Programming for Linux PREEMPT_RT:
How to do it the right way?

Configuration of the Linux PREEMPT_RT kernel and beyond

Alexander Bähr
Open Source Automation Development Lab (OSADL) eG

Project planning: hardware requirements

Our company is planning a project. The **control system** for the new model of a manufacturing machine has to be designed. The hardware has to fulfill some general aspects:

- Area of application
- Environment
- Required hardware connections, power supply
- Structural conditions, etc.

List of requirements (real-time)

.....and for the real-time application:

- ☐ Real-time properties (worst-case latency of 500 µs)
- ☐ Real-time capable network interface
- ☐ Isolated core for a real-time application

Hardware selection, from Linux perspective

Select hardware:

- **Architecture** supported by Linux/PREEMPT_RT
- Suitable for the field of application, especially with regard to the application **requirements:**
  - certain real-time properties must be fulfilled, in our case a **worst-case latency of 500 µs**
  - **real-time** capable **network interface**
  - **isolated core** for running a **real-time application**
- The **OSADL QA-Farm** (https://www.osadl.org/?id=850) can be helpful for a preselection

Estimated real-time capabilities

Selected Hardware:

- x86 Intel Core i5-8265UE
  - 1600 MHz
  - 4 core / 8 threads
  - Expected worst case latency, calculated with the rule of thumb ~63 µs

$$t_{Lat} = 10^5 * \frac{1}{freq} \Rightarrow t_{Lat} = 10^5 * \frac{1}{(1.6 * 10^9)^{\frac{1}{s}}} = 62{,}5\,\mu s$$

Getting the Kernel

Selection criteria for the **kernel version**:

- Preferably select the **latest longterm** version.
  (LTS → https://www.kernel.org/category/releases.html)
- Take a less recent sublevel, if the latest longterm release is not supported for real-time.
- Take a more recent kernel version if needed features are not available, but prepare for later upgrading to the subsequent longterm version.

Getting the Kernel

Get **Kernel** and **PREEMPT_RT** patches either:

- from git
  https://git.kernel.org/pub/scm/linux/kernel/git/rt/linux-stable-rt

or by:

- Downloading the sources of the vanilla Kernel from
  https://www.kernel.org/pub/linux/kernel/v[x].[y]/
- and the corresponding PREEMPT_RT patch from
  https://www.kernel.org/pub/linux/kernel/projects/rt/v[x].[y]/
- and patching the Kernel with PREEMPT_RT (e.g. by using quilt)

Getting the Kernel

Get **Kernel** and **PREEMPT_RT** patches either:

- from git
  https://git.kernel.org/pub/scm/linux/kernel/git/rt/linux-stable-rt

or by:

- Downloading the sources of the vanilla Kernel from
  https://www.kernel.org/pub/linux/kernel/v[x].[y]/
- and the corresponding PREEMPT_RT patch from
  https://www.kernel.org/pub/linux/kernel/projects/rt/v[x].[y]/
- and patching the Kernel with PREEMPT_RT (e.g. by using quilt)

Configuring the Kernel

Enable CONFIG_PREEMPT_RT under "***General Setup***"

- *Preemption Model → (x) Fully Preemptible Kernel (Real-Time)*

(only available in expert mode, CONFIG_EXPERT)

Disable:

- CONFIG_SLUB_CPU_PARTIAL
- CONFIG_SLUB_DEBUG
- CONFIG_DEBUG_PREEMPT

(Attention: These are enabled in many distro configurations)

Configuring the Kernel

- Disable *Kernel hacking* → *Debug Oops, Lockups and Hangs* →
  - *Detect Hung Task (CONFIG_DETECT_HUNG_TASK)*
  - *Detect Soft Lockups (CONFIG_SOFTLOCKUP_DETECTOR)*
  - *Detect Hard Lockups (CONFIG_HARDLOCKUP_DETECTOR)*
- Since many **debug** options can cause latencies, *e.g.* DEBUG_LOCKDEP, only activate these when they are needed.

Configuring the Kernel

In order to have tracing possibilities the following options can safely be configured on a production system:

- Kernel hacking → Tracers →
  - *Kernel Function Tracer*
  - *Enable kprobes-based dynamic events*
  - *Enable uprobes-based dynamic events*

Configuring the Kernel (optional)

Optional (only available with the OSADL add-on patches)

- *CPU/Task time and stats accounting → Provide individual CPU usage measurement based on idle processing*
- *Kernel patchset support → Enable access to patchset.tar.gz through /proc/patchset.tar.gz*

Configuring the Kernel (optional)

