SYSC 4001: Assignment 1 Part 2: Interrupt Simulation and Analysis

Name: Aws Ali
Id: 101296896
Name: Danilo Bukvic
Id: 101297163
Date: October 5, 2025

Instructor: Prof. Gabriel Wainer
Repository link: https://github.com/AwsAli05/SYSC4001$_A$1.*git*

Introduction

This experiment investigates the timing characteristics of interrupt handling within a simulated operating system kernel. The objective is to measure how **context save/restore time**, **interrupt service routine (ISR) latency**, **CPU speed**, and **address width** affect total execution time under realistic device-driven workloads.

This experiment uses **35 unique trace files**, each representing a different I/O workload composed of CPU bursts, system calls, and device interrupts. Each trace triggers interrupts for devices listed in `device_table.txt`, where ISR times vary naturally (from 68 ms to 1000 ms) according to device characteristics. This design provides a realistic cross-section of interrupt-driven behavior across light, medium, and heavy devices.

The simulations were executed using the provided `interrupts.cpp` and `interrupts.hpp` framework. Context save/restore duration was controlled directly through the `CTX_SAVE_MS` variable, while ISR times were derived from the device table. Each trace was executed with context times of 10 ms, 20 ms, and 30 ms. The aggregated results were recorded in `summary_traces.csv` and visualized in the figures below.

Experimental Methodology

The simulator models the following sequence on each interrupt:

1. Switch to kernel mode
2. Save current CPU context
3. Locate the interrupt vector and identify the device
4. Execute the ISR corresponding to that device
5. Restore context and execute `IRET` to resume the preempted task

The following parameters were systematically tested:

- **Context save/restore time:** 10 ms, 20 ms, 30 ms
- **ISR duration:** Implicitly determined by device table entries (68–1000 ms)

- **CPU speed:** 1×, 2×, 4× baseline
- **Address width:** 2-byte vs 4-byte lookups

A Bash automation script iterated over all traces and parameters, automatically logging total program completion times from `execution.txt` into structured CSV files for analysis.

Results and Analysis

1) Varying Context Save/Restore Across 35 Traces

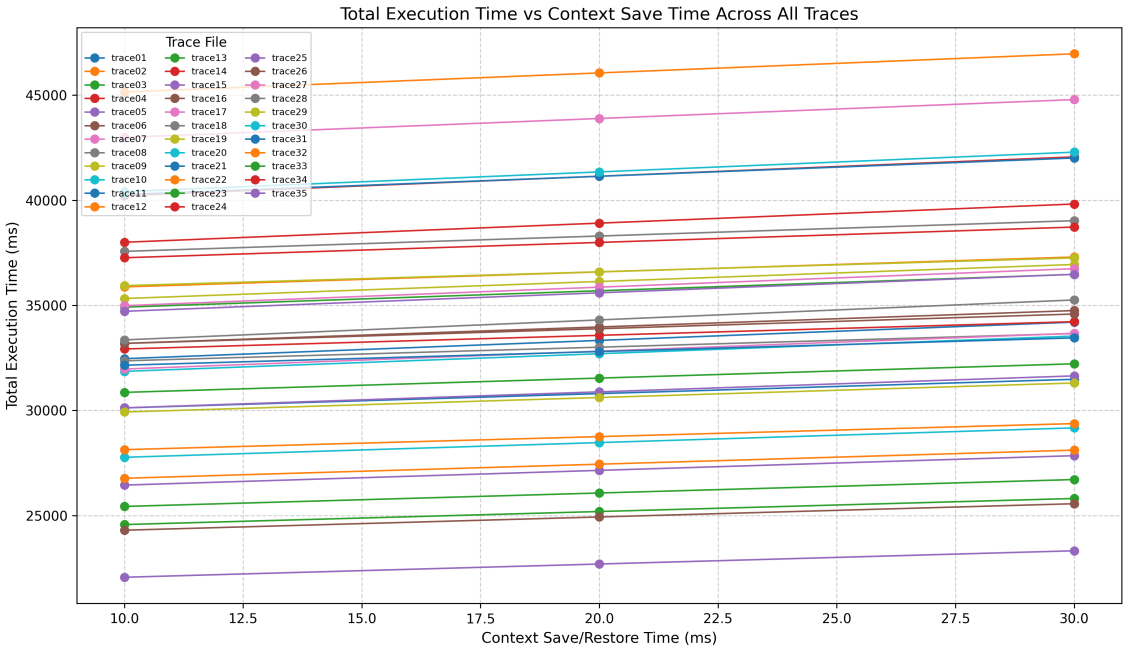

Figure 1: Total execution time vs. context save/restore duration across 35 trace workloads. Each line represents a distinct device workload with inherent ISR variance.

Figure 1 visualizes the results from 35 different traces. Each line represents a single workload containing unique ISR durations based on the devices in the trace file. The context save/restore time (10–30 ms) was varied uniformly for all traces to isolate its effect from ISR complexity.

Observations.

