SYSC 4001

Assignment 1
Design and Implementation of an Interrupts Simulator
Report

Student 1: George Tzemenakis, 101296691
Github handle: TheRealRisu (Risu)

Student 2: Sabari Mathiyalagan, 101296257
Github handle: SabariMathiyalagan (Sabari Mathiyalagan)

Github Repository:

https://github.com/SabariMathiyalagan/SYSC4001_A1.git

Analysis of Results:

Effect of Changing Save/Restore Context Time:

The effects of changing the save/restore context time can be seen between traces and their execution files, especially how long the entire interrupt boilerplate takes. Trace 6, Trace 7, and Trace 20 were all run with the same ISR time of 40ms, but the save/restore context times were varied by 10, 20, and 30 ms, respectively. This caused the entire interrupt boilerplate to also vary by intervals of 10ms as shown below:

```
51, 1, switch to kernel mode
52, 10, context saved
62, 1, find vector 14 in memory position 0x001C
63, 1, load address 0X0165 into the PC
64, 40, SYSCALL: run the ISR (device driver)
```

Execution 6

```
30, 1, switch to kernel mode
31, 20, context saved
51, 1, find vector 10 in memory position 0x0014
52, 1, load address 0X07B0 into the PC
53, 40, SYSCALL: run the ISR (device driver)
```

Execution 7

```
25, 1, switch to kernel mode
26, 30, context saved
56, 1, find vector 2 in memory position 0x0004
57, 1, load address 0X0695 into the PC
58, 40, SYSCALL: run the ISR (device driver)
```

Execution 20

As seen above, the time it takes for trace 6 to get ready to process the interrupt is (64 - 51) = 13 ms, while in trace 7 it is 23 ms, and in trace 20 it is 33 ms. Therefore, increasing the save/restore context time will increase the execution time of the overall process. Whenever there is a software interrupt, this effect can be seen and thus the more software interrupts a program has, the more of an effect the save/restore context time has on the execution length.

Effect of Varying ISR Activity Time:

Varying the ISR activity time affects the amount of time that the CPU is busy handling interrupt-related tasks, preventing normal program execution. In our simulation, since the “check for error” step in the ISR process takes the remaining amount of time required for the I/O device to complete its ISR, the total execution time of the entire program does not change with ISR activity time. Trace 6 and trace 16, trace 17, and trace 11 have the same save/restore context time, yet have increasing amounts of ISR activity time, starting at 40 in Trace 6, and increasing to 200 in trace 11. Reflected in their execution files, the increase of ISR time causes each

interrupt (both SYSCALL and END_IO) to occupy the CPU for longer durations before control is returned to the main process. The device table reflects the average amount of time it takes for a device to complete its I/O Task. In some cases, if the ISR activity time is too large, the CPU will not have enough time to complete the ISR and transfer all data from device to memory.

In execution 6, with an ISR activity time of 40ms, we can see that the time it takes for the CPU to run the ISR and transfer all data to memory is 40ms each. The remaining amount of time is spent checking for errors, before continuing. In this case, the CPU has enough time to fully run the ISR time before continuing normal execution.

```
62, 1, find vector 14 in memory position 0x001C
63, 1, load address 0X0165 into the PC
64, 40, SYSCALL: run the ISR (device driver)
104, 40, transfer data from device to memory
144, 376, check for errors
```

(Execution 6)

In execution 17, with an ISR activity time of 120ms, we can see that the time it takes for the CPU to run the ISR and transfer all data to memory is 120ms each. In this example, there is not enough time for the CPU to complete the full data transfer from device to memory. As a result the process ends early with no time remaining for error checking. This is undesirable, as incomplete data transfers can lead to data loss and other problems.

```
46, 1, find vector 5 in memory position 0x000A
47, 1, load address 0X048B into the PC
48, 120, SYSCALL: run the ISR (device driver)
168, 91, transfer data from device to memory
259, 0, check for errors
```

(Execution 17)

<u>Effect due to Addresses of 4 Bytes Vs 2 Bytes:</u>

Varying the vector_size in the program has no impact on the overall execution time of the program, but rather on which memory address the CPU fetches the ISR from. This can be seen in the difference between trace 20 and trace 23, as shown below:

```
56, 1, find vector 2 in memory position 0x0004
```

Execution 20

```
56, 1, find vector 2 in memory position 0x0008
```

Execution 23

In both of these, the traces had a software interrupt regarding device 2; however, both executions attempted to fetch the device from two different locations. This is because trace 20 was executed with a vector size of 2, while trace 23 was executed with a vector size of 4 and thus the CPU searched for vectors at memory positions 0x0004 and 0x0008, respectively.

Effect due to Faster and Slower CPU:

When comparing the effects of fast and slow CPU burst times, the difference in processing speed only impacts how quickly the CPU completes normal execution tasks, and how no effect on the time it takes to complete interrupt handling, as ISR and device operations run independently from normal execution tasks. This is shown through two examples, trace 14, which had a smaller (faster) CPU burst time, and trace 25, which had a larger (slower) CPU burst time. Since the values of the CPU burst time in trace 14 were smaller than the values in trace 25, the CPU during execution 14 spent less total time on user-level processing between interrupts, while the CPU during execution 25 spent longer on the same processes.

```
551, 185, check for errors
736, 1, IRET
737, 20, CPU Burst
757, 1, switch to kernel mode
```

(Execution 14)

```
1259, 572, check for errors
1831, 1, IRET
1832, 60, CPU Burst
1892, 1, switch to kernel mode
```

(Execution 25)