

Overview of Real Time Linux

Ed Hursey

ECE, University of Colorado

Boulder CO

A stylized, teal-colored silhouette of a mountain range is positioned in the bottom right corner of the slide, extending from the right edge towards the center.

Real Time

- ◆ Real Time isn't about high throughput or low latencies.
- ◆ Its about predictability and consistency.
- ◆ Vanilla Linux is not predictable or consistent
 - System calls, disk I/O, Interrupts, CPU Intensive Processes can make it difficult to determine when a task/process will get to run.
- ◆ A Real Time O/S must be able to schedule events so they happen in a deterministic amount of time.
- ◆ Real Time patches and features attempt to make the Linux Kernel more deterministic.

What is Real Time Linux?

- ◆ Simply put it is a series of patches, features and updates to the vanilla Linux kernel to make it behave more like an RTOS
- ◆ There are numerous (well over 30) bundled solutions available, both commercial and open source.
 - Bundled means someone has already went to the trouble of incorporating the things into the kernel and distributes it that way.
 - See <http://www.realtimelinuxfoundation.org/> for a short list.
- ◆ A large majority of the real time features are also freely available as patches and some are even incorporated into the next version of the Linux kernel. Patches??

Real Time Linux

- ◆ This isn't (too) new.
- ◆ Some of the most important patches have been in development and testing since the late 90's.
- ◆ Abundant amount of information available online
 - Too much in some cases, hard to tell what is current and what has been superseded.
- ◆ MontaVisa Software was one of the first companies to distribute a Real Time Linux – September 2000.
- ◆ Other companies in the RTOS and Linux markets have also developed distributions (Wind River, Novell)

WindRiver Distribution

- ◆ Pros (my take)
 - Over 100 patches
 - Well tested
 - Good support (if you can pay)
 - New IDE (Eclipse based) allows for common development environment between VxWorks and RT Linux – very nice compared to Tornado
- ◆ Cons
 - Still have to build and apply all the patches to your kernel
 - Furthermore only works with specific versions of kernel 2.4.10 and 2.6.2 (I think)
 - Difficult licensing model
 - As far as x86 is concerned, I only got to work on a 686, due to fact that is all our educational version had
 - Why not just use open source and configure kernel and use patches as you need – this was the option I used

Patches / Configuration

- ◆ Real Time Preempt Patch
- ◆ Low Latency Patch
- ◆ $O(1)$ Scheduler

realtime-preempt Patch

- ◆ Vanilla Kernel is not preemptible
 - Means code running in kernel runs to completion = latency
 - Kernel is well written and regulated but still have unbounded limit of how long in kernel = problem
- ◆ The realtime-preempt patch (configuration) changes this
 - Makes kernel preemptible like user space
 - If higher priority task becomes runnable then it can run (maybe) – see next slide
 - A version was first developed by Monta Vista, then maintained by Robert Love, and the version contained in the 2.6.18 kernel is from Ingo Molnar.
- ◆ Very helpful in real time applications where you need a task to run as soon as it becomes runnable.
- ◆ Arguments have been made that it will lower throughput due to complexity, however testing has shown little to no effect
- ◆ Other issue is that CPU variables require more careful handling since the kernel can be preempted.

Not Preemptable

- ◆ The kernel is not preemptible in these cases:
 - While handling interrupts
 - While doing “bottom half” processing
 - While holding a spin-lock, read-lock, write-lock (protect from other processors (SMP) – the latest patch has attempted to minimize these
 - While executing scheduler

Low Latency Patch

- ◆ Similar goal to realtime-preempt patch – allow scheduler to run more often
- ◆ Different approach
 - Rather than attempting to implement preemption in a kernel that wasn't designed for it, this patch adds specific preemption points in blocks of code that could execute for long stretches of time.
 - So when the kernel iterates over large data structures, it finds a way to safely call the scheduler, which may involve dropping a lock, scheduling and reacquiring the lock.
 - This is much more difficult to test/determine if anything has broken

Which is better?

- ◆ The developers of each patch were very passionate about which was better
- ◆ RTOS are preemptible – so from that standpoint the preempt patch seems to make sense, but altering the kernel that way can be difficult and perhaps hard to maintain
- ◆ The low latency patch showed large latency improvements
- ◆ Why not both?
 - The two patches have been combined are now in Ingo Molnar's realtime-preempt patch
 - And have been added to the 2.6.18 kernel as a configuration option – see details at end of slides

Draw Backs

- ◆ The biggest draw back is throughput.
 - Most processes (like disk I/O) have the greatest throughput if run uninterrupted.
 - Constant preemption and context switching affects this throughput.
 - Preemption has overhead.
 - Scheduling has overhead.
- ◆ Everyone agrees that throughput could be an issue, but testing continues to show that throughput isn't affected.
- ◆ Complexity
 - These patches add complexity that could lead to hard to maintain and/or difficult to debug code
 - But isn't the kernel already pretty complex?

