DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

CONCURRENT AND REAL TIME PROGRAMMING
[INQ0091623] AA 2022-23

Lab 11

Realtime Linux Tunings

Gabriele Manduchi <gabriele.manduchi@unipd.it>

Andrea Rigoni Garola <andrea.rigonigarola@unipd.it>

TOPICS:

Realtime is about timing precision on performing some operations.

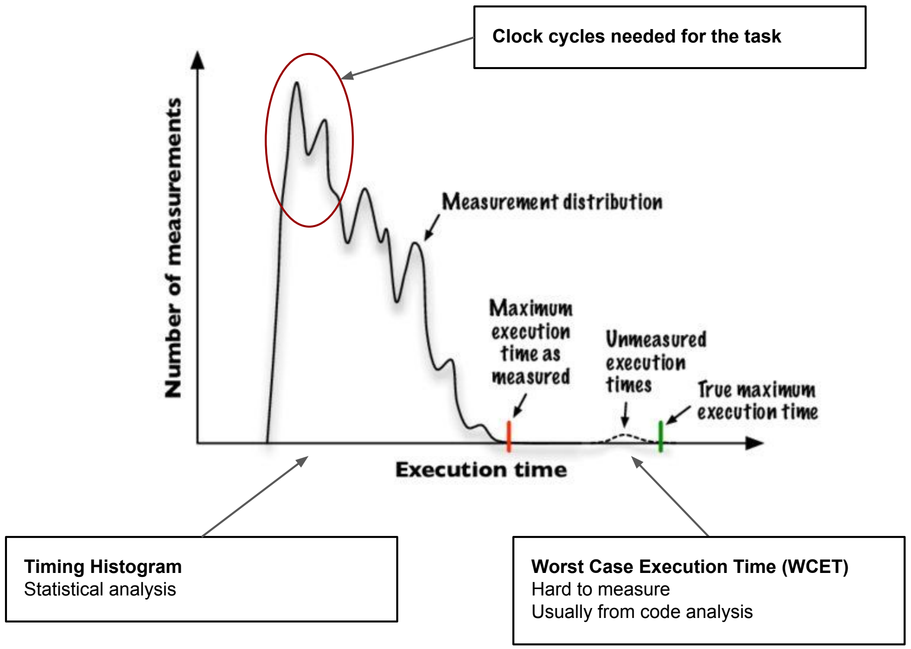
So it reflects into:

1. Measure time delays for a running and interrupt in case the time spans over a limit that would affect other tasks. This also needs a precise knowledge of what time actually is in every moment!

2. Remove as much as possible the noise that comes from other non related tasks and causes the jitter in the process operation times.

HO TO DO THAT:

- Design application with care !
- Tune the operative system

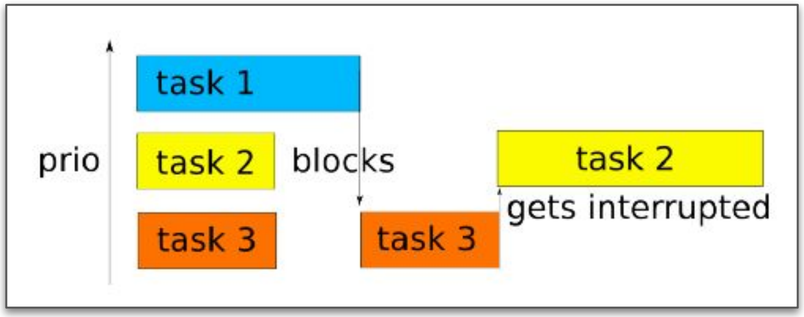

Clock cycles needed for the task
Number of measurements
Measurement distribution
Maximum execution time as measured
Unmeasured execution times
True maximum execution time
Execution time
Timing Histogram
Statistical analysis
Worst Case Execution Time (WCET)
Hard to measure
Usually from code analysis

Realtime in GNU Linux

Adapting GNU Linux to act in real-time

General requisites of a Realtime Operating System:

- Deterministic timing behaviour
- Task Preemption
- Task Priority (Inheritance / Ceiling)



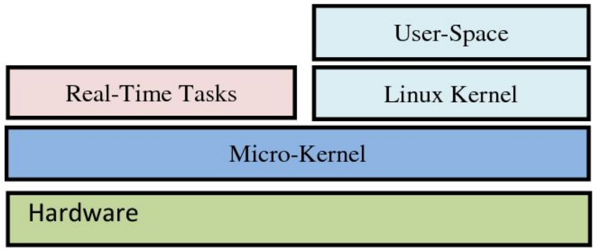
Avoid the unfortunate case of Task Priority inversion where a shared resource with a low priority task affect the actual timing of a realtime task.

Linux to Linux-RT

Two general approaches have been proposed:

1. Dual-Kernel configuration

   This approach relies on splitting the entire operating system in two pieces: the standard kernel and a micro-kernel that is specifically designed to handle real-time processes.



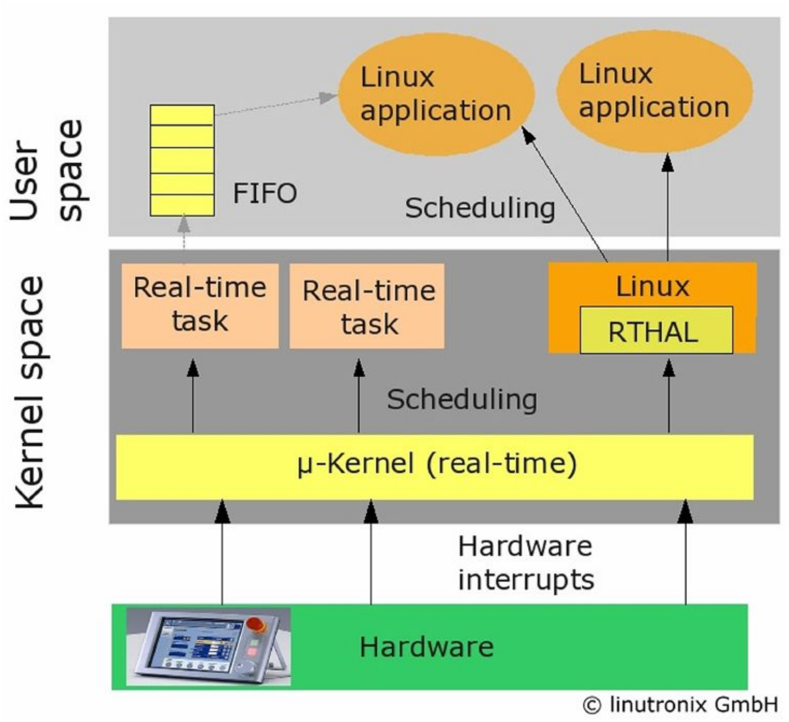
2. Single Kernel / In-Kernel

   The Linux kernel itself can be adapted (patched) to provide a better scheduling of the real-time processes by improving prehemption of all the kernel internal functions.

