>>> Operating Systems And Applications For Embedded Systems
>>> Real-time Programming

Name: Mariusz Naumowicz
Date: 27 sierpnia 2018

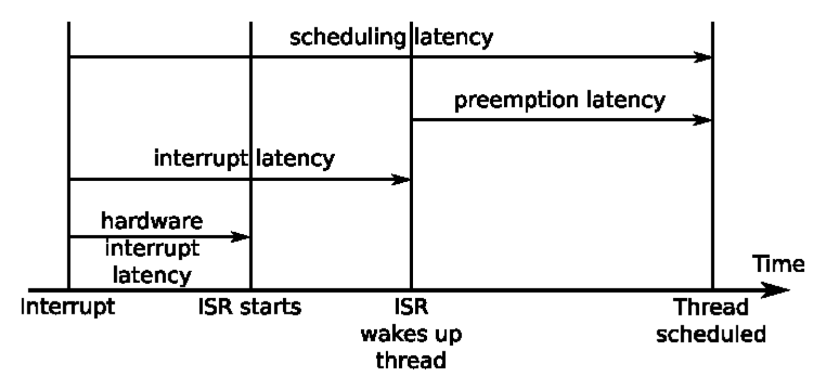
>>> Plan

1. Real-time Programming
    - Identifying the sources of non-determinism
    - Scheduling latency
    - Kernel preemption
    - The real-time Linux kernel
    - Thread priorities
    - PREEMPT RT patches
    - Threaded interrupt handlers
    - Preemptible kernel locks
    - cyclictest
    - cyclictest no preemption
    - cyclictest standard preemption
    - RT preemption cyclictest
    - Ftrace
    - Further reading

>>> Identifying the sources of non-determinism

* Scheduling: Real-time threads must be scheduled before others so they must have a real-time policy, SCHED_FIFO, or SCHED_RR. Additionally they should have priorities assigned in descending order starting with the one with the shortest deadline, according to the theory of Rate Monotonic Analysis.
* Scheduling latency: The kernel must be able to reschedule as soon as an event such as an interrupt or timer occurs, and not be subject to unbounded delays.
* Priority inversion: This is a consequence of priority-based scheduling, which leads to unbounded delays when a high priority thread is blocked on a mutex held by a low priority thread. User space has priority inheritance and priority ceiling mutexes; in kernel space we have rt-mutexes which implement priority inheritance and which I will talk about in the section on the real-time kernel.
* Accurate timers: If you want to manage deadlines in the region of low milliseconds or microseconds, you need timers that match. High resolution timers are crucial and are a configuration option on almost all kernels.
* Page faults: A page fault while executing a critical section of code will upset all timing estimates. You can avoid them by locking memory, as I describe later on.
* Interrupts: They occur at unpredictable times and can result in unexpected processing overhead if there is a sudden flood of them. There are two ways to avoid this. One is to run interrupts as kernel threads, and the other, on multi-core devices, is to shield one or more CPUs from interrupt handling. I will discuss both possibilities later.

>>> Scheduling latency



>>> Kernel preemption

* CONFIG_PREEMPT_NONE: no preemption
* CONFIG_PREEMPT_VOLUNTARY: enables additional checks for requests for preemption
* CONFIG_PREEMPT: allows the kernel to be preempted

>>> The real-time Linux kernel

* is running an interrupt or trap handler
* is holding a spin lock or in an RCU critical section. Spin lock and RCU are kernel locking primitives, the details of which are not relevant here
* is between calls to preempt_disable() and preempt_enable()
* hardware interrupts are disabled

