Carleton University

Department of Systems and Computer Engineering

SYSC 4001 Operating Systems Fall 2025

Assignment 1

This assignment must be completed in **teams of two students** and submitted **in two stages**, each with its own deadline: an initial **individual** submission followed by a **final** collaborative submission. In the first stage, one student is responsible for Exercise I.a), the other for Exercise I.b), and both must individually complete portions of Exercises I.c) and I.f). Any student who fails to submit their individual portion or scores below 50% will receive a **zero for the entire assignment**. For the final submission, teams must submit the **complete** assignment, incorporating both individual and collaborative components. The programming section must be completed using the Pair Programming technique, and the final grade will be calculated as the sum of all assignment parts.

YOU ARE REQUIRED TO FORM GROUPS ON BRIGHTSPACE BEFORE THE SUBMISSION OF THE ASSIGNMENT. To form groups, go to "Tools"> "Groups". Scroll through the list of group options available. The group would be "<Lab section> - Assignment 1". The "Lab Section" will be your respective to the section. The deadline to form groups is **22nd September**. Brightspace will auto-assign your groups after this deadline. You will not be permitted to change groups after this deadline for **ANY** reason.

More information can be found here:
https://carleton.ca/brightspace/students/viewing-groups-and-using-groups-locker/

Please submit the assignment using the electronic submission on Brightspace. The submission process will be closed at the deadline. No assignments will be accepted via email.

Part I – Concepts [2 marks].

Answer the following questions (from Chapter 1, 2 Silberschatz + lectures, marks in brackets)

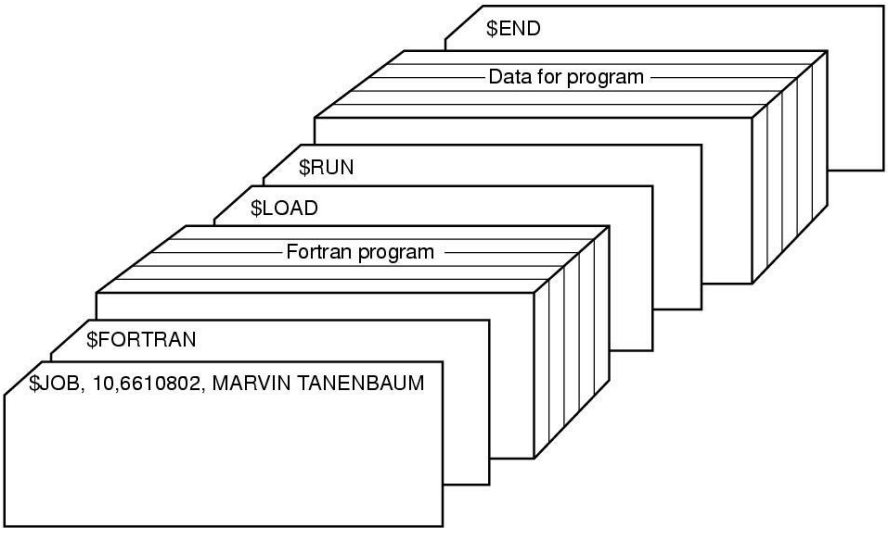
a) [0.1 mark] Explain, in detail, the complete Interrupt mechanism, starting from an external signal until completion. Differentiate clearly what part of the process is carried out by hardware components, and what is done by software. **[Student 1]**

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]**

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.).

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

e) [0.4 mark] Batch Operating Systems used special cards to automate processing and to identify the jobs to be done. A new job started by using a special card that contained a command, starting with $, like:



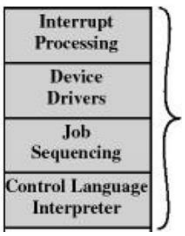
For instance, the $FORTRAN card would indicate to start executing the FORTRAN compiler and compile the program in the cards and generate an executable. $LOAD loads the executable, and $RUN starts the execution.

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?

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**).

g) [0.4 marks] A simple Batch OS includes the four components discussed in class:



Suppose that you have to run a program whose executable is stored in a tape. The command $LOAD TAPE1: will activate the *loader* and will load the first file found in TAPE1: into main memory (the executable is stored in the User Area of main memory). The $RUN card will start the execution of the program.

Explain what will happen when you have the two cards below in your deck, one after the other:

$LOAD TAPE1:

$RUN

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.

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)

Part II – [3 marks] Design and Implementation of an Interrupts Simulator

The objective of this section is to build a small simulator of an interrupt system, which could be used for performance analysis of different parts of the interrupt process. This simulator will also be used in Assignment 2 and 3.

