



Real-time - Timers and scheduling latency

Objective: Learn how to handle real-time processes and practice with the different real-time modes. Measure scheduling latency.

After this lab, you will

- Be able to apply the rt-preempt patches and be more familiar with kernel configuration settings related to real-time.
- Be able to check clock accuracy.
- Be able to start processes with real-time priority.
- Have compared scheduling latency on your system, between a standard kernel and a kernel with real-time preempt patches.

Setup

Go to the `/home/<user>/felabs/realtime/rttest` directory.

For this lab, you must have compiled a 2.6.29.x kernel for the Calao board, with the default configuration. Boot the Calao board by mounting the root filesystem available at `/home/<user>/felabs/realtime/rttest/nfsroot/` with NFS.

Please stay with a 2.6.29.x version, as this is the most recent version with both Xenomai and PREEMPT_RT support.

Install netcat on your host, by running:

```
apt-get install netcat
```

Download CodeSourcery's 2009q1 toolchain at:

<http://www.codesourcery.com/sgpp/lite/arm/portal/release858>

Choose "IA32 GNU/Linux TAR"

Untar it.

Add `/home/<user>/felabs/realtime/rttest/arm-2009q1/bin` to your PATH.

Using high-resolution timers

Have a look at the `rttest.c` source file available in `root/` in the `nfsroot/` directory. See how it shows the resolution of the `CLOCK_MONOTONIC` clock.

Now compile this program:

```
arm-none-linux-gnueabi-gcc -o rttest rttest.c -lrt
```

Execute the program on the board. Is the clock resolution good or bad? Compare it to the timer tick of your system, as defined by `CONFIG_HZ`.

Obviously, this resolution will not provide accurate sleep times, and this is because our kernel doesn't use high-resolution timers. So let's enable the following options in the kernel configuration:

- `CONFIG_HIGH_RES_TIMERS`
- `CONFIG_ATMEL_TCLIB`
- `CONFIG_ATMEL_TCB_CLKSRC`

You will have to log with username root and no password.

We are using a glibc toolchain because glibc has better support for the POSIX RT API than uClibc. In our case, when we created this lab, uClibc didn't support the `clock_nanosleep` function used in our `rttest.c` program.



Recompile your kernel, boot your Calao board with the new version, and check the new resolution. Better, isn't it ?

Testing the non-preemptible kernel

Now, do the following tests:

- Test the program with nothing special and write down the results.
- Test your program and at the same time, add some workload to the board, by running `do1oad 300` on the board, and using `netcat 192.168.0.100 5566` on your workstation when you see the message "Listening on any address 5566" in order to flood the network interface of the Calao board (where 192.168.0.100 is the IP address of the Calao board)
- Test your program again with the workload, but by running the program in the `SCHED_FIFO` scheduling class at priority 99, using the `chrt` command.

Testing the preemptible kernel

Recompile your kernel with `CONFIG_PREEMPT` enabled, which enables kernel preemption (except for critical sections protected by spinlocks).

Re-do the simple tests with this new preemptible kernel and compare the results.

Testing the real-time patch

Get the `PREEMPT_RT` patch from

<http://www.kernel.org/pub/linux/kernel/projects/rt/>, and apply it to the kernel source code. Configure the kernel with `CONFIG_PREEMPT_RT` and recompile it. The kernel should now be fully preemptible, including in critical sections, and tasks can have higher priorities than interrupts.

Repeat the simple tests with this real-time preemptible kernel and compare the results.

Testing Xenomai scheduling latency

Get Xenomai from its download area:

<http://download.gna.org/xenomai/stable/>

Untar Xenomai.

Prepare the kernel for Xenomai compilation, for instance, if your Linux version is 2.6.29.6 and Xenomai version is 2.5.0:

```
xn=/home/<user>/felabs/realtime/rttest/xenomai-2.5.0
lnx=/home/<user>/felabs/realtime/rttest/linux-2.6.29.6
$xn/scripts/prepare-kernel.sh --arch=arm --linux=$lnx
```

You can copy the `.config` file from a previous compile job, then launch kernel configuration again and enable the options:

- `CONFIG_XENOMAI`
- `CONFIG_XENO_DRIVERS_TIMERBENCH`



Other options of interest (ARM specific) are:

- `CONFIG_ARM_FCSE_GUARANTEED`
- `CONFIG_XENO_HW_UNLOCKED_SWITCH`

Read the help associated with these options, decide whether you want to enable them, please try not to choose the same combo as your neighbor, so as to be able to compare your results.

Compile `rttest` for the Xenomai POSIX skin:

```
DESTDIR=/home/<user>/felabs/realtime/rttest/nfsroot/  
export DESTDIR  
CFL=`$DESTDIR/bin/xeno-config --posix-cflags`  
LDF=`$DESTDIR/bin/xeno-config --posix-ldflags`  
arm-none-linux-gnueabi-gcc $CFL -o rttest rttest.c $LDF
```

Now boot the board with the new kernel.

Re-run the tests, compare the results.

Testing Xenomai interrupt latency

Run the following commands:

```
echo 0 > /proc/xenomai/latency  
modprobe xeno_timerbench
```

Then measure the interrupt latency with and without load, running the following command:

```
latency -t 2
```

