SYSC 4001A L3
Assignment 1

Group 28
Caroline Twelves 101301400
Jules Wong 101309829

Part 1 - Concepts

a) [0.1 mark] Explain, in detail, the complete Interrupt mechanism, starting from an external signal until completion. Differentiate clearly what part of the process is carried out by hardware components, and what is done by software. **[Student 1]**

An operating system is driven by interrupts. When a hardware device detects a condition for an interrupt to occur, it sends an interrupt signal to the CPU. The signal goes through the Interrupt Controller, a piece of software that decides priority and notifies the CPU. The Program Counter and other registers are stored to keep the CPU's information safe, and the device which caused the interrupt is identified through the drivers. The device driver will also clear the interrupt signals after double checking what happened. The job sequence will then update its queue and the saved registers will be restored.

b) [0.1 marks] Explain, in detail, what a System Call is, give at least three examples of known system calls. Additionally, explain how system Calls are related to Interrupts and explain how the Interrupt hardware mechanism is used to implement System Calls. **[Student 2]**

A system call is an instruction the user can give to the computer, software to software, to ask it to perform a task, called a system call. A system call is a call to an interrupt that the kernel must execute. The kernel must make this call because it involves accessing to protect the memory. Examples of system calls are; read(), write() and print(text).

The ISR is in kernel mode and the interrupt is called from user mode, thus the CPU is also in user mode. In this case the hardware of the interrupt must change the CPU to kernel mode to allow it the permission to execute the interrupt.

c) [0.1 marks] In class, we showed a simple pseudocode of an output driver for a printer. This driver included two generic statements:
i. check if the printer is OK
ii. print (LF, CR)
Discuss in detail all the steps that the printer must carry out for each of these two items (**Student 1** should submit answer ii., and **Student 2** should submit answer i.).

i. Use a sensor to check if the printer is done printing
Check if you have paper
If so, print the next letter

ii. The steps that the printer must carry out for each item in (LF, CR) come from typewriters.
LF: Line Feed, advances by one line down the page once the current line is done printing.
CR: Carriage Return, returns to the original side of the page once the line is done printing, e.g. if a line is printed from left to right, the printer must return to the other side of the page.
The printer has to verify it is done printing its content for the line and continue to the LF and CR steps.

d) [0.4 marks] Explain briefly how the off-line operation works in batch OS. Discuss advantages and disadvantages of this approach.

Offline operation was used when tapes were still in action. A human operator would load a tape to something acting as a buffer/driver to load the tapes, which another operator then loaded into what acted as the CPU, which executed the commands of the tapes. These were then loaded by an operator into another buffer/driver which would print the output, and could be stored by another operator .

Because the tasks for the machine are written off-line aka not on the actual computer itself, this is called an off-line operation.

Advantages to this approach is that a human would be able to check manually if the CPU/ any of the machines were in use before loading the tapes, thus eliminating needing to sense if any tapes were in it. Additionally, it works as a cost saving measure as the computer didn't need to be online for as long and programmers could prepare the cards in advance, maximizing its operation time.

Disadvantages to this method include over-reliance on human operators. If an operator were elsewhere, such as at lunch or talking to someone, time would be lost on them loading and unloading the tapes. Another disadvantage was that the CPU would only be able to execute one thing at a time. The system also makes it difficult to test for issues, as another version of the card would have to be written and added to the queue, and optimizing operation time is no longer necessary in the modern day thanks to advances in computing technology.

For instance, the $FORTRAN card would indicate to start executing the FORTRAN compiler and compile the program in the cards and generate an executable. $LOAD loads the executable, and $RUN starts the execution.

i. [0.2 marks] Explain what would happen if a programmer wrote a driver and forgot to parse the "$" in the cards read. How do we prevent that error?

ii. [0.2 marks] Explain what would happen if, in the middle of the execution of the program (i.e., after executing the program using $RUN), we have a card that has the text "$END" at the beginning of the card. What should the Operating System do in that case?