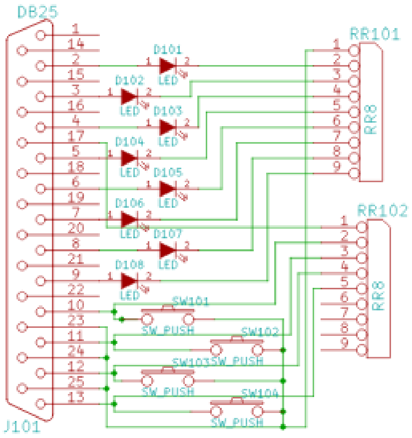
Under "*Kernel hacking*", available with the OSADL add-on patches:

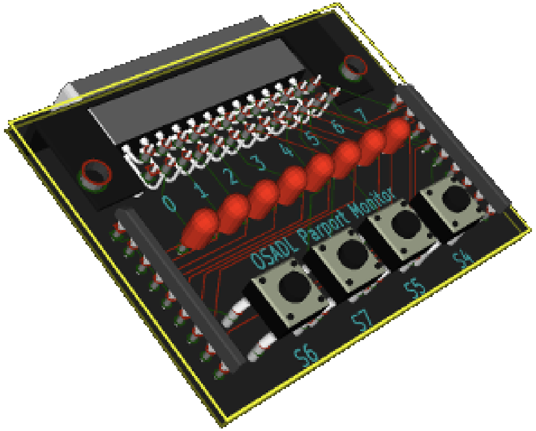
- *Enable kernel built-in latency histograms at → Kernel hacking → Tracers →*
  - *Missed Timer Offsets Histogram*
  - *Scheduling Latency Tracer*
  - *Scheduling Latency Histogram*
  - *Context Switch Time, Histogram, CPU/Task time and stats accounting*
  - *Provide individual CPU usage measurement based on idle processing*

Configuring the Kernel (optional)

Device driver to facilitate low-level kernel debugging via the parallel port under "***Device Drivers***", available with the OSADL add-on patches:

- *Misc devices → Raw output driver for parallel port*





Building the Kernel

```
$ make -j16
..
..
Kernel: arch/x86/boot/bzImage is ready (#1)
$ make modules_install install
..
..
$ reboot
..
```

Booting the real-time Kernel

Check if real-time preemption model is enabled:

```
$ uname -srv
Linux project 5.10.41-rt42 #1 SMP PREEMPT_RT Mon Mar
29 14:26:03 CET 2023
```

Configuring the operating system

Set the **scaling governor** to "performance"

- only required while running a real-time application, should be restricted to the applicable core:

```
$ for i in /sys/devices/system/cpu/cpu*/cpufreq/scaling_governor
do
  echo performance > $i
done
```

Configuring the operating system

Disable **sleep states** that can interfere with real-time requirements:

List the available **sleep states**:

```
$ ls -d1 /sys/devices/system/cpu/cpu0/cpuidle/state?
/sys/devices/system/cpu/cpu0/cpuidle/state0
/sys/devices/system/cpu/cpu0/cpuidle/state1
/sys/devices/system/cpu/cpu0/cpuidle/state2
/sys/devices/system/cpu/cpu0/cpuidle/state3
/sys/devices/system/cpu/cpu0/cpuidle/state4
/sys/devices/system/cpu/cpu0/cpuidle/state5
/sys/devices/system/cpu/cpu0/cpuidle/state6
/sys/devices/system/cpu/cpu0/cpuidle/state7
/sys/devices/system/cpu/cpu0/cpuidle/state8
```

Configuring the operating system

Disable **sleep states** that can interfere with real-time requirements:

List the **latencies** (in microseconds) caused by a particular state:

```
$ cat /sys/devices/system/cpu/cpu0/cpuidle/state?/latency
0
2
10
70
85
124
200
480
890
```

Configuring the operating system

To enable sleep states that are allowed, depending on the requirements, set the **maximum latency (µs)** in the psedo device

`/dev/cpu_dma_latency`

This device must be opened by a program, then written to and kept open throughout the run of the program, e.g. setting to "400" enables only sleep states the transition time of which is below 400 µs, in our case state 0 -> state 6.

```
int fd = open("/dev/cpu_dma_latency", O_WRONLY);
write(fd, "400", 3);
```

Configuring the operating system

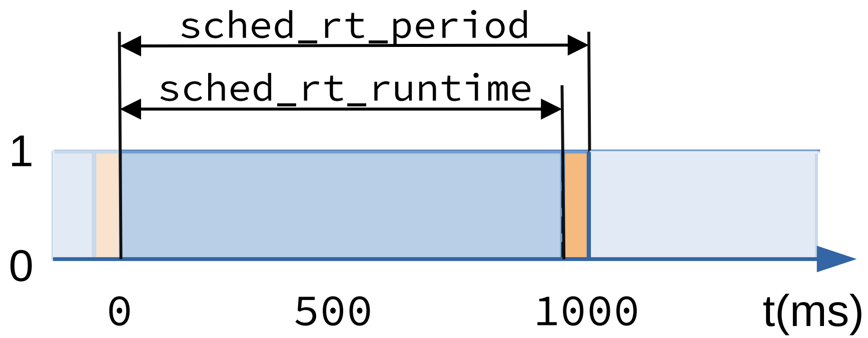
Completely disable CPU sleep states

- only required while running a real-time application, may be restricted to the applicable core and if the latency is too long as given in */sys/devices/system/cpu/cpu[0-9]*/cpuidle/state*/latency*):

```
$ for i in /sys/devices/system/cpu/cpu[0-9]*
  do
    cd $i
    for j in cpuidle/state*/disable
    do
      echo 1 > $j
    done
  done
```

