ROS-Timer Latency Tests

Jonas Sticha

1 Abstract

As modern real-time systems become more and more complex, their software architectures include a middleware to connect the systems' software components. This middleware must fulfill hard real-time requirements. To validate whether ROS can meet such requirements, a series of tests have to be conducted.

As a first step, the latencies of the ROS-Timer function have been measured under different circumstances. The results of the first tests, as well as the exact Hardware/Software modules of the test system, are depicted in this paper, allowing independent validation of our test results. Also, everyone is welcome to use the test suite to validate the real-time capabilities of any other Software/Hardware system.

2 Hardware

The hardware used to run the tests:

- Platform: PandaBoard ES OMAP4460
- Data carrier: Sandisk Extreme Pro SDHC UHS-I card 8GB
- Power supply: 5.0V, 4.0A

3 Software

3.1 Operating System

The tests were run on a normal Linux system and on a Linux system with full kernel preemption support. The build process is explained in the following sections. The exact software-versions I used are specified in Table 1. My build system for the whole build as described in Section 3.1.1 was a server with Fedora 20, the real-time Linux kernel (Section 3.1.2) however, was built on a notebook, running with Ubuntu 13.04.

Note: We expect that the build machine has no impact on the test results.

3.1.1 Linux 3.4.0

To build a Linux image with the 3.4.0 Linux kernel and all the necessary software, first follow the instructions on the ros-Wiki page “Installation OpenEmbedded/Yocto”[A]. However, instead of cloning the bitbake and openembedded-core git repositories, download the YOCTOPROJECT from the official website[0] and clone the other repositories into the extracted YOCTOPROJECT folder. Next move into the meta-ros folder and add “timer-tests” to the IMAGE_INSTALL variable in meta-ros/recipes-core/images/core-image-roscore.bb. Also you have to rename the “boost_%.bbappend” recipe in recipes-support/boost/ to “boost_1.54.0.bbappend”. Then move into the meta-openembedded directory and checkout the “dora” branch.

Optional: If you also want to have cyclictest included in the image, simply add “rt-tests” to the IMAGE_INSTALL variable in meta-ros/recipes-core/images/core-image-roscore.bb.

To build the image for the PandaBoard target platform, we need to clone the meta-ti layer git repository[1] into the YOCTOPROJECT folder and modify the SRC_URI and SRCREV variables in the kernel recipe meta-ti/recipes-kernel/linux/linux-omap4_3.4.bb, to checkout the correct branch and commit of the kernel sources as specified in Table 1 in the row “kernel-ubuntu”. Then add the meta-ti layer path to your bblayers.conf and change the MACHINE variable in your local.conf to “pandaboard”.

Finally to bake the image, execute the following two commands:

1. source oe-init-build-env
2. bitbake core-image-roscore

On success, the image files are placed in build/tmp/deploy/images/pandaboard/. Those can now be installed to an SD card by invoking the create_mmc.sh script from that directory, with the path to your SD card device file (/dev/sdX) and the file system you want to use (I chose ext4).

3.1.2 Linux 3.4.0-rt17 PREEMPT RT

To build the same image as described in Section 3.1.1 but with a PREEMPT RT Linux kernel, you first follow all the steps in Section 3.1.1. After that, create a new directory and there, execute the following commands to compile the PREEMPT RT Linux kernel:

1. git clone git://dev.omapzoom.org/pub/scm/integration/kernel-ubuntu.git &
2. cd kernel-ubuntu

ROS-Timer Latency Tests http://bmwcarit.github.io/ros_realtime_tests/
2. git checkout b3d5eeb10553e4bc0c3b250a4d06d43c4ab397a9
3. wget http://hbrobotics.org/wiki/images/8/88/Patch-3-4-9-rt17.patch.doc
4. patch -p1 < ./Patch-3-4-9-rt17.patch.doc
5. wget -O .config http://hbrobotics.org/wiki/images/c/c7/Config-3-4-9-rt17.doc

Then add "#include <linux/cache.h>" to security/apparmor/sid.c and compile the kernel by executing "make ARCH=arm CROSS_COMPILE=arm-linux-gnueabi uImage". Make sure you have gcc-arm-linux-gnueabi installed on your build system, otherwise the make command will fail. If the compilation was successful, the kernel should now be placed at arch/arm/boot/uImage. Now simply replace the uImage on the boot partition of the SD card with the one we just compiled and you have a bootable real-time Linux image.

<table>
<thead>
<tr>
<th>Software</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>YOCTO_PROJECT</td>
<td>10.0.1 “Dora”</td>
</tr>
<tr>
<td>meta-ros</td>
<td>SHA1 ID “097668a6de955511b58f20ef12fcee96ace333874”</td>
</tr>
<tr>
<td>meta-openembedded</td>
<td>Branch “dora”, SHA1 ID “40ef0f3713eb1628655e484feac0cebf810737b4”</td>
</tr>
<tr>
<td>meta-ti</td>
<td>SHA1 ID “aaecf01794687af7c78cf98a979a3db8b69938a”</td>
</tr>
<tr>
<td>kernel-ubuntu</td>
<td>Branch “ti-ubuntu-3.4-1485”, SHA1 ID “b3d5eeb10553e4bc0c3f250a4d06d43c4ab397a9”</td>
</tr>
<tr>
<td>ros_realtime_tests</td>
<td>SHA1 ID “83d50bc52e3821f1602e8d581dce04c3e3e4aa”</td>
</tr>
<tr>
<td>gcc [fedora system]</td>
<td>Version 4.8.2 20131212 (Red Hat 4.8.2-7)</td>
</tr>
<tr>
<td>gcc [ubuntu system]</td>
<td>Version 4.7.3 (Ubuntu/Linux 4.7.3-1ubuntu1)</td>
</tr>
<tr>
<td>gcc-arm-linux-gnueabi</td>
<td>Version 4.4.7.2-1</td>
</tr>
</tbody>
</table>

