SUSE

15 OCTOBER 2022

Realtime Linux

20 години

OpenFest 2022

Sofia, Bulgaria

Bogdan Lezhepekov <bogdan.lezhepekov@suse.com>

Radoslav Kolev <radoslav.kolev@suse.com>

SUSE – COMPANY SNAPSHOT

SUSE solutions are powering thousands of enterprise customers' mission-critical workloads, including electronic banking systems and enterprise applications, autonomous vehicles, satellite operation centers, and life-saving medical devices.

Acquired Rancher in 2020

FAST FACTS

- 13 out of 15 largest FinServ firms
- 14 out of 15 largest aerospace firms
- 10 out of 10 largest automotive firms
- 13 out of 15 largest pharma firms
- 5 out of 5 largest technology firms
- 19% YoY growth for SUSE
- 88% YoY growth for SUSE Rancher
- 50% ACV growth in cloud
- Member of CNCF board and TOC
- An independent leader in open source – SUSE SA (FRA)

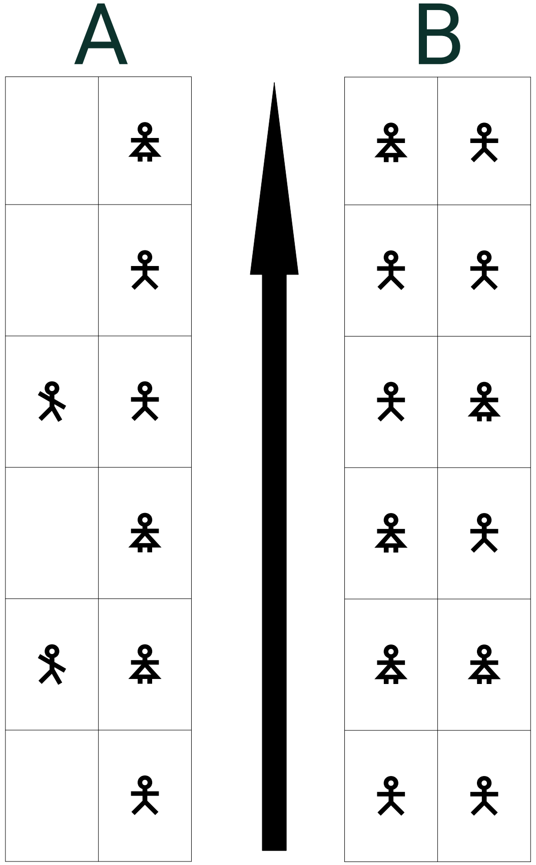
INDUSTRY INITIATIVES & ASSOCIATIONS

Cloud Native Computing Foundation, Open Container Initiative, The Linux Foundation, LF Networking, ELISA (Enabling Linux in Safety Applications), O-RAN Alliance, 5GAA Automotive Association, gaia-x, OpenPOWER, Open Mainframe Project, openHPC

PARTNERSHIPS

aws, Microsoft, Google, Alibaba Cloud, arm, Hewlett Packard Enterprise, Oracle, Huawei, vmware, DXC.technology, Dell, SAP, intel, IBM, Fujitsu, AMD, Cisco, HCL, NVIDIA, Lenovo

The ***CORRECT*** way to use an escalator



Optimizing for throughput or latency

Throughput - the tasks performed by a computer over a period of time

Latency - the delay between cause and effect, reaction time

A real-time system is actually slower (lower throughput) than a non-RT one!

