Enable real-time capabilities of the mainline kernel

Carsten Emde

< C.Emde@osadl.org >

Revision History	
Revision 0.1	2006-12-07
First release	
Revision 0.2	2006-12-13
CONFIG_DEBUG_PREEMPT must be set to obtain /proc/latency_trace	
Revision 0.3	2006-12-26
Omitted to initialize /proc/sys/kernel/preeempt_max_latency	
Revision 0.4	2007-03-19
Added paragraph on Ubuntu, thanks to Alessio Igor Bogani	
Revision 0.5	2007-04-25
Removed .config settings, provided menuconfig description, added latency histogram	
Revision 0.6	2007-05-10
Introduced latency fighting	
Revision 0.7	2007-05-31
Added latency trace example	
Revision 0.8	2007-10-09
Adapted to the download directory at kernel.org, removed dead links to Fedora prebuilt kernels	
Revision 0.9	2008-08-29
Adapted to 2.6.24.7-rt17, added a section on ketchup, added debug fs, mentioned stress tools	
Revision 0.10	2008-12-10
Adapted to 2.6.24.7-rt23, updated kernel CONFIGs	
Revision 0.11	2009-11-07
Adapted to 2.6.31.5-rt17, updated kernel CONFIGs, dirs and files	

Table of Contents

1. Introduction

In August 2006, a large part of the realtime-preempt kernel patch that is maintained by Ingo Molnar, Thmomas Gleixner and Steven Rostedt was merged into the mainline kernel and is now immediately available. A number of functions, however, still is only available, if the realtime-preempt patch is applied. It is planned to merge the vast majority of these functions into the mainline kernel before the end of 2008.

In addition to the traditional installation method that requires download, patch and recompilation of the kernel, an apt repository is now provided that greatly facilitates the installation of the realtime-preempt patch. This mechanism, however, is currently only available when the Ubuntu Feisty distribution is used. It is expected that Redhat Enterprise Linux and Fedora will contain prepatched realtime-preempt kernels in the near future.

This HOWTO explains how to install, configure, test and use the realtime- preempt kernel patch in Ubuntu Feisty (7.04) and other 2.6 systems.

2. Installation

2.1. Ubuntu Feisty (7.04)

Add, as root, to your **/etc/apt/sources.list** (remember to do a backup of this file) the following line:

```
# deb http://www.texware.it/ubuntu feisty/
```

Then execute:

```
# wget -q http://www.texware.it/ubuntu/feisty/BBA3222D.gpg -O- | sudo apt-key add -
# sudo apt-get update
# sudo apt-get install linux-realtime
```

The provided packages include realtime-enabled kernels as well as the cyclictest utility (see below).

Thanks to Alessio Igor Bogani for making these kernels available. Additional information is given here.

2.2. Other 2.6 systems

2.2.1. Download and patch the kernel manually

Download the mainstream kernel sources and the corresponding realtime-preempt patch, for example **linux-2.6.31.5** and **patch-2.6.31.5-rt17**:

```
# cd /usr/src/kernels
# wget http://www.kernel.org/pub/linux/kernel/v2.6/linux-2.6.31.5.tar.bz2
# wget http://www.kernel.org/pub/linux/kernel/projects/rt/patch-2.6.31.5-rt17.bz2
```

Unpack the kernel:

```
# tar -jxvf linux-2.6.31.5.tar.bz2
```

Rename the kernel directory:

```
# mv linux-2.6.31.5 linux-2.6.31.5-rt17
```

Apply the patch:

```
# bunzip2 patch-2.6.31.5-rt17.bz2
# cd linux-2.6.31.5-rt17
# patch -p1 <../patch-2.6.31.5-rt17
```

2.2.2. Download and patch the kernel using ketchup

Alternatively, you can let **ketchup** do the above work for you automatically:

```
# cd /usr/src/kernels
# mkdir tmp
# cd tmp
# ketchup -r -G 2.6.31.5-rt17
```

Matt Mackall’s and Steven Rostedt’s **ketchup** is available at http://people.redhat.com/srostedt/rt/tools/ketchup-0.9.8-rt3. To install it, type

```
# wget -O /usr/local/bin/ketchup http://people.redhat.com/srostedt/rt/tools/ketchup-0.9.8-rt3
# chmod +x /usr/local/bin/ketchup
```

