address and the value of the relocation register. The address translation takes some time but the time is predictable, which is more important. The programmer doesn't need to know exactly where the program is located in the memory, as the relocation register will be loaded with the starting address of the program. The drawbacks here are that it's not that fast as real-addressing mode and we still have no memory protection.

The third and last approach we'll investigate is a full virtual memory functionally. Now the address translation is made by using page tables and a translation look-aside buffer (TLB). The benefits of this approach is the fact that we now can load programs at any location in the memory and not in blocks as in the other approaches, it also provides memory protection between different processes. But there are drawbacks… This approach demands a memory management unit, which is not always available as we discussed earlier. There are lots of real-time systems without disk drives, and therefore swapping may be a problem (can be solved by using flash memory). The address translation takes more time than it does with the other approaches and more important, the time spent on address translation is not predictable (consider a miss in the TLB).

2 VxWorks

VxWorks is an operating system from *Wind River Systems* for hard real-time systems, and is a very popular RTOS used in automobiles, consumer devices, network switches and routers, etc. VxWorks is also used in the two rovers *Spirit* and *Opportunity* that began exploring Mars in 2004. (Silberschatz, 2005)

According to *Wikipedia* the name VxWorks is believed to come from another RTOS named VRTX. In the early eighties VRTX didn't worked very well and *Wind River* made an extension that made it better. The name VxWorks is believed to stand for "VRTX that Works". Later *Wind River* developed a new kernel and used it instead of VRTX.

The VxWorks system is organized around the *Wind microkernel* with minimal features. To add more features such as networking, file systems, memory management etc. specialized libraries are included. By this approach the system will only have the features that are actually needed and the footprint will be as small as possible, which is desirable for real-time systems. (Silberschatz, 2005)

2.1 Memory Management in VxWorks

The latest version of VxWorks (v6) has some improvements comparing to VxWorks 5 and earlier, considering memory management. Here we'll discuss the details about memory management in both VxWorks 6 and VxWorks 5.

2.1.1 Memory protection

In earlier versions of VxWorks all applications are run in kernel mode, as there is no difference between user and kernel modes. That gives all application access to the whole address space. This can lead to problems considering memory protection. To solve this issue in VxWorks 5 and earlier the solution is to include virtual memory library called VxVMI, allowing applications to mark data areas private and make it only accessible by the tasks it belongs to. The problem with this approach is that you will need a memory management unit to use this library. (Silberschatz, 2005)

VxWorks 6 has a different approach but is still supporting the approach above for backward compatibility. VxWorks 6 are now supporting user-mode and kernels-modes. The kernel runs in kernel mode and VxWorks real-time processes (RTPs) are running in user-mode. The kernel is protected from the RTPs and the RTPs are protected from each other. This approach needs, as well as the approach for VxWorks 5, a memory management unit.
(Wind River, 2006a)

2.1.2 Addressing translation

Earlier versions of VxWorks use a non-overlapping addressing model (real-addressing mode). VxWorks 6 uses another model with full virtual memory support, but is, again, still supporting the older model.

The benefits of letting VxWorks 6 supporting the earlier model are: it's more determinism as no translation tables are required, it'll provide backward compatibility for existing drivers and applications and no memory management unit are required (supported though). (Wind River, 2006a)

Without the VxWorks 6 model, we cannot use the memory protection with user- and kernel-mode described above.

2.1.3 Memory Allocation

One issue in memory management is fragmentation. As C++ and Java are getting more widely used in device development, the performance of allocating memory dynamically is more important. Earlier versions of VxWorks are using a first-fit allocation algorithm. All free memory blocks are stored in a linked list. When allocating memory, the algorithm simply walk through the list to find a free block big enough. (Wind River, 2006b)

For static allocation without freeing memory this approach is very fast, but if we are dynamically freeing and allocating, the memory will be much fragmentized. Larger blocks will be split in two, and the free block left will over time get very small. These smaller blocks will most likely never be used, and allocating means we have to walk over these small blocks every time because they are too small for the first-fit algorithm. This leads to bad worst-case and average performance. (Wind River, 2006b)

Instead the best-fit algorithm is used in VxWorks 6. Now all free blocks are stored in a binary tree, sorted according to their size. When allocating it simply traverse the tree, looking for a free block big enough, after splitting the block in two, the free block left over will be added to the binary tree. To maintain the binary tree is a little more complex and needs a little more overhead than it does for the first-fit algorithm, but it has a much better time complexity. (Wind River, 2006b)