1. If a programmer wrote a driver and forgot to parse the $ in the cards read, the program the driver is interfacing with will not be able to function normally without being able to interpret privileged commands. The program would not register the card as a new driver, and simply keep executing it as if it were the previous card it received. This would break the program as it would be running one type of execution where it should be running another, therefore printing gibberish. To prevent this a $ must be placed before the cards read. This could be fixed by implementing better checks on drivers, making it a mandatory requirement to parse the $ in the cards read.
2. If a card has the text $END at the beginning of the card, this is an example of the program attempting to use a privileged instruction. This mistake would cause the system to break. The system would execute the card $END while intending to run data, causing the program to print out gibberish. The operating system should be shut off, and the cards reorganized.

f) [0.2 marks] Write examples of four privileged instructions and explain what they do and why they are privileged (**each student should submit an answer for two instructions, separately, by the first deadline**).

Privileged instructions are specialized commands running at the machine level and can only be executed by the monitor. They are called privileged because they are used to allocate resources in the system, run program files and manage the behaviour of the system. These are put in place to prevent any instructions being sent that will fundamentally change anything about the programs. An interrupt should occur if a program tries to run a privileged instruction.

$RUN: initiates the execution of the program loaded into the user area of main memory
$END: signals the end and the ability to release memory or other resources previously allocated
$LOAD: signals the act of a card being loaded

You must provide a detailed analysis of the execution sequence triggered by the two cards, clearly identifying the routines illustrated in the figure above. Your explanation should specify which routines are executed, the order in which they occur, the timing of each, and their respective functions—step by step. In your response, include the following:

i. A clear identification and description of the routines involved, with direct reference to the figure.

ii. A detailed explanation of the execution order and how the routines interact.

iii. A step-by-step breakdown of what each routine performs during its execution.

First, the control language interpreter interprets $LOAD and sends this privileged instruction to call the loading routine. The loader will request the tape via the device drivers, which will in turn issue the I/O command to the tape device. The interrupt processor will suspend the CPU and freeze the Program Counter while waiting and resume when the tape read is done. The program is then copied into the User area of main memory. The job sequencing will then ensure that the next command, $RUN, is interpreted next. The control language interpreter will again interpret $RUN, and initiate the execution of the program. In summary, these four components each have specific roles. The Control Language Interpreter interprets the instructions $LOAD and $RUN and determines which actions to take next. The Device Drivers handle communication with the hardware to fetch the program in Tape 1 into memory. Job Sequencing ensures that after the tape is loaded, $RUN will be processed afterwards. The Interrupt Processing will handle the interrupts at crucial moments, between instructions, I/O signals, etc.

h) [0.3 marks] Consider the following program:

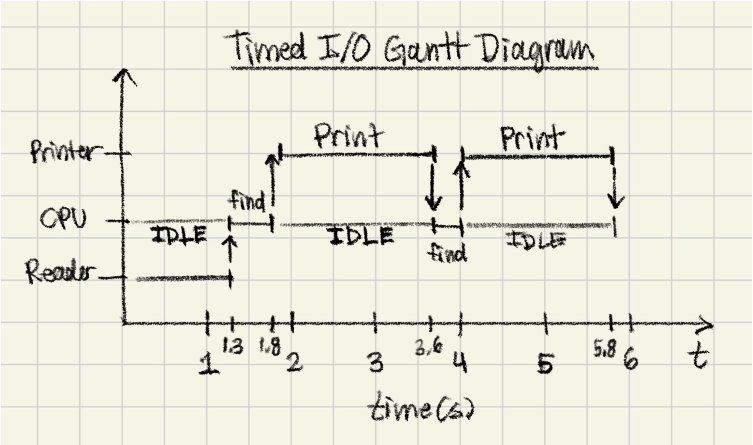
```
Loop 284 times {
        x = read_card();
        name = find_student_Last_Name (x);  // 0.5s
        print(name, printer);
        GPA = find_student_marks_and_average(x); // 0.4s
        print(GPA, printer);
}
```

Reading a card takes 1 second, printing anything takes 1.5 seconds. When using basic timing I/O, we add an error of 30% for card reading and 20% for printing. Interrupt latency is 0.1 seconds.