2.2.3. Configure and build the kernel

After the patched kernel is available on your system, copy the configuration file as provided by your distribution to the file **.config**:

```
# cp /boot/config-‘uname -r‘ .config
```

Invoke the text based kernel configuration menu:

```
# make menuconfig
```

The preemption model is selected along with the processor type and features:

```
Processor type and features  --->
      Preemption Mode (Complete Preemption (Real-Time))  --->
```

Debug, trace and diagnostic tools are part of the "kernel hacking" settings, for example:

```
Kernel hacking  --->
   [*] Tracers  --->
      --- Tracers
      [*]   Kernel Function Tracer
      [*]   Interrupts-off Latency Tracer
      [*]     Interrupts-off Latency Histogram
      [*]   Preemption-off Latency Traver
      [*]     Preemption-off Latency Histogram
      [*]   Scheduling Latency Tracer
      [*]     Scheduling Latency Histogram
      [*]   Missed timer offsets histogram
```

These settings, however, are only useful during development and evaluation. Some of them are better disabled when the system goes into production. The scheduling latency and the missed timer offsets histograms, however, only slightly interfere with the system and may, thus, be used to continuously register the wakeup latency under production conditions. In any case, make sure that stack overflow checking is disabled, since this may produce additional latencies when enabled:

```
Kernel hacking  --->
      [ ] Check for stack overflows
```

The new kernel is then compiled, linked and installed as usual.

```
# make
# make modules_install install
```

When the system is rebooted, the newly provided kernel becomes part of the boot menu and can be selected.

Should you happen to be the proud owner of a multi-core processor, be sure to specify the **-j <jobs>** option of **make** where **<jobs>** is twice the number of cores your processor has, as this will speed up kernel compilation considerably.

3. Testing and using the realtime-preempt patch

3.1. Built-in tools

The kernel version must now contain the tags **PREEMPT** and **RT** such as

```
uname -v
#42 SMP PREEMPT RT Fri Nov 6 18:55:29 CET 2009
```

or something went completely wrong, otherwise.

In order to access the built-in diagnostic tools, the debug file system must be mounted. This is either done manually by entering

```
# mount -t sysfs nodev /sys
# mount -t debugfs nodev /sys/kernel/debug
```

or by appending the lines

```
nodev /sys sysfs defaults 0 0
nodev /sys/kernel/debug debugfs defaults 0 0
```

to the file **/etc/fstab** to have the debug file system mounted automatically at boot time.

Note

In earlier kernels, the built-in diagnostic tools were immediately available in the **proc** file system that is normally mounted by default.

The histograms of the wakeup latencies (one per CPU) are available here:

```
# ls /sys/kernel/debug/tracing/latency_hist/wakeup_latency/CPU?
```

From 2.6.2.31 onwards, the files are in:

```
# ls /sys/kernel/debug/tracing/latency_hist/wakeup/CPU?
```

This configuration feature is always available in the prebuilt Ubuntu kernels; in the locally created kernels it is only available, if configured as shown above. The granularity of the histograms amounts to one microsecond:

```
# grep -v " 0$" /sys/kernel/debug/tracing/latency_hist/wakeup*/CPU0
#Minimum latency: 0 microseconds.
#Average latency: 7 microseconds.
#Maximum latency: 39 microseconds.
#Total samples: 7069336
#There are 0 samples greater or equal than 10240 microseconds
#usecs           samples
    0             249884
    1             120023
    2             338781
    3             197834
    4             210872
    5             150366
    6              45870
    7            1053204
    8             564637
    9             273756
   10             190432
   11             483611
   12             328509
   13              44716
   14               72925
   15               59304
   16               28927
   17               10836
   18                2821
   19                 543
   20                 110
   21                  82
   22                  63
   23                  61
   24                  32
   25                  21
   26                   9
   27                   4
   28                   3
   29                   1
   30                   3
   31                   2
   32                   4
   33                   2
   39                   1
```

By default, all histograms, kernel function tracing etc. are disabled. To enable, for example, the wakeup latency histograms, type

```
echo 1 >/sys/kernel/debug/tracing/latency_hist/enable/wakeup
```

To reset the histogram counters, the following script may be used

```
#!/bin/sh
TRACINGDIR=/sys/kernel/debug/tracing
HISTDIR=$TRACINGDIR/latency_hist

if test -d $HISTDIR
then
  cd $HISTDIR
  for i in ‘find . | grep /reset$‘
  do
    echo 1 >$i
  done
fi
```

Some more documentation is available in the kernel source directory **Documentation/trace/histograms.txt**.