Configuring the operating system

Set *RT_Throttling by setting the ratio between*
- *RT_Period*
- *RT_Runtime*




```
$ cat /proc/sys/kernel/sched_rt_period_us
1000000
$ cat /proc/sys/kernel/sched_rt_runtime_us
950000
```

RT_Throttling can be disabled by:

```
$ echo -1 >/proc/sys/kernel/sched_rt_runtime_us
```

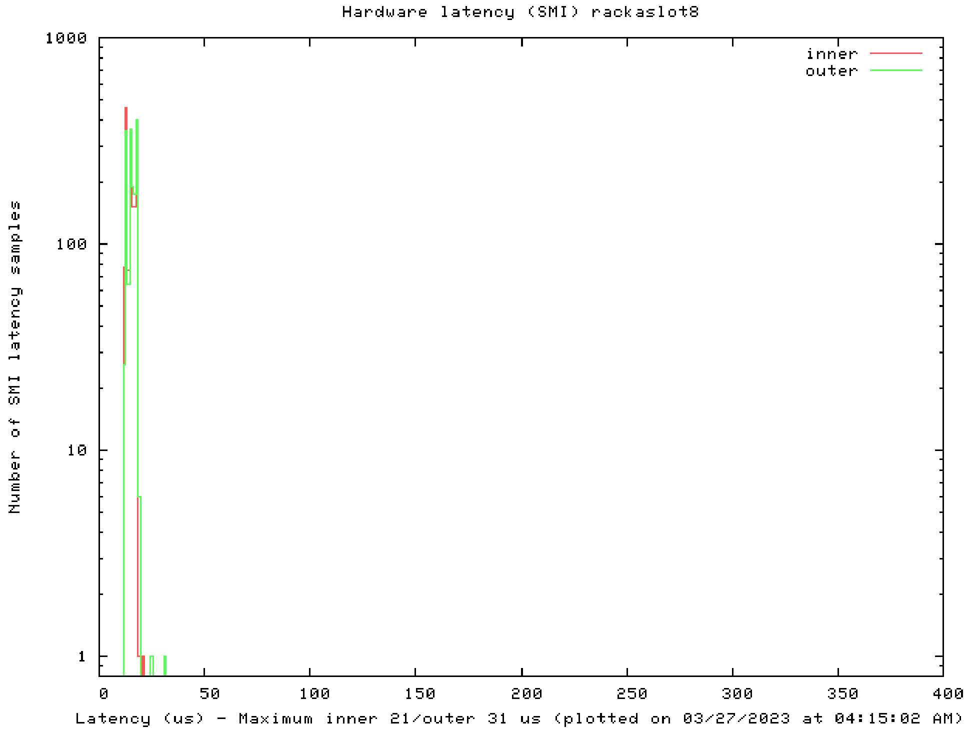
System latencies induced by hardware (SMI/NMI)

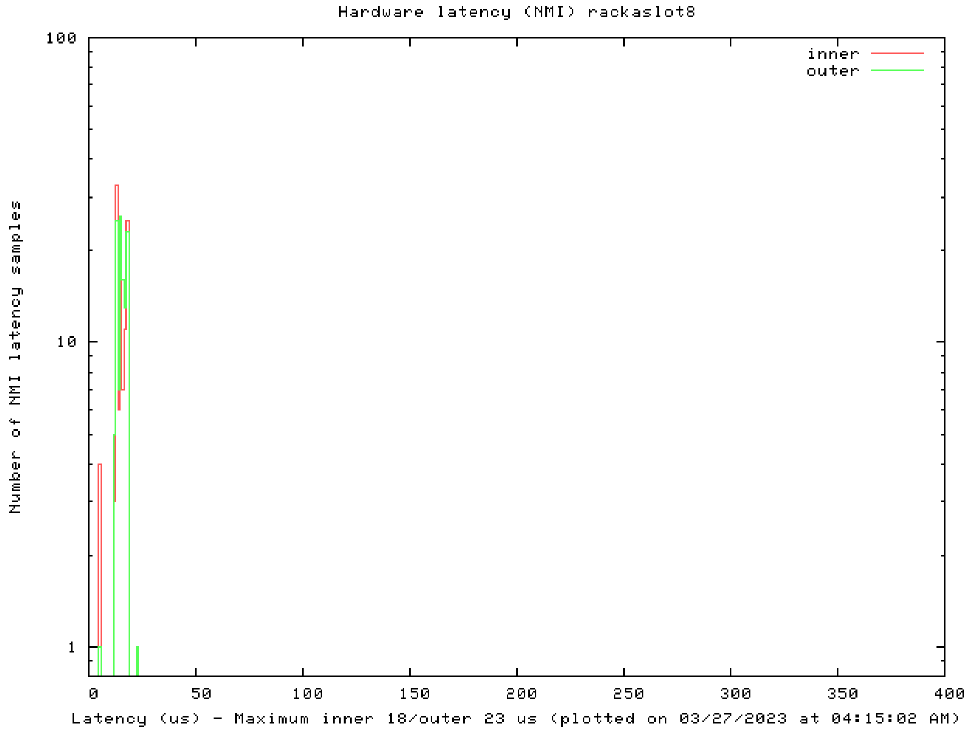
SMIs/NMIs are set up and serviced by BIOS code and not by the Linux kernel. Though, they can spend an inordinate amount of time in the handler (sometimes up to milliseconds).

