SYSC 4001 Assignment 1 Report

Group Members: Aryan Kumar Singh (101299776), Srivathsan Murali (101287534)
GitHub Repo: https://github.com/CryptidFiles/SYSC4001_A1

NOTE: For Test Cases 1–11, a consistent input trace (test_trace.txt) was used to isolate the effects of varying context time, ISR duration, vector size, and CPU speed. Separate trace files were introduced from Case 12 onward to simulate distinct device delay scenarios.

# Test Case 1: Baseline (original parameters)
Parameters: Context = 10 ms, ISR = 40 ms, Vector = 2 bytes, CPU = 1.0×

This case serves as the reference model for all subsequent variations. The system displays stable interrupt servicing where user execution and kernel-mode transitions coexist smoothly. The baseline highlights a well-balanced interaction between computation and interrupt handling, forming the standard of comparison for evaluating changes in context, ISR duration, and CPU speed.

# Test Case 2-4: Vary context save/restore time (10ms, 20ms, 30ms)
Configuration: Context Time = 10 ms → 30 ms, ISR = 40 ms, Vector = 2 bytes, CPU = 1.0×

As context time increases, each interrupt introduces a visibly longer pause before resuming normal execution. At around 20 ms, responses feel slightly delayed; by 30 ms, mode transitions noticeably stretch the timeline. The system still completes all operations correctly, but the CPU spends a larger share of its time saving and restoring registers rather than doing useful work. These results confirm that inefficient context management directly contributes to cumulative latency in multitasking environments.

# Test Case 5-7: Vary vector address size (2 bytes, 4 bytes, 8 bytes)
Configuration: Context = 10 ms, ISR = 40 ms, Vector = 2–8 bytes, CPU = 1.0×

Changing the vector-table width has only a minor effect. Even when the table doubles or quadruples in size, lookup operations occupy a tiny portion of total runtime. The trace sequences appear nearly identical, showing that vector resolution remains a lightweight task compared with the rest of interrupt servicing. The finding reinforces that CPU and context timing dominate performance, while vector width mainly influences memory layout and scalability rather than speed.

# Test Case 8-11: Vary CPU speed (0.5, 1.0, 2.0, 4.0)

Configuration: Context = 10 ms, ISR = 40 ms, Vector = 2 bytes, CPU Speed = varied

Lowering CPU speed roughly doubles the perceived runtime, with longer CPU bursts and slower transitions between tasks. At 0.5×, the processor feels busy but sluggish, and interrupts dominate the schedule. Restoring the normal 1.0× speed returns the balanced pattern observed in the baseline.

Increasing to 2.0× compresses CPU activity into noticeably shorter intervals, improving responsiveness without altering the sequence of events. At 4.0×, however, gains plateau—the trace still shows identical ordering, but fixed-time operations such as context saving and ISR work now occupy most of the timeline. This reveals the point where the system becomes limited by I/O latency rather than raw compute speed.

# Test Case 12-16: Vary ISR activity time (20, 80, 120, 160, 200)
Configuration: ISR activity time: **varied**, rest are same as base case
Input trace: test_trace2.txt

The key finding with variest ISR activity times is that a universally "good" ISR time does not exist, only ISR times appropriate for specific device characteristics and system requirements. Around the 120ms mark emerges as a critical collapse point, where system efficiency drops precipitously from sustainable levels to a state where overhead begins to dominate productive work. An interesting case occurs when a small duration for each ISR activity is initialized. For the device number 15 (68ms for delay), during the SYSCALL the ISR activities take up 40ms each, exceeding the required time needed for the device.

# Test Case 17: Fast Context Time but Slow ISR Activity Time
Configuration: Context Time = 5ms, ISR Time = 200ms, Vector Size = 2 bytes, Devices 9 (1000ms) and 19 (652ms)
Input trace: test_trace3.txt

This configuration renders the fast context switching optimization almost negligible as the excessively slow ISR execution of 200ms dominates the interrupt timeline. The fast context switching optimization led to an overhead time of 162ms while the actual I/O time took 4504ms, which is 75% of the total useful time of CPU and I/O. The assumption that slow ISRs are inherently inefficient proves incorrect as the case shows a favourable efficiency ratio as useful I/O operations substantially outweigh work of the overhead.

# Test Case 18: Slow Context Time but Fast ISR Activity Time

Configuration: Context Time = 30ms, ISR Time = 20ms, Devices: 16 (68ms) and 1 (100ms)
Input trace: test_trace4.txt

This case uncovers a paradoxical efficiency phenomenon. Despite the fast 20ms ISR time, the 30ms context switching creates a situation where the overhead-to-work ratio becomes inverted. For fast devices like Device 16 (68ms total), the system spends 25ms on overhead versus 68ms on useful I/O operations. This reveals that for high-frequency, low-latency I/O operations, context switching overhead becomes the dominant performance factor regardless of ISR efficiency.

# Test Case 19: High overhead
Configuration: Context Time = 30ms, ISR Time = 200ms, Vector Size = 8 bytes, CPU Speed = 0.5x, Devices: 19 (652ms) and 8 (1000ms)
Input trace: test_trace5.txt

Despite being the worst-case configuration for completing a user program, the extremely slow devices (Device 8 at 1000ms) still provide substantial useful work time whereas the faster device (Device 0 at 110ms) spends approximately half of the interrupt activity handling on overhead work. The analysis shows that even in this heavily penalized scenario, the system achieves approximately ms of productive I/O work per interrupt. Optimization strategies should differ based on I/O characteristics: latency-sensitive systems must minimize absolute overhead, while throughput-focused systems can tolerate higher overhead if it enables greater useful work capacity.

# Test Case 20: Low Overhead
Configuration: Context Time 5ms, ISR Activity Time 20ms, and Vector Size 1 byte
Input trace: test_trace6.txt

This optimized configuration reveals an "optimization threshold" where additional effort to reduce interrupt overhead provides negligible real-world benefits. While the system achieves an impressive 24ms one interrupt overhead, the analysis shows that further reductions provide minimal benefits for certain workloads. Reducing overhead further would yield progressively smaller improvements in overall efficiency.