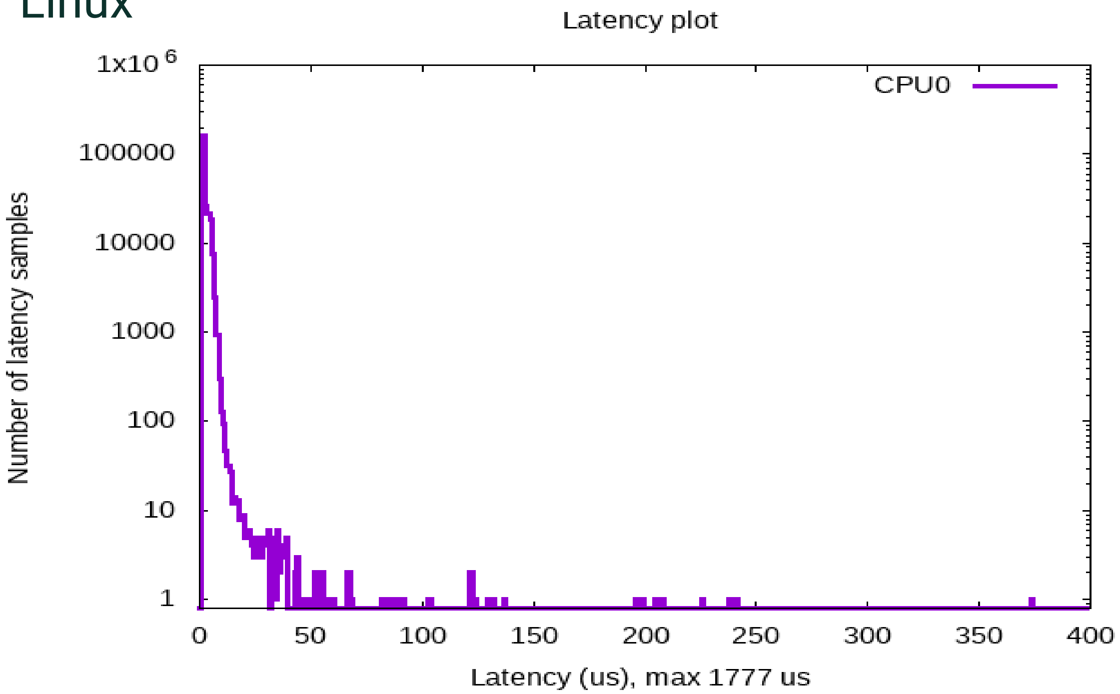
- Context switching takes time
- Bigger chance of CPU cache misses
- Real-time systems require some 'slack'

But it puts an upper bound on latency and minimizes jitter.

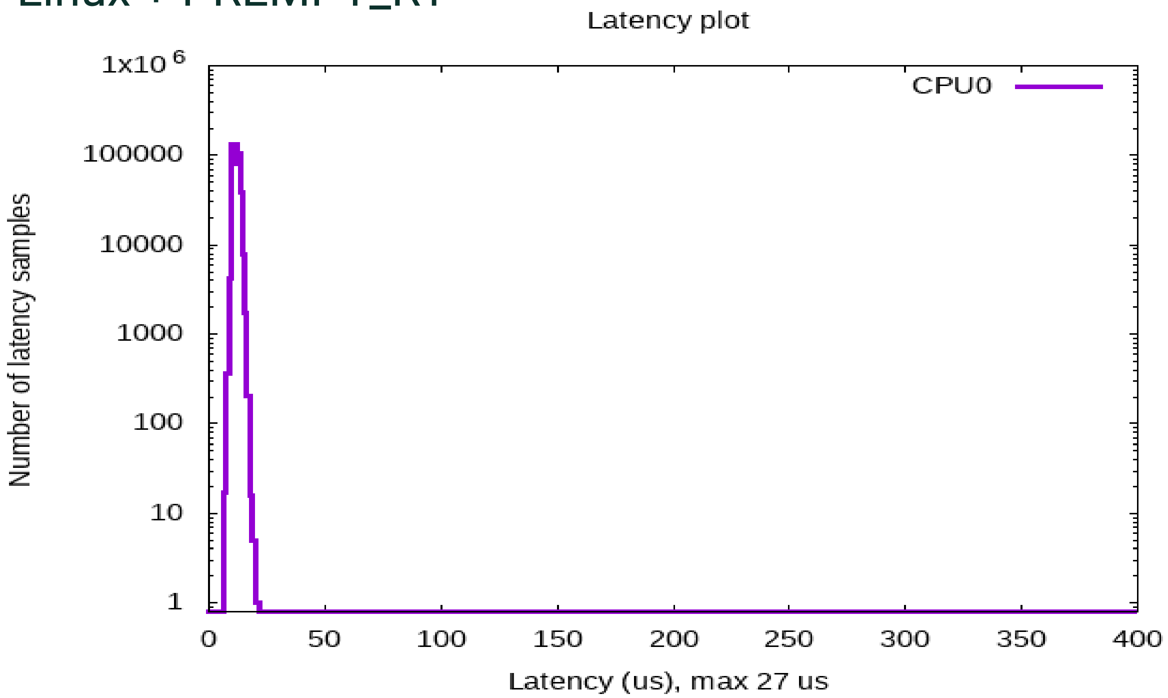
Real-time operating systems

A system is said to be real-time if the total correctness of an operation depends not only upon its logical correctness, but also upon the time in which it is performed – Wikipedia

Linux



Linux + PREMPT_RT



Real-time systems – soft & hard

Hard real-time systems:

missing a deadline is a total system failure

- Car ECUs, pacemakers, industrial robot control, avionics
- Examples: no OS (bare metal), QNX, classic AUTOSAR, FreeRTOS, Integrity, Zephyr, etc.