Popular Dual-kernel approaches

RTAI

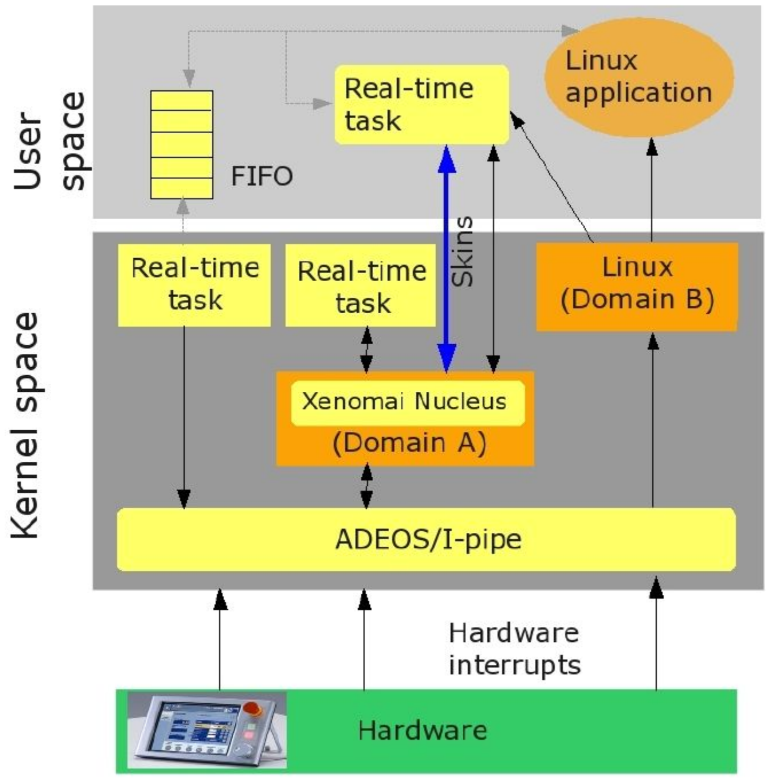
- Prof. Paolo Mantegazza, University of Milano
- Dual-Kernel approach
- Realtime in kernel-space
- Realtime in user-space very limited
- Design goal: Lowest latencies
- Supported platforms: x86, x86_64, and a couple of ARM platforms



Popular Dual-kernel approaches

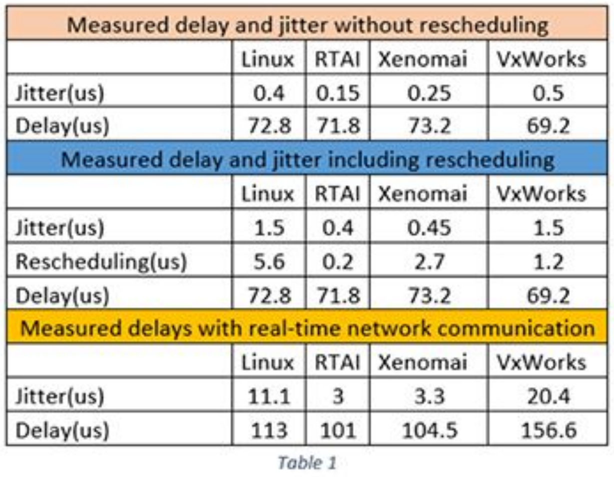
Xenomai

- Private company founded in 2001
- Realtime in userspace
- Skins can emulate the API of different RTOSes
- Dual-Kernel approach
- Supported platforms: x86, x86_64, PowerPC, ARM, ia64



Known issues of dual-kernel approaches

- Special API
- Special tools and libraries
- Microkernel needs to be ported for new HW and new Linux versions
- Bad scaling on big platforms



Measured delay and jitter without rescheduling				
	Linux	RTAI	Xenomai	VxWorks
Jitter(us)	0.4	0.15	0.25	0.5
Delay(us)	72.8	71.8	73.2	69.2
Measured delay and jitter including rescheduling				
	Linux	RTAI	Xenomai	VxWorks
Jitter(us)	1.5	0.4	0.45	1.5
Rescheduling(us)	5.6	0.2	2.7	1.2
Delay(us)	72.8	71.8	73.2	69.2
Measured delays with real-time network communication				
	Linux	RTAI	Xenomai	VxWorks
Jitter(us)	11.1	3	3.3	20.4
Delay(us)	113	101	104.5	156.6

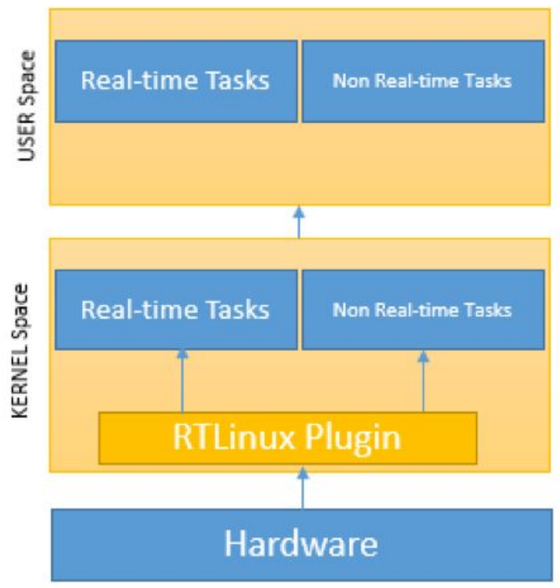
Table 1

Barbalace, A. Luchetta, G. Manduchi,M. Moro, A. Soppelsa, and C. Taliercio. Performance comparison of vxworks, linux, rtai, and xenomai in a hard real-time application. Nuclear Science, IEEE Transactions on, 55(1):435–439, 2008.

Single Kernel

The **PREEMPT_RT** patch

- In-Kernel approach
- Founded by: Thomas Gleixner, Ingo Molnar
- Huge community
- Most of the features already made it into "Mainline"
- POSIX realtime
- Highly accepted in the community



How Preempt RT brings Realtime to Linux?

It looks like the real-time (RT) patches for the Linux kernel are almost to the point of being fully upstream in the mainline Linux kernel. Merged for Linux 5.15 is the PREEMPT_RT locking core that represents a bulk of the outstanding RT patches.

http://lkml.iu.edu/hypermail/linux/kernel/2108.3/05759.html

MAIN CHANGES:

The RT-Preempt patch converts Linux into a fully preemptible kernel. This is done through:

- Making in-kernel locking-primitives (using **spinlocks**) preemptible by reimplementation with **rtmutexes**.
- **Critical sections** protected by i.e. spinlock_t and rwlock_t are now preemptible. The creation of non-preemptible sections (in kernel) is still possible with raw_spinlock_t (same APIs like spinlock_t).
- Implementing **priority inheritance** for in-kernel spinlocks and **semaphores**.
- Converting interrupt handlers into preemptible kernel threads: The RT-Preempt patch treats soft interrupt handlers in kernel thread context, which is represented by a task_struct like a common user space process. However it is also possible to register an IRQ in kernel context.
- Converting the old Linux timer API into separate infrastructures for high resolution kernel timers plus one for timeouts, leading to user space POSIX timers with high resolution.

Install PREEMPT_RT patched kernel

Most of the PREEMPT_RT patches are already merged into latest kernel versions:

[GIT pull] locking/core for **v5.15-rc1**

But if you want a pure last patched RT kernel you need to install a specific version:

There are two git repositories hosting the source code of the Linux mainline kernel versions with the additional PREEMPT_RT Patch.

http://git.kernel.org/cgit/linux/kernel/git/rt/linux-rt-devel.git

http://git.kernel.org/cgit/linux/kernel/git/rt/linux-stable-rt.git

Or you can install from archLinux user repository (AUR)

https://wiki.archlinux.org/title/Realtime_kernel_patchset

Linux Realtime tuning

Linux Scheduler

EVOLUTION OF LINUX SCHEDULER OVER TIME:

- **O(n) Scheduler [Linux 2.4 to 2.6]:** The scheduler would simply iterate over the tasks, applying a goodness function (metric) to determine which task to execute next. At every context switch it needs to find the next task candidate to run in O(n), when n is the number of runnable processes. Although this approach was relatively simple, it was relatively inefficient, lacked scalability, and was weak for real-time systems.

- **O(1) Scheduler [Linux 2.6 to 2.6.12]:**

  The O(1) scheduler kept track of runnable tasks in a run queue (actually, two run queues for each priority level, one for active and one for expired tasks), which meant that to identify the task to execute next, the scheduler simply needed to dequeue the next task off the specific active per-priority run queue.

