Advanced Operating Systems, Fall 2025 - Undergrad Project: Kernel Performance Monitoring on Raspberry Pi 4

In this project, you will explore low-level performance monitoring in Linux by implementing a kernel module on Raspberry Pi 4. You will use the ARM Performance Monitoring Unit (PMU) to collect microarchitectural statistics, analyze different workloads, and understand how hardware events reflect program behaviors. You will also extend your implementation to support finer-grained control and optionally modify the Linux scheduler for advanced measurement.

Due

Submit your full package (source code and report) to LMS before **Dec 5th, Friday, 11:59:59pm**.

- Include your student ID in the archived or compressed file name.
- No late submission will be accepted.

Background – ARM PMU

The ARM Performance Monitoring Unit (PMU) is a hardware feature that allows counting various microarchitectural events, such as the number of executed instructions, cache references, and cache misses. On Raspberry Pi 4, which uses the ARM Cortex-A72 core, you can access PMU counters through kernel code using specific system registers.

You can find details about PMU registers and event encodings in the ARM Developer Guide (*ARM Architecture Reference Manual*). You may also refer to the Linux kernel source tree, especially the file /arch/arm64/kernel/perf_event.c, to see how existing kernel drivers use PMU interfaces.

Part 1 – Kernel Module for PMU Statistics

Implement a kernel module that collects the following statistics using PMU:

- Number of instructions executed.
- Number of cache references and cache misses for L1 data/instruction caches.
- Number of cache misses for the last-level cache (LLC).

Expose the results to user space via /proc or /sys interface. You can create a simple read-only entry to print the measured values.

Part 2 – Measure Workload Behavior

Using your kernel module, measure the following Linux programs:

- openssl speed sha256
- stream (with multiple configurations)
- bzip2 (with multiple files of different sizes)

Collect PMU data for each program and discuss how their behaviors differ in terms of instruction count and cache activity. Identify which workload is more CPU-bound or memory-bound.

Part 3 – Add Pause/Resume Support

Extend your kernel module to support **pause** and **resume** functionalities for performance monitoring. Create a simple interface (e.g., through /proc/pmu_control) to start and stop measurements. This feature will allow you to measure specific phases of program execution separately.

Part 4 – Phase-wise Measurement Example

Write a simple program that demonstrates the use of pause/resume. For example:

- **Scenario 1:** Load a large file, then compute its hash value.
- **Scenario 2:** Initialize a data structure, then perform computation.
- **Scenario 3:** Run compression followed by encryption.

You can implement any program by yourself as long as they have multiple phases with different characteristics. Each phase should be measured independently using your kernel module.

Part 5 (Bonus) – Context-Switch Awareness

For the bonus task, modify the Linux kernel scheduler so that PMU counting automatically pauses and resumes during context switches. This ensures that PMU values only correspond to the process being measured. Look into context_switch() in kernel/sched/core.c for integration points.

Report

Write a report of up to **four pages** (five pages allowed if you complete the bonus part). The report should follow this structure:

- **System setup & Implementation:** One page summarizing how you implemented the kernel module and pause/resume control.
- **Measurement results and Discussion :** Three pages showing results (graphs, tables) for the three workloads and your phase-based example. Analyze your results and explain performance differences.

If you completed the bonus part, describe your scheduler modification approach and how it affects measurement accuracy.

Evaluation

Component	Weight
PMU kernel module functionality	20%
Measurement and workload analysis	20%
Pause/resume implementation	20%
Report quality and discussion	30%
Bonus (context-switch aware PMU)	+10%

Notes

- Your kernel module must load/unload properly using insmod and rmmod.
- Use a recent Linux kernel on Raspberry Pi 4.
- Avoid busy loops or excessive kernel logging.
- You may use AI tools (e.g., ChatGPT) to assist coding or debugging