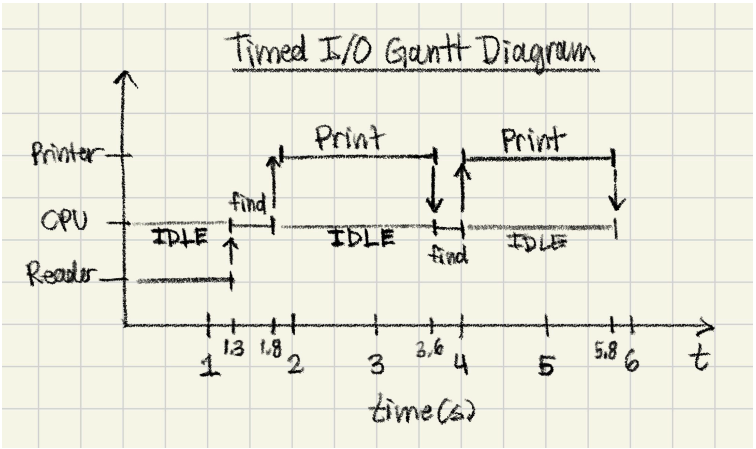
For each of the following cases: create a Gantt diagram which includes all actions described above as well as the times when the CPU is busy/not busy, calculate the time for one cycle and the time for entire program execution, and finally briefly discuss the results obtained.

i) Timed I/O
ii) Polling
iii) Interrupts
iv) Interrupts + Buffering (Consider the buffer is big enough to hold one input or one output)

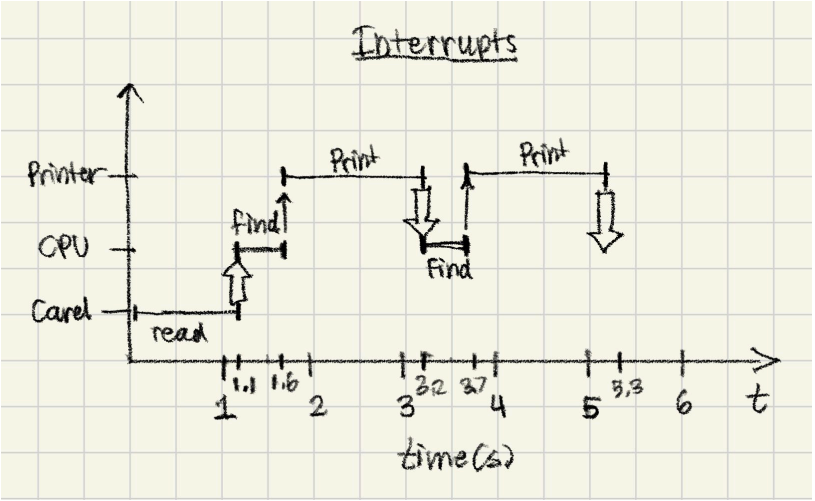
1. Time for one cycle in Timed I/O is 5.8 seconds. Total time is 1647.2 seconds



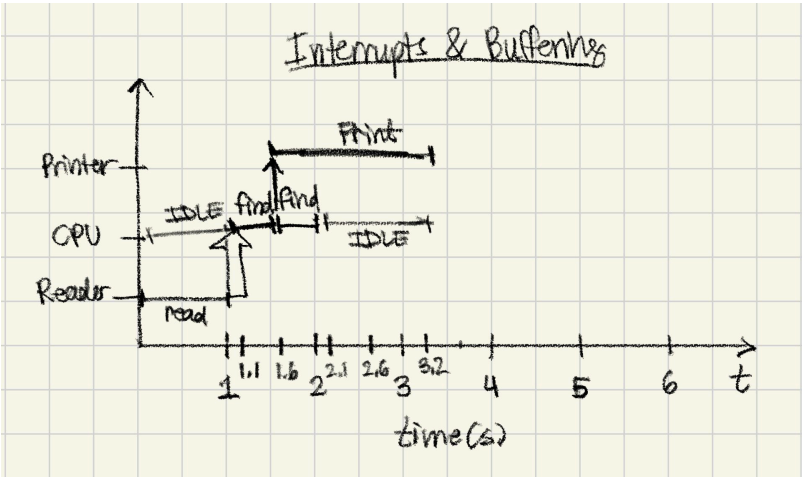
2. Time for one cycle in Polling is 5.0 seconds. Total time is 1420 seconds



