SYSC4001 Assignment 1 Group 10

Student 1: James Gohl 101299043
Student 2: Magdi Hajjaj 101299841
Date: 03-10-2025

Part II – Design and Implementation of an Interrupt Simulator:

For part 2 of assignment 1, the objective was to build a simulator that models the interrupt process of a computer. The goal of this simulator is to be used to analyze the performance of different parts of the interrupt process. For the assignment, the simulator was built based on the template code in C++ and various computer interrupt simulations were performed, and the results were analyzed.

For the creation of the simulator, some assumptions had to be made. This was due to ambiguity in the instructions as well as the provided examples. Both TAs in our lab section agreed and said that stating the process we used to create the simulator based on the ambiguity in the instructions was the correct way to proceed. These are explained in the next section, which explains how the simulator was built.

The CPP simulator works by taking input from a trace file, which contains various tasks a computer may perform. The tasks used in this simulator are system calls, end of I/O interrupt and CPU bursts. The CPU bursts in the trace file contain the length of the burst, while the number associated with the system calls and the end I/O interrupt corresponds to a device number. The simulator works by reading each line from the trace file and outputs the computer's process for the line, and calculates the total time. For a CPU burst, the simulator simply adds the time of the burst to the total and outputs the time at which the CPU burst occurred. For a system call or an end I/O interrupt, the simulator outputs the entire process and its length to the output file. It shows the length of each step, such as switching to kernel mode, saving context, finding vector, loading the address, as well as specific tasks if it is a system call or an end IO, followed by an IRET instruction. The length of the specific tasks totals the length of the ISR determined by the time listed in the device table. The other lengths for the generic task, such as switching to kernel mode, are standardized. Although the lab manual says to choose a time for each ISR activity (such as 40ms), it also states that the ISR should take as long as specified in the device table. The device table length made more sense as each device in real life would have a varying ISR time. The Tas in our lab section both agreed, so this was the assumption made.

After the simulation was built, various simulation tests were performed, and the results were analyzed. The various tests and their results are as shown:

1. **Testing the effects of varying the time taken to save the context**

To test the effects of different lengths of context saving, 3 varying trace files were each simulated 3 separate times. The first time with a context save time of 10ms, then 20ms and finally 30ms. When analyzing the total completion times between the 3 different

context times, it was found that the total time of the simulation increased linearly based on the proportions of instructions that were interrupted. For example, trace 1 with 10ms had a total time of 3202ms, while with 20ms it had 3302ms and with 30ms 3402ms. With each increase of 10 m/s, the total runtime of the trace increased by 100ms. This linear increase was the same for all traces tested. Although 30ms is 3 times larger than 10ms, it did not increase the total time by 3 times. This is because the total percentage of time the computer is being used is significantly more than the time it is saving contexts. These results make sense as a time increase of a computer task should only proportionally increase by the number of times the computer performs said task. However, if the time it takes to save the context was significantly longer, it would have had a more noticeable effect on the overall system. However, this is not realistic as CPU bursts, I/O and system call tasks tend to be more time-consuming than basic tasks.

2. **Testing the effects of a fast CPU vs. a slow CPU:**

To test the effects of a fast or slow CPU, various simulations were performed. To simulate a fast CPU, the trace file contained only CPU bursts of 5 ms, while for the slow CPU, bursts of 50 ms (10 times longer) were used. Each interrupt in the simulation was kept the same between the slow and the fast CPU simulations. The results of this simulation showed that even though the CPU bursts were slower by ten times in the one trace file, the total time was hardly even different between the slow and the fast CPU. This was because the device times used for this simulation were way larger than the amount of time for the CPU bursts. For each system call, there is a corresponding end I/O interrupt. This means that the device time occurs twice per the time of one CPU burst. Just like with the differences in context save times, again this simulation demonstrates that the total used by the computer is only as much as the total of tasks performed and their length. Even if a task takes a long time, its contribution to the system's total time can be minuscule if shorter or longer tasks occur at higher frequencies.

3. **Testing the effects of a longer ISR time:**

To test the effects of increasing or decreasing the length of an ISR, simulations were performed on trace files of the same length. One trace file contained only devices with an ISR time of 63ms, and one with an ISR time of 1000ms. The results showed that the longer ISR times had a significant effect on the total time. The one simulation had a difference in total time of around 700%. No matter how fast a CPU is, its total time to execute programs can only be as fast as its I/O devices. This is only the case for computers that can execute only one task at a time, as in this simulator. This is not the

case in today's computers, which would be executing other tasks while waiting for an I/O device to interrupt.

4. Conclusion:

In conclusion, the simulator has demonstrated that, in a computer, the instructions that occur at greater frequency or of significant length have the most noticeable effect on total computer run time. This was demonstrated in the simulations explained in 1-3 as well as additional simulations, which all came to the same conclusions. A computer executing one program at a time can only be as fast as each program and its I/O devices. The question of increasing the size of the vector table's address size was also explored. It was determined that this will not have a noticeable effect on the run time, provided the bus is the correct width for the increase in size. This is because the CPU can simply fetch each bit in parallel. However, if the bus is not the correct size, it will slow down the total time as it will have to request data from the vector table multiple times.

GitHub Repository Link:
https://github.com/JamesGohl/SYSC4001_A1