A comparison between FreeRTOS and RTLinux in embedded real-time systems

Karl Andersson (karan496@studeasdasdasdasdant.liu.se)
Robert Andersson (roban197@student.liu.se)

2005-11-18

Abstract

This report compares two real-time system in different sizes more precise RTLinux and FreeRTOS. We start up with a listing of different implementation and feature in the two system and move on to compare the differences in them. The thought of the report is to be used as a guide when choosing between different sized operating systems. Not entirely surprising we reach the conclusion that you should choose the right system for the right project. Why and what makes something the right system is some of the question we try to answer in the report. We find that FreeRTOS is a very small operating system, making it suitable for small applications on small platforms. RTLinux is more suitable when more complexity, scalability and processing power is needed.

Contents

1 Authors and Affiliation

Karl Andersson (student LiTH) karan496@student.liu.se
Robert Andersson (student LiTH) roban197@student.liu.se

2 Introduction

For many years computers have been embedded in all kinds of devices. They are controlling our microwave ovens, helping us drive our cars, printing our documents and doing many other things. In many of these devices it is important that the job is done in a specified time, thus making it a real-time system. Often running in these real-time systems are real-time operating systems. They provide an easy way of constructing a real-time system without having to reinvent the wheel. There are many different real-time operating systems available today, both commercial and open source. Therefore selecting one can be a difficult task.

In this report we compare the two real-time operating systems FreeRTOS and RTLinux. They are both open source and interesting from a technical point of view. We begin by giving an introduction to each operating system in sections 3 and 4. In section 5 we give a short definition of what an embedded system is. Section 6 is the main part of this report. Here we compare different features of the two operating systems. Namely: Size, platform support, features, scalability, schedulers and ease of development. In section 7 we conclude what we have written earlier, and give a guide to when to select each operating system.

3 Introduction to FreeRTOS

3.1 What is FreeRTOS?

FreeRTOS is a free real-time operating system kernel. It is aimed at small embedded real-time systems. Since most of the code is written in the c programming language, it is highly portable and has been ported to many different platforms. Its strength is its small size, making it possible to run where most (or all) other real-time operating systems won't fit.[1]

3.2 License

FreeRTOS is licensed under the gnu General Public License (GPL), with an exception. If you link FreeRTOS to other independent modules using only the FreeRTOS API interface, you can distribute your code under different licenses than the GPL. This exception makes it possible to use FreeRTOS in commercial applications without paying any royalties. You can of course also use it without the API restriction if you use the GPL license on your own code.[6]

3.3 Platforms

FreeRTOS is ported to many platforms. A few of these include:

- Different versions of the ARM processor

- Microchip PICMicro
- Atmel AVR (MegaAVR)
- Motorola/Freescale HCS12 Processors

For each supported platform, the code includes a demo project demonstrating how to use the code on that specific platform.[7]

3.4 Developers

FreeRTOS is developed by Richard Barry. Some ports to different platforms are also contributed from third parties.[7]

3.5 Design

According to its web page[3], FreeRTOS is designed to be:

- Simple
- Portable
- Concise

This means that unlike most operating systems, FreeRTOS has very few features. Mainly three functions are provided by the kernel:

- A scheduler, for scheduling tasks (Preemptive or cooperative)
- Message queue handling
- Semaphores for synchronization

3.6 What can it be used for?

A typical use of the FreeRTOS kernel is when one needs to have some kind of (time critical) control algorithm while also providing user-control and sensor monitoring. Propose we have a system with some sensor that needs an amount of processing which leads to a control signal (say a water flow). It's critical that we measure at controlled timings to make sure that there isn't an overflow. This will be our highest priority process. We then have an LCD-display which must be updated with the current flow and how much to put in each bottle. Also we might have an keypad that needs to be read so we can configure the system in some way. We now have three tasks that needs to be executed with different priority.

The FreeRTOS solution to this problem might be an AVR with three different tasks each of them given a specific priority. Then we schedule the tasks to run at different intervals. The preemptive scheduler of FreeRTOS makes sure that the most important task (the control loop) is always run at time. We now have a small system that performance this task.

4 Introduction to RTLinux

4.1 History and licensing

RTLinux was originally developed by V. Yodaiken at the New Mexico Institute of Technology[16]. It was later released by FSMLabs and a GPLed version is currently maintained at rtlinux-gpl.org. The GPL-version is free to use in any GPLed project while the FSMLabs *pro version* can be used for a cost in non-GPLed commercial projects.

4.2 What is RTLinux?

RTLinux strives to integrate a non real-time OS with a real-time OS. Since it is hard to get a non real-time OS to work nicely together with real-time tasks RTLinux instead separates the non-real-time system from the hardware and acts as a layer in between. This makes it possible for RTLinux to run real-time tasks with very low latency and very high predictability[12]. Linux itself is run as the idle-process in RTLinux and is therefore only run when there is no real-time task that needs to be run. You then get the nice tools in Linux we have all grown to love like ps, python, Perl(?) and so on while still being able to load real-time tasks such as data gathering processes.