To detect hardware latencies:

```
$ hwlatdetect
```

System latencies induced by hardware (SMI/NMI)





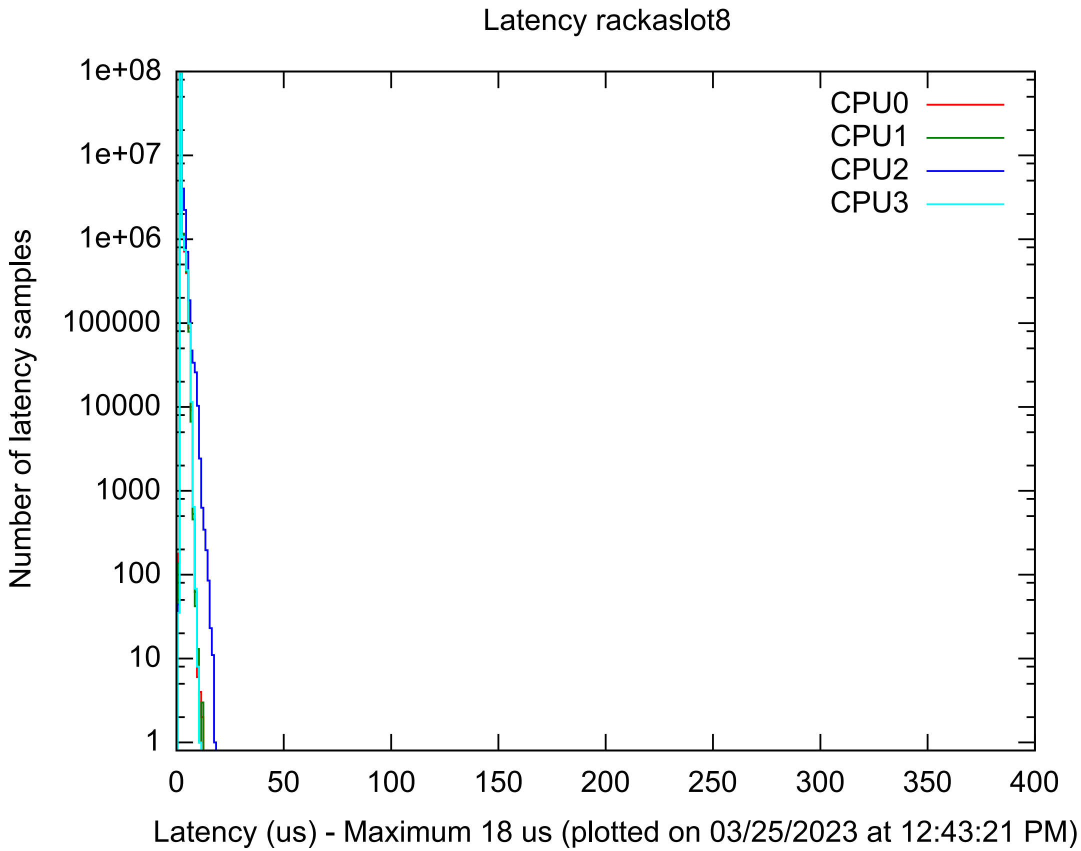
Measuring real-time capabilities

Installation and usage of ***cyclictest***

- *Cyclictest* is part of the *rt-tests*, available as tarball on *https://mirrors.edge.kernel.org/pub/linux/utils/rt-tests/* or via git *git://git.kernel.org/pub/scm/utils/rt-tests/rt-tests.git*

```
$ cyclictest -l100000000 -m -Sp98 -i200 -h400 -q >hist.txt
```

