SYSC 4001-A
Operating Systems

Assignment 1 - Report

Naima Mamun, 101276213
Tabia Farid, 101258535

Submitted: October 5, 2025

Part II: Design and Implementation of an Interrupts Simulator

In this experiment, the interrupt handling process was simulated in C++ using a trace-based model that mimics how the CPU responds to system calls and I/O completions. Each event in the trace file was processed sequentially, and the program recorded every step of the interrupt routine, including saving and restoring context, running the Interrupt Service Routine (ISR), transferring data, checking for errors, and returning to user mode. The total execution time was measured by advancing a global clock that accumulates the time spent on both CPU work and interrupt-related overhead. By varying key parameters such as save/restore context time and ISR activity time, the experiment demonstrates how different stages of the interrupt process influence the system's overall performance and reliability.

When the save/restore context time was varied between 10 ms, 20 ms, and 30 ms, the total execution time increased in a linear trend. Each interrupt requires saving the CPU's current state before entering kernel mode and restoring it afterward, so increasing this value directly adds overhead to every interrupt handled. Since both the save and restore steps occur once per interrupt, doubling the context-switch duration effectively adds twice the change in delay per interrupt. In the case of trace1.txt, we observed a clear linear relationship between the context-switch time and the total execution time. When the context time parameter was set to 10 ms, the total runtime was 3832 ms. Increasing it to 20 ms raised the runtime to 4032 ms, and further increasing it to 30 ms resulted in 4232 ms. Since the trace includes 10 total interrupts, each additional 10 ms of context time adds approximately 20 ms of extra work per interrupt (accounting for both saving and restoring context). Experimenting with another trace file, trace2.txt, also demonstrates this pattern, as shown by its three associated execution files. These results align with the expected behaviour, that higher context-switch times directly increase overall system overhead in proportion to the number of interrupts handled.

Next, we experimented by varying the ISR activity time between 40 ms and 200 ms. From trace 3 through trace 10, each trace is paired with two execution files, the first run uses a smaller ISR activity time, and the second run increases it by 10 ms. The ISR time begins at 50 ms for trace 3 and rises by 10 ms for each new trace, reaching 200 ms by trace 10. This gradual increase allows us to observe how longer interrupt service routines affect system behaviour and overall execution time.

Across these traces, the trend is consistent, as the ISR activity time grows, total execution time initially increases steadily but eventually leads to failures where the program halts early with a "DEVICE TIMEOUT / DATA LOSS" error. This happens because each device has a fixed delay, and when the ISR routine takes too long relative to that delay, it exceeds the device's timing window. Additionally, this misuses the CPU's computation time as the CPU cannot run other

tasks of equal or lower priority, until the ISR is complete. In early traces, such as trace 3, where the ISR time was only 50 ms, most devices completed successfully, and the system ran nearly the entire trace before hitting a timeout near the end. As the ISR grew to 60 ms and higher, the same devices began to fail earlier, showing that small increases in ISR duration can quickly push certain devices past their limits.

By the time we reach traces 6 and 7 (with ISR times 110–140 ms), the system becomes noticeably more fragile. Devices with smaller delay values, like device 9 in trace 6 or device 7 in trace 7, can no longer complete even a single full interrupt cycle without timing out. Each increment of 10 ms in ISR time shortens how far the simulation can progress before failing, because the total time spent inside the ISR doubles for SYSCALL events. These results show how the most timing-sensitive device determines the overall reliability of the system. If even one device's required delay is smaller than the ISR's execution time, the system cannot finish its interrupt sequence.

When the ISR activity time reaches 150 ms or higher (traces 8 to 10), the effect becomes more noticeable. In these later traces, the simulation often fails almost immediately after the first SYSCALL because the ISR now consumes so much time that it instantly exceeds the device's allowed delay. Only traces that involve devices with larger delays, such as those in trace 9, can still complete, because those devices have enough processing slack to tolerate longer ISR times. The pattern across all traces demonstrates a clear linear-then-cliff relationship, where execution time rises steadily with ISR length while all devices meet their timing constraints, but once ISR time crosses a device's threshold, the run terminates abruptly due to a timeout.

If we had 4-byte addresses instead of 2, the system would probably slow down since it would take longer to fetch and handle larger addresses during each interrupt. On the other hand, if we had a faster CPU, everything would likely run smoother, the context saves, restores, and ISR executions would finish faster, meaning fewer timeouts and overall shorter total execution times. So, bigger addresses might slightly increase overhead, while a faster CPU would make the system more efficient and responsive.

In summary, traces 3 through 10 illustrate how increasing the ISR activity time gradually shifts the system from stable performance to failure. At lower ISR values, execution time increases predictably due to added overhead, and at higher ISR values, performance collapses as devices miss their service deadlines. The experiment shows that interrupt handling speed is very important, since even small delays in the ISR routine can lead to missed device responses, making efficient ISR design essential for reliable system performance.

Our GitHub Repository Link:
https://github.com/tabiafarid/SYSC4001_A1