SYSC 4001

Assignment 1 Simulator Code Report

Chris Canton - 101000259

Joshua Nicholls - 101211810

26th September, 2025

Change the value of the *save/restore context* time from 10, to 20, to 30ms. What do you observe?

As you increase the context switch time it can be seen that the overhead increases for the same task. The added overhead is equal to the number of context switches multiplied by the duration of the context switch. Below shows the final CPU burst for the same trace running with a 10,20 and 30 ms context switch duration.

10ms Context Switch Duration

31529, 10, context saved

31539, 1, find vector 4 in memory position 0x0008

31540, 1, load address 0X0292 into the PC

31541, 40 ENDIO: run the ISR (Device Driver). 4

31581, 210 Check Device 4 status.

31791, 63, CPU Burst.

20ms Context Switch Duration

32119, 20, context saved

32139, 1, find vector 4 in memory position 0x0008

32140, 1, load address 0X0292 into the PC

32141, 40 ENDIO: run the ISR (Device Driver). 4

32181, 210 Check Device 4 status.

32391, 63, CPU Burst.

30ms Context Switch Duration

32709, 30, context saved

32739, 1, find vector 4 in memory position 0x0008

32740, 1, load address 0X0292 into the PC

32741, 40 ENDIO: run the ISR (Device Driver). 4

32781, 210 Check Device 4 status.

32991, 63, CPU Burst.

As can be seen the total time for an entire trace to be completed is increased as you increase the context switch duration adding overhead and as a consequence lowering throughput. If context switches are inefficient or called more than needed this affects the overall performance of the system. This is even more important in systems that multitask as the performance gain from multitasking is lowered if the time to switch tasks becomes a large enough portion of performance increase either due to duration or quantity of switches.

Vary the ISR activity time from between 40 and 200, what happens when the ISR execution takes too long?

40ms ISR Duration

19152, 40, SYSCALL: run ISR (Device Driver) run.

19192, 40, Transfer data from device to memory

19232, -12, Check for errors.

19220, 51, CPU Burst.

- Since all times from the device table are above 40ms this causes only 1 issue. With addition of the time from the Transfer data activity the device with a 68ms duration is shorter than the time taken for the Syscal to run and then run its own activities.

80ms Context Switch Duration

2503, 80, SYSCALL: run ISR (Device Driver) run.

2583, 80, Transfer data from device to memory

2663, -8, Check for errors.

2655, 25, CPU Burst.

- When the ISR duration is increased this timing error occurs more frequently. With 80ms ISR duration the device's duration is shorter than the required duration for the ISR to run.

120ms Context Switch Duration

2503, 120, SYSCALL: run ISR (Device Driver) run.

2623, 120, Transfer data from device to memory

2743, -88, Check for errors.

2655, 25, CPU Burst.

- Like with the 80ms duration the 120ms duration increases the frequency of the error to 16 occurrences with the same trace file.

160ms Context Switch Duration

30044, 160, SYSCALL: run ISR (Device Driver) run.

30204, 160, Transfer data from device to memory

30364, -55, Check for errors.

30309, 58, CPU Burst.

- With an ISR duration of 160ms the error occurred 25 times.

200ms Context Switch Duration

10257, 200, SYSCALL: run ISR (Device Driver) run.

10457, 200, Transfer data from device to memory

10657, -250, Check for errors.

10407, 26, CPU Burst.

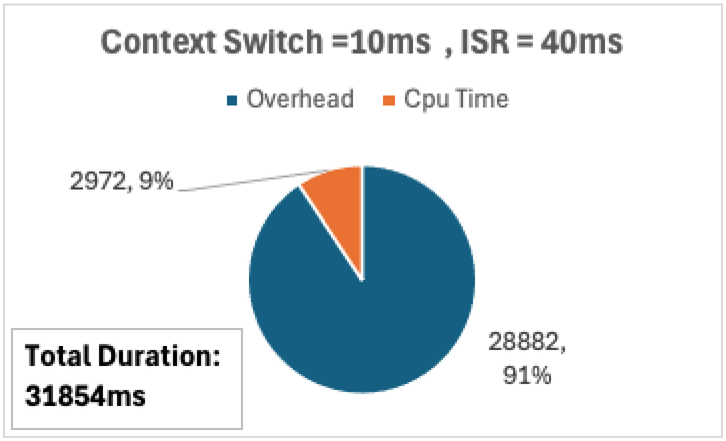
- With the ISR duration set to 200ms the number of occurrences is still 25 but the degree to which they were late has increased considerably.

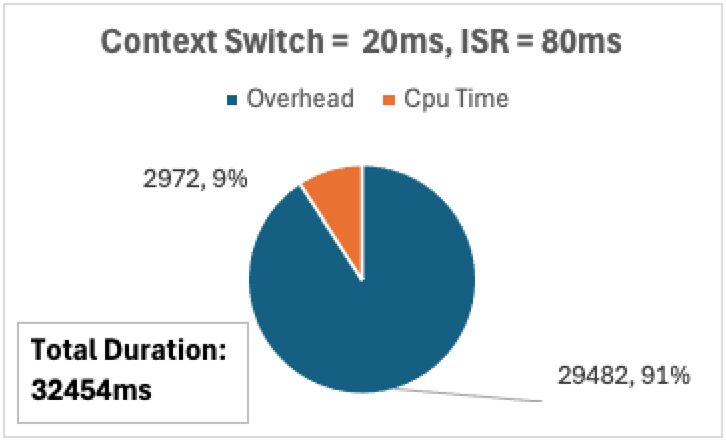
These tests show clearly that properly managing timing and efficiently using the systems resources are crucial for a system to be robust. Proper error handling is also very important for a system that may be used in many different ways as shown by the range of our testing. It should be noted that you could increase the duration of the ISR so that it exceeded all of your device duration in which case the trace files itself is completely useless.