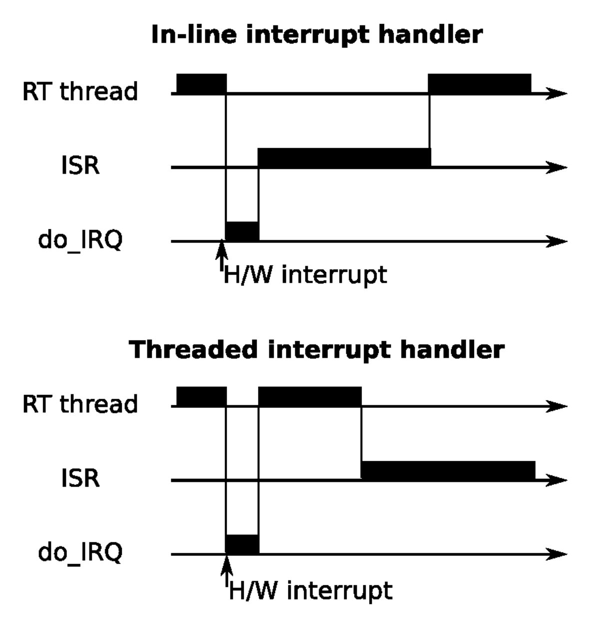
>>> Thread priorities

1. The POSIX timers thread, posixcputmr, should always have the highest priority.
2. Hardware interrupts associated with the highest priority real-time thread.
3. The highest priority real-time thread.
4. Hardware interrupts for the progressively lower priority real-time threads followed by the thread itself.
5. Hardware interrupts for non-real-time interfaces.
6. The soft IRQ daemon, ksoftirqd, which on RT kernels is responsible for running delayed interrupt routines and, prior to Linux 3.6, was responsible for running the network stack, the block I/O layer, and other things. You may need to experiment with different priority levels to get a balance.

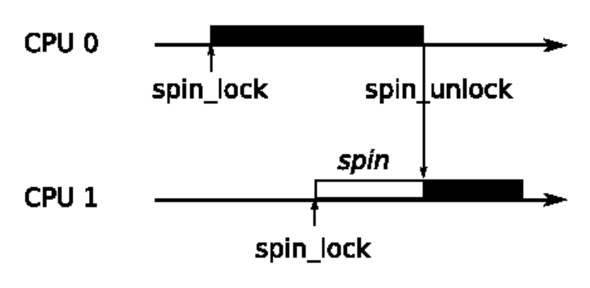
>>> PREEMPT_RT patches

https://www.kernel.org/pub/linux/kernel/projects/rt cd linux-4.1.10
zcat patch-4.1.10-rt11.patch.gz | patch -p1

>>> Threaded interrupt handlers



>>> Preemptible kernel locks



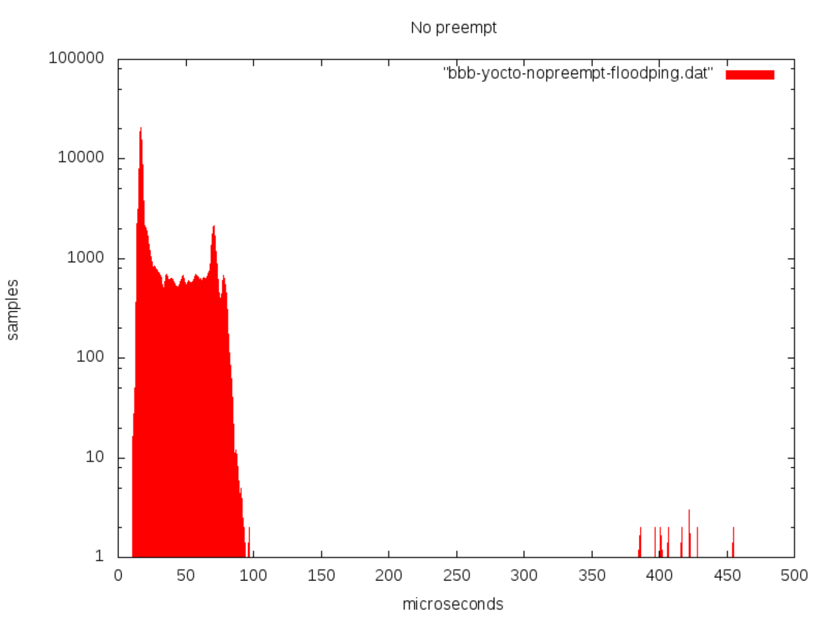
>>> cyclictest

```
bitbake core-image-rt
```

