**Here is the link to the project repository:** https://github.com/sajohna/SYSC4001_A1/tree/main

**Overall Analysis:**
From general analysis of the 20 test case simulations we did, we were able to understand the process of an interrupt driven I/O system. For each test case, we varied the number of iterations through the interrupt cycle, as well as changing the time of CPU bursts and the address of the I/O device. On average, the timing for context switching was about 22 ms and 2ms for vector lookup and PC loading. The time between the "end of I/O" and "cpu burst" instruction varies through the traces, showing how the interrupt latency changes per simulation. Each input file tests different possibilities, such as very long CPU bursts, multiple lookups to the same I/O device, bound-testing at the end of files, and only interrupting once. Altogether, our program handled each case correctly and with ease, and these test cases illustrated to us how the interrupt process changes in different situations.

Here are the results of changing specific factors within the simulations:
**- Change the value of the save/restore context time from 10, to 20, to 30ms. What do you observe?**
There will be screenshots appended to this report to show the results of our testing but the full execution result file can be found in the github repository under output_files/report_results/changing_saverestore_context_time.

Results when save/restore context time is 10ms:

```
45   2062, 84, CPU burst
46   2146, 1, switch to kernel mode
47   2147, 10, context saved
48   2157, 1, find vector 7 in memory position 0x000E
49   2158, 1, load address 0X00BD into the PC
50   2159, 152, execute I/O
51   2311, 1, execute IRET
52   2312, 1, switch out of kernel mode
53   2313, 10, context reloaded
54   2323, 38, CPU burst
55   2361, 20, end of I/O 7: interrupt
56   2381, 90, CPU burst
```

Results when save/restore context time is 20ms:

```
45   2101, 1, end of I/O 4: interrupt
46   2102, 84, CPU burst
47   2186, 1, switch to kernel mode
48   2187, 10, context saved
49   2197, 1, find vector 7 in memory position 0x000E
50   2198, 1, load address 0X00BD into the PC
51   2199, 152, execute I/O
52   2351, 1, execute IRET
53   2352, 1, switch out of kernel mode
54   2353, 10, context reloaded
55   2373, 38, CPU burst
56   2411, 20, end of I/O 7: interrupt
57   2431, 90, CPU burst
```

Results when save/restore context time is 30ms:

```
45   2142, 84, CPU burst
46   2226, 1, switch to kernel mode
47   2227, 10, context saved
48   2237, 1, find vector 7 in memory position 0x000E
49   2238, 1, load address 0X00BD into the PC
50   2239, 152, execute I/O
51   2391, 1, execute IRET
52   2392, 1, switch out of kernel mode
53   2393, 10, context reloaded
54   2423, 38, CPU burst
55   2461, 20, end of I/O 7: interrupt
56   2481, 90, CPU burst
```

Each interrupt requires saving context on entry and restoring context on IRET. So increasing the time for these instructions would overall increase the time taken for interrupt handling. This is seen through the screenshots

as the save/context time is increased, so is the execution time. Thus, the overall execution time increases while the CPU efficiency decreases as more time is spent dealing with interrupt overhead than the actual workload.

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

There will be screenshots appended to this report to show the results of our testing but the full execution result file can be found in the github repository under output_files/report_results/changing_ISR_activity_time. We decided to increase the ISR activity time in increments of 40.

ISR time is 40:

```
2182, 84, CPU burst
2266, 1, switch to kernel mode
2267, 40, context saved
2307, 1, find vector 7 in memory position 0x000E
2308, 1, load address 0X00BD into the PC
2309, 152, execute I/O
2461, 1, execute IRET
2462, 1, switch out of kernel mode
2463, 10, context reloaded
2473, 38, CPU burst
2511, 20, end of I/O 7: interrupt
2531, 90, CPU burst
```

ISR is 80:

```
2342, 84, CPU burst
2426, 1, switch to kernel mode
2427, 80, context saved
2507, 1, find vector 7 in memory position 0x000E
2508, 1, load address 0X00BD into the PC
2509, 152, execute I/O
2661, 1, execute IRET
2662, 1, switch out of kernel mode
2663, 10, context reloaded
2673, 38, CPU burst
2711, 20, end of I/O 7: interrupt
2731, 90, CPU burst
```