The probability of splitting a larger block is smaller than for the first-fit algorithm and the time complexity for allocation is much better O(log(n)) for the best-fit algorithm, compared with O(n) for the first-fit algorithm. We have much better worst-case performance and the best-fit algorithm has more deterministic allocation time. (Wind River, 2006b)

3 RTLinux

RTLinux is an operating system from FSMLabs for hard real-time system. It's used to test jet engines, automated telescopes, control robots, autonomous underwater vehicles, etc. (FSMLabs, 2006)

3.1 General information

There are two techniques to develop Linux into a hard real-time operating system. One way is to extend an existing kernel with real-time operations to make it possible to write applications with strict time constraints. One of these extensions is called RTAI, Real-Time Application Interface. (Yodaiken, 1997a)

The second way is to add a real-time kernel that works as an abstraction layer between the regular Linux kernel and the hardware. This means that the regular Linux kernel will have no direct communication with the hardware. (Yodaiken, 1997a)

This report will focus on the second technique of constructing a real-time operating system based on an existing Linux kernel. One of the most common real-time Linux systems that use this second approach is RTLinux, Real-Time Linux, and this chapter will focus on this system. (Yodaiken, 1997a)

3.2 Why isn't Linux suitable for real-time?

Why can't we use regular Linux to handle real-time tasks? Listed below are some of the reasons of why regular Linux isn't suitable for real-time tasks.

1) System calls aren't preemptible. A process currently running cannot be pre-empted before it's ready to leave the kernel. (Abbott, 2003)
2) The process of swapping pages in and out of memory is, for practically use, made unbounded. This means that we can't predict how long time it's going to take to swap the page in/out of the memory, so we can't predict the delay. (Abbott, 2003)
3) The scheduler can at any time run a task with low priority even though there are high priority task ready to run. This might happen if the low priority tasks have been waiting for a while. (Abbott, 2003)

3.3 The basic idea behind RTLinux

The intention when constructing RTLinux was to make it able to run common programs while still permitting real-time functions to operate in a predictable and low-latency environment. The developers of RTLinux choose to extend an existing kernel into a real-time operating system, which simply runs the regular non-real-time kernel at the lowest priority. Real-time instructions runs with higher priority and the system are fully preemptable.
(Yodaiken, 1997a)

One big problem with the regular Linux kernel is that it can, at any time, suspend any user-level task if it has exceeded its given time. This means that a critical process can be suspended, which can have terrifying consequences. (Yodaiken, 1996)

In RTLinux a new abstraction layer is added, called the virtual machine. This layer is placed between the hardware and the regular Linux kernel. For the standard kernel this new real-time

kernel appears to be hardware. Any interrupts made by the regular kernel is intercepted by the real-time kernel and is only executed when the real-time kernel is idle. The new real-time kernel also introduces its own fixed-priority scheduler. (Yodaiken, 1997a)

3.4 Predictability in RTLinux

Predictability in real-time operating systems is very important otherwise we can't set any time constraints. No time constraint means that it can not handle tasks, which requires a hard real-time operating system. To obtain predictability, RTLinux is constructed so that the regular Linux kernel can't, in any way, delay the real-time operating system. At the same time the real-time kernel is very small and simple which reduces the chance of errors and makes the system delays more predictable. (Fu, 1999)

RTLinux only offers primitive tasks with only statically allocated memory, no memory protection, a simple scheduler containing no protection against impossible schedules, a shared memory as the only synchronization primitives between real-time tasks, and a limited range of operations on the first-in-first-out queues connecting real-time tasks to Linux processes. The real-time kernel is constructed in this way to minimize unpredictable delays. (Yodaiken, 1997b).

A side effect of that RTLinux doesn't have any memory protection or virtual memory is that it becomes less secure. A real-time task can in fact bring down the whole system.

3.5 Memory Management in RTLinux

All regular (non-real-time) processes are handled by the regular Linux kernel. In this report we want to focus on the real-time kernel so information about the regular Linux kernel's memory management is omitted.

RTLinux is build to use a processor with support for a memory management unit, but there is no memory protection implemented. (OCERA, 2002)

3.5.1 Memory protection

The real-time tasks run in kernel space which means that response times are very short. On the other hand there is no memory protection in the kernel so it becomes less secure. (OCERA, 2004)

3.5.2 Addressing translation

As mention above the RTLinux kernel has no memory protection at all and it also doesn't use virtual memory. This means that the logical address is the same as the physical. This is called real-addressing. All this for the purpose of making the real-time kernel as predictable as possible and reduce the chance of unpredictable delays. (OCERA, 2004)

3.5.3 Memory allocation

No dynamic memory allocation is available at all. All memory that a thread want to use must be allocated before the real time thread starts. (Yodaiken, 1997b)

4 Comparison of VxWorks and RTLinux

It's finally time for the highlight of the report, the comparison of memory management in VxWorks and RTLinux.