Latency plot of the real time system



Hardware:
x86 Intel Core i5-8265UE @1600 MHz,
Linux 5.10.41-rt42
Expected maximum latency,
calculated with the rule of thumb:

$$t_{Lat} = 10^5 * \frac{1}{freq} \Rightarrow t_{Lat} = 10^5 * \frac{1}{(1.6 * 10^9)^{\frac{1}{s}}} = 62{,}5\,\mu s$$

Additional measurement
(only available with OSADL add-on patches)

Internal latency measurement with built-in kernel histograms:

- Mount virtual debug filesystem:

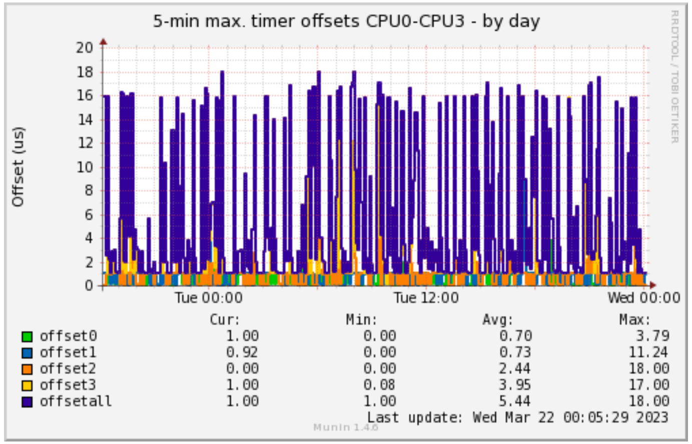
```
$ mount -t sysfs nodev /sys
$ mount -t debugfs nodev /sys/kernel/debug
```

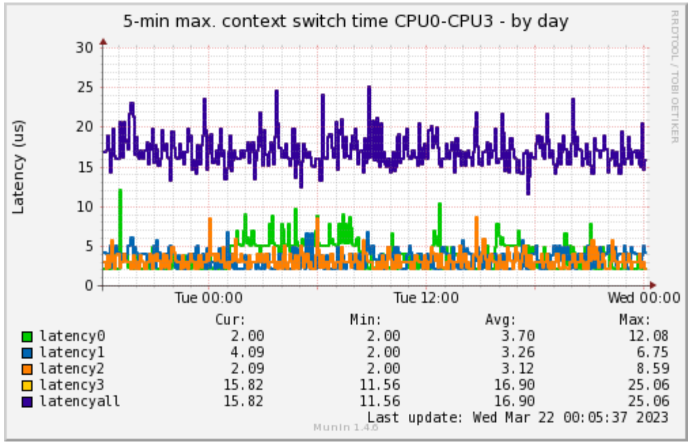
- Enable histograms (missed_timer_offsets, wakeup, switchtime, timerandwakeup, timerwakeupswitch)

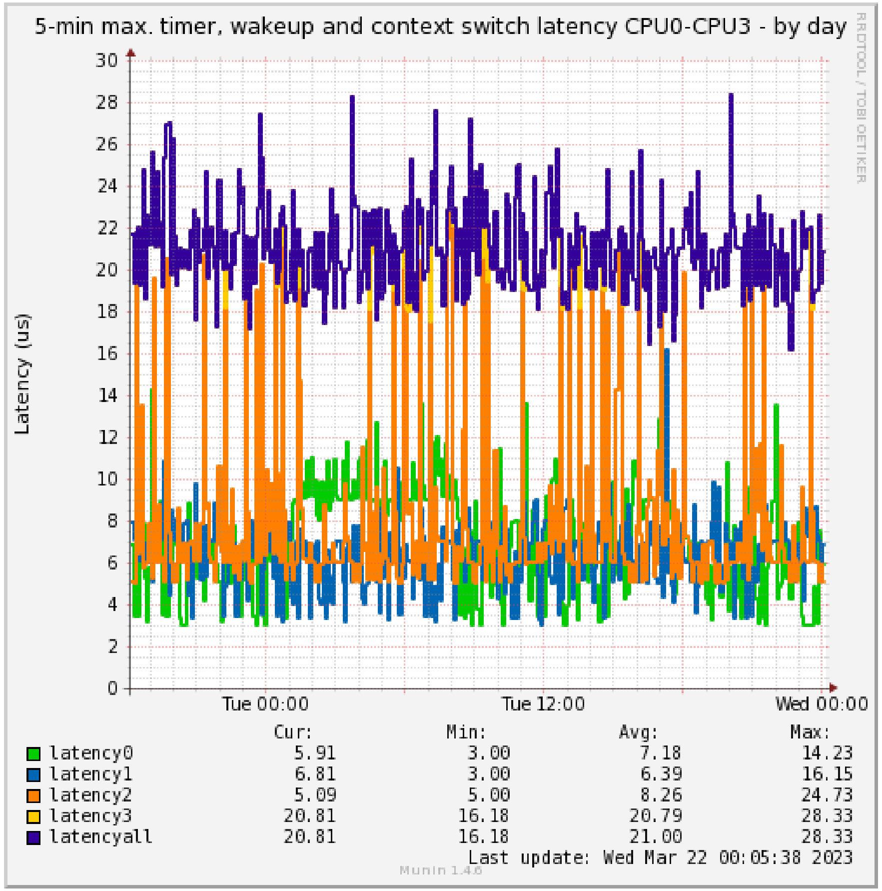
```
$ for i in /sys/kernel/debug/latency_hist/enable/*
do
echo 1 > $i
done
```

