SYSC4001 - Assignment 1 Report

In this report, we will be discussing our analysis and results obtained from running multiple tests on the simulation code to understand the interrupt system and its interaction with the OS. Below you'll find tables of a few of the cases we ran with various test conditions to see the different outcomes. We will be describing each test condition in detail in this document, like save/restore context time, ISR execution and more.

The simulation tests changed the save/restore context time in increments of 10ms (from 10ms to 30ms) systematically to assess the effect on total program execution time of each workload and interrupt activity pattern we evaluated. The important observation is that increasing the context save/restore time consistently increases overall execution time. The reason is that every interrupt saves and restores CPU context. Even small increases in context saving and restoring times add up to a substantial overhead over so many frequent interrupts.

Trace File	Context Save Time (ms)	Effect on Runtime
trace.txt	10 → 20 → 30	Runtime increases linearly with context time
trace_1.txt	10 → 20 → 30	Noticeable increase with each 10 ms step
trace_2.txt	10 → 20 → 30	Total execution grows significantly
trace_3.txt	10 → 20 → 30	Overhead accumulates, impacting total time
trace_4.txt	10 → 20 → 30	Runtime nearly doubles moving from 10 to 30ms
trace_5.txt	10 → 20 → 30	Moderate increase in total simulation time
trace_6.txt	10 → 20 → 30	Sharp increase in runtime due to frequent interrupts
trace_7.txt	10 → 20 → 30	Largest increase in heavy-load scenario
trace_8.txt	10 → 20 → 30	Noticeable increase in overhead
trace_9.txt	10 → 20 → 30	Runtime impacted by frequent context saves

The ISR (Interrupt Service Routine) execution time matters when determining how quickly the OS serves hardware interrupts and resumes ordinary process operation. The ISR time increases above the baseline from 40ms to higher alternatives (up to 200ms allowed) and meaningfully affects the overall completion time of programs that are simulated. Below is a table summarizing our findings and observations.

Test	Trace File	ISR Time (ms)	Key Observation
10	trace_10.txt	40 → 120 → 200	Minimal interrupts; runtime grows steadily with ISR time
11	trace_11.txt	40 → 120 → 200	More frequent IO; ISR rise causes marked slowdown
12	trace_12.txt	40 → 120 → 200	Dense; doubling ISR time about doubles overall runtime
13	trace_13.txt	40 → 120 → 200	IO-heavy; high ISR makes processing much slower
14	trace_14.txt	40 → 120 → 200	Mixed pattern; ISR overhead obvious with more events
15	trace_15.txt	40 → 120 → 200	Frequent context switches; higher ISR multiplies overhead
16	trace_16.txt	40 → 120 → 200	Back-to-back interrupts; bottleneck at 200ms
17	trace_17.txt	40 → 120 → 200	Low-moderate IO; runtime grows with ISR duration
18	trace_18.txt	40 → 120 → 200	Short workload; high ISR slows even small traces
19	trace_19.txt	40 → 120 → 200	Dense bursts; doubling ISR time roughly doubles runtime
20	trace_20.txt	40 → 120 → 200	Varied bursts and IO; heavy ISR overhead in busier sections

How does the difference in speed of these steps affect the overall execution time of the process?
The speed variations on context save/restore time and ISR execution time can greatly affect overall process execution time. As these times increase, the overhead of each interrupt is also increased, and for processes with high interrupt frequency or many IO events, this overhead will quickly add up. For example, when going from a context save time of 10ms to a context save time of 30ms, not only will the context save time increase, but the overall transition time from user mode to kernel mode will increase as well, increasing the amount of time before the program can continue the execution of instructions. Longer ISR execution times mean the processor is spending more cycles on servicing interrupts and less upon the computations or other IO it must perform. Therefore, in systems with a high frequency of interrupts or multiple IO device requests, small amounts of increases in these delays can lead to large percentage increases in the overall runtime of the program and contribute to delays and reduced responsiveness in the system.

Ask yourselves other interesting questions and try to answer them through simulations. For instance: what happens if we have addresses of 4 bytes instead of 2? What if we have a faster CPU?

To start with, I made a change to the size of the interrupt vector address from 2 bytes to 4 bytes. This requires more to be saved and restored in the interrupt routine, slightly increasing overhead time for each interrupt. Although the difference may not seem substantial, the ramifications became evident after I began testing a CPU with frequent interrupts, such as one that uses IO-but CPU-bound routines. In those systems, where the CPU still finished the normal computation fast, the overhead took a larger and larger percentage of runtime as I sped up CPU and reduced instruction time and burst time. This is meant to illustrate the lesson that, on occasion, minor changes to hardware can have significant impact on overall performance, such as the width of the address.

Next, I tested a CPU with a much faster performance profile and reduced the instruction time and burst time. Still the CPU was completing the normal computing from the user in a fraction of the time, but time spent handling interrupts and context switches had adjusted to be roughly the same. As the CPU was too much faster, these overhead times became an increasingly larger portion of overall runtime. To summarize the point here, merely speeding up hardware does not improve overall performance unless the software is also (or especially the interrupt service routines), being optimized.