Lightweight Instrumentation for Accurate Performance Monitoring in RTOSes

Bruno Forlin[a], Kuan-Hsun Chen[a], Nikolaos Alachiotis[a], Luca Cassano[b], Marco Ottavi[a,c]
[a]University of Twente, The Netherlands, [b]Politecnico di Milano, Italy, [c]University of Rome Tor Vergata, Italy,
[a]{b.endresforlin, k.h.chen, n.alachiotis, m.ottavi}@utwente.nl, [b]luca.cassano@polimi.it

***Abstract*—Evaluating performance metrics in embedded systems poses challenges, particularly due to the limited set of tools available for monitoring performance counters. In addition, performance evaluation frameworks for Real-Time Operating Systems (RTOSes) often lack the sophistication and capabilities available in general-purpose operating systems like Linux, which benefit from utilities such as `perf_event`. To bridge this gap, this paper presents an accurate and low-overhead instrumentation utility tailored for RTOSes. Our approach utilizes performance monitoring counters to observe individual user applications within the RTOS environment. Importantly, it enables comprehensive application monitoring by strategically placing probes at points of inherent system interference, thereby minimizing additional overhead. A pre-calibration of these probes allows for fine-grained measurements within user applications. This results in the elimination of 100% of the overheads for most counters in our test configuration, impacting the context switch by only three additional instructions per monitored counter.**

***Index Terms*—Hardware Perfomance Counters, Monitoring, RISC-V, RTOS, Trace**

I. Introduction

Embedded processors are crucial in diverse applications, necessitating thorough validation in performance, energy consumption, real-time responsiveness, and security. Performance Monitoring Counters (PMCs) play a crucial role in debugging and performance analysis in real-time systems. While PMCs are widely used in desktop and server environments through standard interfaces like PAPI, their adaptation in embedded contexts, through interfaces like ePAPI and PRL, faces limitations such as lack of task separation and security risks due to direct PMC access. PMCs are specialized registers in modern processors that monitor low-level system metrics, such as Core-State Dependent (CSD) metrics like cache hits and misses, and Process-Specific (PS) metrics like retired instructions.

These counters, which can be either fixed-function or programmable, are key in debugging and performance analysis. PMCs operate in either free-running or interrupt-based modes, continuously incrementing events until cleared. In Linux, supervisor-level permissions are required for PMC access, managed via the `perf_event` interface. However, PMC measurements can be affected by external sources like OS activity, hardware interrupts, and system calls, leading to non-determinism and inaccuracies [1]–[4]. These issues are compounded in different architectures and kernel versions, where unclear documentation and inconsistent tool implementations can lead to overcounting and variations in readings. Most contamination sources are unrelated to the PMC itself, stemming instead from system activity and the impact of shared resources, affecting both PS and CSD metrics.

This work was funded by Key Digital Technologies Joint Undertaking (KDT JU) under the TRISTAN project (101095947).

Addressing these challenges, we propose a low-overhead, accurate, and security-conscious instrumentation utility for Real-Time Operating Systems (RTOSes), akin to Linux's `perf_event`. Our method, integrating low-level instrumentation into the RTOS's portable layer of FreeRTOS [5], isolates counters and events for each user application, minimizing measurement variability and enhancing tools like ePAPI and PRL. We utilize a RISC-V SoC (SiFive FE310-G002) for our proof-of-concept, ensuring isolation between kernel and user-level code. Our method isolates PMC values during context switches and employs low-level CSR calls for precise PMC sampling. This setup allows for comprehensive monitoring of context switches with minimal overhead and employs run-time calibration to nullify the instrumentation's measured overheads.

Our approach involves isolating PMC values during context switches and using low-level CSR calls for precise PMC sampling, thereby enabling comprehensive monitoring of context switches with minimal implementation overhead, while using run-time calibration to nulify the measured overheads from the instrumentation. Performance Monitoring Counters (PMCs) are specialized registers available in most modern processors for monitoring low-level system metrics. Each physical register is attached to a specific event. Monitored events can stem from **Core-State Dependent** (CSD) metrics (e.g., cache hits and misses, branch predictions) or from **Process-Specific** (PS) metrics (e.g., retired instructions, number of branches). They are often used for debugging and performance analysis. Our contributions are threefold:

1) **Precise Instrumentation**: Implementing accurate RTOS instrumentation to monitor user-level code, ensuring task isolation for system counters.
2) **Validation and Implementation**: Validating our implementation on the SiFive FE310-G002 SoC within FreeRTOS, with detailed low-level implementation exposed and adaptable to various system requirements.
3) **Security-Conscious**: Retaining PMC control in the highest privileged mode, while offering a low-overhead interface for user-level instrumentation requests.

979-8-3503-4859-0/DATE24/© 2024 EDAA

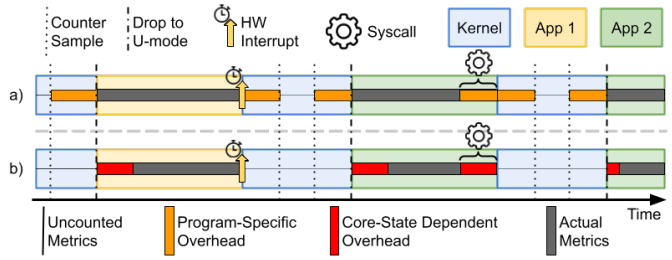

Fig. 1: Overheads on PS-related events (a) and CSD-related events (b) from OS intereference.