- Histograms per CPU in /sys/kernel/debug/latency_hist/*/CPU*

Built in histograms (results)







List of requirements (real-time)

- ☑ Real-time properties (worst-case latency of 500 µs)
- ☐ Real-time capable network interface
- ☐ Isolated core for a real-time application

Isolate cores for the real-time specific tasks

In order to keep the influence on the real-time processes as low as possible, it is recommended to run them on **isolated cores**. In the given use-case, we will therefore reserve one core for the operation of the network interface and one for the real-time application:

Core	Isolation	
1 (#0)	no	System applications
2 (#1)	no	System applications
3 (#2)	yes	Network interface (Interrupts)
4 (#3)	yes	Reserved for real-time application

Isolate cores for the real-time specific tasks

The cores can be **isolated** by setting the following kernel commandline parameters:

- `isolcpus` -> Isolate a given set of CPUs from disturbance
- `rcu_nocbs` -> Specified list of CPUs is set to no-callback mode from boot
- `nohz_full` -> Stop the tick on the specified list of CPUs whenever possible

Isolation of core 3(#2) and 4(#3):

```
BOOT_IMAGE=/boot/vmlinuz-5.10.41-rt42 isolcpus=2,3 nohz_full=2,3 rcu_nocbs=2,3
```

Isolate cores for the real time specific tasks

Move away **housekeeping threads** from isolated CPUs:

- switch specified CPUs off/on during boot process (*e.g.* in /etc/rc.local or via script)

```
# echo 0 > /sys/devices/system/cpu/cpu2/online
# echo 0 > /sys/devices/system/cpu/cpu3/online
# echo 1 > /sys/devices/system/cpu/cpu2/online
# echo 1 > /sys/devices/system/cpu/cpu3/online
```

Network IRQ routing on specified, isolated core

Irqbalance is a service which can reassign various IRQs to system CPUs depending on the workload involved. To avoid this on the RT system:

```
$ systemctl disable irqbalance
```

Set the default IRQ **affinity** for all interrupts:

```
$ cd /proc/irq
for i in [0-9]*
do
  echo 0-1 >$i/smp_affinity_list 2>/dev/null
done
```

Network IRQ routing on specified, isolated core

Set the **affinity** of the IRQ of the specific network interface (here: 124-128) to force the IRQ on the specific CPU core #3:

```
# for i in /proc/irq/12[4-8]
do
  echo 3 >$i/smp_affinity_list
done
```

- Note: The affinity of the interrupt threads follows the hardware routing.

Network IRQ routing on specified, isolated core

Set the **priority** of the **network interrupt threads** (irq/12[4-8]-enp1s0) in our case to 80
Note: By default all IRQ threads run at SCHED_FIFO 50

```
$ for i in `pgrep 'irq/[0-9]*-enp1s0'`
do
  chrt -fp 80 $i
done
```

Also set the **priority** of the **ksoftirqd** on the specific core

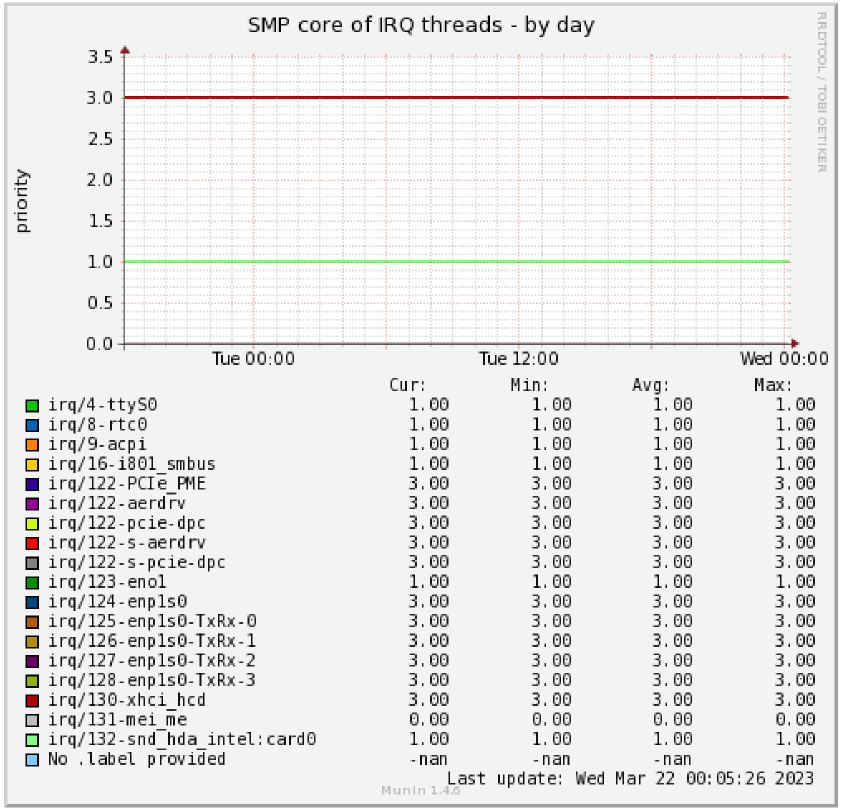
```
$ chrt -fp 80 `pgrep 'ksoftirqd/3'`
```