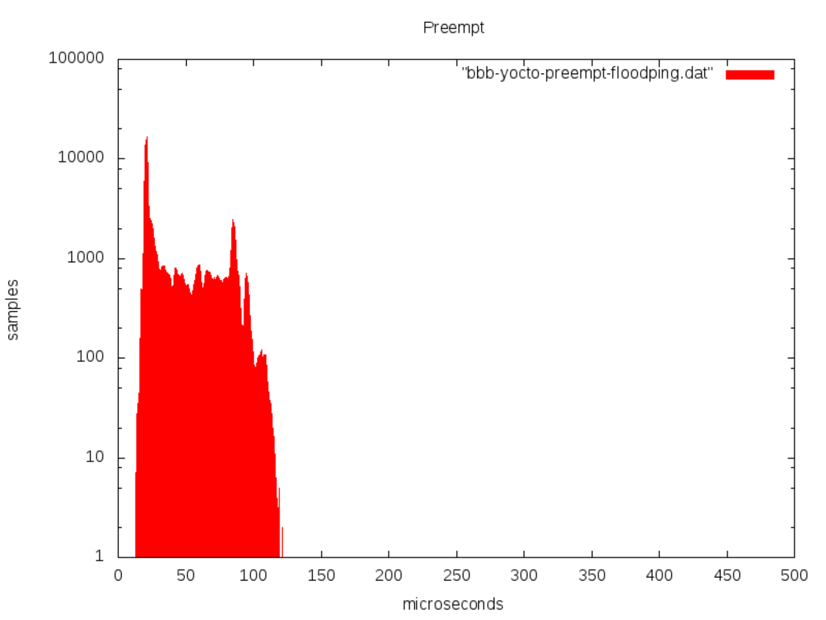
If you are using Buildroot, you need to add the package, BR2_PACKAGE_RT_TESTS in the menu Target packages | Debugging, profiling and benchmark | rt-tests.

```
cyclictest -p 99 -m -n -l 100000 -q -h 500 > cyclictest.data
```

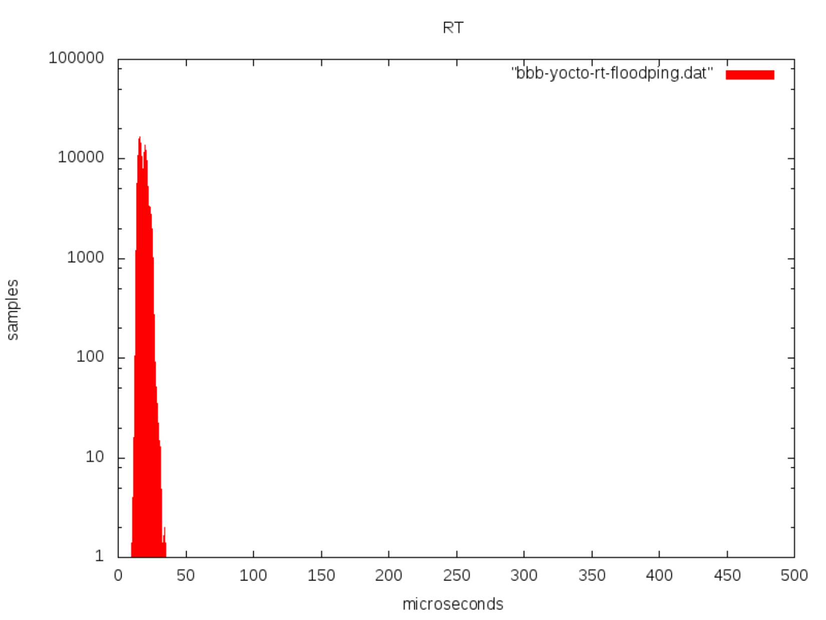
>>> cyclictest no preemption



>>> cyclictest standard preemption



\>>> RT preemption cyclictest



>>> Ftrace

* irqsoff: CONFIG_IRQSOFF_TRACER traces code that disables interrupts, recording the worst case
* preemptoff: CONFIG_PREEMPT_TRACER is similar to irqsoff, but traces the longest time that kernel preemeption is disabled (only available on preemptible kernels)
* preemptirqsoff: it combines the previous two traces to record the largest time either irqs and/or preemption is disabled
* wakeup: traces and records the maximum latency that it takes for the highest priority task to get scheduled after it has been woken up
* wakeup_rt: the same as wake up but only for real-time threads with the SCHED_FIFO, SCHED_RR, or SCHED_DEADLINE policies
* wakeup_dl: the same but only for deadline-scheduled threads with the SCHED_DEADLINE policy

```
echo preemptoff > /sys/kernel/debug/tracing/current_tracer
echo 0 > /sys/kernel/debug/tracing/tracing_max_latency
echo 1 > /sys/kernel/debug/tracing/tracing_on
sleep 60
echo 0 > /sys/kernel/debug/tracing/tracing_on
```

>>> Further reading

* Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications by Buttazzo, Giorgio, Springer, 2011
* Multicore Application Programming by Darryl Gove, Addison Wesley, 2011

>>> References

C. Simmonds.
Mastering Embedded Linux Programming.
Packt Publishing, 2015.