- **CFS Scheduler**

  Linux 2.6.13. up to latest (5.15.7 8 December 2021)

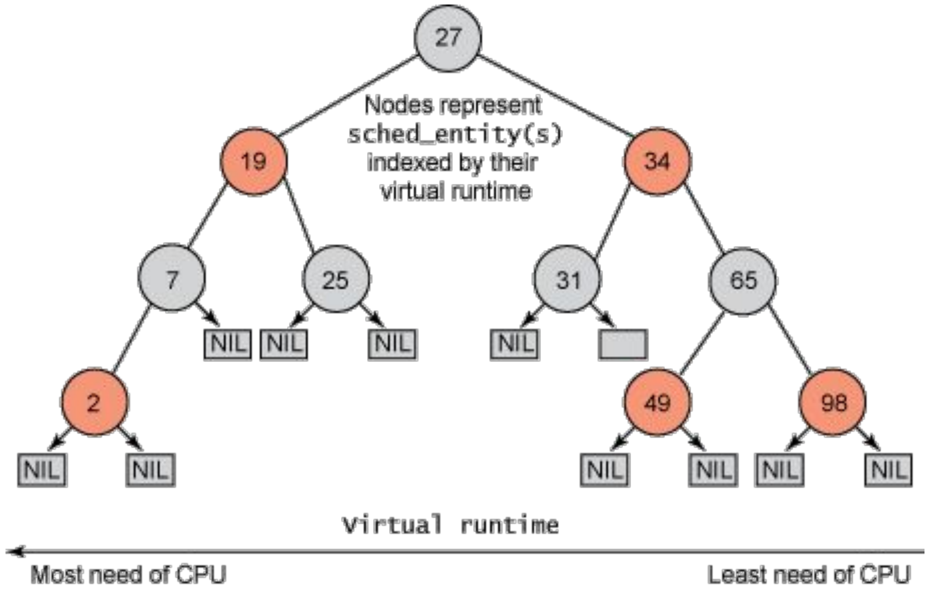
CFS scheduler implementation

The main idea behind the CFS is to maintain balance (fairness) in providing processor time to tasks. This means processes should be given a fair amount of the processor.

But rather than maintain the tasks in a run queue the CFS maintains a time-ordered **red-black tree**.

red-black tree:

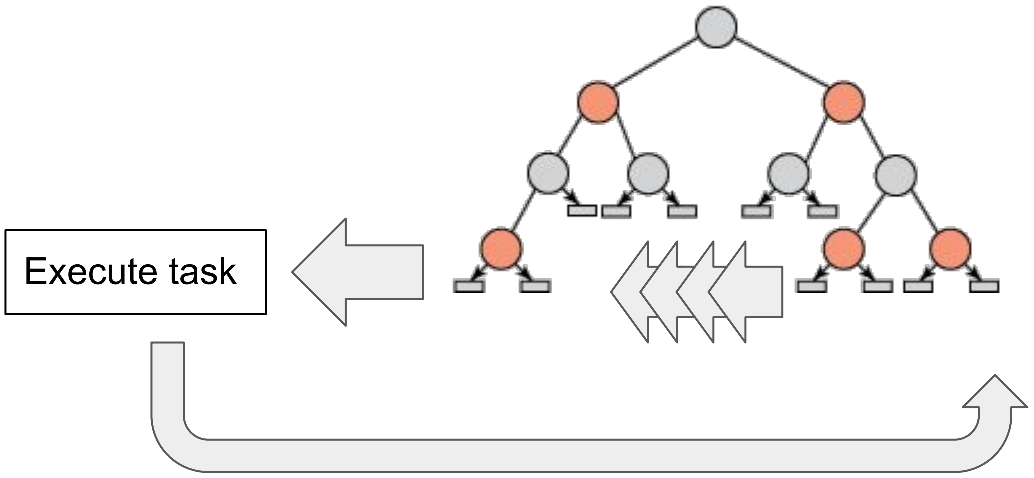
- self-balancing, which means that no path in the tree will ever be more than twice as long as any other.
- all operations on the tree occur in O(log n) time (where n is the number of nodes in the tree). This means that you can insert or delete a task quickly and efficiently.



CFS scheduler implementation

The scheduler then, to be fair, picks the left-most node of the red-black tree to schedule next to maintain fairness.

The task accounts for its time with the CPU by adding its execution time to the virtual runtime and is then inserted back into the tree if runnable. In this way, tasks on the left side of the tree are given time to execute, and the contents of the tree migrate from the right to the left to maintain fairness. Therefore, each runnable task chases the other to maintain a balance of execution across the set of runnable tasks.



Linux uses two main scheduling policies categories:

1) **normal policies:** Threads with a priority of 0.

SCHED_OTHER (sometimes called **SCHED_NORMAL**)

This is the default thread policy and has dynamic priority controlled by the kernel. The priority is changed based on thread activity.

2) **realtime policies:** Threads with priority 1 - 99 (99 the highest).

- **SCHED_FIFO** (First in, first out)

SCHED_FIFO threads always have a higher priority than SCHED_OTHER threads. Any thread created as a SCHED_FIFO thread has a fixed priority and will run until it is blocked or preempted by a higher priority thread, and it will always immediately preempt any currently running SCHED_OTHER.

- **SCHED_RR** (Round-Robin)

SCHED_RR is a modification of SCHED_FIFO. Threads with the same priority have a quantum and are round-robin scheduled among all equal priority SCHED_RR threads. This policy is rarely used.

modify scheduling of a process

Scheduler policy

- sched_setscheduler(2) Set the scheduling policy and parameters of a specified thread.
- sched_getscheduler(2) Return the scheduling policy of a specified thread.

SYNOPSIS

```
#include <sched.h>

 int sched_setscheduler(pid_t pid, int policy,
                        const struct sched_param *param);

 int sched_getscheduler(pid_t pid);
```

pid is the ID of the thread that can be obtained with pthread_self()

SCHED_OTHER	the standard round-robin time-sharing policy;
SCHED_BATCH	for "batch" style execution of processes; and
SCHED_IDLE	for running *very* low priority background jobs.
SCHED_FIFO	a first-in, first-out policy; and
SCHED_RR	a round-robin policy.

modify priority of a process

Thread priority

- getpriority(2) Return the nice value of a thread, a process group, or the set of threads owned by a specified user.
- setpriority(2) Set the nice value of a thread, a process group, or the set of threads owned by a specified user.

SYNOPSIS

```
#include <sys/resource.h>

 int getpriority(int which, id_t who);
 int setpriority(int which, id_t who, int prio);
```

The scheduling priority of the process, process group, or user, as indicated by *which* and *who* is obtained with the **getpriority**() call and set with the **setpriority**() call. The process attribute dealt with by these system calls is the same attribute (also known as the "nice" value) that is dealt with by nice(2).

Target processes are specified by the values of the ***which*** and ***who*** arguments. The *which* argument may be one of the following values: PRIO_PROCESS, PRIO_PGRP, or PRIO_USER, indicating that the *who* argument is to be interpreted as a process ID, a process group ID, or an effective user ID, respectively. A 0 value for the *who* argument specifies the current process, process group, or user.

modify nice of a process

nice change process priority

SYNOPSIS

```
#include <unistd.h>

 int nice(int inc);
```

nice() adds *inc* to the nice value for the calling thread. (A higher nice value means a lower priority.) The range of the nice value is +19 (low priority) to -20 (high priority). Attempts to set a nice value outside the range are clamped to the range.

Traditionally, only a privileged process could lower the nice value (i.e., set a higher priority). However, since Linux 2.6.12, an unprivileged process can decrease the nice value of a target process that has a suitable **RLIMIT_NICE** soft limit.