A template code can be downloaded from here: https://github.com/sysc4001/Assignment-1
The template contains the following files:

- interrupts.cpp
- interrupts.hpp
- build.sh
- trace.txt
- vector_table.txt
- device_table.txt

You are expected to edit *interrupts.cpp* with your simulation code according to the instructions below. Go through *interrupts.hpp* to get an idea of all the helper functions you have at your disposal.
The *build.sh* is a script file. Run it (using `source build.sh`) to compile your code and to generate the executable in the 'bin' directory. *trace.txt* is an example input file we provide you. Feel free to make more input files as necessary to solve the assignment. *vector_table.txt* and *device_table.txt* are the vector table file and device table file (explained more below), feel free to play around with the values in these files to see how the output changes.

i) Input data

The simulator will receive, as an input, a trace of a single program. The system has one CPU and a couple of I/O devices. A pseudocode of the program is as follows:

```
main() {
initialize_variables();    //CPU burst (*1)
loop{  //Do the following in a loop
x = read_input();    //read 'x' from some input device (*2)
y = calculate(x);    //CPU burst; perform operations on x (*3)
print_output(y);     //Send 'y' to some output device
delete(y, x);        //CPU burst; free memory
       }
}
```

The trace information is stored in a table as follows:

Activity	Duration **OR** device number

The first column, **Activity,** refers to the type of activity carried out by the program (CPU burst, I/O etc.), and the second column is either **Duration** (in milliseconds) or **device number** of the device that caused the interrupt; the data is comma separated. For example, a table looks like this:

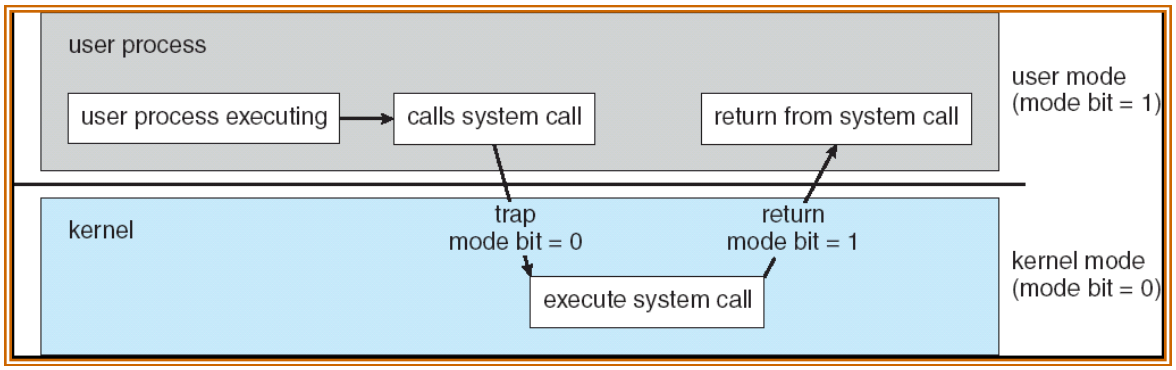
```
CPU, 50
SYSCALL, 7
END_IO, 7
CPU, 100
SYSCALL, 12
END_IO, 12
CPU, 20
...
```

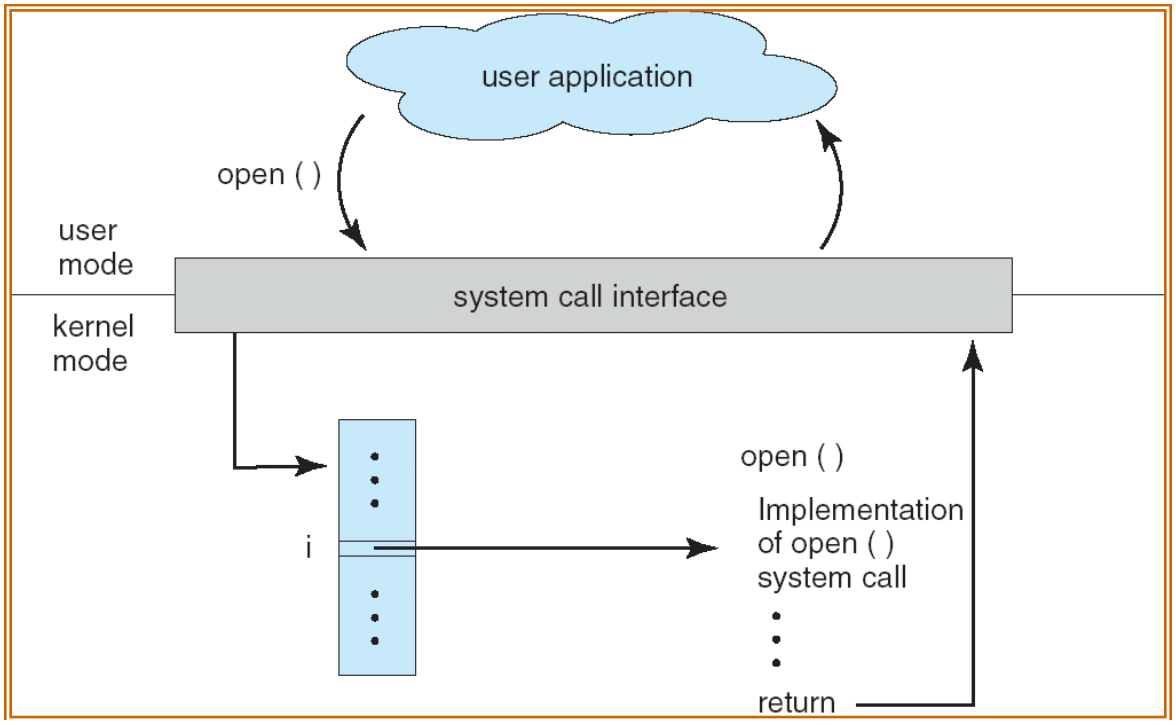
Explanation of the trace: First, we start with the initialization of variables (50 ms of CPU, i.e., *1 in the program above); then, we read from an input device 7; this also means that the system call is in position 7 of the vector table (SYSCALL, 7, called by *2 above). Then, the end of interrupt is called for device 7. The next line corresponds to *3, i.e., processing of the data by the *calculate* function. Finally, another SYSCALL for device 12 is called and so on. Each iteration of the loop, a new device may interrupt the system (this is not visible from the pseudo code, but can be seen in the trace)