ISR time is 120:

```
2502, 84, CPU burst
2586, 1, switch to kernel mode
2587, 120, context saved
2707, 1, find vector 7 in memory position 0x000E
2708, 1, load address 0X00BD into the PC
2709, 152, execute I/O
2861, 1, execute IRET
2862, 1, switch out of kernel mode
2863, 10, context reloaded
2873, 38, CPU burst
2911, 20, end of I/O 7: interrupt
2931, 90, CPU burst
```

ISR at 160:

```
2662, 84, CPU burst
2746, 1, switch to kernel mode
2747, 160, context saved
2907, 1, find vector 7 in memory position 0x000E
2908, 1, load address 0X00BD into the PC
2909, 152, execute I/O
3061, 1, execute IRET
3062, 1, switch out of kernel mode
3063, 10, context reloaded
3073, 38, CPU burst
3111, 20, end of I/O 7: interrupt
3131, 90, CPU burst
```

ISR time is 200:

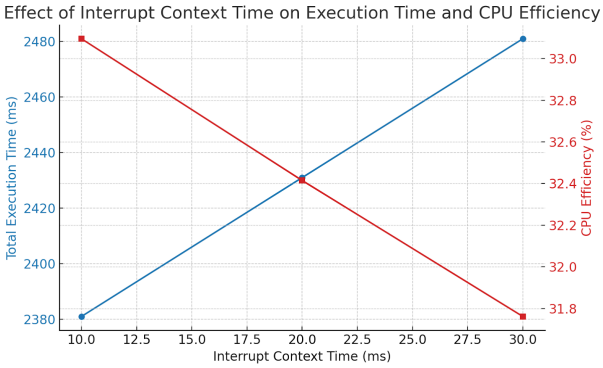
```
2822, 84, CPU burst
2906, 1, switch to kernel mode
2907, 200, context saved
3107, 1, find vector 7 in memory position 0x000E
3108, 1, load address 0X00BD into the PC
3109, 152, execute I/O
3261, 1, execute IRET
3262, 1, switch out of kernel mode
3263, 10, context reloaded
3273, 38, CPU burst
3311, 20, end of I/O 7: interrupt
3331, 90, CPU burst
```

Increasing the ISR activity team creates more delays when processing multiple interrupts as one interrupt takes longer to execute. This is shown through the screenshots as the execution time increases more significantly each time. This also further decreases the CPU efficiency as the interrupt handling overhead has increased. Theoretically, the longer the ISR takes, the longer it is kept in kernel mode which increases the risk of the OS missing other interrupts which could cause issues. So ISR times should always be kept as short as possible.

**- How does the difference in speed of these steps affect the overall execution time of the process?**

To demonstrate this, we have graphed the results of the previous simulations regarding changing the context time and ISR time with the computed CPU efficiency. The CPU efficiency percentage was calculated by dividing the total CPU burst time by the total execution time then multiplying by 100 for each simulation. Since for all simulations, the input trace_1 file was used, the CPU burst time was consistent at 788 ms. This was calculated by adding all the CPU burst log times from the result file.

Here is the graphed results from changing the context time and ISR Activity:





The linear relationship can be seen that as the interrupt context time increases, the total execution time increases and CPU efficiency decreases. The relationship can be seen that as the ISR activity time increases, the total execution time increases and CPU efficiency decreases. In an OS, this is not ideal as this can lead to many issues, decrease efficiency and throughput. It is best to have the most useful OS by having the least required context and ISR activity time so that the CPU is more efficient.

**- What happens if we have addresses of 4 bytes instead of 2?**

To compare the difference of changing the addresses from 2 to 4 bytes, we shall compare the execution times before and after the change. The changes include the addresses stored in the vector tables, which will be converted all from 2 to 4 bytes and a delay implemented to simulate fetching 4 bytes instead of 2.

Through simulation we saw that the execution time takes longer due to the time taken to now process the larger address than before. The results are put in the github repository under output_files/report_results/changing_bytes.

**-What if we have a faster CPU?**

A faster CPU would reduce both CPU burst time and interrupt execution time, causing a substantially faster overall execution time. The interrupt delay itself would remain the same time, as would switching in and out of kernel mode and saving/accessing memory. The execution time would go down, but the CPU sleeping time would stay the same, and ultimately the system would be using the CPU for a smaller percentage of the overall time. It should be noted that a faster CPU would use more energy and be a more expensive computer.