3.2. External testing tool

In addition, Thomas Gleixner made available the test tool **cyclictest**, that allows to better and more spcifically determine the realtime capabilities of a given system. The sources can be downloaded from here. If you installed the Ubuntu apt packages (see above), the **cyclictest** tool is available immediately. The git repository contains other useful test programs for real-time systems.

Download, unpack and compile the tool

```
# git clone git://git.kernel.org/pub/scm/linux/kernel/git/clrkwllms/rt-tests.git
# cd rt-tests
# make
```

To run one test thread per CPU or per CPU core, each thread on a separate processor, type

```
# ./cyclictest -a -t -n -p99
```

On a non-realtime system, you may see something like

```
T: 0 ( 3431) P:99 I:1000 C: 100000 Min:      5 Act:   10 Avg:   14 Max:   39242
T: 1 ( 3432) P:98 I:1500 C:  66934 Min:      4 Act:   10 Avg:   17 Max:   39661
```

The rightmost column contains the most important result, i.e. the worst-case latency of 39.242 milliseconds. On a realtime-enabled system, the result may look like

```
T: 0 ( 3407) P:99 I:1000 C: 100000 Min:      7 Act:   10 Avg:   10 Max:      18
T: 1 ( 3408) P:98 I:1500 C:  67043 Min:      7 Act:    8 Avg:   10 Max:      22
```

and, thus, indicate an apparent short-term worst-case latency of 18 microseconds.

Running **cyclictest** only over a short period of time and without creating appropriate real-time stress conditions is rather meaningless, since the execution of an asynchronous event from idle state is normally always quite fast, and every - even non-RT system - can do that. The challenge is to minimize the latency when reacting to an asynchronuous event, irrespective of what code path is executed at the time when the external event arrives. Therefore, specific stress conditions must be present while **cyclictest** is running to reliably determine the worst-case latency of a given system.

3.3. Latency fighting

If - as in the above example - a low worst-case latency is measured, and this is the case even under a system load that is equivalent to the load expected under production conditions, everything is alright. Of course, the measurement must last suffciently long, preferably 24 hours or more to run several hundred million test threads. If possible, the **-i** command line option (thread interval) should be used to increase the number of test threads over time. As a role of thumb, the thread interval should be set to a value twice as long as the expected worst-case latency. If at the end of such a test period the worst-cae latency still did not exceed the value that is assumed critical for a given system, the particular kernel in combination with the hardware in use can then probably be regarded as real-time capable.

What, however, if the latency is higher than acceptable? Then, the famous "latency fighting" begins. For this purpose, the **cyclictest** tool provides the **-b** option that causes a function tracing to be written to **/sys/kernel/debug/tracing/trace**, if a specified latency threshold was exceeded, for example:

```
# ./cyclictest -a -t -n -p99 -f -b100
```

This causes the program to abort execution, if the latency value exceeds 100 microseconds; the culprit can then be found in the trace output at **/sys/kernel/debug/tracing/trace**. The kernel function that was executed just before a latency of more than 100 microseconds was detected is marked with an exclamation mark such as

```
qemu-30047 2D.h3 742805us : __activate_task+0x42/0x68 <cyclicte-426> (199 1)
qemu-30047 2D.h3 742806us : __trace_start_sched_wakeup+0x40/0x161 <cyclicte-426> (0 -1)
qemu-30047 2DNh3 742806us!: try_to_wake_up+0x422/0x460 <cyclicte-426> (199 -5)
qemu-30047 2DN.1 742939us : __sched_text_start+0xf3/0xdcd (c064e442 0)
```

The first column indicates the calling process responsible for triggering the latency.

If the trace output is not obvious, it can be submitted to the OSADL Latency Fight Support Service at `<latency-fighters@osadl.org>` In addition to the output of **cat /sys/kernel/debug/tracing/trace**, the output of **lspci** and the **.config** file that was used to build the kernel in question must be submitted. We are sure you understand that OSADL members will be served first, but we promise to do our best to help everybody to successfully fight against kernel and driver latencies.