NOTE: NICE VS PRIORITY (PR)

Nice value is a user-space and priority PR is the process's actual priority that use by Linux kernel.
In linux system **priorities are 0 to 139 in which 0 to 99 for real time and 100 to 139 for users**.
nice value range is -20 to +19 where -20 is highest, 0 default and +19 is lowest.

EXAMPLE:

```
  PID USER       PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
 4586 ipc-adm+  20   0 1303900 605152  92844 S  30,6 29,3   3:52.88 firefox
 3985 ipc-adm+  20   0  258588 124508  63072 S  12,2  6,0   0:40.04 compiz
 3092 root      20   0  172392  56164  25980 S   6,1  2,7   0:30.13 Xorg
```

PR = 20 + NI

start a program with positive nice (user)

```
nice -n 10 sleep 10m &
nice -n 19 sleep 10m &
```

start a program with negative nice (root)

```
nice -n -10 sleep 10m &
nice -n -20 sleep 10m &
```

look for a specific program name with nice and priority using top

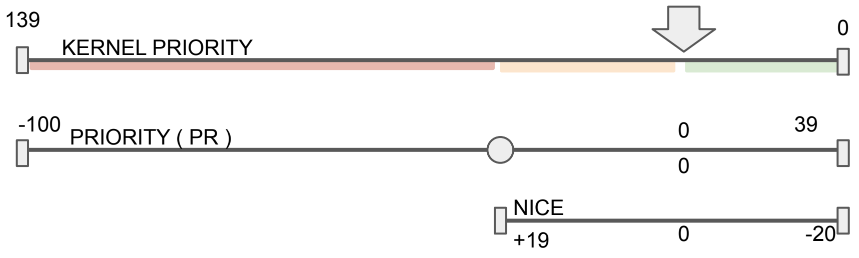
```
top -p `pgrep -d "," sleep`
```

```
    PID USER       PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
3869619 andrea     30  10    6144    292    208 S   0.0   0.0   0:00.00 sleep
3869658 andrea     39  19    6144    292    208 S   0.0   0.0   0:00.00 sleep
3869712 root       10 -10    6112    936    848 S   0.0   0.0   0:00.00 sleep
3869735 root        0 -20    6112    940    848 S   0.0   0.0   0:00.00 sleep
```

look program nice and priority

ps a -o pid,pri,ni,cmd

```
3869619   9  10 sleep 10m
3869658   0  19 sleep 10m
3869712  29 -10 sleep 10m
3869735  39 -20 sleep 10m
```



starting a realtime process

NAME

```
chrt - manipulate the real-time attributes of a process
```

SYNOPSIS

```
chrt [options] priority command argument
chrt [options] -p [priority] PID
```

DESCRIPTION

```
chrt sets or retrieves the real-time scheduling attributes of an existing PID,
or runs command with the given attributes.
```

POLICIES top

```
 -o, --other
      Set scheduling policy to SCHED_OTHER (time-sharing
      scheduling). This is the default Linux scheduling policy.

  -f, --fifo
      Set scheduling policy to SCHED_FIFO (first in-first out).

  -r, --rr
      Set scheduling policy to SCHED_RR (round-robin scheduling).
      When no policy is defined, the SCHED_RR is used as the
      default.
```

$\boxed{PR_{realtime} = -1 - rt\_prior}$

starting a REAL TIME process

```
sudo chrt -r 30 sleep 10m &
sudo chrt -r 50 sleep 10m &

top -p `pgrep -d "," sleep`

    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
3886921 root     -31   0    6112    940    852 S   0.0   0.0   0:00.00 sleep
3886945 root     -51   0    6112    936    848 S   0.0   0.0   0:00.00 sleep
```

```
ps a -o pid,pri,ni,cmd

3886921  70   - sleep 10m
3886945  90   - sleep 10m
```

Realtime linux test tools: **rt-tests**

rt-tests is a test suite, that contains programs to test various real time Linux features.

The following programs are part of the rt-tests:

- **cyclictest: latency detection**
- cyclicdeadline
- deadline_test
- **hackbench**
- **hwlatdetect**
- pip_stress
- pi_stress
- pmqtest
- ptsematest
- queuelat
- rt-migrate-test
- signaltest
- sigwaittest
- ssdd
- svsematest

INSTALL (archlinux): # pacman -Sy rt-tests

Hardware Latency Detector

Inside kernel there is a special module called: hwlat_detector.ko that traces system latencies induced by the behavior of certain underlying hardware or firmware , independent of Linux itself (sometime called System Management Interrupts SMI).

Usage:

Write the ASCII text “hwlat” into the current_tracer file of the tracing system (mounted at /sys/kernel/tracing or /sys/kernel/tracing). It is possible to redefine the threshold in microseconds (us) above which latency spikes will be taken into account.

```
# echo hwlat > /sys/kernel/tracing/current_tracer
# echo 100 > /sys/kernel/tracing/tracing_thresh
```

in /sys/kernel/tracing:

- tracing_threshold - minimum latency value to be considered (usecs)
- tracing_max_latency - maximum hardware latency actually observed (usecs)
- tracing_cpumask - the CPUs to move the hwlat thread across
- hwlat_detector/width - specified amount of time to spin within window (usecs)
- hwlat_detector/window - amount of time between (width) runs (usecs)
- hwlat_detector/mode - the thread mode

Hardware Latency Detector

As an easier way the tool ***hwlatdetect*** can be also exploited:

```
hwlatdetect  [  --duration=<time>  ]  [--threshold=<usecs>  ]  [--window=<time interval> ]
             [--width=<time interval> ] [--report=<path> ] [--cleanup ] [--debug ] [--quiet ]
```

The hardware latency detector module works by hogging all of the cpus for configurable amounts of time (by calling stop_machine()), polling the CPU Time Stamp Counter for some period, then looking for gaps in the TSC data. Any gap indicates a time when the polling was interrupted and since the machine is stopped and interrupts turned off the only thing that could do that would be an SMI.

System management interrupt (SMI)

System management interrupts are high priority unmaskable hardware interrupts which cause the CPU to immediately suspend all other activities, including the operating system, and go into a special execution mode called system management mode (SMM) (Intel SDM, Vol.1 3.1). Once the system is in SMM, the interrupt is handled by firmware code.

The concepts of SMI and SMM are specific to x86, but similar high privilege processor modes exist on some other architectures. For example, on ARM there is a Secure Monitor mode that manages the switches between the Secure and Non-secure processor states.

hwlatdetect example

```
# hwlatdetect --duration=10s
hwlatdetect:  test duration 60 seconds
        detector: tracer
        parameters:
                Latency threshold: 10us
                Sample window:     1000000us
                Sample width:      500000us
                Non-sampling period:  500000us
                Output File:       None

Starting test
test finished
Max Latency: Below threshold
Samples recorded: 0
Samples exceeding threshold: 0
```

```
# hwlatdetect --duration=10s
hwlatdetect:  test duration 10 seconds
   detector: tracer
   parameters:
        Latency threshold: 10us
        Sample window:     1000000us
        Sample width:      500000us
        Non-sampling period:  500000us
        Output File:       None

Starting test
test finished
Max Latency: 28us
Samples recorded: 2
Samples exceeding threshold: 2
ts: 1639305616.400704793, inner:11, outer:14
ts: 1639305620.512866407, inner:28, outer:6
```

cyclictest

One of the programs in rt-tests is called cyclictest, which can be used to verify the maximum scheduling latency, and for tracking down the causes of latency spikes. cyclictest works by measuring the time between the expiration of a timer a thread sets and when the thread starts running again.