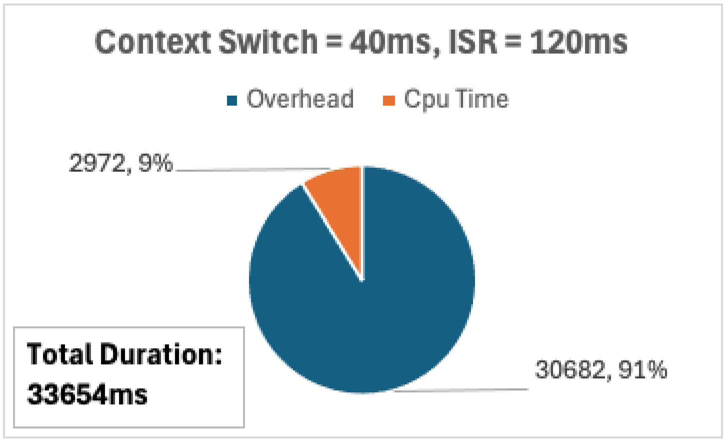
How does the difference in speed of these steps affect the overall execution time of the Process? :

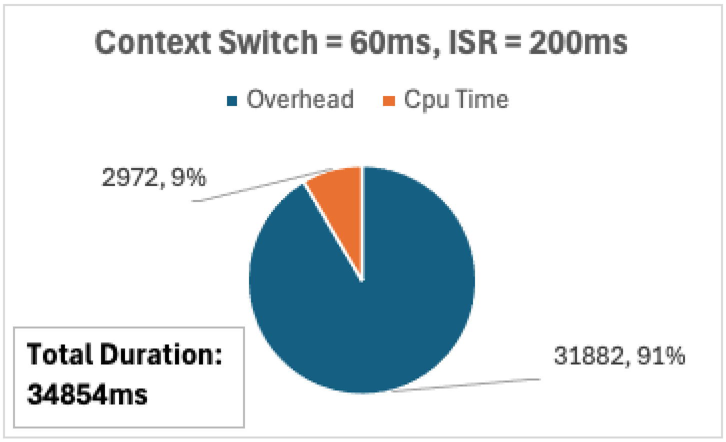
If we assume the ISR duration is not increased past the required time for the IO (as this is really an issue with errors not execution time) we can gain insight into the effect of increased durations. As you can see as you increase ISR like with the context switch time, only the overhead increases but the CPU uptime remains the same. When these durations are increased they affect only the total uptime of the system and not the CPU execution time. This means that as a percentage of the total uptime CPU throughput is lowered as you increase factors like context switch time and time to complete an ISR. This effect changes from program to program as the number of system calls and length of individual CPU bursts depend on the specific trace file but it still holds that as you increase these factors you lower throughput. This demonstrates clearly the need for a very quick interrupt handling system to make multi tasking in systems practical. (if the switching of jobs takes too long, and you have too many at a certain point you may be better to just not multi task in extreme cases.) Another important observation from the last two questions is that in both cases a large portion of the time was from the I/O times from the device table.

You can process this data using a Python script, Excel spreadsheet or any other tools for separating the overhead from the actual work of your program (i.e., CPU use for processing and actual I/O needed by the program):









The above tests show us a few important things. First as we increase things like the ISR time, or context switch time it does not affect the amount of time the cpu spends processing. What it does do is increase total time for the trace file to complete. This means the cpu is working for less percentage of the total time as ISR or context switch time increases. In other words, throughput is decreased. When plotting data from the test it is interesting to see that although numerically you can see a difference as a percentage of the total time the difference is not particularly large. This however is just a consequence of small trace files and the effect would compound the longer the trace file was.

What happens if we have addresses of 4 bytes instead of 2? What if we have a faster CPU:

Since the current number of addresses with 2 bytes is 2^N bits , which is 2^16 then if we have 4 bytes we have 2^32 possible Isr addresses. This means the number of ISR addresses is much , much larger and you can have many more SYSCALLS . A faster CPU would decrease both the time to execute the CPU processes and the context switches and ISR's. Unfortunately as mentioned earlier so much of the total time consisted of I/O operations (time read from device table) that even a faster CPU would improve the throughput but not solve the bottleneck.

How to improve the simulator? :

The first thing is to add function to flag traces whose device time is incompatible with the ISR times. That way we would never get the negative duration for any activity. Beyond that, the simulator does not have many features, like scheduling, multitasking etc. Building upon the current simulator would make for a more realistic simulation that could comprise more than just the Sycall process and interface and simulate more parts of the operating system. Another issue with the simulator is how it is actually sequences, breaking blocks up even more would allow me to build in more operating system function without rebuilding parts of the system. Although there is some error handling, more complete error and debug messages could be included. Lastly, handling of slightly malformed traces (like double space between comma and put time) could be handled by prompting the user asking if the input is correct or if they would like to delete the white space and continue.