

A Quantitative Analysis of the Real-time Capabilities of Linux with PREEMPT_RT

This paper investigates the real-time capabilities of Linux, both for the vanilla kernel and for kernels with a real-time patch (PREEMPT_RT) applied. Scheduling latency is our parameter of interest and it is measured across various load conditions for both the kernels. Our experiments suggest that a real-time patch can improve the deterministic behaviour of Linux, thereby offering an alternative to proprietary, hard real-time operating systems.

The distinguishing characteristic of real-time systems is their ability to respond to events in a timely fashion, i.e., they are deadline oriented. Further, depending on how important deadlines are to the system, real-time operating systems are classified into hard and soft real-time. In the case of the latter, these systems provide a ‘best-effort’ service; they aim to complete tasks within the deadline, but deadline misses result in performance degradation and not a system failure. For instance, an ATM machine may not be able to meet a transaction deadline and may timeout, which is acceptable. Hard real-time systems, on the other hand, have imperative deadlines that they cannot afford to miss without causing serious damage to the system. An example would be an airbag inflation system in an automobile. The airbag must inflate within a few milliseconds of an accident; else it could be fatal to the driver.

Some of the popular real-time operating systems currently being used are VxWorks, QNX, eCOS and FreeRTOS. Even though Linux is designed as a GPOS (general-purpose operating system), the Linux kernel also has the features to customise the required options for use in embedded systems. The vanilla Linux kernel supports soft real-time capabilities but not hard real-time features. However, since Linux is the most prevalent operating system used by developers of embedded systems, this paper aims to inspect and analyse the real-time capabilities of Linux and look at ways to further augment these capabilities. Early Linux (1.x) had no facility for kernel pre-emption.

Once a task started executing in kernel mode, it could not be pre-empted and had to voluntarily give up kernel access. Linux 2.x introduced SMP and the Big Kernel Lock. Linux 2.6 was the first kernel to introduce pre-emption. However, spinlock sections were not preemptible. 2.6 comes with configuration options: CONFIG_PREEMPT_VOLUNTARY and CONFIG_PREEMPT. Despite using the CONFIG_PREEMPT option, critical sections protected by spinlocks are not pre-emptible. As a result, worst case latencies are still significantly high and not suitable for the strict deadlines of hard real-time systems. Therefore, to use Linux as a hard, real-time OS, modifications need to be made to the kernel.

There are two different approaches to providing a real-time performance with Linux:

1. Improving the Linux kernel pre-empt ability.
2. Adding a new software layer beneath the Linux kernel with full control of interrupts and processor key features.



A few ways to achieve this are:

- PREEMPT_RT
- RTLinux
- Xenomai
- RTAI

PREEMPT_RT: Originally referred to as the sleeping spinlocks patch, it was developed by Ingo Molnar and Thomas Gleixner for the 2.6.22 kernel. It provides full kernel pre-emption by replacing spinlocks with mutexes.

RTLinux: This uses a micro-kernel approach that runs the entire Linux kernel as a fully pre-emptive process. RTLinux was developed in FSM Labs, which was later acquired by Wind River. Therefore, currently there are two versions of RTLinux — a proprietary version offered by Wind River and an open source version that is available under GPL.

Xenomai: This uses a dual-kernel method, where non-real-time tasks are run on the Linux kernel and real-time tasks are run on the Xenomai kernel. The two kernels communicate via a virtual interrupt controller. Xenomai was meant to provide real-time support to user space applications on a Linux based platform.

RTAI: This is a community developed extension to the Linux kernel, which lets users write applications that can meet strict real-time deadlines. It consists of a patch that introduces a hardware abstraction layer. It also exposes a set of services, which makes programming on RTAI easier.

An experimental setup

Some of the features of PREEMPT_RT are sleeping spinlocks, threaded interrupts, high resolution timers and priority inheritance. We chose the PREEMPT_RT patch and applied it to the vanilla Linux kernel for our experiments. The only necessary configuration for a real-time Linux kernel is selecting 'Fully Pre-emptible Kernel' in the *menu config* options.

The test environment: All the tests were performed on two virtual machines (vanilla and real-time patched) created on VirtualBox. The tests were conducted on two different