Here is the result of a typical test run:

```
# cyclictest --smp -p98 -m
# /dev/cpu_dma_latency set to 0us
policy: fifo: loadavg: 239.09 220.49 134.53 142/1304 23799

T: 0 (23124) P:98 I:1000 C: 645663 Min:      2 Act:    4 Avg:    4 Max:      23
T: 1 (23125) P:98 I:1500 C: 430429 Min:      2 Act:    5 Avg:    3 Max:      23
T: 2 (23126) P:98 I:2000 C: 322819 Min:      2 Act:    4 Avg:    3 Max:      15
T: 3 (23127) P:98 I:2500 C: 258247 Min:      2 Act:    5 Avg:    4 Max:      32
```

It shows a four CPU core system running one thread (SCHED_FIFO) per core at priority 98, with memory locked, the system is also under a high load due to running hackbench in a separate terminal (next slide). What is most interesting is the max scheduling latency detected, in this case 32 usecs on core 3.

put system under load

An idle kernel will tend to show much lower scheduling latencies, it is essential to put some load on it to get a realistic result. This can be done with another utility in the rt-tests package called ***hackbench***.

It works by creating multiple pairs of threads or processes, that pass data between themselves either over sockets or pipes.

To make it run longer add the -l parameter:

```
# hackbench -l 1000000
```

WARNING:

your system will become quite unresponsive because of the threads load..

Tuning BIOS parameters

Some BIOS options can be tuned to reduce **System independent noise**

1. **disable Power Management**

   Anything that tries to save power by either changing the system clock frequency or by putting the CPU into various sleep states can affect how quickly the system responds to external events.

2. **disable Error Detection and Correction (EDAC) units**

   EDAC units are devices used to detect and correct errors signaled from Error Correcting Code (ECC) memory. Usually EDAC options range from no ECC checking to a periodic scan of all memory nodes for errors. The higher the EDAC level, the more time is spent in BIOS, and the more likely that crucial event deadlines will be missed.

3. **tune the System Management Interrupts (SMI) WITH CARE !**

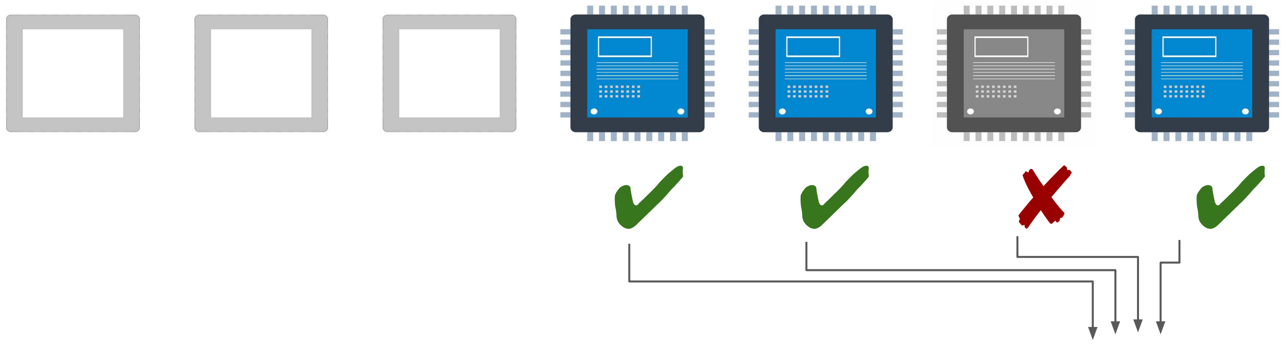
   SMIs are a facility used by hardware vendors ensure the system is operating correctly. Typically used for thermal management, remote console management (IPMI), EDAC checks, and various other housekeeping tasks. The SMI interrupts are usually not serviced by the running operating system, but by some code in the BIOS.

   WARNING: Removing the ability for your system to generate and service SMIs can result in catastrophic hardware failure.

CPU Binding

Real-time environments need to minimize or eliminate latency when responding to various events. Ideally, **interrupts (IRQs) and user processes can be isolated from one another on different dedicated CPUs.**

Identify CPU: with the CPU mask



CPU BIT_MASK: 0000 0000 0000 0000 0000 0000 0000 1101

The CPU mask is typically represented as a 32-bit bitmask (0x0000000d in the example).

Assign to a specific CPU unit:

- **Processes:** the process will be scheduled to run on that particular CPU.
- **IRQ:** once the interrupt occurs the handler code will be run at the specified CPU.

1 - IRQ Binding

If we are running a I/O critical realtime code we can improve the timing of the code by assigning the I/O interrupt to ONE or FEW cores.

WHY ??

> Interrupts are generally shared evenly between CPUs. This can delay interrupt processing through having to write new data and instruction caches, and often creates conflicts with other processing occurring on the CPU.

In this way, the code and data structures needed to process this interrupt will have the highest possible likelihood to be in the processor data and instruction caches. The dedicated process can then run as quickly as possible, while all other non-time-critical processes run on the remainder of the CPUs.

This can be particularly important in cases where the speeds involved are in the limits of memory and peripheral bus bandwidth available. Here, any wait for memory to be fetched into processor caches will have a noticeable impact in overall processing time and determinism.

1 - IRQ Binding: Disabling the irqbalance

Disabling Daemon:

The purpose of irqbalance is distribute hardware interrupts across processors on a multiprocessor system in order to increase overall computation performance. It can be controlled by the systemd **irqbalance.service**.

```
# systemctl status irqbalance
irqbalance.service - irqbalance daemon
  Loaded: loaded (/usr/lib/systemd/system/irqbalance.service; enabled)
  Active: active (running) …

# systemctl stop irqbalance
# systemctl disable irqbalance
```

Ban a specific CPU

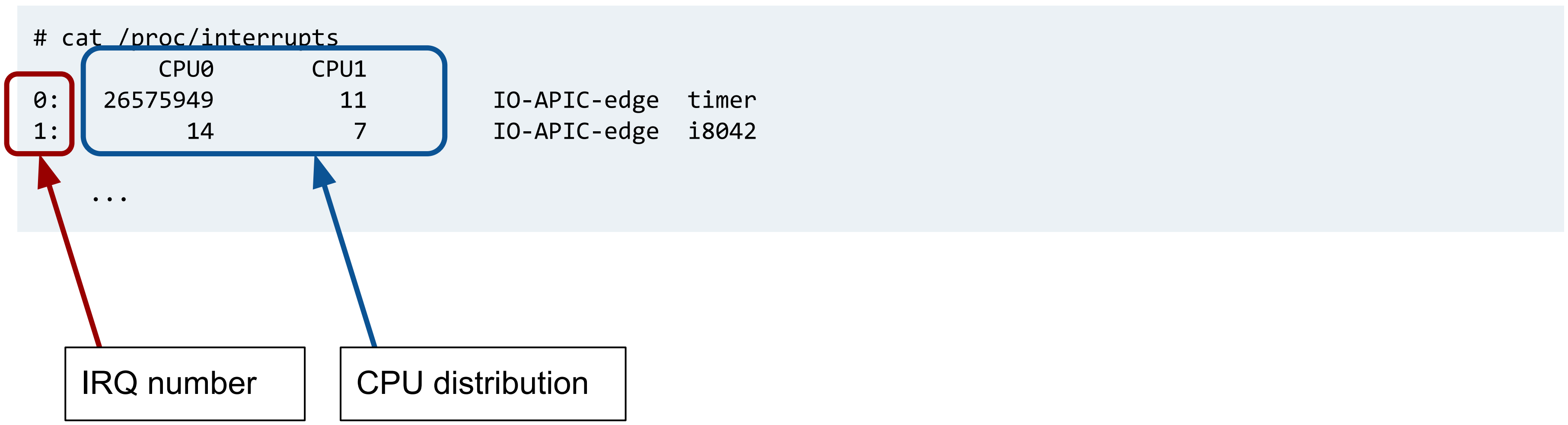
It is also possible to ban a single CPU instead of stopping balance completely:

