Tips, Tricks, and Gotchas

Linux Real-Time Tuning

Grațian Crișan

gratian.crisan@ni.com,
gratian@gmail.com
About me

- I work for NI (formerly known as National Instruments)
  - Makes hardware & software for test, measurement, and automation
  - Member and supporter of Real-Time Linux Collaborative Project
  - Real-Time OS group for the past decade
    - PREEMPT_RT based Linux kernels
    - Embedded 32-bit ARM and x86_64 systems
    - Distribution based on OpenEmbedded/Yocto
  - Maintainer for the Linux kernel shipping on NI’s Real-Time hardware
About this presentation

Covered:

• Real-Time
• Tools
• Tuning
• Safety nets
• Gotchas

Not covered:

• Implementation details
• In-depth review of tools
• RT alternatives
Real-Time == Deterministic response to stimulus

Events can be:
- Asynchronous
- Synchronous (clock driven)

We want the latency to be:
- Predictable
- Bounded
Bremsvorgang aktiv

Stadtverkehr

Achtung!

Ankunft: 15:34 Uhr

30 32
Measuring latency
Cyclic test

Simulate the load:
- iperf (network)
- fio (disk)
- hackbench (scheduler)

```c
static void *timerthread(void *param)
{
    while (!shutdown) {
        clock_gettime(clock, &before);
        clock_nanosleep(clock,…, &interval);
        clock_gettime(clock, &after);

        latency = after - before - interval;
        /* compute statistics/histogram */
    }
}
```
Histograms
static void *timerThread(...) {
    while (!shutdown) {
        clock_gettime(&before);
        read_inputs(...);
        process_data(...);
        update_outputs(...);
        clock_gettime(&after);
        latency = after - before;
        /* statistics */
    }
}

I/O latency
“Single point” tests
Total system latency

CPU(s) → I/O → ADC → DAC → ... → CPU(s)
Other tools

- RTEval: https://wiki.linuxfoundation.org/realtime/documentation/howto/tools/rteval
- LTP: https://wiki.linuxfoundation.org/realtime/documentation/howto/tools/ltp
- RTLA: https://docs.kernel.org/tools/rtla/index.html

RTLA: Real-time Linux Analysis Toolset - Daniel Bristot De Oliveira, Red Hat
Thursday, Jun 23, 4:55pm; Room 203/204 (Level 2)
Debugging
Debugging tools

- ftrace
- trace-cmd
- Kernel Shark
- LTTng, etc.
- perf
- bpftrace, bcc
- GPIO + oscilloscope
Kernel

Patch (for now):

CONFIG_PREEMPT_RT = y

General Setup
Preemption Model (Fully Preemptible Kernel (Real-Time)) → (X) Fully Preemptible Kernel (Real-time)

Verify with:
```
# uname -a
Linux NI-PXIe-8880-03096F84 5.15.40-rt43-00095-g915fbd285457 #1 SMP PREEMPT_RT Tue May 24 16:02:43 CDT 2022 x86_64 x86_64 x86_64 GNU/Linux
```
Scheduling policy and priority

- Identify RT workloads
- Assign scheduling policy & priority\(^1\)[\(^2\)]:
  - SCHED_FIFO: 1-98 priority
  - SCHED_DEADLINE: runtime, deadline, period
- Also adjust RT priorities for:
  - IRQ threads, kernel threads, etc.
- Run everything else as:
  - SCHED_OTHER or lower

\(^1\) [https://man7.org/linux/man-pages/man1/chrt.1.html](https://man7.org/linux/man-pages/man1/chrt.1.html)
CPU affinity

- Partition CPUs:
  - `cpuset` (cgroup v1), `chrt`, `sched` syscalls

- IRQ affinities:
  - `/proc/irq/default_smp_affinity`
  - `/proc/irq/*/smp_affinity`

- Kernel workqueue threads:
  - `find /sys/devices/virtual/workqueue -name "cpumask"`

- Isolate CPUs for sensitive real-time workloads:
  - kernel parameters: `isolcpus=7 nohz_full=7`
  - `CONFIG_NO_HZ_FULL`
RCU

CONFIG_RCU_NOCB_CPU = y

General Setup →
RCU Subsystem →
[*] Offload RCU callback processing from boot-selected CPUs