4.3 Platforms

RTLinux supports platforms like x86, PowerPC, ALPHA and others.

4.4 What can it be used for?

RTLinux being a bigger system than FreeRTOS would have a more common use where more CPU heavy processing is required. One can also see the good use of already available tools in Linux. Propose we have a video surveillance of something, perhaps a moving part that must move. We here use RTLinux to take the picture in from some kind of camera then we process these pictures to make a control signal. Perhaps whether to emergency stop or not. This is a somewhat critical task therefore we assign this to a task in RTLinux. We then want the possibility to extract the videos to a monitor, this can easily be accomplished by the tools in Linux. We simply let our rt-task take the picture and after it has processed it and given the control signal it sends the picture through a FIFO to Linux. We then use Linux to send this feed over TCP/IP to some other computer. By separating in this way we get a clean division between what is real-time and what is just complex code.

5 What is an embedded system?

5.1 Definition

It doesn't seem to exist any hard definition of what an embedded system is. Wikipedias definition is, however:

"An embedded system is a special-purpose computer system, which is completely encapsulated by the device it controls. An embedded system has specific requirements and performs pre-defined tasks, unlike a general-purpose personal computer."[15]

5.2 Examples

Even given the above definition, it is hard to say whether a system can be regarded as embedded or not. PDA's, for instance, were originally regarded as embedded systems. Today, however, PDA's are becoming more and more versatile and often perform more than a few pre-defined tasks. Some examples that can (at least in most cases) be considered embedded are:

- Photocopying machines
- Hard disk drive controllers
- Fuel injection regulators for car engines
- Microwave ovens
- Printers

5.3 Platforms

An embedded system can be implemented on many different platforms, ranging from small microcontrollers to large multi-processor environments.

6 Comparison

6.1 Size

One major difference between FreeRTOS and RTLinux are their sizes. FreeRTOS running on an AVR has a footprint (the amount of ROM used) of approximately 4.4 kilobytes.[4] RTLinux on the other hand is relatively scalable. The Linux kernel can be stripped of functionality you don't need. But even given that, the footprint is generally measured in megabytes (or in some rare cases hundreds of kilobytes).

When it comes to RAM requirements Linux can be scaled down to only use a few megabytes. In practical applications, however, you often need more than that. That is because you probably want to run some tasks under the Linux kernel, not just the Linux kernel itself).FreeRTOS has much smaller requirements when it comes to the amount of memory. A very simple setup, running two tasks, a scheduler, a queue for communication and a semaphore on an 8 bit architecture would use in the vicinity of 200 bytes (excluding of course the amount of memory that the tasks themselves need).[5]

6.2 Platform support

Because of the difference in size, the two operating systems supports different platforms. While RTLinux supports architectures like x86, FreeRTOS generally runs om smaller microcontrollers. FreeRTOS supports a greater number of platforms than RTLinux does. This is probably because one goal of the FreeRTOS project is to write portable code and the fact that all kernel code is contained in just three files.[10] RTLinux on the other hand is much more complex (much due to the Linux kernel) and therefore much harder to port to new platforms.

6.3 Features

While size is a major difference between FreeRTOS and RTLinux, so are the functions they provide. FreeRTOS is a very minimalistic operating system. It provides only some basic scheduling, inter-process communication (IPC) and semaphores for synchronization.[3] RTLinux on the other hand can provide all the things that a normal Linux distribution can. This includes networking support, graphical environments (X11), web servers and much more. These features are, however, run in the Linux kernel. That means that they are not strictly speaking part of the real-time system.

While the separation of real-time and non real-time tasks has advantages, it also introduces some problems. Real-time tasks and normal tasks will at some time need to communicate with each other. This can be done in many ways, but a common way is to use a FIFO (First In First Out) buffer. In this case, however, a normal FIFO buffer will not suffice. Say, for instance, that a real-time task is writing to the buffer while a normal task is reading from it. If the buffer becomes full, the real-time task can't be blocked until the non real-time task has made room in the buffer (which is how a normal FIFO-buffer would be implemented). To overcome this problem RTLinux provides a real-time FIFO. The difference between a real-time FIFO and a normal FIFO is that on the real-time side of it the calls to push data into the buffer are non-blocking and allows data to be overwritten.[18] Beside semaphores RTLinux also provides condition variables and mutexes for synchronization.[14]

6.4 Scalability

Scalability in systems is getting increasingly more important today. The authors thinks that scalability is and will be one of the most important aspect of future systems. Therefore it's interesting to look into this matter when choosing between real-time systems.