ii) Simulation implementation

Using the information on the input file, you must simulate the advance of the program, logging every activity carried out by the program in an output file.

For system calls, the simulation should try to reproduce the behavior of this state diagram (from Silberschatz *et al.*). The complete detailed behavior of interrupts is found in the video posted on Brightspace.





Your code should use the trace table to simulate all the execution steps. Every step with its simulated time should be stored in an output file called *execution.txt*. This file should be submitted.

ASSUME THAT YOU ALWAYS HAVE AN I/O DEVICE AVAILABLE (that is, do not worry about waiting queues at the I/O devices: whenever you request an I/O, the I/O starts immediately).

Each vector is 2 bytes long.

You will need to include a Vector Table as follows (provided to you in *vector_table.txt*):

Interrupt(device) Number	**ISR Address**
...	
7	0x0E
...	
12	0x1B
...	
20	0x28
...	
22	0x16

The purpose of the interrupt vector table is to locate and load the address of the interrupt service routine (ISR) into the program counter (PC).

And a device table (provided to you in *device_table.txt*):

Device number	**Delay (ms)**
...	
7	110
...	
12	600
...	

The purpose of the device table is to give you the average time it would take for that device to complete its I/O task.

The simulation should include all the steps in the interrupt process, which should be in the output trace:

- Switch to/from kernel mode (time: negligible; use a duration of 1 ms)
- Save/restore context (say 10ms)
- Depending on the interrupt number, calculate where in memory the ISR start address is (time: negligible; use a duration of 1 ms)
- Get ISR address from vector table (time: negligible; use a duration of 1 ms)
- Execute ISR body. Include all the activities carried by the ISR (each activity: say 40ms
- Execute IRET (time: negligible; use a duration of 1 ms)

The *execution.txt* file should include the following information, comma separated:

Time of the event	Duration of the event	Event type

Example: The output execution trace in file *execution.txt* could look as follows (**comments are to clarify; should not be included in the file)**:

```
...
590, 1, switch to kernel mode
591, 2, context saved
593, 1, find vector 7 in memory position 14
594, 1, obtain ISR address
595, 24, call device driver // Sample ISR line (all lines must add up to 110ms)
...
705, 1, IRET          //IRET after 110ms (from the device table for device 7)
706, 50, CPU burst    //Just an example CPU Burst
756, 1, switch to kernel mode
757, 1, context saved
758, 1, find vector 7 in memory position 14
759, 35, Store information in memory  //Sample END_IO ISR line.
...
869, 1, IRET //IRET called after 110ms (from the device table for device 7)
...
```

Assume that the ISR of both END I/O and SYSCALL for the same device takes the same amount of time.

We will submit test cases in BrightSpace; the TAs will mark these cases first. You must include your own test cases as well.

The program must be written in C++ (or C, they are mostly backwards compatible).

You must run numerous simulation tests and discuss the influence of the different steps of the interrupt process:

- Change the value of the *save/restore context* time from 10, to 20, to 30ms. What do you observe?
- Vary the ISR activity time from between 40 and 200, what happens when the ISR execution takes too long?
- How does the difference in speed of these steps affect the overall execution time of the process?
- You can process this data using a Python script, Excel spreadsheet or any other tools for separating the overhead from the actual work of your program (i.e., CPU use for processing and actual I/O needed by the program).
- Ask yourselves other interesting questions and try to answer them through simulations. For instance: what happens if we have addresses of 4 bytes instead of 2? What if we have a faster CPU

You should execute at least 20 simulation cases (more are preferred), analyze the results, and write a report (1-3 pages long), in pdf format. Submit the report in a *report.pdf* file.

For each of the simulation tests above, you should analyze the results obtained, and

Submission guideline:

Your code must be pushed to github. Name the repository *SYSC4001_A1*. The repository must contain:

- interrupts.cpp
- interrupts.hpp
- build.sh
- input_files
    - This is a folder containing your input files
- output_files

    - This is a folder containing your execution files
- vector_table.txt
- device_table.txt
- report.pdf

YOU MUST ALSO SUBMIT THE **REPORT.PDF** ON THE BRIGHTSPACE SUBMISSION PAGE. Include the link to your repository in this pdf. Also, in the description text box, include the link to your repository.