Soft real-time systems:

missing a deadline degrades the quality of service, but doesn't lead to critical failure

- Audio/video transmission, computer games
- Examples: RT Linux, Windows 10 IoT, etc.

Linux OS Basic Concepts

Kernel vs Userspace

Userspace isolates ‘user’ processes from the core sub-systems of the operating system.

Kernel

- CPU is in priviledged mode.
- Provides abstraction for security, hardware, and internal data structures.
- A kernel process has direct and unrestricted access to system resources.
- All processes share a single virtual address space.
- A kernel process can access any memory block.

Userspace

- CPU is in unpriviledged mode.
- Userspace processes communicate with the Kernel via API called system call.
- All processes get separate virtual address space.
- A user process can access memory allocated to it, otherwise sigmentation fault.

Interrupts

An interrupt is a sort of signal that the hardware can send when it requires CPU time. Linux has to deal with two types of interrupts: hardware interrupts and software interrupts.

- Normally, handled in two parts: **top** half and **bottom** half.
- Top half executes critical code as soon as the hardware interrupt is received.
- Bottom half is scheduled by software interrupt and does the most expensive calculations.

The concept of preemption

Preemption is a property of a multi-tasking operating system, in which the CPU can be interrupted in the middle of executing code and assigned other tasks. It is a way of implementing multitasking.

- The decision to preempt a task is taken by the scheduler.
- Critical for an RTOS to be able to ensure a higher priority task overtakes a lower priority task.
- For RTOS any task should be preemptible, both in userspace and kernel.

Preemption models in non-RT Linux

The Linux kernel implements several preemption models. The desired model is selected at build time of the kernel.

- **No Forced Preemption (CONFIG_PREEMPT_NONE)** (server): The traditional Linux preemption model, geared towards throughput. System call returns and interrupts are the only preemption points in the kernel.
- **Voluntary Kernel Preemption (CONFIG_PREEMPT_VOLUNTARY)** (Desktop): This option reduces the latency of the kernel by adding more “explicit preemption points” to the kernel code [. . .] at the cost of slightly lower throughput. In addition to explicit preemption points, system call returns and interrupt returns are implicit preemption points.

Kernel Locks

Locks are synchronisation primitives that arbitrate concurrent accesses to a resource.

Spinning Lock

- Spinlocks will busy-wait until the lock is freed.
- Spinlocks will disable preemption when taken.
- The spinlocks are most easily added to places that are completely independent of other code (for example, internal driver data structures that nobody else ever touches).
- Types of spinning locks:
  - spinlock_t
  - rwlock_t
  - raw_spinlock_t

Sleeping Locks

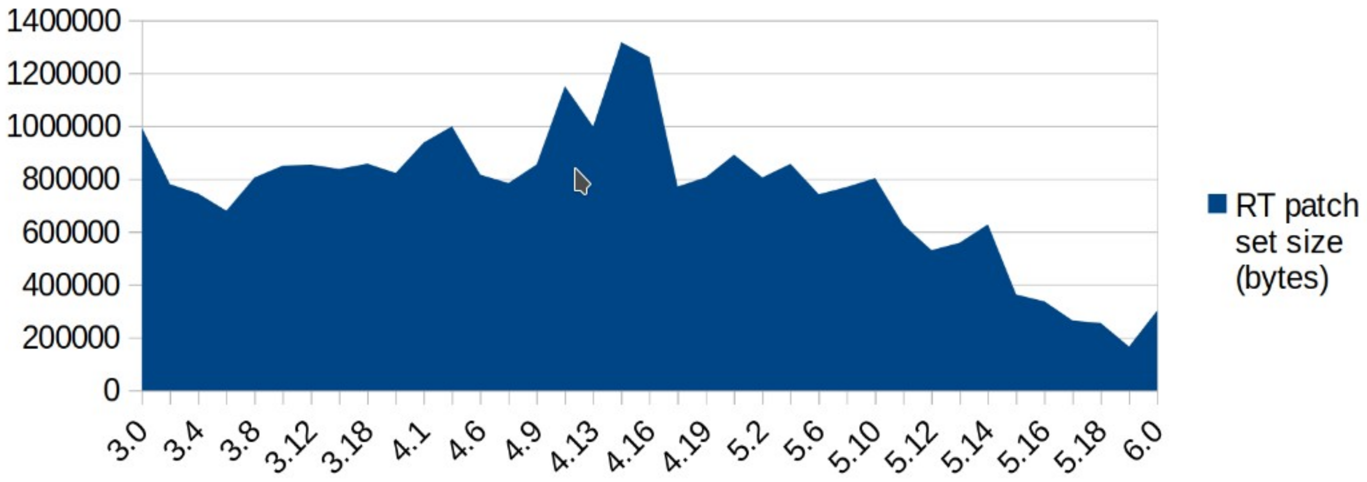
- Sleeping locks will sleep and schedule while waiting.
- Types of sleeping locks :
  - Mutex
  - rt_mutex
  - Semaphore
  - rw_semaphore