Control at boot via kernel parameters:

rcu_nocbs[=cpu-list]
rcu_nocb_poll

Verify with:

```
# ps ax | grep rcuop
  15 ?  S   0:21 [rcuop/0]
  28 ?  S   0:00 [rcuop/1]
...
```
Memory

Avoid memory allocations in real-time contexts:
```c
malloc();
```

Consider resolving symbols at start-up:
```bash
# LD_BIND_NOW=1
# export LD_BIND_NOW
```

Lock pages in memory:
```c
#include <sys/mman.h>
int mlockall(MCL_CURRENT | MCL_FUTURE);
```

Delay the vmstat timer far away into the future:
```bash
sysctl vm.stat_interval=999
```
Clock sources

Check the current clock source:

```bash
# cat /sys/devices/system/clocksource/clocksource0/current_clocksource
tsc
```

On Intel hardware pick TSC if available:

```bash
# cat /sys/devices/system/clocksource/clocksource0/available_clocksource
tsc hpet acpi_pm
```

Don’t forget about the trace clock:

```bash
# cat /sys/kernel/debug/tracing/trace_clock
[local] global counter uptime perf mono mono_raw boot x86-tsc
```
Power management

Disable CPU frequency scaling:

```bash
CONFIG_CPU_FREQ = N
```

Disable power management at boot via kernel parameters[1]:

```bash
intel_pstate= [X86]
intel_idle.max_cstate= [KNL,HW,ACPI,X86]
processor.max_cstate= [HW,ACPI]
```

Disable c-states at run-time:

```bash
for CSTATE in /sys/devices/system/cpu/cpu*/cpuidle/state[^0]/disable;do
echo 1 > $CSTATE
done
```

Firmware configuration

Disable:

- Power management: P-states, C-states
- SMT (hyper-threading)[1]
- Intel Turbo Boost
- EDAC or configure to lowest functional level
- Unused peripherals and legacy hardware
- Vendor specific options that affect performance

[1] Core scheduling can be an alternative in kernels >= 5.14 (https://lwn.net/Articles/861251)
Removing safety nets

Disable RT throttling:
```bash
# echo -1 > /proc/sys/kernel/sched_rt_runtime_us
```

Disable clocksource watchdog:
```bash
tsc=nowatchdog
```

Disable soft-lockup detector:
```bash
nosoftlockup
```

Disable both lockup detectors:
```bash
nowatchdog
```

Ignore corrected errors:
```bash
mce=ignore_ce
```
Memory

Don’t overcommit memory:

```bash
# echo 2 > /proc/sys/vm/overcommit_memory
# sysctl -w vm.overcommit_ratio=<ratio>
```

Prioritize processes to kill:

```bash
# echo 1000 > /proc/self/oom_score_adj
# echo -17 > /proc/12465/oom_adj
```

Decide what to do when out of memory:

```bash
# echo 1 > /proc/sys/vm/panic_on_oom
```
Security mitigations

```
mitigations=
[X86,PPC,S390,ARM64] Control optional mitigations for CPU vulnerabilities. This is a set of curated, arch-independent options, each of which is an aggregation of existing arch-specific options.

off

Disable all optional CPU mitigations. This improves system performance, but it may also expose users to several CPU vulnerabilities.
Equivalent to: nopti [X86,PPC]
kpti=0 [ARM64]
nospectre_v1 [X86,PPC]
nobp=0 [S390]
nospectre_v2 [X86,PPC,S390,ARM64]
spectre_v2_user=off [X86]
spec_store_bypass_disable=off [X86,PPC]
ssbd=force-off [ARM64]
l1tf=off [X86]
mds=off [X86]
tsx_async_abort=off [X86]
kvm.nxhuge_pages=off [X86]
no_entry_flush [PPC]
no_uaccess_flush [PPC]
```
System Management Interrupts (SMI)

- High priority un-maskable hardware interrupts, handled in firmware
- Used for temperature management, legacy hardware emulation, hardware bugs etc.
- The OS is unaware of transitions to/from System Management Mode (SMM)
- x86 specific but other architectures have similar privileged modes:
  - e.g., Secure Monitor Mode on ARM
  - [https://wiki.linuxfoundation.org/realtime/documentation/howto/debugging/smi-latency/start](https://wiki.linuxfoundation.org/realtime/documentation/howto/debugging/smi-latency/start)
Interrupts

Request threaded interrupt handlers:

switched driver to explicitly call: request_threaded_irq()
MMIO CPU stalls

~400 µS added latency when accessing TPM chip
Priority inversions

Priority inversions occur when a process with a high priority (H) preempts a process with a low priority (L) on a lock. This results in the high-priority process blocking on the lock and waiting for the low-priority process to release it. The waiting process then runs, executing for an unbounded amount of time, leading to a situation where the low-priority process is starved for resources.

