Overview of Real Time Operating Systems
Fall 2012 Independent Study

Ian O'Hara

October 2, 2012

At UPenn's Modlab (GRASP Subsidiary)
Advisor: Dr. Mark Yim

1 DEFINITIONS

The basic terminology of real time systems is taken from [1]. Unless otherwise noted, assume these definitions can be found there.

System A mapping of a set of inputs into a set of outputs

Response Time The time a system takes to map a set of inputs to the corresponding set of outputs

Real Time System (RTS) A system where there is a bound set on the response time of a system. When this bound is not met, the system is considered in a failed state.

Hard Real Time System (HRTS) A RTS in which failure to meet the response time bound leads to catastrophic failure. IE: Response times must be deterministically met.

Soft Real Time System (SRTS) A RTS in which the response time bounds must be met the majority of the time. IE: Performance is degraded, but not destroyed, when response times are not met.

Process Preemption A process preemption is the suspension of a process so that another higher priority process can run. This can occur for a number of reasons, and is essentil in RTS.

It is importantant to note that while RTS often deal with "fast" times, a RTS does not need to be fast. It simply needs to have explicit bounds on response times.
Another phrasing from [2] is that a Real Time Application is one in which there are operational deadlines between some event being triggered and the applications response to that event. The use of a Real Time Operating System (RTOS) gives the programmer calculated (hard real time) or measurement predicted (softer real time) response times.

2 INTRODUCTION

3 WHAT IS REAL TIME

The definition of real time is ambiguous in scope. Almost any problem can be casted as a "real time" problem (eg: shipping a package across the US can be considered real time if you need it to be there in 5 days).
In the context of robots, real time demands are typically associated with sensor input and control loops. EG: Read this sensor every 1ms and run a control loop at 100us.

4 WHY

Why should we bother with Real Time Operating Systems (RTOS)? This needs to be answered and justified, and can be done by looking at the different ways in which we can program, in this case, robots.
On the one hand we have high level generic operating systems, like Linux, which provides a mechanism for:

a. Running many independent processes

b. Memory management and safety. Once process in user space cannot corrupt the memory of another process or the kernel (as long as the system has an MMU, and most systems running Linux do).

c. Interfacing with many different pieces of hardware without having to write low level device specific code (i2c bus, spi bus, etc all are just simple entities we can use)

d. Developing in-situ

e. Using many different programming languages

f. Taking advantage of open source code without worrying about it being specific to your hardware

However, along with the benefits of abstracting away all of the low level details for us, a generic operating system does not normally care about meeting timing deadlines. For instance, a desktop user does not care if their mouse position is updated exactly once every 1 ms. If the update time fluctuates between 100ms and 100us, the user will not notice a difference. The same goes for most typical uses of generic operating systems; scientific number crunching, web browsing, text editing, and many others do not need consistent timing. The "real time" general use computers face is video games which require consistent 30-60hz update rates. However, even in this case, the failure to meet the deadlines does not result in catastrophic failure; just lag.

On the opposite side of the spectrum is the microcontroller approach. In this scenario, the programmer himself "writes" the operating system. He is responsible for coordinating and scheduling everything that happens. This can be nice because it allows for absolute control of what happens when - timing is in the programmer's hands, and has the resolution of the microcontroller's clock. However, the advantages of the absolute control of a microcontroller are accompanied by many disadvantages:

a. You must develop offboard, cross compile, and then use a binary loading mechanism to load code onto the microcontroller.

b. The programmer is responsible for everything.

c. Device interfaces are at the lowest level. Most microcontrollers have basic libraries in the wild, but the platform specific nature of microcontroller programming makes for many platform specific libraries that accomplish the same funcionality. This results in a lot of un-maintained dead libraries. **TODO: Justify and source**

d. The programmer must know the minutia of any microcontroller used.

e. Switching hardware often involes re-writting significant portions of the code since it is very hardware specific.

f. There is no memory management - an errant memory access can change anything. This includes mux registers, timer registers, or literally anything that has a memory address.

The middle ground is the RTOS which provides the advantages of having an operating system and the advantages of having the guaranteed timing of a microcontroller. However, this comes with the added complexity of having to be, to some level, aware of how your operating system works. Also, writing code must be done in such a manner that it can be real time.

5 LINUX

5.1 Operating System

An Operating System is a piece of software that does two things:

a. Provides an abstracted interface to the underlying hardware

b. Manager of the execution of a users programs (ie: sharing the CPU between all users' processes)

An operating system does its best to give each process the impression that the resources of the entire computer are at its disposal at all times. In the absence of inter-process communication, the operating system does its best to make each process think it is the only think running on a computer.

The main functions of a kernel, which is the core part of an OS that is always in memory and running, are memory management, process scheduling, interfacing to the hardware, file management, and communication with external devices and networks. [15]

5.2 Task Scheduling

Linux uses `nice()` to control task priority. Priority levels are -20 (high priority) to 19 (low priority). You can see tasks as a column in `top` output (Column "NI" column)
The SCHED_FF, SCHED_RR, SCHED_OTHER, and SCHED_FIFO scheduling policies define how a task is treated will be used. The `SCHED_OTHER` option is default.

5.3 Context switching and Preemption

Context switching occurs when the kernel changes the "current" task, be it to another kernel task or to a user space process. This occurs often behind the scenes to make it look as if all of the processes running on a system are running simultaneously. In reality, only one runs at a time.
Preemption is the suspension of a current running task in favor of another higher priority one.
See [14] for an excellent explanation of the different ways linux knows it is safe to switch context, and how different tasks (including kernel tasks) can be preempted.