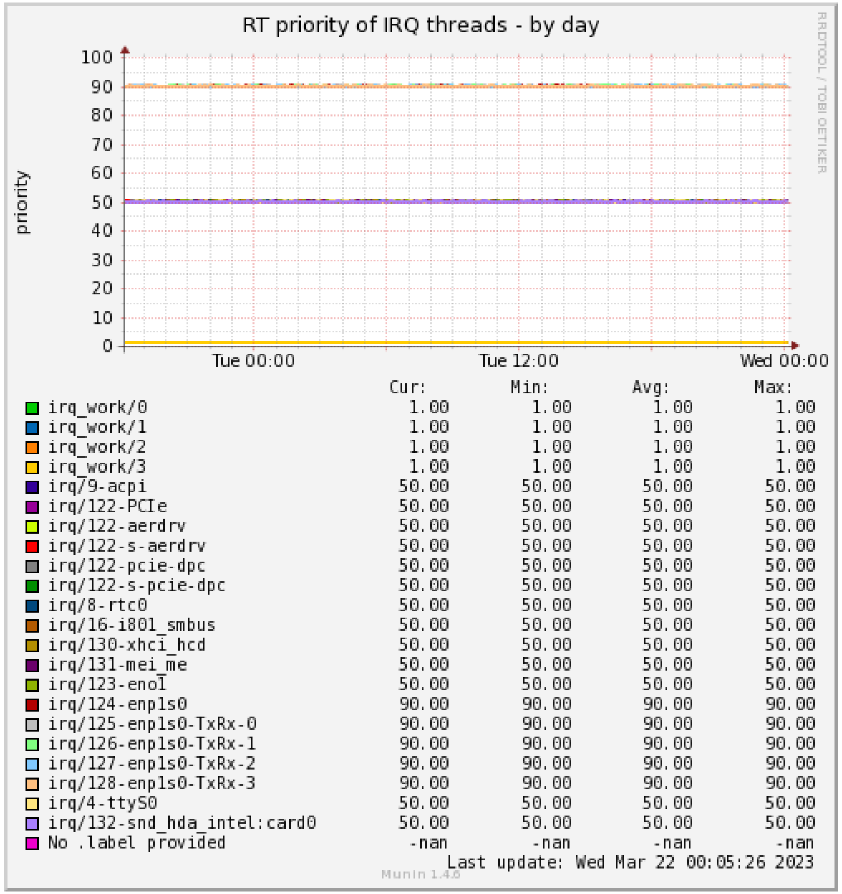
Top (IRQs + Priorities)

```
top - 18:57:43 up 32 days, 37 min,  2 users,  load average: 0.64, 0.77, 0.75
Tasks: 177 total,   1 running, 176 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.0 us,  0.2 sy,  0.0 ni, 99.8 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
MiB Mem :   3798.8 total,    480.1 free,    177.2 used,   3141.4 buff/cache
MiB Swap:   8064.0 total,   8064.0 free,      0.0 used.   3555.1 avail Mem

P      PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
3       46 root      20   0       0      0      0 S   0.0   0.0   0:00.00 cpuhp/3
3       51 root     -81   0       0      0      0 S   0.0   0.0   0:00.00 ksoftirqd/3
3       53 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 kworker/3:0H-events
3       77 root      20   0       0      0      0 I   0.0   0.0   0:35.72 kworker/3:1-events
3      168 root       0 -20       0      0      0 I   0.0   0.0   0:08.21 kworker/3:1H-events
3      206 root     -51   0       0      0      0 S   0.0   0.0   0:00.00 irq/130-xhci_hcd
3     1691 root     -81   0       0      0      0 S   0.0   0.0   0:17.36 irq/124-enp1s0
3     1692 root     -81   0       0      0      0 S   0.0   0.0   0:41.85 irq/125-enp1s0-TxRx-0
3     1693 root     -81   0       0      0      0 S   0.0   0.0   0:32.68 irq/126-enp1s0-TxRx-1
3     1694 root     -81   0       0      0      0 S   0.0   0.0   0:03.31 irq/127-enp1s0-TxRx-2
3     1695 root     -81   0       0      0      0 S   0.0   0.0   0:05.66 irq/128-enp1s0-TxRx-3
3     2115 root      20   0       0      0      0 I   0.0   0.0   0:00.00 kworker/3:2
3   225706 root      20   0  142012    716    612 S   0.0   0.0   0:00.00 turbostat
```