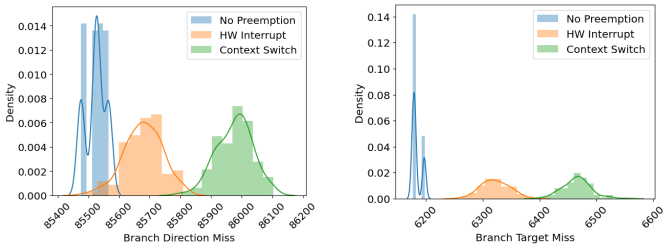

Fig. 2: Distribution of CSD metrics related to shared resources.

TABLE I: Measured overheads compared to the manually verified metrics for the 2 calibration functions lead us to the probe overheads of each type of call.

Counter Description	Syscall Calibration			Timer Calibration		
	Measured	Counted	Probe	Measured	Counted	Probe
Cycles	61	2[c]	59	46[a]	2[c]	44
Retired Instructions (RI)	43	2	41	34[b]	2	32
Integer store RI	28	0	28	28	0	28
System RI	7	0	7	7	0	7
Integer arithmetic RI	8	0	8	2	1	1
JAL RI	2	2	0	1	1	0
JALR RI	1	0	1	0	0	0
Branch/jump target misprediction	2	2[c]	0	0	0	0
Pipeline flush from CSR write	5	0	5	5	0	5

[a] 45.5 $\pm$ 1.120 [b] 34 $\pm$ 0.708 [c] Estimated

II. Low-Overhead PMC Utility

Our system integrates low-overhead instrumentation into the OS for effective monitoring, minimizing additional overhead and the impact on measurements. Using PMC, the system dynamically adjusts for context switches, interrupts, system calls, and exceptions, allowing precise overhead removal and understanding their timing and nature. Figure 1 shows the OS's effect on PS and CSD metrics. To mitigate ISR interference, counters are sampled right after the context is pushed to the stack, managed by the `context_switch_handler`. This process avoids altering the user application's context and limits the implementation overhead.

The system calibrates performance counters for each system call and interrupt, using known routines at startup for accurate ground-truth. This ensures that counter values reflect real-world scenarios and are used by the kernel without adding measurable code overhead. The calibration, especially for system call stubs, is crucial for precise measurements, with the calibration process and results for 1000 samples are detailed in Table I. The approach also allows for the variation in OS routines and adapts to different contexts, with counter access mainly restricted to the OS.

Validation: For our implementation test case, we developed a minimalist Proof-of-Concept (PoC) kernel, handling multiple user-level stacks and based on the RISC-V's GCC porting layer for FreeRTOS. The experiments were conducted on a SiFive FE310-G002 SoC, which incorporates the E31 Core Complex, supporting the RISC-V ISA performance monitoring standards and capable of executing U-mode and M-mode code. In our Coremark validation against a Manual Performance Utility (Manual PMC), we analyzed various metrics, including 2 fixed counters and aggregated instruction events. The results demonstrated that our instrumented kernel aligns well with 'Manual PMC' results in non-interference scenarios, confirming its effectiveness in accurately collecting performance metrics and reducing overheads, particularly for Program-Specific (PS) counters.

However, Core-State Dependent (CSD) counters, due to their susceptibility to transient execution and the inability of software to control their state, present a greater challenge. Figure 2 shows the varying impacts of OS interference, especially on branch direction and miss-prediction metrics. Notably, context switches significantly affect these metrics, a finding supported by 'no preemption' scenario results. This underscores the value of our RTOS instrumentation in fine-grained performance metric collection and in mitigating overheads for PS events, despite challenges with CSD counters.

III. Conclusions and Future Work

In this paper, we presented a methodology for instrumenting RTOSes that combines low overhead with high accuracy. Validated on a RISC-V microcontroller using FreeRTOS's context switch mechanism, this architecture-agnostic approach can be easily adapted to other platforms. It involves simple kernel patching at the (portable) architectural level. Our method, which offers always-on monitoring with minimal probe effect, integrates smoothly with the OS and completely eliminates program-specific overheads, while optimally mitigating shared hardware resource overheads. Future work will explore extending this instrumentation to more complex hardware setups and its potential for monitoring diverse application patterns.

References

[1] V. M. Weaver, D. Terpstra, and S. Moore, "Non-determinism and overcount on modern hardware performance counter implementations," in *IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2013, pp. 215–224.

[2] V. M. Weaver, "Self-monitoring overhead of the linux perf_event performance counter interface," in *IEEE International Symposium on Performance Analysis of Systems and Software*, 2015, pp. 102–111.

[3] Y. Liu and V. M. Weaver, "Enhancing PAPI with Low-Overhead rdpmc Reads," in *Programming and Performance Visualization Tools*. Springer International Publishing, 2019, pp. 3–20.

[4] S. Das, J. Werner, M. Antonakakis, M. Polychronakis, and F. Monrose, "Sok: The challenges, pitfalls, and perils of using hardware performance counters for security," in *IEEE Symposium on Security and Privacy*, 2019, pp. 20–38.

[5] FreeRTOS, "Freertos-kernel: Portable gcc risc-v code," 2023, accessed: 2023-09-12. [Online]. Available: https://github.com/FreeRTOS/FreeRTOS-Kernel/tree/main/portable/GCC/RISC-V