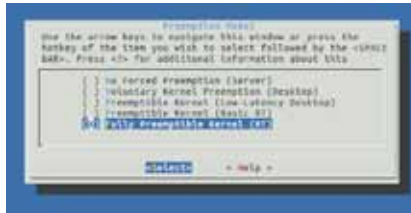


Figure 1: Kernel configuration options using menu config

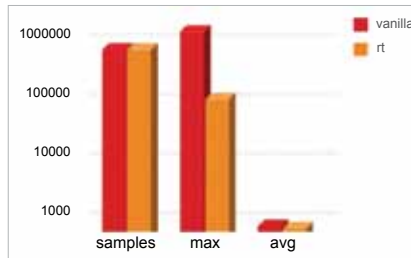


Figure 2: Latency results under no-load conditions

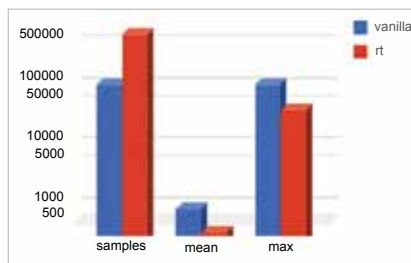


Figure 3: Latency results with CPU-intensive load in user mode

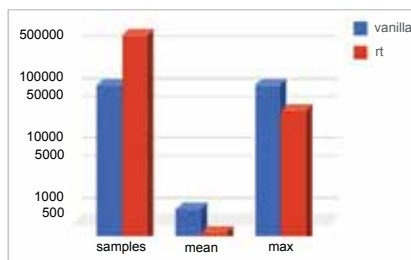


Figure 4: Latency results with CPU-intensive load in kernel mode

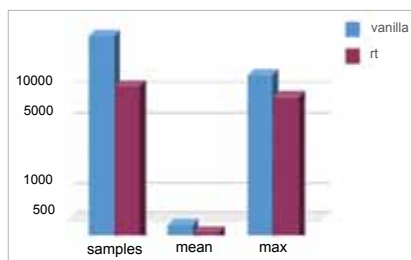


Figure 5: Latency results with memory-intensive load

kernel versions and patches too. The details are as mentioned below:

Linux distribution: Ubuntu 16

Kernel version: 4.9.40 and 4.11.12

PREEMPT_RT patch version: RT30 and RT14

Cores: 1- 4, depending on the tests

RAM: 2GB

Scheduling latency: Scheduling latency is the time taken by the kernel to schedule a task. It is measured as the difference between the time a task wakes up and the time it is run.

The task execution time depends on scheduling latency. An RTOS needs to be deterministic and predictable and, therefore, scheduling latency is of paramount importance. Hence, this is chosen as our parameter of interest.

Latency test – Cyclicttest: Cyclicttest is a part of RT-tests and is used to measure scheduling latency. It starts running a non-real-time master thread, which starts a defined number of threads with a real-time priority. These threads are woken up periodically by an expiring timer. The difference between the programmed and effective wake-up times is calculated. In this manner, minimum, average and maximum latency values (in microseconds) are calculated and printed.

Loads:

1. CPU-intensive applications
2. Memory-intensive applications
3. Hackbench
4. KCompile

Implementation and results

Latency under no-load conditions:

Cyclicttest was run on both kernels with no external load (Figure 2).

Latency with a CPU-intensive load: A simple infinite *for* loop was used as a user mode CPU-intensive load, and the same configuration with an additional system call to get the process ID was used as the kernel mode load. Cyclicttest was run under these conditions (Figures 3 and 4).

Latency with a memory-intensive

load: The load, in this case, is created using the *dd* command to create a 500MB file of zeros.

Latency with Hackbench as load: Hackbench is a stress test on the Linux scheduler. It creates a certain number

of pairs of threads, which communicate through pipes/sockets, and estimates how long it takes for each pair to send data back and forth.

Hackbench was used with the following options:

```
$ hackbench -p -g 1 -f 2 -l 10000000 -s 1
```

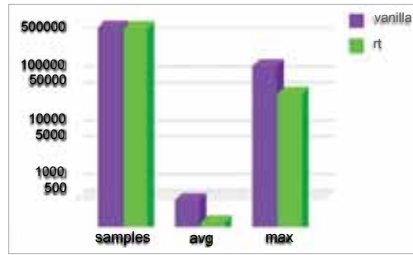


Figure 6: Latency results with Hackbench as load

How long Hackbench runs for is not known in advance; therefore, it is run with an arbitrarily large value of loops and forcefully terminated when Cyclictest finishes.


Latency with *kcompile* as load: To use *kcompile* as a load, we chose Rteval, which is written in Python to use two loads: Hackbench and a Linux kernel compile in periodic loops. While the loads are running, the Cyclictest program is run to measure system performance under these circumstances. After a specified duration, the loads are stopped and Cyclictest outputs are analysed by Rteval. We have run Rteval with just *kcompile* as load.

Future work

The above experiments and results have proved that the PREEMPT_RT patch with the GPOS Linux kernel can be used for hard real-time tasks. Linux is open source software published with the General Public License. Therefore, using Linux for hard real-time systems is cost-effective compared to using proprietary RTOS like VxWorks, etc. With this, we can eliminate the licence as well as royalty costs. So, the

end product would be cheaper in the competitive market.

Scheduling latency was the only parameter evaluated in this project. However, there are several other parameters like interrupt latency, I/O latency and network latency which also contribute to the deterministic behaviour of an RTOS. All of these can be measured and used to evaluate the performance of

the vanilla kernel versus the real-time kernel. **END** 

References

- [1] Stankovic, John A., and Raj Rajkumar: 'Real-time operating systems' Real-Time Systems 28.2-3 (2004): 237-253
- [2] Real-time Linux wiki. <https://rt.wiki.kernel.org/index.php/Cyclictest>
- [3] Felipe Cerqueira, Bjorn B. Brandenburg 'A Comparison of Scheduling Latency in Linux, PREEMPT RT, and LITMUSRT', Max Planck Institute for Software Systems (MPI-SWS)
- [4] Oliveira, Daniel Bristot, and Romulo Silva Oliveira: 'Timing analysis of the PREEMPT RT Linux kernel' Software: Practice and Experience 46.6 (2016): 789-819.
- [5] High resolution timers: https://rt.wiki.kernel.org/index.php/High_resolution_timers
- [6] Hackbench man page: <http://manpages.ubuntu.com/manpages/zesty/man8/hackbench.8.html>
- [7] Examining load average: <http://www.linuxjournal.com/article/9001>

By: K. Deepika Raj, Prashanthi S.K. and Dr B. Thangaraju

The authors are all associated with the International Institute of Information Technology, Bengaluru.