Master-Studiengang Robotics Engineering
FH Technikum Wien, Höchstädtplatz 6, A-1200 Wien

Virtualization of a Real-Time Operating System for Robot Control with a Focus on Real-Time Compliance

Student: Pamuk, Halil Ibrahim, BSc., **PK:** 51842568
Supervisor: Rauh, Sebastian, MSc. BEng.

***Abstract:** This thesis investigates the virtualization of a real-time operating system, which is built with Yocto, employs hard real-time through Xenomai 3, and is emulated using QEMU/KVM. The primary objective was to bridge the latency gap between virtualized and bare metal versions to ensure deterministic and reliable behavior, which is crucial for real-time robotics. Initial latency measurements showed a high gap between bare metal and virtualized setups. Extensive tuning across BIOS, kernel, host OS, QEMU/KVM, and guest OS reduced the worst-case latency from 707.622 µs to 17.134 µs, close to the bare metal performance of 10.709 µs. The improvement was validated using a robotic application, comparing tuned virtualization with untuned and hardware versions.*

***Keywords:** Virtualization, Real-Time Systems, Latency Reduction, Robot Control*

1. Introduction

In today's industrial production and automation, robots must react to their environment and perform time-critical tasks within strict time constraints. Delays or errors can have catastrophic consequences in some cases. Virtualizing real-time operating systems (RTOS) for robotic control offers many advantages over traditional hardware-based approaches, especially when it comes to scaling [1], flexibility, and costs [2]. However, virtualization also introduces overhead and latency [3] [4]. In this master's thesis, the goal was the virtualization of a real-time operating system to control a robot, with a focus on compliance with real-time determinism. Specifically, the aim was to bring the latency of the virtualized Salamander 4 closer to that of the bare metal version, as initial latency measurements revealed a significant latency gap between the bare metal and virtualized setup.

2. Methodology

In order to address this gap, an extensive tuning process was carried out to achieve real-time performance and determinism. These tunings were added sequentially and tested together. This involves configurations spanning the BIOS, kernel, host operating system (OS), and QEMU/KVM virtualization layer, as demonstrated in [5]. The guest OS is already based on Xenomai and therefore, no additional modifications were necessary. Ubuntu 22.04.4 LTS served as the host operating system, while Salamander 4, built with Yocto and based on Linux 5.15.94, was the guest OS. Xenomai provided real-time capabilities for the guest and its `latency` tool was used to measure the scheduling latency of the real-time OS. QEMU with KVM was employed for virtualization. Trace-cmd and Kernelshark were used for kernel tracing and visualization of trace data. The testing robot was connected to the virtual CPU via a VARAN bus interface.

3. Implementation

The initial latency values were first measured with the `latency` program of the Xenomai tool suite. The tests were conducted with a sampling period of 100 µs, using a periodic user-mode task, and were assigned a priority of 99. If any sample's latency value was greater than 100 µs, this was considered an overrun. There was a significant initial gap in latency statistics between the virtualized Salamander 4 and the Salamander 4 on bare metal. To address this gap, an extensive tuning process was carried out to achieve real-time performance and determinism. These modifications were added sequentially and tested together. Previous tunings were not reverted when moving to the next tunings. First, the BIOS was configured, followed by the kernel. The BIOS settings were not reverted when moving to the kernel configurations. Next, the host was configured, but the BIOS and kernel settings remained unchanged. The guest configurations were not changed since Xenomai already provides real-time capabilities. Finally, QEMU/KVM settings were applied. On top of that, a robotic application further confirmed the improvements. The difference between the command time and the time the signal reaches the PWM was measured 1,000 times for the untuned, tuned, and hardware versions of Salamander 4.

4. Results

This section summarizes the results after the real-time performance tunings, both in terms of the `latency` program and the robotic application. Figure 1 displays the change in latency after each real-time performance tuning.

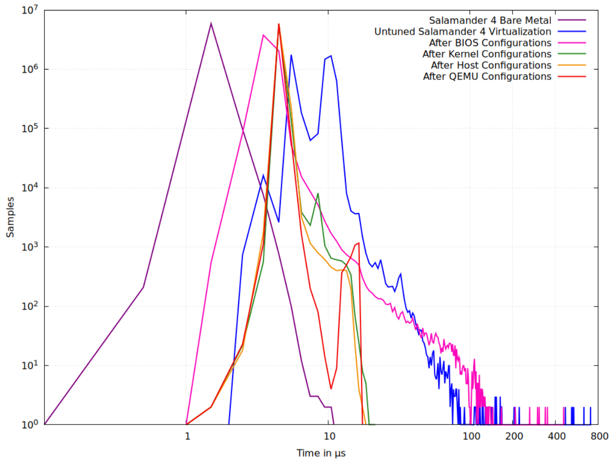

Figure 1: Comparison of Latency Distribution of Salamander 4 under different Configurations

Table 1 gives an overview of the most important metrics of the measurements after each tuning.