6.4.1 FreeRTOS scalability

Since FreeRTOS is a small real-time system it's hard for it to scale beyond the target of the platform. Therefore we only choose to look how it scales in the platform. When using the preemptive scheduler it gives you the possibility of adding more tasks to the system easily. If these tasks are idle priority (lower then all other) one can even add them without effecting the rest of the system.[8]

6.4.2 RTLinux scalability

Even though RTLinux can't be scaled all the way down to really really small platforms it can scale down to ARM. Upwards it scales even better allowing you to run it all the way up to full grown "home computer systems". The scheduler being of simple highest priority first allows adding of more idle tasks without effecting the rest of the system. More advanced type of schedulers like EDF and rate monotonic scheduler allows for adding more tasks with the system itself verifying whether it will work or not.[11]

6.5 Scheduler

The scheduler in an RT-system is very important in the means to ensure that tasks completes before their deadlines. In contrast to a regular scheduler for a general purpose system it is not the main task of the scheduler to ensure 'fair' distribution of CPU-time. Instead it is critical that tasks gets to run enough to complete before their deadline. To ensure that this happens there are many different approaches. A commonly used method is simply to let the task with highest priority run before all tasks with lower priority. This works for a soft real-time system but for hard real-time it's necessary that the system provides a better guarantee then a "best-effort".[11]

6.5.1 FreeRTOS scheduler

FreeRTOS uses a highest priority first scheduler. That means that the task with highest priority is always run. This is achieved by having a preemptive scheduler that at a tick-interrupt decides if the current task is allowed to continue executing or if it needs to be switched for another task. To be able to do this FreeRTOS demands that every task is assigned with a priority. The scheduler then uses this priority to schedule the task with highest priority. Tasks that have the same priority is given "fair" process time by round robin.[9] We can see the advantages of having such a simple and small scheduler for a small embedded system. Still one loses the possibility to achieve hard real-time by not having any kind of deadline based scheduling. The developer is given the choice of having a preemptive or a cooperative scheduler. In preemptive mode a task can be preempted unlike in cooperative mode where it's up to all tasks to give away the CPU "often" enough so higher priority tasks get to run.

6.5.2 RTLinux scheduler

In RTLinux the developer is given a lot of choice when it comes to scheduler. The modular way that RTLinux is built in makes it easy to change different parts of the system. A simple insmod gives the possibility to change scheduler. RTLinux comes with a couple of schedulers designed for different things. First of all it has a basic highest priority first scheduler that simply uses the priority of a task and schedules it first. The task itself then releases the CPU when done or gets preempted if a task with higher priority needs to run. RTLinux also implements something of a "earliest deadline first" originally written by Ismael Rippol.[13] Earliest deadline first is implemented using the periodic feature of RTLinux. Assuming that every tasks deadline is when it is next to be run again one can implement a fast EDF. The earliest deadline first scheduler is in theory optimal since it (in theory) can schedule tasks to 100% CPU-usages. In practice this however is not the same due to overheads.[11] As it idle process RTLinux runs a usual Linux kernel. This give the effect that when there is no rt-tasks that can run Linux gets to run. Of

course this could very easily lead to starvation of Linux and thus effectively disabling Linux. But since the only real importance of a real-time system is to run the real-time tasks this isn't really a problem.

6.5.3 Comparison

Now lets look at the difference between the two systems. Both systems implement the simple highest priority first scheduler but since FreeRTOS targets the smaller CPUs and not full grown PC-systems RTLinux got the advantage to have a lot more CPU and memory for the scheduler. Therefore its easy to see why RTLinux can implement the more advanced schedulers like EDF and rate monotonic scheduler. Also there might not be the need for a more advanced scheduler in the FreeRTOS-size since the tasks running will be of pretty simple kind.

Some points:

- RTLinux scheduler give the possibility for hard real-time scheduling
- FreeRTOS has a simple highest priority first scheduler
- FreeRTOS has MUCH less scheduling overhead
- Both are preemptive

6.6 Ease of development

The development of real-time applications in FreeRTOS is fairly straight forward. Since most of the operating system is written in the c programming language, so are the real-time tasks themselves. Per default only one task exists in the system. This is the idle task. Any other tasks need to be created. When created, the task is assigned a priority number. Higher priority number means that the task is run before all tasks with a lower priority number. The idle task has priority number 0. All tasks are written as functions that, by convention, never should return. Therefore they are generally written as an infinite loop. The other functions implemented in FreeRTOS (semaphores and message queues) are also very easy to use. When using semaphores, however, it is up to the developer to make sure that no low-level tasks blocks higher level tasks.[2]

In RTLinux real-time tasks are written as kernel modules. In RTLinux, the development is divided into two parts, the time-critical tasks running as real-time threads and the non time-critical tasks running as normal pthreads in Linux. Much of the work when writing RTLinux code seems to be establishing some sort of communication between the real-time code and the non real-time code. See the comparison of features for a description of the real-time FIFO. Writing the non real-time code is very much like regular application development in Linux. When it comes to the real-time code it is very different. Although the normal Linux API functions are available for the real-time code as well, the use of these often introduce a source of unpredictability and should be avoided.[17]

7 Conclusions

When choosing a real-time operating system it is important to know what features you need. Our comparison of FreeRTOS and RTLinux has shown that two real-time operating systems can in fact be very dissimilar.

