RT-Tests Overview

Goal

Explore some of the Real Time test tools from the `rt-tests` package.

Install `rt-tests` package

Copy the package from the ex7 to the target.

```
host$ cd exfiles/ex7/rt-tests

host$ scp rt-tests-1.1-r0.armv5e.rpm root@192.168.7.2:
```

Install the package on the target.

```
target# rpm -ivh rt-tests-1.1-r0.armv5e.rpm
```

List the test tools included in the `rt-tests` package.

```
target# rpm -ql rt-tests
```

Overview

In general, most of the test tools will start multiple threads (or processes), and test the latency when acquiring/releasing a resource (lock, socket, message queue, signal, etc). The tools produce similar outputs
Min = minimal measured latency
Cur = latency of the current thread/process
Avg = average latency during the test
Max = maximum measured latency

The aim is obviously to get bounded predictions on the worse (maximum) latency, when using different inter-process communication and lock mechanisms coupled with different scheduling policies.

Lab Limitations

We are not booting the target with a real time kernel, but it would be fairly easy to recompile a kernel with support for PREEMPT_RT and re-run the `rt-tests` suite to compare the effects of the real time patches. It should also be possible to explore alternative schedulers such as Xenomai.

Also note that your target might be utilising a single core (SMP might be disabled), so advanced CPU affinity options and the CPU migration tool will only take into account a single core.

Test Descriptions from Manpages

The manpages for the `rt-tools` are not included on the target, so the following table is a summary of what the tools do. Most tests will simply report Min/Avg/Max latency measurements.

Tool	Description from manpages
/usr/bin/cyclictest	To measure latencies, Cyclictest runs a non real-time master thread (scheduling class SCHED_OTHER) which starts a defined number of measuring threads with a defined real-time priority (scheduling class SCHED_FIFO). The measuring threads are woken up periodically with a defined interval by an expiring timer (cyclic alarm). Subsequently, the difference between the programmed and the effective wake-up time is calculated and handed over to the master thread via shared memory. The master thread tracks the latency values and prints the minimum, maximum, and average latencies.
/usr/bin/hackbench	Hackbench is both a benchmark and a stress test for the Linux kernel scheduler. It's main job is to create a specified number of pairs of schedulable entities (either threads or traditional processes) which communicate via either sockets or pipes and time how long it takes for each pair to send data back and forth.
/usr/bin/pi_stress	**pi_stress** is a program used to stress the *priority-inheritance* code paths for POSIX mutexes, in both the Linux kernel and the C library. It runs as a realtime-priority task and launches *inversion machine* thread groups. Each inversion group causes a *priority inversion* condition that will deadlock if *priority inheritance* doesn't work.
/usr/bin/pip_stress	This program demonstrates the technique of using priority inheritance (PI) mutexes with processes instead of threads. The way to do this is to obtain some shared memory - in this case with mmap that backs a pthread_mutex_t since this will support PI.
/usr/bin/pmqtest	The program **pmqtest** starts pairs of threads that are synchronized via mq_send/mw_receive() and measures the latency between sending and receiving the message
/usr/bin/ptsematest	The program **ptsematest** starts two threads that are synchronized via pthread_mutex_unlock()/pthread_mutex_lock() and measures the latency between releasing and getting the lock.
/usr/bin/rt-migrate-test	Test real-time multiprocessor scheduling of tasks to ensure the highest priority tasks are running on all available CPUs
/usr/bin/sendme	The program sendme uses the backfire driver to send a signal from driver to user. It then reads the timestamp from the driver and calculates the time intervals to call the driver and to receive the signal from the driver.
/usr/bin/signaltest	signal roundtrip test
/usr/bin/sigwaittest	The program **sigwaittest** starts two threads or, optionally, forks two processes that are synchronized via signals and measures the latency between sending a signal and returning from sigwait().

Tool	Description from manpages
/usr/bin/svsematest	The program **svsematest** starts two threads or, optionally, forks two processes that are synchronized via SYSV semaphores and measures the latency between releasing a semaphore on one side and getting it on the other side