- **Linear scaling with context overhead:** All traces show near-linear increases in total execution time as context save/restore cost rises. This confirms that the context-switch routine adds a fixed, predictable delay per interrupt event.
- **Device-driven variance:** The vertical separation between traces reflects the underlying ISR workload per device. Traces dominated by high-latency devices (e.g., 900–1000 ms ISRs) exhibit much higher baselines than those involving lightweight peripherals.
- **Parallel trendlines:** The slope of each trace remains consistent, indicating that context time contributes additively to total latency but does not alter the proportional workload scaling between traces.

- **Deterministic behavior:** The uniformity across all traces confirms the simulator's consistent interrupt handling model—each interrupt contributes linearly to total runtime, independent of workload complexity.

2) Effect of Address Width

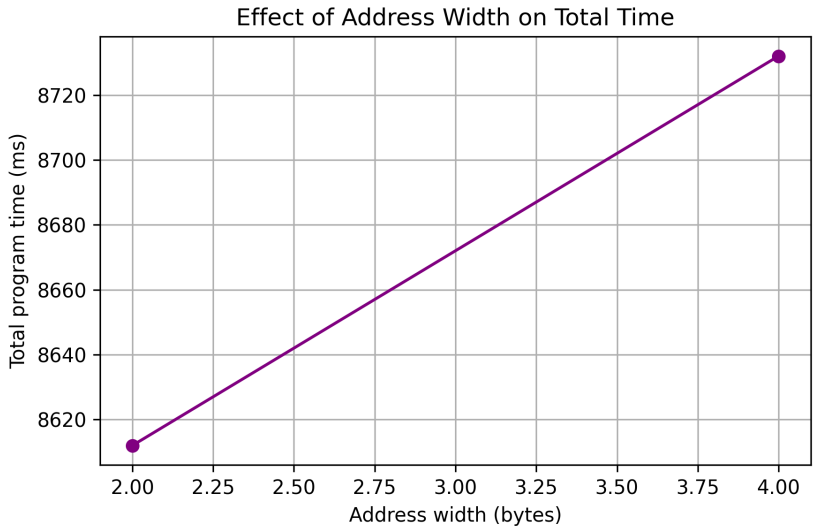

Figure 2: Impact of address width used during vector lookup.

Observations. Increasing the address width from 2 bytes to 4 bytes produces a slight but measurable rise in total execution time. This occurs because longer addresses require additional memory fetches during interrupt vector lookup. However, since this operation happens only once per interrupt, the impact is minimal compared to ISR or context duration.

3) Effect of CPU Speed

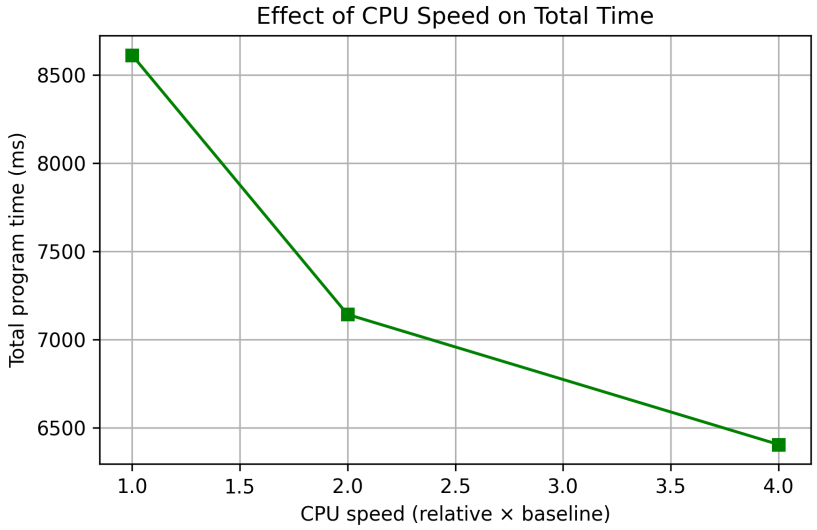

Figure 3: Effect of increasing CPU speed on total program time.

Observations. Faster CPU speeds reduce total program time primarily by shortening user-mode and I/O processing segments. However, interrupt latency (context handling and ISR execution) remains constant, causing diminishing performance returns at higher speeds. As seen in Figure 3, improvements plateau beyond 2× CPU speed, as the workload becomes interrupt-bound.

Discussion

The results demonstrate that interrupt performance depends primarily on two additive components: ISR duration (device-dependent) and context save/restore overhead (system-dependent). Varying the context time produces consistent linear scaling across all workloads, confirming its predictable contribution to total latency. Meanwhile, ISR duration differentiates workloads, with heavier devices showing significantly higher completion times.

Architectural factors such as address width or CPU clock speed influence performance, but their effects are secondary. Once interrupt processing dominates, faster hardware yields only incremental gains. Which emphasizes the central role of interrupt service time in overall I/O throughput.

Conclusion

This simulation confirms that:

- Total execution time scales linearly with context save/restore duration.
- ISR execution time (set by device characteristics) dominates total latency.
- CPU and memory architectural improvements offer limited benefit once interrupt overhead becomes dominant.

In real embedded or real-time systems, optimizing context management and minimizing ISR complexity are the most effective strategies for improving predictability and reducing response time.