One of the biggest differences between FreeRTOS and RTLinux are their sizes. One thing that may need to be considered is what platforms are available for the project. The platform choice is often directly related to the processing power needed or restrictions when it comes to physical size. FreeRTOS supports many small processors and microcontrollers. If a microcontroller can do the job at hand, then FreeRTOS is probably the way to go. On the other hand, if much processing power is needed, the platform of choice is probably something larger like an x86 system. Then RTLinux is a good choice.

Beside from their platform support, the two operating systems are very different in what features they have. FreeRTOS is very simple, and though that can be considered a bad thing, a small project can probably benefit from the simplicity. Large projects, however, will profit from the extra functions in RTLinux.

Many real-time projects often include one part that is time-critical and one that isn't. If the non time-critical part is a relatively large part of the project then RTLinux has many thing to offer. If a network connection, a graphical interface or something else fairly complex is needed, RTLinux definitely has the upper hand.

One last thing to consider is the economics. In a commercial project, the ability to use the GPL license doesn't always exist. In this case it can be worth noting that FreeRTOS can be used in a commercial project without paying royalties, while RTLinux can't.

The scheduler in the two system is very different in the aspect that FreeRTOS is smaller and more simple while RTLinux of course has a bigger and more complex scheduler. Although FreeRTOSs scheduler works good with limited amount of predefined-tasks (an embedded system). It or you will probably run in to troubles when trying to do thing close to the system limit. This is because there is no more advanced scheduler than highest priority first. Both systems might run into problems when the idle-process get starved. Though not necessarily problems with the RT-task but problems like logging and controlling. Especially if RTLinux is used for data processing.

If you have a small well defined system where an AVR-sized microcontroller will suffice FreeRTOS probably is the way to go. But if you have a bigger and likely expanding system RTLinux will give you the possibility to expand it more easily. Though one must think about that FreeRTOS it being a simpler system your engineers will probably get a more system-wide understanding making debugging and development easier.

To conclude: Use the right system for the right project.

References

[1] Barry, Richard *FreeRTOS.org*
<http://www.freertos.org/main.html>

[2] Barry, Richard *FreeRTOS.org - API*
<http://www.freertos.org/a00106.html>

[3] Barry, Richard *FreeRTOS.org - Design*
<http://www.freertos.org/a00014.html>

[4] Barry, Richard *FreeRTOS.org - FAQ*
<http://www.freertos.org/FAQMem.html#ROMUse>

[5] Barry, Richard *FreeRTOS.org - FAQ*
<http://www.freertos.org/FAQMem.html#RAMUse>

[6] Barry, Richard *FreeRTOS.org - License ans Warranty*
<http://www.freertos.org/a00114.html>

[7] Barry, Richard *FreeRTOS.org - Ports*
<http://www.freertos.org/a00090.html>

[8] Barry, Richard *FreeRTOS.org - Real Time Application Design*
<http://www.freertos.org/tutorial/solution2.html>

[9] Barry, Richard *FreeRTOS.org - RTOS Implementation*
<http://www.freertos.org/implementation/index.html>

[10] Barry, Richard *FreeRTOS.org - Source Organization*
<http://www.freertos.org/a00017.html>

[11] Silberschatz, Galvin & Gagne *Operating System Concepts 7. ed.* p. 704-707

[12] Yodaiken, Victor *The RTLinux Manifesto*
<http://www.systematic.co.za/datasheets/rtlmanifesto.pdf> chapter 4

[13] Yodaiken, Victor *The RTLinux Manifesto*
<http://www.systematic.co.za/datasheets/rtlmanifesto.pdf> chapter 5

[14] Ripoll, Ismael *RTLinux versus RTAI (2002)*
<http://rtportal.upv.es/comparative/rtl_vs_rtai.html>

[15] *Wikipedia - Embedded system*
<http://en.wikipedia.org/wiki/Embedded_system>

[16] *Wikipedia - RTLinux*
<http://wikipedia.org/wiki/RTLinux>

[17] Sherer, Matt *Writing Applications with RTLinuxPro*
<http://www.fsmlabs.com/writing-simple-applications-with-rtlinux-2.html>

[18] Sherer, Matt *Writing Applications with RTLinuxPro*
<http://www.fsmlabs.com/writing-simple-applications-with-rtlinux-3.html>