PREEMPT_RT patchset

The PREEMPT_RT patchset

But why?

- First made available for Linux v2.6.11
- Slowly being merged into mainline
- One major hurdle remains - printk



graphic by Bootlin

What does PREEMPT_RT bring to the Linux kernel?

- Fully Preemptible Kernel (CONFIG_PREEMPT_RT) (RT): All kernel code is preemptible except for a few selected critical sections. Additionally, large preemption disabled sections are substituted by separate locking constructs. This preemption model has to be selected in order to obtain real-time behavior.

- With PREEMPT_RT, spinlock_t and rwlock_t will become sleeping locks

- Almost all interrupt handlers are threaded

Configuring the realtime system

CPU affinity and CPU isolation

Linux provides means to 'pin' a given process to CPU – the CPU affinity mechanism. You can also set constraints on CPU cores that allow you to allocate cores for your tasks.

Practical tips:

- Make sure that a process won't be migrated to another core.
- Dedicate cores for specific tasks.
- Optimize the data-path if a process deals with data handled by a specific CPU core.
- Ease the job of the scheduler's CPU load-balancer, whose complexity grows non-linearly with the number of CPUs.
- Kernel can also schedule other processes on the CPU cores you have chosen, therefore consider CPU isolation (isolcpus) to allocate fully pre-allocate the resource.
- Better to run RT processes with non-RT on the same cores.

IRQ affinity

- By default Linux handles IRQs on a specific core – CPU 0.
- Consider balancing IRQ handlers between the cores.
- Pinning and isolation of CPU is also possible in case of IRQ.
- The irqbalance tool monitors and distributes the irq affinty to spread the load across CPUs.

Schedulling classes

Non-RT classes

- SCHED_OTHER – default class, time sharing algoritm.
- SCHED_BATCH – similar to SCHED_OTHER, designed for CPU-intensive loads that affect the wakeup time.
- SCHED_IDLE - low priority class, tasks run only when there is nothing to do.

RT classes

RT tasks can be assigned a priority between 0 and 98 (by chrt command). Priority 99 is reserved for housekeeping tasks.

A scheduling algoritm matters only for tasks with equal priorities.

- SCHED_FIFO – first in, first out algoritm.
- SCHED_RR - similar to SCHED_FIFO but with a time-sharing round-robin.
- SCHED_DEADLINE - for tasks doing recurrent jobs, extra attributes are attached to a task
  - A computation time, which represents the time the tasks needs to complete a job
  - A deadline, which is the maximum allowable time to compute the job
  - A period, during which only one job can occur.

Caveats for real-time code

- Kernel space:
  - Avoid using raw_spinlock_t.
  - Avoid forcing non-threaded interrupts if possible
- User space:
  - Proper initialization is crucial and it doesn't have to be RT.
  - During initialization make sure that you:
    - Pre-allocate, lock and pre-fault memory
    - Create and configure threads
    - Configure the scheduling parameters
    - Configure the CPU affinity and isolation
- Hardware:
  - NMIs and low level firmware
  - Hyperthreading
  - Idle states & CPU frequency scaling
  - NUMA

Resources & tools

https://bootlin.com/training/preempt-rt/

https://wiki.linuxfoundation.org/realtime/documentation/howto/tools/rt-tests

https://docs.kernel.org/trace/hwlat_detector.html

https://github.com/fenrus75/powertop

https://www.kernel.org/doc/Documentation/trace/ftrace.txt

SUSE

Thank you

For more information, contact SUSE at:

+1 800 796 3700 (U.S./Canada)

Frankenstrasse 146

90461 Nürnberg

www.suse.com