Table 1 Versions

3.2 timer_tests Test Suite

To run the timer_tests test suite, you first start the roscore process. After that you invoke the timer_tests binary with a set of arguments:

 - amount_measurements Amount of repetitions.
 - timeout_value_us Timeout value to use for measurements (in microseconds).
 - testnode_rt 0/1 Specifies whether the test node should run with normal (0) or real-time (1) priority. Note: In the latter case the rt_sched_policy argument has to be specified.
 - [rt_sched_policy 'FIFO'/'RR'] Specify the real-time scheduling algorithm. It can be either FIFO or Round-Robin.

Example: timer_tests 2000000 1000 1 FIFO (used for the sample in Section 4).

After the measurements are completed, the results are stored in log files in the directory in which the test suite was invoked. Those log files can be plotted using gnuplot. A sample gnuplot script for that can be found in the ros_realtime_tests git repository. To use the script to plot your test results, set the marked variables with the corresponding values and invoke it by executing "gnuplot plotTestRes.gnuplot".

4 Test results

In all tests, the timer_tests test suite was run with two million repetitions but under different conditions:

- 1 / 10 milliseconds timeout
- standard / real-time process priority
- Round-Robin / FIFO scheduling
- normal / PREEMPT RT Linux kernel
- idle state / 80% CPU load

The most important insights we gained so far are:
Timeout values smaller than 1 millisecond aren’t processed correctly, leading to disproportionally high latencies.

If the test node is running with real-time priority, the timer never expires too early (no negative latencies).

If the test node is running with real-time priority, the latencies reported to the callback function by the ROS-Timer become quite accurate: The average difference is approximately 6 microseconds and the peak differences are below 150 microseconds compared to an average difference of 50 - 150 microseconds with peak values of up to two milliseconds, when running the test node with normal process priority.

The average latencies differ significantly depending on the timeout value. Depending on the setup, the average latencies with a timeout value of 1 and a timeout value of 10 milliseconds differ by 30 to 50 microseconds.

Process priority and scheduling policy have to be set before calling `ROS::init()`, otherwise not all threads run with the intended priority. This can lead to extremely high latencies.

However, the first test results indicate that with the right setup, hard real-time requirements could be met, as none of the measured latencies on the PREEMPT RT Linux system with the test node running with real-time priority, exceeded 400 microseconds. Though, to prove this statement the empirical evidence has to be strengthened through additional and longer test runs. This is work in progress. The plot below shows the latencies that occurred on the PandaBoard with 3.4.0-rt17+ PREEMPT RT Linux kernel, a CPU load of 80%, a timeout value of one millisecond and the test node running with real-time priority and Round-Robin scheduling. In this scenario, the average latency was 77 microseconds with a maximum latency of 157 microseconds.

In the following sections, you can find all plots of the results from the complete set of test scenarios.

### 4.1 Linux 3.4.0 no CPU load
4.2 Linux 3.4.0 80% CPU load
### 4.3 Linux 3.4.0-rt17+ PREEMPT RT no CPU load
4.4 Linux 3.4.0-rt17+ PREEMPT RT 80% CPU load
### ROS-Timer Latency Tests

<table>
<thead>
<tr>
<th>Priority Type</th>
<th>1ms Timeline</th>
<th>10ms Timeline</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Normal Priority</strong></td>
<td><img src="image" alt="1ms Normal Priority Timeline" /></td>
<td><img src="image" alt="10ms Normal Priority Timeline" /></td>
</tr>
<tr>
<td>MIN: -992us</td>
<td>AVG: 251us</td>
<td>MAX: 2455us</td>
</tr>
<tr>
<td><strong>Real-time Priority, Round-Robin Scheduling</strong></td>
<td><img src="image" alt="1ms Round-Robin Scheduling Timeline" /></td>
<td><img src="image" alt="10ms Round-Robin Scheduling Timeline" /></td>
</tr>
<tr>
<td>MIN: 34us</td>
<td>AVG: 77us</td>
<td>MAX: 157us</td>
</tr>
<tr>
<td><strong>Real-time Priority, FIFO Scheduling</strong></td>
<td><img src="image" alt="1ms FIFO Scheduling Timeline" /></td>
<td><img src="image" alt="10ms FIFO Scheduling Timeline" /></td>
</tr>
<tr>
<td>MIN: 33us</td>
<td>AVG: 77us</td>
<td>MAX: 205us</td>
</tr>
</tbody>
</table>

---

Copyright (C) 2014, BMW Car IT GmbH

Document generated by eLyXer 1.2.5 (2013-03-10) on 2014-04-24T11:12:30.329000