The diagram illustrates the sequence of events:
- **L acquires lock**
- **H preempts L**
- **H blocks on lock**
- **unbounded latency**
- **L releases lock**

This unbounded latency can cause the system to become unresponsive and degrade performance.
unbounded latency
Priority inheritance

H preempts L
H blocks on lock
H acquires lock
M runs

bounded latency

L acquires lock
L priority boosted to H prio
L releases lock
Priority inheritance
Lack of priority inheritance support

**With** priority inheritance support:

- `pthread_mutex_*`

**Without** priority inheritance support:

- `pthread_barrier_*`
- `pthread_cond_*`
- `pthread_rwlock_*`
- `sem_*`

- `FUTEX_LOCK_PI/UNLOCK_PI`
  (enabled via mutex attributes)

- `FUTEX_WAIT/WAKE`
- `FUTEX_WAIT_BITSET/WAKE`

**No way** of setting priority inheritance attribute on `std::mutex()`
Partial solution

- librtpi\textsuperscript{[1]}\textsuperscript{[2]}

  \textsuperscript{[1]} https://github.com/dvhart/librtpi
  \textsuperscript{[2]} https://github.com/gratian/librtpi

- PI mutex and condvar

- Taking suggestions for RT-aware libraries implementing POSIX locks
Interrupt priority inversions

Context:
- Watchdog functionality implemented in a CPLD hanging of a I\textsuperscript{2}C bus
- It can be configured to fire an interrupt (as opposed to a straight reset)

Behavior:
- High priority watchdog interrupt fires
- To acknowledge the interrupt slow I\textsuperscript{2}C transfers need to happen
- I\textsuperscript{2}C interrupt has low priority
- Some unrelated mid-priority irq preempts the I\textsuperscript{2}C interrupt
Futex “trick”

diff --git a/kernel/futex.c b/kernel/futex.c
index c15ad276fd15..9c0393631d02 100644
--- a/kernel/futex.c
+++ b/kernel/futex.c
@@ -3954,6 +3954,10 @@ long do_futex(u32 __user *uaddr, int op, u32 val, ktime_t *timeout,
     case FUTEX_CMP_REQUEUE_PI:
         if (!futex_cmpxchg_enabled)
             return -ENOSYS;
+    default:
+        /* debug: catch non-pi futexes */
+        if (task_is_realtime(current))
+            force_sigsegv(SIGSEGV);
     }

switch (cmd) {
Thread 2 "low_th" received signal SIGSEGV, Segmentation fault.
[Switching to Thread 0x7fffffff7dc1640 (LWP 2441)]
...

(gdb) bt
#0  futex_wait (private=0, expected=0, futex_word=0x404144 <start_barrier+4>) at 
   ../sysdeps/nptl/futex-internal.h:146
#1  futex_wait_simple (private=0, expected=0, futex_word=0x404144 <start_barrier+4>) at 
   ../sysdeps/nptl/futex-internal.h:177
#2  __pthread_barrier_wait (barrier=0x404140 <start_barrier>) at pthread_barrier_wait.c:184
#3  0x000000000000401514 in low_tf (p=0x0) at pi.c:91
#4  0x00007ffff7fa3d08 in start_thread (arg=0x7fffffff7dc1640) at pthread_create.c:481
#5  0x00007fffffff7ec0123 in clone () at ../sysdeps/unix/sysv/linux/x86_64/clone.S:95
Summary

• Real-Time tools

• Tuning knobs

• Removing safety nets

• Gotchas to avoid
Summary

• Real-Time tools
• Tuning knobs
• Removing safety nets
• Gotchas to avoid