4.1 Memory protection

All versions of VxWorks are supporting memory protection in some way. VxWorks 6 does it by providing full virtual memory functionality and VxWorks 5 and earlier does it by marking special data areas as private using the VxVMI-library. Both of these approaches demand a memory management unit. The support of memory protection can be disabled making VxWorks work without a memory management unit.

All though RTLinux is build to be used together with a memory management unit; it has no support of memory protection at all.

The drawback of not supporting memory protection is that just one small failure from the programmer of an application or device driver can actually bring down the whole system.

The drawback of supporting memory protection is that it demands a memory management unit and the time spent for address translation may be unpredictable.

4.2 Address translation

Both VxWorks and RTLinux are using real-addressing mode (VxWorks 6 is also supporting full virtual memory functionality), allowing fast address translations as the logical address is simply equal to the physical address.

One of the benefits of using real-address mode translations is that there is no need of a memory management unit. More information about real-addressing mode can be found in the section about memory management in the first chapter of the report.

4.3 Memory allocation

VxWorks 5 and earlier are actually supporting dynamic allocation, even if it's not very efficient, as we already investigated. VxWorks 6 solves this issue by using the best-fit allocation algorithm, which is more deterministic than first-fit used in VxWorks 5.

RTLinux lacks the support of dynamic allocation and all memory used in a real-time task must be allocated when the task is starting.

The benefit of not supporting dynamic allocation is that simpler allocation algorithms can be used. But the lack of dynamic allocation support also results in no support for programming languages as C++ and Java, which is getting more widely used in real-time system development.

5 Conclusion

According to the comparison we have made, it seems like VxWorks have better support for more complex memory management as virtual memory and dynamic memory allocation.

It's surprising that RTLinux aren't supporting more complex memory management than it does. RTLinux is supposed to be built for processors with a memory management unit (probably because it's running on top of a general Linux kernel), but it isn't supporting any management models that demand a MMU.

The reason for this could be that dynamic allocation and virtual memory along with other complex models are still considered not enough predictable for use in a real-time system. As dynamic allocating programming languages as C++ and Java are getting more widely used in real-time system development it's important to develop a deterministic allocation algorithm.

Even if dynamic allocation never can be as deterministic as pre allocation done when a task is starting, it feels like the people at Wind River are looking into the future with VxWorks 6 by providing and developing support for dynamic allocation.

References

Printed references

Abbott, Doug (2003). *Linux for embedded and real-time applications*, Elsevier Science Publishers

Silberschatz, Galvin and Gagne (2005). *Operating System Concepts, 7th ed.* John Wiley & Sons

Online References

FSMLabs (2006). Hard Real Time Case Studies [www] <http://www.fsmlabs.com/case-studies.html> 2006-11-17

Fu, Kevin (1999). An Interview with Victor Yodaiken [www] <http://www.acm.org/crossroads/xrds6-1/yodaiken.html> 2006-11-17

OCERA (2002). RTOS State of the Art Analysis [www] <http://mnis.fr/ocera_support/rtos/book1.html> 2006-11-17

OCERA (2004). Memory protection in a threaded system [www] <http://www.ocera.org/archive/upvlc/public/reports/memory-protection/memory-protection.pdf> 2006-11-11

Wind River (2006a). VxWorks 6 [www] <http://www.windriver.com/products/product-notes/vxworks6-product-note.pdf> 2006-11-14

Wind River (2006b). Memory Allocation in VxWorks 6.0 [www] <http://www.windriver.com/whitepapers/vxworks_memory_allocation_wp.pdf> 2006-11-14

Yodaiken, Victor (1997a). The RT-Linux approach to hard real-time [www] <http://rtlinux.lzu.edu.cn/documents/papers/whitepaper.html> 2006-11-13

Yodaiken, Victor (1997b). An Introduction to RTLinux [www] <http://www.linuxdevices.com/articles/AT3694406595.html> 2006-11-12

Yodaiken, Victor and Barabanov, Michael (1996). Real-Time Linux [www] <http://rtlinux.lzu.edu.cn/documents/papers/lj.pdf> 2006-11-14