Measurement results of the isolated IRQs

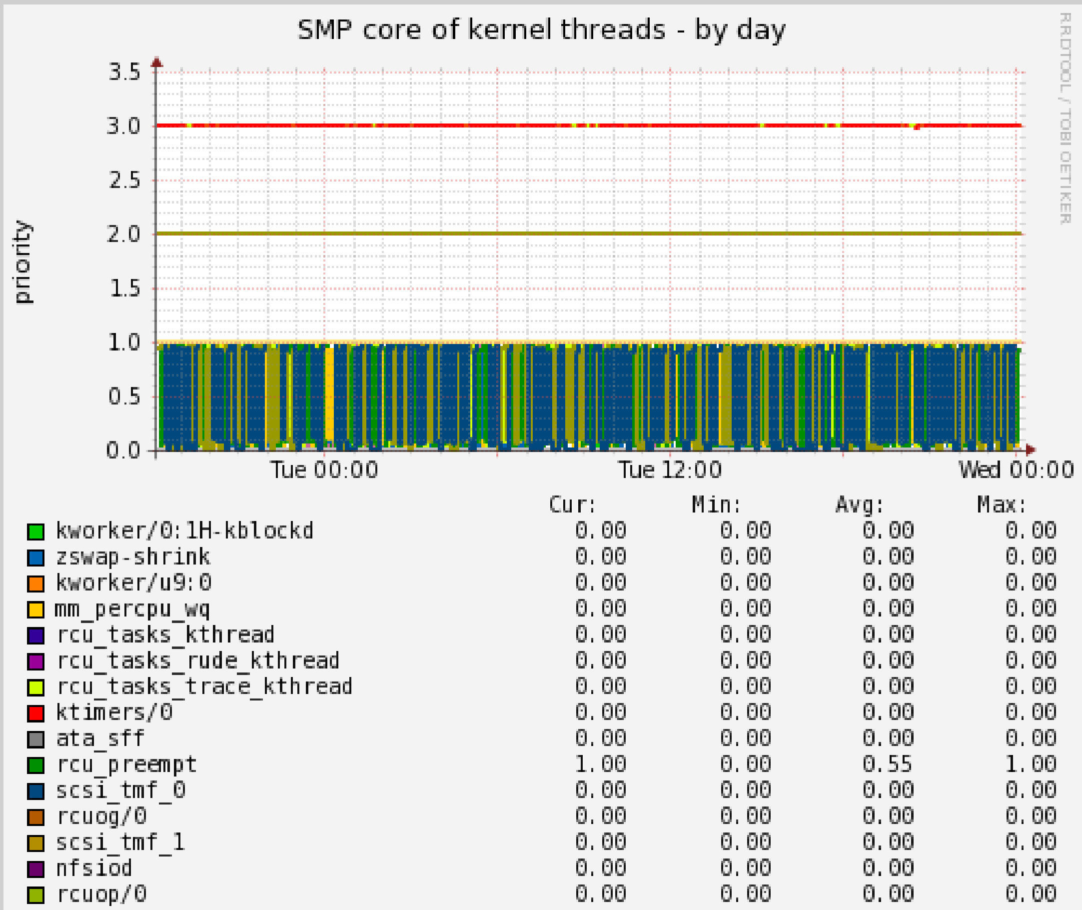




Set the affinity of the kernel and RT threads

- Determine the process IDs of all **kernel threads** and set their affinity mask to **0x3** (cores allowed: #0, #1)
  - This can be done and verified with the script at: https://www.osadl.org/?id=3661
  - The affinity can only be set for threads without the PF_NO_SETAFFINITY flag
- Set the affinity mask of the related **user-space** application to **0x4** and its priority to 97 (to run the RT task on core #2)

Measurement results of the Kthreads



List of requirements (real-time)

- ✔ Real-time properties (worst-case latency of 500 µs)
- ✔ Real-time capable network interface
- ✔ Isolated core for a real-time application

List of requirements (real-time)

- ✔ Real-time properties (worst-case latency of 500 µs)
- ✔ Real-time capable network interface
- ✔ Isolated core for real-time application

FULFILLED - PASSING THE UNIT OVER TO THE APPLICATION DEVELOPMENT DEPARTMENT