3. Time for one cycle on interrupt is 5.3 seconds. Total time is 1505.2 seconds



4. Time for one cycle on interrupt & buffer is 3.2 seconds. Total time is 908.8 seconds



Part 2 - Simulations

[Link to Code Repository](https://github.com/carolinetwelves/SYSC4001_A1/) (https://github.com/carolinetwelves/SYSC4001_A1/)

The test cases we prepared involve varying the save/restore context time, as well as varying the time of the interrupt service routines. The simulations run were organized as follows:

Table 1: Simulation Test Case Numbering Guide

ISR/SRC	10	20	30
40	#10-40	#20-40	#30-40
67	#10-67	#20-67	#30-67
100	#10-100	#20-100	#30-100
120	#10-120	#20-120	#30-120
150	#10-150	#20-150	#30-150
200	#10-200	#20-200	#30-200

In the resulting files from the simulation, various versions of execution.txt, there were 10 different types of activity or actions recorded.

Table 2: Types and Names of Actions

Actions Constituting Useful CPU Time	CPU Burst End I/O Call Device Driver ISR Activity
Other Actions	Switch Back to User Mode Switch to Kernel Mode Context Saved Find Vector Load Address IRET

Keeping in mind these classifications, the results of the simulation were as follows:

Table 3: Simulation Test Cases Results

Test Case #	SRC Time	Total ISR Time	Total Useful CPU Time	Total Time	% Usefulness
#10-40	300	15251	32274	32724	98.62
#20-40	600	15251	32274	33024	97.73
#30-40	900	15251	32274	33324	96.85
#10-67	300	16061	33084	33534	98.66
#20-67	600	16061	33084	33834	97.78
#30-67	900	16061	33084	34134	96.92
#10-100	300	17051	34074	34524	98.70
#20-100	600	17051	34074	34824	97.85
#30-100	900	17051	34074	35124	97.01
#10-120	300	17651	34674	35124	98.72
#20-120	600	17651	34674	35424	97.88
#30-120	900	17651	34674	35724	97.06
#10-150	300	18551	35574	36024	98.75
#20-150	600	18551	35574	36324	97.94
#30-150	900	18551	35574	36624	97.13
#10-200	300	20051	37074	37524	98.80
#20-200	600	20051	37074	37824	98.02
#30-200	900	20051	37074	38124	97.24

Changing the save/restore context time to a certain value makes the CPU run for that specified number of milliseconds whenever it switches into kernel mode. Changing the time of the interrupt service routines increases the amount of time it takes to call the device driver.

As seen in the table, the SRC times naturally change when the SRC time value varies. This is unaffected by changing the ISR time, as there are only 3 values for SRC time shown in Table 2 that correspond to the 3 test values. It is only logical that they would not have any effect. The % usefulness, or percentage of time the simulation is running where useful actions are occurring goes down across trials as the SRC value increases. This is also expected as the Context Saved actions are not part of our useful actions classification as seen in Table 2.

Specifically, the total time taken to run actions labelled as “Call Device Driver” is directly related to the ISR time. This again makes sense as this action is part of the Interrupt Service Routines. The total times for the CPU burst, END I/O, find, load address, IRET as well as switching back to user mode as well as switching back to kernel mode, were all constant across all the simulation cases.