SYSC 4001

ASSIGNMENT 1 Report

September 26th 2025

Sammy Eyongorock-101246652-Student 1

Paul Harry Theuma-101299749-Student 2

Github Link: https://github.com/Esammy-88/SYSC4001_A1

Objective

The goal of this assignment was to design and implement a simulator that models the operation of an interrupt-driven system.
The simulator processes a sequence of CPU bursts, system calls (SYSCALL), and interrupt completion events (END_IO) while tracking timing and overheads associated with context switching, interrupt handling, and I/O device delays.
This simulation provides a controlled environment for analyzing the performance of interrupt handling mechanisms, allowing for the observation of how different timing factors (context save time, ISR activity duration, and device delays) affect overall program execution time.

Methodology

The simulator was developed in C++ using the provided template (interrupts.cpp and interrupts.hpp).
Input traces represent program execution sequences that alternate between CPU and I/O operations.
Two supporting files were used:

- Vector Table: Maps device numbers to interrupt service routine (ISR) addresses.
- Device Table: Defines the average I/O completion delays per device.

The simulator was designed to output an execution trace (execution.txt) containing timestamped system events such as:

- CPU bursts
- Context saving and restoring
- Switching between user and kernel modes
- ISR execution
- Device interrupt completions

Each event was assigned a fixed simulated duration based on system configuration constants (e.g., 1 ms for mode switch, 10–30 ms for context save/restore, 40–200 ms for ISR execution).
Simulation cases were executed by systematically varying:

1. Context save/restore duration (Supposed to be 10 ms, 20 ms,30 ms, but we used random variables for more analysis)

2. ISR activity duration (40 ms to 200 ms)

Each configuration was recompiled and executed with different input traces that were stored under input_files/, producing individual logs in output_files/.

Analysis

-Changing the value of Context Save/Restore from 10, 20 to 30 shows a linear increase in the total execution time. Each interrupt requires context saving twice.

-Increasing the ISR linearly in the range 40-200ms shows a linear relationship with the total execution time. When ISR takes too long, there is a delay in other interrupts and CPU work, reducing the system's efficiency.

-The longer the respective steps, the longer the execution time is. This shows a direct relationship between save/restore context time and total execution time, similarly with ISR activity time and total execution time. The longer the interrupt, the longer the total execution time.

-Changing the address size from 2bytes to 4 bytes will probably have close to no effect on the functioning of the simulation.

-A faster CPU will benefit all parts of the simulation. Useful work and overhead benefit from a faster CPU.

-From overhead analysis, useful CPU work is ~1800ms, interrupt overhead (total time - useful work) is ~3260ms, the efficiency is 36%

-Device delays do not affect CPU usage but increase the total time.

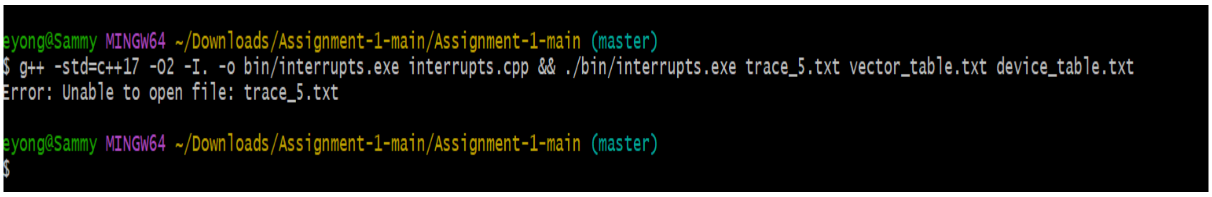
-Context switching and ISR execution are bottlenecks

An example of a challenge faced

We were unable to run trace_5.txt because the file was still in the test cases folder.

To fix this, we simply had to bring out the test cases from the test cases folder into

the Assignment-1-main folder



```
eyong@Sammy MINGW64 ~/Downloads/Assignment-1-main/Assignment-1-main (master)
$ g++ -std=c++17 -O2 -I. -o bin/interrupts.exe interrupts.cpp && ./bin/interrupts.exe trace_5.txt vector_table.txt device_table.txt
Error: Unable to open file: trace_5.txt

eyong@Sammy MINGW64 ~/Downloads/Assignment-1-main/Assignment-1-main (master)
$
```

Discussion

From the experiments, it became evident that interrupt handling contributes measurable overhead to system performance.
In a real operating system, this overhead corresponds to kernel-mode time and affects latency-sensitive applications.
Optimizing interrupt performance, therefore, requires:

- Minimizing context switch durations via lightweight state saving.
- Keeping ISR routines short, offloading longer work to deferred processes.
- Efficiently managing vector tables to reduce ISR lookup overhead.
- Balancing device concurrency and CPU workload to maintain throughput.

The simulation results qualitatively match expectations from real systems:
longer context operations and ISR times produce slower response, while high device delays affect total runtime but not CPU efficiency.

Conclusion

The interrupt system simulator successfully models the interaction between CPU bursts, system calls, and I/O completion interrupts.
By varying different timing parameters, the study demonstrated how interrupt overhead impacts system performance.
Key conclusions:

- Context save/restore times directly add to system overhead per interrupt.
- ISR execution times determine service latency and CPU blocking time.
- Device delays extend the total elapsed time but not CPU work.

- Workload composition (CPU-bound vs I/O-bound) dictates sensitivity to these factors.

This simulator provides a solid platform for further experiments in later assignments, such as multi-interrupt scheduling or performance optimization analysis.