O(1) Scheduler

- ◆ Also developed by Ingo Molnar
 - First attempt in 1998
 - Current version started in 2001
 - Incorporated into 2.5 development tree in 2002
- ◆ Purpose - a good scheduler for Linux that is $O(1)$ in its entirety: wakeup, context-switching and timer interrupt overhead

0(1) Scheduler con't

- ◆ Areas of improvements:
 - Performs well independently of how many tasks there are in the system.
 - ◆ The old scheduler was inefficient in certain cases (eg. JVMs)
 - The way interactive tasks are handled is improved.
 - ◆ Interactive tasks are now detected via a separate, usage-statistics-driven mechanism
 - Interactive tasks are snappy under heavy load

Demo

- ◆ 2.6.18 Kernel with realtime-preemption configured
- ◆ Using POSIX threads with SCHED_FIFO scheduling policy
- ◆ Bt878 Video Card with Standard Linux drivers using Video4Linux API
- ◆ Threads:
 - One grabbing current frame
 - One executing centroid location
 - One doing kernel system calls

Kernel Configuration

- ◆ Steps I used to configure my kernel
 - Started with SUSE Linux 10.1 (selected all kernel development, developer options etc on install (gcc etc))
 - Downloaded 2.6.18.3 kernel from www.kernel.org (the full kernel, not patches, clicked on "F", saved to /home/<user>/src)
 - Go into /home/<user>/src/ directory
 - Extract
 - ◆ **tar xfvj /home/<user>/src/linux-2.6.18.3.tar.bz2**
 - Go into /home/<user>/src/linux-2.6.18.3 directory
 - Clean up previous stuff
 - ◆ make mrproper
 - Configuration
 - ◆ make menuconfig
 - Enable realtime-preempt patch
 - ◆ Go into Processor type and Features menu (enter)
 - ◆ Go into Preemptible Kernel (enter)
 - ◆ Select Preemptible Kernel (Low-Latency Desktop)
 - ◆ You will then be back at the Processor type and features menu (make sure the Preempt The Big Kernel Lock) is now selected
 - ◆ Then exit and save the configurateion
 - Build
 - ◆ make clean
 - ◆ make bzImage
 - ◆ make modules
 - ◆ su (switch to root user, so need to type root username)
 - ◆ make modules_install

Kernel Configuration con't

- ◆ Create initial RAM disk
 - `Mkinitrd -k /home/<user>/src/linux-2.6.18.3/arch/boot/bzImage -i /boot/initrd-2.6.18.3-rt -M /home/<user>/src/linux-2.6.18.3/System.map`
 - Note the thing after the `-i` in the above command is what you are creating so you can name it what ever you want, I added the `(-rt)` to signify the real time stuff is in that kernel. The `bzImage` file and the `System.map` file should already exist
- ◆ Installation
 - `cp arch/i386/boot/bzImage /boot/bzImage-2.6.18.3-rt`
 - `cp System.map /boot/System.map-2.6.18.3-rt`
 - `ln -s /boot/System.map-2.6.18.3-rt /boot/System.map`
- ◆ Grub configuration (add new item to boot menu)
 - Add the following at the end of the file

```
title Real Time Kernel (2.6.18.3)
root(hd0,0)
kernel /boot/bzImage-2.6.18.3-rt root=/dev/hda1
initrd /boot/initrd-2.6.18.3-rt
```
 - Then you should be able to reboot and select the new kernel to boot with

◆ Questions and Discussion



References

- ◆ <http://www.linuxdevices.com/news/NS3989618385.html>
- ◆ <http://www.linuxdevices.com/articles/AT4185744181.html>
- ◆ <http://www.linuxdevices.com/news/NS9566944929.html>
- ◆ <http://www.linuxdevices.com/articles/AT8267298734.html>
- ◆ <http://www.realtimelinuxfoundation.org/events/rtlws-2005/papers.html>
- ◆ <http://www.realtimelinuxfoundation.org/>
- ◆ <http://deadc0de.blogspot.com/2006/10/linux-real-time-kernel.html>
- ◆ <http://kerneltrap.org/node/517>
- ◆ <http://www.digitalhermit.com/linux/Kernel-Build-HOWTO.html>