SYSC 4001 - OPERATING SYSTEMS

ASSIGNMENT 1

GROUP - 22

Student 1 - **Peter Tourkoyiannis**
# - 101262960
Student 2 - **Akshwin Sellathurai**
# - 101225500

Part 1

Question A

Hardware: A signal is received by the cpu from the interrupt event and the signal is generated. The cpu then checks if interrupts have been enabled. A signal is then sent back to acknowledge the initial interrupt signal. The cpu then stores the state of the current program into memory, which will be retrieved when the interrupt is complete. The signal is compared with a vector table to match with an address to point to the ISR routine (both hardware/software). This is then loaded and run by the cpu. Upon completion, the interrupt signal/flag is turned off and the previous program running in loaded back from memory and continues executing.
Software: Switches to kernel mode if interrupts enabled. It then stores the current program/state the cpu is currently in memory. The vector table is stored in memory (both hardware/software). It retrieves the old program previously running to return the cpu to the previous state before the interrupt. If the interrupt is of lower priority than the current program running, it is then scheduled to run the ISR but does not run right away.

Question B

A system call is an entry point that allows the user to make a request for a service from the kernel. Some examples are open() which opens a file, read() to read a file, write to write a file and fork() this creates a new process. System calls actually use interrupts be accomplished and they both entail going from user to the kernel mode. The interrupt hardware mechanism in system calls work by initially having a user issue an interrupt. The hardware then takes the context from the interrupt and gives over control to the system call handler. The OS executes the file access. When done the control is given back to the user. 1.

Question C-i

All the steps required to check if the printer is ok would be firstly having the driver check the printers status register. Hardware sends back status bits to represent the status of the printer. Driver takes in the bits returned. Finally for the example, if the status bit returned is “ready” then the printer can continue. If the return is

Question C-ii

The printer must first check if it is ok (i.e no errors such as jams, no ink). Then the line feed must occur. A signal will be sent to turn the motor to advance the paper up one line. When this is completed, a signal would be sent back either confirming its completion or an error. If all clear, then the signal for the carriage return would be sent. The motor to return the carriage to the right most position is then initiated, then a signal is sent back. If the signal is ok, then the driver can then continue outputting more characters through that printer.

Question D

In a batch operating system, off-line operation means jobs are first prepared on an input medium like punched cards outside the main computer and then processed automatically in groups ("batches") without user interaction. This approach reduces CPU idle time and allows large volumes of work to be handled efficiently with minimal operator intervention, but it also has drawbacks being there's no immediate feedback or interaction, so errors can only be corrected after a batch finishes, and turnaround times are generally slower and less flexible for tasks needing real-time input.

Question E-i

If a programmer writes a driver and forgets to check for the $ at the start of each card, the system would treat control cards (like `$FORTRAN`, `$SLOAD`, `$RUN`) as normal data cards. This means it would try to process them as part of the program or data instead of as commands, this most likely causes errors or unpredictable behavior. To prevent this is to make the input routine explicitly check for a leading $ on each card before processing it, and then send those cards to the job control interpreter instead of the program input stream

Question E-ii

While a program is executing under `$RUN`, a new card appears with `$SEND` at the start, the operating system should not just feed it to the running program as data. Instead, it should intercept the card because the "$" marks it as a control card. The OS then decides whether to queue it for after the current job or stop the current job and process the new command, depending on its scheduling policy. In other words, the OS must always monitor the input stream for control cards, even during execution, so that job control commands aren't misinterpreted as normal data.

Question F

Student 1: Switching between kernel mode and user mode is a privileged instruction done by the operating system to prevent the user from any accidental errors that could corrupt the system. Also to help with the security of the system. Accessing memory locations are privileged instructions as one user's program could overwrite another user's program or the operation system itself. This could also help with data security as only the operating system would know where data is stored.

Student 2: Enabling and disabling interrupts is a privileged instruction as they can be used to hog the system in the event that a malicious program is running. The same can be said with I/O instructions as it can lead to system instability if a file is stored incorrectly, at the wrong location. This can lead to programs (even the OS) to be overwritten and critical data is lost. **NOTE:** Similar to why memory locations are also a privileged instruction.

Question G

When the card `SLOAD TAPE1` is read, the Control Language Interpreter recognizes it as a control command and, through Job Sequencing, activates the loader routine. The loader uses the Device Driver for the tape drive to read the first file from TAPE1 into the User Area of main memory, with Interrupt Processing handling tape-read completions or errors. Once the executable is fully loaded, the system returns to the interpreter to read the next card. When the `SRUN` card appears, the interpreter recognizes it as a request to execute the loaded program and, again via Job Sequencing, sets up the execution environment and dispatches the CPU to the program's starting address. From this point on, the program runs out of main memory, using the system's device drivers for any I/O and relying on interrupt processing for event handling during execution. This sequence ensures that the program is first correctly loaded from tape and then started under the batch operating system in an orderly, automated fashion.

Part 2

Introduction

The goal of this assignment is to simulate and collect and analyze data from how interrupts affect general scheduling. The tests simulate different save times and ISR activity times from 10,20,30 and 40, 60, 80, 100, 120, 160, 200 respectively.

Table 1: Data collected from the saving times and the ISR chunk times.

Test Number	Save Time (ms)	ISR Chunk Time (ms)	Total Time (ms)
1	10	40	30965
2	10	60	31125
3	10	80	31405
4	10	100	31765
5	10	120	32085
6	10	160	32205
7	10	200	32965
8	20	40	31565
9	20	60	31725
10	20	80	32005
11	20	100	32365
12	20	120	32685
13	20	160	32805
14	20	200	33565
15	30	40	32165
16	30	60	32325
17	30	80	32605
18	30	100	32965
19	30	120	33285
20	30	160	33405
21	30	200	33405

Github Link To Code

https://github.com/Peter-Tourkoyiannis/SYSC4001_A1.git

Observations

After observing the data, it is clear that increasing the overhead and ISR time increases the total time. It seems that increasing the overhead increases it uniformly while increasing the ISR time increases it in an exponential manner.

Increasing the address bytes from 2 to 4 should not change the time in the system created as it is dealt with in the boilerplate. In practice, assuming the system can handle a larger address, would take longer as the hardware would need to take more time to get to the address (Think bigger but same speed memory etc.) It should be noted though that it would not increase the time significantly.

A faster CPU would allow for more instructions to pass through making it faster. However, this would require that it is not constantly waiting on external factors, such as I/O, as that is where the main bottleneck would be.

Note: The way the code was originally set up (NOT THE FINAL VERSION), I had used a function to stop the ISR when it was done (I believe the min, can't remember), even when the ISR chunk was not completed yet. However, the final version was updated to run the entire chunk even if the SYSCALL duration had expired. This was done as the same value was outputted for each save time. This led to the chuck time having no impact, hence the change.

Conclusion

The program successfully processes from an input file, simulate SYSCALLS, I/O and CPU Burst cycles. With the data we can prove that the more overhead and the longer chunk times result in a more inefficient system