SYSC 4001

Assignment 1

John Patterson 101265900

Bhavesh Bhatia 101187314

Date: 2025-10-05

https://github.com/JPatt03/SYSC4001_A1

Part I – Concepts

1.a.)
First an external signal occurs such as some event like moving a mouse or pressing a key on a keyboard which are hardware components and will send a signal to the processor. It finishes the current instruction and then is paused and saves the current state to return to this point later. The interrupt service routine is going to happen which will call the interrupt specific handler which will handle that specific interrupt to handle that specific event in the software. After the interrupt service routine is complete the processor will then return to the point where it was paused and resume.

1.b.)

What is a System Call?

System Calls are a method in which software can interface with features and methods provided by the operating system. These are features that are not accessible in the “user mode” of the system, and therefore it must switch to “kernel mode” to utilize them. Three examples of system calls are open() ReadFile() and write()**.** System Calls utilize something called a “Software Interrupt”. It uses something referred to as a “trap”, which goes to a specific place in the interrupt vector. This will then trigger a service routine within the operating system which will switch the system to kernel mode. Using a parameter in the interrupt instruction provided, the kernel is able to tell what system call has been called. It will proceed to execute the requested system call.

1.c.)

1.c.i.)

The printer must check if the printer currently has power, without power it cannot print, check if the printer is currently loaded with paper and ink, check if the printer is currently printing, and lastly check if the printer is jammed

1.c.ii.)

The printer will return the motor to its initial position and Move the carriage to the next line

1.d.)

The off-line operation works by a card reader that will read data or a tape drive reads this data and writes it to a magnetic tape to store this data. These jobs are batched together and loaded in the computer's main memory. The OS will execute the jobs that are batched together in an order after the output is written onto the magnetic tape. An advantage is that the cpu is always being used and jobs are run in batches.A disadvantage is if there was an error you would have to wait for the whole batch to run before going back to fix this.

1.e.)

1.e.i)

If the programmer forgot to parse the “$”, all cards with “$” on them would be treated as normal code and be loaded by the loader into memory. This would cause compilation errors, and unintended outputs from the program. This can be prevented by having the drivers already stored in memory and preventing programmers from having access to the commands needed to run the driver using privileged instructions.

1.e.ii)

The program would immediately stop executing. The operating system should find and load the next job and begin execution.

1.f.)

Assuming that this is asking about the privileged instructions on a higher level, not the instruction names themselves, as those were not listed in the slides or textbook.

Halt Instructions

Halting the CPU puts the CPU into an idle low-power state, which is useful for when the CPU has no work to do. This instruction is privileged because if you were to be able to halt the CPU outside of kernel mode you risk halting it when it shouldn't be halted. If this were to happen a user may halt the system using software without also providing means for it to start its fetch execute cycle again. This could lead to software causing the entire computer to freeze.

I/O Instructions

I/O Instructions provide a way for the system to interface with I/O Devices. This instruction is privileged because if users could freely access I/O Instructions that could lead to accidentally overwriting memory where it shouldn't be overwritten. This could be drivers, data currently being accessed by another piece of software, or other files crucial for the operating system to function as required

Context Switching

This is used when multitasking is needed to share the CPU between multiple processes. The scheduler gives priority to these processes and context switching will execute the highest priority by stopping a process and starting the higher priority one. After that the next highest will be run which is where the switching is done so the current process is paused and will continue where it left off from before.

Timer Management

This instruction can bypass and interfere with pausing tasks currently running if they are hogging the CPU so other tasks can run when required. If the user was able to modify or interfere at the wrong instance this will cause errors or delays that's why it can only be run in kernel mode and not user mode.

1.g.)

Interrupt Processing: Handles and interprets asynchronous events that are generated by I/O devices.

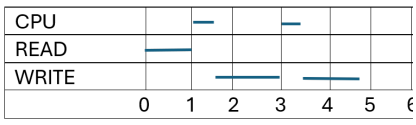
Device Drivers: Contains software that allows the system to interface with I/O devices.
Job Sequencing: Manages the flow of jobs and the major steps that must be taken in order to run a job.
Control Language Interpreter: Parses cards that contain "$" commands. These commands are in control language, and tells the system what actions must be done in order to execute the program provided.

Sequence of events:
The control language interpreter reads the $LOAD TAPE1 card, the control language interpreter parses the command and sends a signal to the job sequencer, the job sequencer calls the loader and sends a signal to the device drivers to execute the driver needed for loading the device driver executes the software required to load data from the input device (tape reader, punch card reader, etc.), the Interrupt Processor receives an interrupt signal from the input device indicating that it has loaded the entire program and may begin execution.

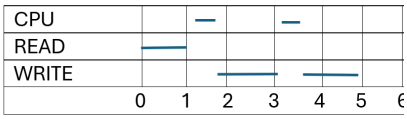
1.h.)

Timed I/O

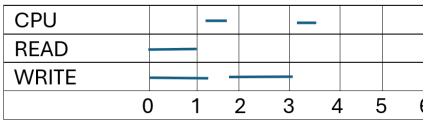
Polling



Interrupt



Interrupt with buffer



Timed I/O is the slowest due to the wasted time from the error. Polling is the 2nd fastest for this software. It's essentially the same as the timed I/O except it does not waste time. The interrupt without the buffer is slower than polling due to interrupt latency. The interrupt with the buffer is the fastest due to being able to read and write simultaneously.

Part II – Design and Implementation of an Interrupts

Simulator

During this assignment, multiple tests were done using trace files. The outputs uploaded are from tests using the activity durations provided. Additional tests were done varying the durations.

Save/Restore Time

Firstly, when varying the save/restore context time, this would increase the execution time of the trace file according to the table list below:

Save/Restore Time (ms)	Execution time increase per interrupt.
10	0
20	20
40	40

This implies that increasing the save/restore time applies a linear increase to the execution time.

ISR Time

As stated in a comment in interrupts.cpp, we could not figure out a way to have all ISR activities take 40ms due to the provided times for each device given in the device table including values that were not multiples of 40. However, in an earnest attempt to complete this portion of the report, we will instead be attempting to predict what this would do.

If every process in the ISR took, for example, twice the amount of time, the total time the execution takes should increase by an amount that is described with the formula below.

$$f(t) = (t - 40) * n$$

For all t > 40

Where f(t) is the total time taken per interrupt in ms, n is the amount of activities done by the ISR, and t is the time taken per interrupt activity. If the ISR takes too long to execute this will lead to lower efficiency as the CPU waits for the ISR.

What if we have 4 byte addresses

With 4 byte addresses, we have the capability to have more I/O devices.

What if we have a faster CPU?

Every instance of CPU execution would be faster.

Interpretation

The trace files that were tested seemed to on average spent slightly more time on interrupts than the CPU execution. Increasing the time to save/restore context or increasing the time of each ISR activity makes this much more of an issue.