Table 1: Comparison of Latency Statistics in Salamander 4 under different Configurations

Tuning	Latency (µs)			Overruns
	Min	Avg	Max	
Bare Metal	0.613	1.380	10.709	0
Untuned Virtualization	2.536	8.940	707.622	43
After BIOS Configurations	0.969	3.948	457.545	94
After Kernel Configurations	2.545	4.811	21.694	0
After Host Configurations	2.591	4.834	18.441	0
After QEMU Configurations	2.614	4.779	17.134	0

The improved latency was also tested with a robotic application. The difference between the command issuance time and the time the signal reaches the PWM was measured 1,000 times for the bare metal, untuned, and tuned versions of Salamander 4. The results are visualized in Table 2.

Table 2: Comparison of Robotic Application Latency of Salamander 4 Configurations

Tuning	Latency (ms)			Std Dev (ms)
	Min	Avg	Max	
Bare Metal	1.211	1.347	1.49	0.082
Untuned Virtualization	3.1	24.603	129.46	13.876
Tuned Virtualization	1.219	2.62	3.988	0.812

5. Discussion

The results show that the real-time performance tunings applied to the virtualized Salamander 4 OS reduced latency and improved determinism. The maximum latency decreased from 707.622 µs to 17.134 µs after tuning BIOS, kernel, host, guest, and QEMU/KVM configurations. Especially the BIOS and kernel configurations played a crucial role in reducing the maximum latency, with a major fall to 21.694 µs. This also eliminated overruns going forward. Host configurations, including the PREEMPT-RT patch, CPU and interrupt affinity, and real-time prioritization of QEMU further reduced latency a little bit down to 18.441 µs. The guest OS already had real-time capabilities through Xenomai, hence the focus was primarily on optimizing the host and virtualization layer to achieve the desired real-time performance. Finally, QEMU/KVM configurations, such as tuning the LAPIC timer advance and using hugepages, brought the latency down to the final worst latency value of 17.134 µs. This value is very close to the bare metal value of 10.709 µs. The goal was set to achieving latency values below 50 microseconds in the duration of the measurement. This goal was achieved. A robotic application in a practical scenario validated the improvement of the tuned virtualization compared to the unmodified version. The difference between command time and signal reaching the PWM was measured 1,000 times for untuned, tuned, and hardware versions of Salamander 4. The worst latency dropped from 129ms to 3.988ms in the developed application. Compared to the hardware version's worst latency of 1.49ms, the tuned virtualization came very close to the determinism and reliability of the hardware.

6. Summary and Outlook

This master's thesis aimed to virtualize a real-time operating system for robot control, focusing on real-time determinism. The goal was to reduce latency in virtualized Salamander 4 to match its bare metal version. Initial tests showed a significant latency gap between virtualized and bare metal versions. Extensive tuning was performed sequentially: BIOS, kernel, host, and QEMU/KVM configurations. This process reduced worst latency from 707.62 µs to 17.134 µs, close to the bare metal value of 10.709 µs, achieving the goal of sub-50 µs latency. A robotic application confirmed these improvements. Measuring the time between command and PWM signal 1,000 times showed worst latency dropping from 129ms to 3.988ms, approaching the hardware version's 1.49ms. Future work could include further optimizations, testing other hypervisors, and evaluating performance under various workloads and stressed situations. Future work could include additional configurations and optimizations of the virtualization layer, investigating other hypervisors and virtualization technologies, and extensive testing under various workloads and stressed situations.

References

[1] Maryam Abbasi et al. "Exploring OpenStack for Scalable and Cost-Effective Virtualization in Education".

[2] Gilberto Taccari et al. "Embedded Real-Time Virtualization: State of the Art and Research Challenges".

[3] Daniel Casini et al. "Latency Analysis of I/O Virtualization Techniques in Hypervisor-Based Real-Time Systems".

[4] Binbin Zhang et al. "Evaluating and Optimizing I/O Virtualization in Kernel-based Virtual Machine (KVM)".

[5] Intel. "Real-Time Performance Tuning Best Practice Guidelines for KVM-Based Virtual Machines".