The /etc/sysconfig/irqbalance configuration file contains a parameter named IRQBALANCE_BANNED_CPUS.

example: IRQBALANCE_BANNED_CPUS=0000ff00 excludes CPUs 8 to 15

Manually Assigning CPU Affinity to Individual IRQs

Check which IRQ is in use by each device by viewing the /proc/interrupts file:



```
# cat /proc/interrupts
        CPU0       CPU1
0:  26575949         11     IO-APIC-edge  timer
1:        14          7     IO-APIC-edge  i8042

   ...
```

To instruct an IRQ to run on only one processor, use the echo command to write the CPU mask, as a hexadecimal number, to the smp_affinity entry of the specific IRQ. In this example, we are instructing the interrupt with IRQ number 142 to run on CPU 0 only:

```
# echo 1 > /proc/irq/142/smp_affinity
```

Manually Assigning CPU Affinity to Individual IRQs

```
andrea@HP:~$ cat /proc/interrupts

           CPU0       CPU1       CPU2       CPU3       CPU4       CPU5       CPU6       CPU7
  0:          7          0          0          0          0          0          0          0  IO-APIC    2-edge      timer
  1:         68          0          0          4          0          0          0          0  IO-APIC    1-edge      i8042
  8:          0          0          0          0          1          0          0          0  IO-APIC    8-edge      rtc0
  9:         17          0          0          0          0          0          0          0  IO-APIC    9-fasteoi   acpi
 12:         85          0          6          0          0          0          0          0  IO-APIC   12-edge      i8042
 16:    1545144          0          0      34277          0          0          0          0  IO-APIC   16-fasteoi   ehci_hcd:usb1
 18:          0          0          0          0          0          0          0          0  IO-APIC   18-fasteoi   ata_generic, i801_smbus
 19:         34          0          2          0          0          0          0          0  IO-APIC   19-fasteoi   firewire_ohci
 20:         34          0          0          0          0          0          0          2  IO-APIC   20-fasteoi   firewire_ohci
 23:     953140        110          0          0          0          0          0          0  IO-APIC   23-fasteoi   ehci_hcd:usb2
 35:    1295705      39872          0          0          0          0          0      21702  PCI-MSI 512000-edge      ahci[0000:00:1f.2]
 37:          0          0          0          0          0          0          0          0  PCI-MSI 2625536-edge      xhci_hcd
 38:      30003          0          0          0          0          0          0         70  PCI-MSI 4194304-edge      xhci_hcd
 39:          0          0          0          0          0          0          0          0  PCI-MSI 4194305-edge      xhci_hcd
 40:          0          0          0          0          0          0          0          0  PCI-MSI 4194306-edge      xhci_hcd
 41:          0          0          0          0          0          0          0          0  PCI-MSI 4194307-edge      xhci_hcd
 42:          0          0          0          0          0          0          0          0  PCI-MSI 4194308-edge      xhci_hcd
 43:          0          0          0          0          0          0          0          0  PCI-MSI 4194309-edge      xhci_hcd
 44:          0          0          0          0          0          0          0          0  PCI-MSI 4194310-edge      xhci_hcd
 45:          0          0          0          0          0          0          0          0  PCI-MSI 4194311-edge      xhci_hcd
 46:         18          0          0          0          0          0          0          0  PCI-MSI 1048576-edge      isci-msix
 47:          0          0          0          0          0          0          0          0  PCI-MSI 1048577-edge      isci-msix
 50:          0         28          0          0          0          0          0          0  PCI-MSI 360448-edge      mei_me
 51:          0          0          0          0          0          0          0        155  PCI-MSI 442368-edge      snd_hda_intel:card1
 52:          0     104903          0          0          0          0          0      77692  PCI-MSI 409600-edge      eno1
 53:          0          0      23676          0          0          0          0       3226  IO-APIC   12-fasteoi   snd_hda_intel:card2
 54:    3927255          0          0          0          0     118517          0          0  PCI-MSI 2621440-edge      nvidia
NMI:          0        322        309        309        314        308        303        301   Non-maskable interrupts
LOC:   14407993   13613376   12624766   13746476   17157197   13651898   12426721   12165389   Local timer interrupts
SPU:          0          0          0          0          0          0          0          0   Spurious interrupts
PMI:          0        322        309        309        314        308        303        301   Performance monitoring interrupts
IWI:          0         23        236          0          2          6          0          2   IRQ work interrupts
RTR:          1          0          0          0          0          0          0          0   APIC ICR read retries
RES:     381318     448509     290968     336207     611861     433995     329030     270004   Rescheduling interrupts
CAL:    4067814    3504689    3274200    3283231    3136112    3175165    3181648    3179877   Function call interrupts
TLB:    5181425    5612943    5762193    5659371    5673287    5730733    5634244    5681658   TLB shootdowns
TRM:          0          0          0          0          0          0          0          0   Thermal event interrupts
THR:          0          0          0          0          0          0          0          0   Threshold APIC interrupts
DFR:          0          0          0          0          0          0          0          0   Deferred Error APIC interrupts
MCE:          0          0          0          0          0          0          0          0   Machine check exceptions
MCP:        388        382        382        382        381        381        381        381   Machine check polls
ERR:          0
MIS:          0
PIN:          0          0          0          0          0          0          0          0   Posted-interrupt notification event
NPI:          0          0          0          0          0          0          0          0   Nested posted-interrupt event
PIW:          0          0          0          0          0          0          0          0   Posted-interrupt wakeup event
```

2 - Process Binding: **taskset**

The `taskset` utility uses the process ID (PID) of a task to view or set the affinity, or can be used to launch a command with a chosen CPU affinity. In order to set the affinity, `taskset` requires the CPU mask expressed as a decimal or hexadecimal number. The mask argument is a bitmask that specifies which CPU cores are legal for the command or PID being modified.

1. To set the affinity of a process that is not currently running, use `taskset` and specify the CPU mask and the process. In this example, `my_embedded_process` is being instructed to use only CPU 3 (using the decimal version of the CPU mask).
   ```
   # taskset 8 /usr/local/bin/my_embedded_process
   ```

2. It is also possible to specify more than one CPU in the bitmask. In this example, `my_embedded_process` is being instructed to execute on processors 4, 5, 6, and 7 (using the hexadecimal version of the CPU mask).
   ```
   # taskset 0xF0 /usr/local/bin/my_embedded_process
   ```

3. Additionally, you can set the CPU affinity for processes that are already running by using the `-p` (`--pid`) option with the CPU mask and the PID of the process you wish to change. In this example, the process with a PID of 7013 is being instructed to run only on CPU 0.
   ```
   # taskset -p 1 7013
   ```

4. Lastly, using the `-c` parameter, you can specify a CPU list instead of a CPU mask. For example, in order to use CPU 0, 4 and CPUs 7 to 11, the command line would contain `-c 0,4,7-11`. This invocation is more convenient in most cases.