5.4 User Control

In standard Linux, users have some control over scheduling. Mentioned in this document is the nice() scheduling priority mechanism. Also, users can take advantage of the sleep() system call (and others of the same flavor such as usleep()) to signify to the kernel that the user process is done what it needs to do for some time. POSIX compliance of an OS stipulates that the sleep() functions return control the process at or after the specified time. The "or after" specification is important because an OS does not need to make any guarantees on when it returns control to the process as long as it is after the specified sleep time. This is an issue when trying to write realtime programs.

6 REAL TIME LINUX

The traditional linux kernel allows one process to preempt other processes in limited cases. Specifically, preemption is controlled by the kernel and occurs only when:
In 2002 the National Institute of Standards and Technology (NIST) had Aeolean Inc. write a report detailing the viability of using Real-Time Linux for control

applications. [15] In it, they note that standard linux is alright for soft realtime applications where scheduling on the order of milliseconds (with several hundred MHz system clock) and occasional missed deadlines are accepted. The report then goes on to say that hard realtime linux can achieve 10 to 100 microsecond timing with no misses. It is important to note that dedicated RTOS can do better than this, but they come without the established (and growing!) Linux community. Linux is active, has thousands of developers contributing, is used in government and universities, and is not going anywhere. This is a major reason for trying our best to make realtime Linux to work for us.

6.1 Considerations

The RT Linux community came up with a list of quality-of-service metrics that they wanted to consider. [13] They are:

a. List of services for which realtime response is supported

b. Probability of meeting a deadline in absensce of hardware failure

c. Allowable deadline (response time) of a task

d. Performance and Scalability (Ambiguous?)

In addition to quality of service, they list 5 other qualities that are desirable in a full real time operating system. All 6 are:

a. Quality of Service

b. Amount of code that must be inspected to assure the quality of service

c. API is provided

d. Minimal added complexity of applications that take advantage of real time

e. Fault isolation: If non-RT code fails, does it affect RT code?

f. What hardware and software is suppored?

The general consensus was that a POSIX conforming api should be provided.

6.2 Development Info

It looks like [6] is highly involved with the Real Time Linux development process. Their Real Time Linux project page [7] cites both Ingo Molnar and Thomas Gleixner as being the lead developers.
The OSADL site cites a few important locations for downloading RTL requirements:

a. The Linux Kernel at kernel.org

b. The Real Time Preempt Patch at [8]

Their precompiled kernels are debian based, which is nice. That is what I want anyway.

a. User-Mode code is running (Kernel preempts it)

b. Kernel returns from a system call or an interrupt back to user space

c. Kernel code blocks on a mutex or explicitly yields

These points are from the FAQ at [2].

6.3 Performance

There are many qualitative (and somewhat subjective) performance metrics used, some of which are mentioned above. However, the two objective quantitative measures of performance for a RTOS are:

a. Event Latency

b. Periodic Jitter

Event Latency is the time it takes a request to be fulfilled. In other words, if my real time process requests control but does not actual get control of the CPU for 10 micro seconds, then the latency is 10 micro seconds.
Periodic Jitter is the variation in time of a periodic repetitive task.

6.4 Testing

There is a set of testing tools on kernel.org that is maintained by Clark Williams at [16].
This section outlines how to use them, and some specific results for the gumstix.

7 NOTES ON OUR USE

We are using the rt Linux release at [8]. We are using the 3.2 kernel, so we are using the "v3.2-rt" branch on the repo.

8 NOTES

Real time software can't have it memory paged out, so in linux a call to `mlockall()` needs to be made. This makes sure all of a program's memory stays in RAM. (FAQ at [2])

References

[1] Laplante, Phillip A. Real Time System Design and Analysis, 3rd Ed. Pscataway, NJ: IEEE Press, 2004. Print.

[2] Real-Time Linux Wiki, https://rt.wiki.kernel.org/index.php/Main_Page,2012-09-18

[3] Yaghmour, K.; Masters, J; Ben-Yossef, G; and Gerem, P. Building Embedded Linux Systems, 2nd Ed. Sebastopol, CA: O'Reilly Media, Inc. 2008. Print.

[4] Gleixner, T. "[ANNOUNCE] 3.0-rc7-rt0". https://lkml.org/lkml/2011/7/19/309. Email

[5] http://www.h-online.com/open/features/Kernel-Log-real-time-kernel-goes-Linux-3-0-1382791.html

[6] Open Source Software for Automation and Other Industries. https://www.osadl.org/Home.1.0.html

[7] OSDL Real Time Linux Project, https://www.osadl.org/Realtime-Linux.projects-realtime-linux.0.html

[8] Real Time Linux Latest Patches, http://www.kernel.org/pub/linux/kernel/projects/rt/

[9] Real Time Linux Installation, https://www.osadl.org/Realtime-Preempt-Kernel.kernel-rt.0.html

[10] Real Time Linux Users mailing list, http://vger.kernel.org/vger-lists.html#linux-rt-users

[11] The Linux Kernel Archives, http://kernel.org/

[12] McKenney, Paul. A Real Time Preemption Overview. http://lwn.net/Articles/146861/

[13] McKenney, Paul. Real Time Patch Acceptance Summary. http://lwn.net/Articles/143323/

[14] Love, R. Linux Kernel Development: Preemption and Context Switching, http://www.informit.com/articles/article.aspx?p=101760&seqNum=3

[15] Aeolean Inc., Introduction to Linux for Real-Time Control, 2002, http://tornasol.datsi.fi.upm.es/ciclope-old/doc/rtos/cache/doc/realtimelinuxreport.pdf

[16] Real Time Linux Testing Tools. git://git.kernel.org/pub/scm/linux/kernel/git/clrkwllms/rt-tests.git