All scheduling system calls

```
nice(2)
       Set a new nice value for the calling thread, and return
       the new nice value.

getpriority(2)
       Return the nice value of a thread, a process group, or the
       set of threads owned by a specified user.

setpriority(2)
       Set the nice value of a thread, a process group, or the
       set of threads owned by a specified user.

sched_setscheduler(2)
       Set the scheduling policy and parameters of a specified
       thread.

sched_getscheduler(2)
       Return the scheduling policy of a specified thread.

sched_setparam(2)
       Set the scheduling parameters of a specified thread.

sched_getparam(2)
       Fetch the scheduling parameters of a specified thread.

sched_get_priority_max(2)
       Return the maximum priority available in a specified
       scheduling policy.

sched_get_priority_min(2)
       Return the minimum priority available in a specified
       scheduling policy.

sched_rr_get_interval(2)
       Fetch the quantum used for threads that are scheduled
       under the "round-robin" scheduling policy.
```

All scheduling system calls (2)

```
sched_yield(2)
          Cause the caller to relinquish the CPU, so that some other
          thread be executed.

sched_setaffinity(2)
          (Linux-specific) Set the CPU affinity of a specified
          thread.

sched_getaffinity(2)
          (Linux-specific) Get the CPU affinity of a specified
          thread.

sched_setattr(2)
          Set the scheduling policy and parameters of a specified
          thread.  This (Linux-specific) system call provides a
          superset of the functionality of sched_setscheduler(2) and
          sched_setparam(2).

sched_getattr(2)
          Fetch the scheduling policy and parameters of a specified
          thread.  This (Linux-specific) system call provides a
          superset of the functionality of sched_getscheduler(2) and
          sched_getparam(2).
```

CPU Isolation: **isolcpu**

isolcpus is a **kernel parameter** to isolate CPUs from the kernel scheduler.

Synopsis

isolcpus= *cpu_number* [, *cpu_number* ,...]

Remove the specified CPUs, as defined by the cpu_number values, from the general kernel SMP balancing and scheduler algorithms. The only way to move a process onto or off an "isolated" CPU is via the CPU affinity syscalls. cpu_number begins at 0, so the maximum value is 1 less than the number of CPUs on the system.

This option is the preferred way to isolate CPUs. The alternative, manually setting the CPU mask of all tasks in the system, can cause problems and suboptimal load balancer performance.

```
andrea@HP:/$ lscpu
Architecture:                    x86_64
CPU op-mode(s):                  32-bit, 64-bit
Byte Order:                      Little Endian
Address sizes:                   46 bits physical, 48 bits virtual
CPU(s):                          8
On-line CPU(s) list:             0-7
Thread(s) per core:              2
Core(s) per socket:              4
Socket(s):                       1
NUMA node(s):                    1
Vendor ID:                       GenuineIntel
CPU family:                      6
Model:                           45
Model name:                      Intel(R) Xeon(R) CPU E5-1620 0 @ 3.60GHz
Stepping:                        7
CPU MHz:                         1197.067
CPU max MHz:                     3800.0000
CPU min MHz:                     1200.0000
BogoMIPS:                        7185.65
L1d cache:                       128 KiB
L1i cache:                       128 KiB
L2 cache:                        1 MiB
L3 cache:                        10 MiB
NUMA node0 CPU(s):               0-7
Vulnerability Itlb multihit:     KVM: Mitigation: VMX unsupported
Vulnerability L1tf:              Mitigation; PTE Inversion
Vulnerability Mds:               Vulnerable: Clear CPU buffers attempted, no microcode; SMT vulnerable
Vulnerability Meltdown:          Mitigation; PTI
Vulnerability Spec store bypass: Vulnerable
Vulnerability Spectre v1:        Mitigation; usercopy/swapgs barriers and __user pointer sanitization
Vulnerability Spectre v2:        Mitigation; Full generic retpoline, STIBP disabled, RSB filling
Vulnerability Srbds:             Not affected
Vulnerability Tsx async abort:   Not affected
Flags:                           fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush dts acpi
mmx fxsr sse sse2 ss ht tm pbe syscall nx pdpe1gb rdtscp lm constant_tsc arch_perfmon pebs bts rep_
                                 good nopl xtopology nonstop_tsc cpuid aperfmperf pni pclmulqdq dtes64 monitor ds_cpl smx
est tm2 ssse3 cx16 xtpr pdcm pcid dca sse4_1 sse4_2 x2apic popcnt tsc_deadline_timer aes xsave avx
                                 lahf_lm epb pti xsaveopt dtherm ida arat pln pts
```

https://unix.stackexchange.com/questions/336017/how-to-detect-if-isolcpus-is-activated/338372

Filesystem tuning for determinism

If your application is accessing disk you can tune the filesystem to be more realtime friendly by disabling journaling.

A journaling file system is a file system that keeps track of changes not yet committed to the file system's main part by recording the goal of such changes in a data structure known as a "journal", which is usually a circular log. In the event of a system crash or power failure, such file systems can be brought back online more quickly with a lower likelihood of becoming corrupted.

Linux: **ext3**, **ext4**, **xfs**, **jfs**, etc … Windows: **NTFS**

Journaling file systems record the time a file was last accessed (***atime***).

Disabling *atime* increases performance and decreases power usage by limiting the number of writes to the filesystem journal.

Edit the /etc/fstab to set the filesystem mount options:

```
/dev/sda1       /       ext4    noatime,nodiratime
```

NOTE: Most of the times the relative timing (access time is only updated if the previous access time is older than the current modify time) is enough and a better option.

```
/dev/sda1       /       ext4    relatime
```

clock source

Multiprocessor systems such as NUMA or SMP have multiple instances of hardware clocks. During boot time the kernel discovers the available clock sources and selects one to use. For the list of the available clock sources in your system, view the `/sys/devices/system/clocksource/clocksource0/available_clocksource` file:

```
# cat /sys/devices/system/clocksource/clocksource0/available_clocksource

tsc hpet acpi_pm
```

In the example above, the **TSC**, **HPET** and **ACPI_PM** clock sources are available.

The **Time Stamp Counter** (**TSC**) is a 64-bit register present on all x86 processors since Pentium.

The **High Precision Event Timer** (**HPET**) is a hardware 64-bit timer counting at a frequency of at least 10 MHz (not in CPU, usually in the southbridge)

The **ACPI Power Management Timer** (or ACPI PMT) is yet another clock device included in almost all ACPI-based motherboards. Its clock signal has a fixed frequency of roughly 3.58 MHz.

The clock source currently in use can be inspected by reading the `/sys/devices/system/clocksource/clocksource0/current_clocksource` file:

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

tsc
```

clock source

Requirements for realtime applications vary on each system.

- Some applications depend on clock resolution, and a clock that delivers reliable nanoseconds readings can be more suitable.

- Applications that read the clock too often can benefit from a clock with a smaller reading cost (the time between a read request and the result).

Generally the tsc is the best option for realtime (the most precise one, since it resides in the CPU)

To optimize the reliability of the TSC clock, you can configure additional parameters when booting the kernel, for example:

- `idle=poll`: Forces the clock to avoid entering the idle state.
- `processor.max_cstate=1`: Prevents the clock from entering deeper C-states (energy saving mode), so it does not become out of sync.

Note however that in both cases there will be an increase in energy consumption, as the system will always run at top speed.

disable CPU power save

Modern processors actively transition to higher power saving states (C-states) from lower states.

Unfortunately, transitioning from a high power saving state back to a running state can consume more time than is optimal for a real-time application.

To prevent these transitions, an application can use the Power Management Quality of Service (PM QoS) interface.

When an application holds the `/dev/cpu_dma_latency` file open, the PM QoS interface prevents the processor from entering deep sleep states, which cause unexpected latencies when they are being exited. When the file is closed, the system returns to a power-saving state.

1. Open the `/dev/cpu_dma_latency` file. Keep the file descriptor open for the duration of the low-latency operation.
2. Write a 32-bit number to it. This number represents a maximum response time in microseconds. For the fastest possible response time, use `0`.

disable CPU power save

CODING EXAMPLE

```
static int pm_qos_fd = -1;

void start_low_latency(void)
{
   s32_t target = 0;

   if (pm_qos_fd >= 0)
       return;
   pm_qos_fd = open("/dev/cpu_dma_latency", O_RDWR);
   if (pm_qos_fd < 0) {
      fprintf(stderr, "Failed to open PM QOS file: %s",
              strerror(errno));
      exit(errno);
   }
   write(pm_qos_fd, &target, sizeof(target));
}

void stop_low_latency(void)
{
   if (pm_qos_fd >= 0)
      close(pm_qos_fd);
}
```

Avoid memory swap

Swapping pages out to disk can introduce a **dramatic latency in any environment**.

To ensure low latency, the best strategy is to have enough memory in your systems so that swapping is not necessary.

- Always size the physical RAM as appropriate for your application and system.

- Use `vmstat` to monitor memory usage and watch the `si` (swap in) and `so` (swap out) fields. It is optimal that they remain zero as much as possible.

And what if we run Out of Memory?

There is a switch that controls OOM behavior in `/proc/sys/vm/panic_on_oom`.

- When set to `1` the kernel will panic on OOM.
- When set to `0` which instructs the kernel to call a function named `oom_killer` on an OOM. Usually, `oom_killer` can kill rogue processes and the system will survive.

Avoid memory swap: oom_killer

It is also possible to prioritize which processes get killed by adjusting the `oom_killer` score. In `/proc/PID/` there are two files named `oom_adj` and `oom_score`. Valid scores for `oom_adj` are in the range -16 to +15. This value is used to calculate the 'badness' of the process using an algorithm that also takes into account how long the process has been running, among other factors. To see the current `oom_killer` score, view the `oom_score` for the process. `oom_killer` will kill processes with the highest scores first.
This example adjusts the `oom_score` of a process with a PID of 12465 to make it less likely that `oom_killer` will kill it.

```
# cat /proc/12465/oom_score
79872
# echo -5 > /proc/12465/oom_adj
# cat /proc/12465/oom_score
78
```

There is also a special value of -17, which disables `oom_killer` for that process. In the example below, `oom_score` returns a value of `0`, indicating that this process would not be killed.

```
# cat /proc/12465/oom_score
78
# echo -17 > /proc/12465/oom_adj
# cat /proc/12465/oom_score
0
```

NETWORK DETERMINISM: avoid TCP

Avoid Transmission Control Protocol (TCP)

TCP can have a large effect on latency. TCP adds latency in order to obtain efficiency, control congestion, and to ensure reliable delivery.

- Do you need ordered delivery?
- Do you need to guard against packet loss?
  Transmitting packets more than once can cause delays.

TCP Nagle algorithm:
If you must use TCP, consider disabling the Nagle buffering algorithm by using `TCP_NODELAY` on your socket. The Nagle algorithm collects small outgoing packets to send all at once, and can have a detrimental effect on latency.

NETWORK DETERMINISM: tuning

Interrupt Coalescing

To reduce the amount of interrupts, packets can be collected and a single interrupt can be generated for a collection of packets. However for realtime systems, requiring a rapid network response, reducing or disabling coalesce is a good practice.

Use the `-C` (`--coalesce`) option with the `ethtool` command to enable.

Congestion

Often, I/O switches can be subject to back-pressure, where network data builds up as a result of full buffers.

Use the `-A` (`--pause`) option with the `ethtool` command to change pause parameters and avoid network congestion.

Infiniband (IB)

Infiniband is a type of communications architecture often used to increase bandwidth and provide quality of service and failover. It can also be used to improve latency through Remote Direct Memory Access (RDMA) capabilities.

NETWORK DETERMINISM: tuning

Turn timestamps off to reduce performance spikes related to timestamp generation. The `sysctl` command controls the values of TCP related entries, setting the timestamps kernel parameter found at `/proc/sys/net/ipv4/tcp_timestamps`.

Turn timestamps off with the following command:

```
# sysctl -w net.ipv4.tcp_timestamps=0
net.ipv4.tcp_timestamps = 0
```

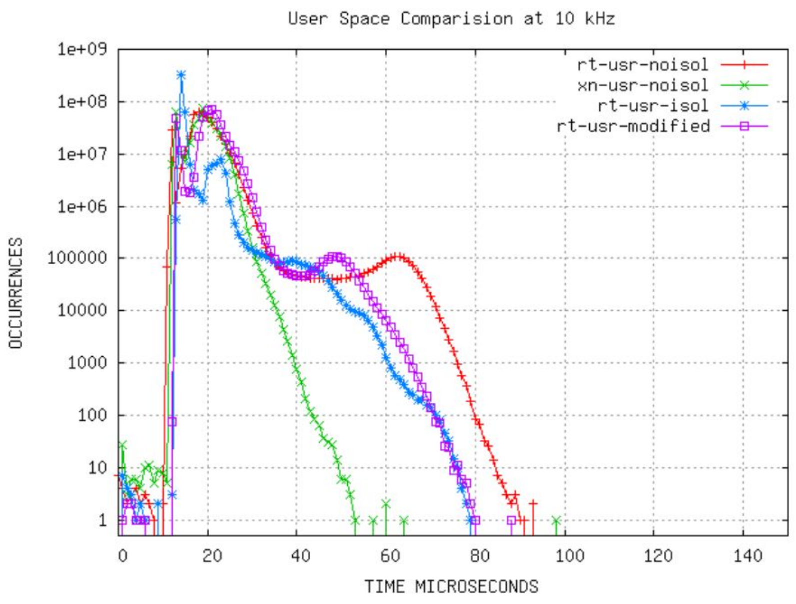
Disable spectre mitigations

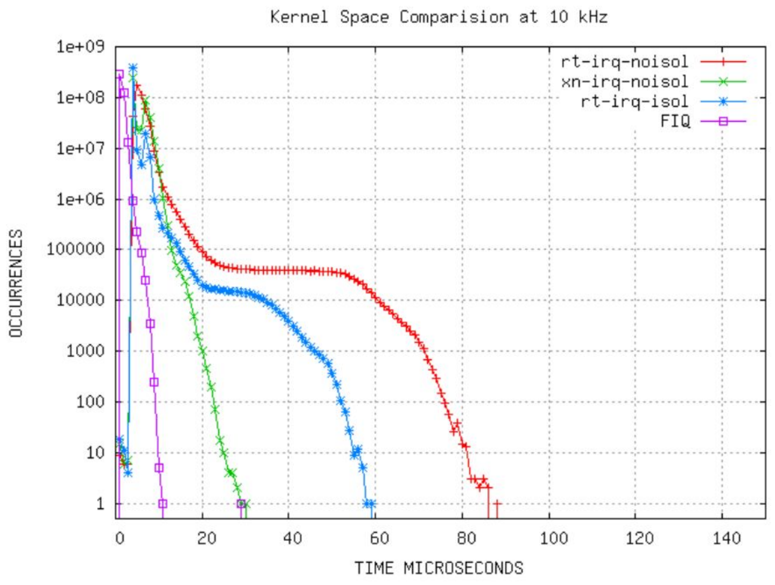
Turn off CPU exploit mitigations

Warning: Do not apply this setting without considering the vulnerabilities it opens up.

Turning off CPU exploit mitigations may improve performance. Use below **kernel parameter** to disable them all:

```
mitigations=off
```


User Space Comparision at 10 kHz
OCCURRENCES
TIME MICROSECONDS
rt-usr-noisol
xn-usr-noisol
rt-usr-isol
rt-usr-modified
1e+09
1e+08
1e+07
1e+06
100000
10000
1000
100
10
1
0
20
40
60
80
100
120
140


Kernel Space Comparision at 10 kHz
OCCURRENCES
TIME MICROSECONDS
rt-irq-noisol
xn-irq-noisol
rt-irq-isol
FIQ
1e+09
1e+08
1e+07
1e+06
100000
10000
1000
100
10
1
0
20
40
60
80
100
120
140

Further readings:

Archlinux improving performance

https://wiki.archlinux.org/title/Realtime_kernel_patchset

https://wiki.archlinux.org/title/improving_performance

https://wiki.archlinux.org/title/CPU_frequency_scaling

RedHat Realtime linux

https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux_for_real_time/8/html/optimizing_rhel_8_for_real_time_for_low_latency_operation/index