Assignment 1: Theory Questions

Esli Emmanuel Konate 101322259 and Nitish Grover 101324174

Carleton University

SYSC 4001 : Operating Systems

Most of our answers were taken from : (Operating Systems 10th Edition Silberschatz, Galvin, Gagne)

Part 1 - Concepts :

a) So the objective of the interrupt mechanism is to allow external devices and also events to send signals to the CPU for an action that needs to be done immediately (immediate attention). There are two important types of steps in the process.

The first steps are initiated by Hardware. First, an external device such as a keyboard, disk or other creates an electrical signal on the IRQ (Interrupt Request). Then the interrupt controller, which is a hardware component, detects the signals and also at the same time it finds out their priority. After that, at the end of each instruction cycle, the CPU will check the interrupt flag to check and see if any interrupts are pending. And, if an interrupt is found, the CPU will continue and complete the current instruction. In addition, the CPU automatically pushes the PC (Program Counter) and PSW (processor status word) onto the stack. Then the CPU sends a signal to the interrupt controller. The signal is like an “Ok” flag to acknowledge the event. Finally, the interrupt controller puts the interrupt vector number on the data bus. The CPU hardware also uses a vector number to index into the interrupt vector table to find the ISR address and also it loads the ISR address into the PC.

Following the Hardware steps, are the steps handled by the software side of things. Now the control is sent to the ISR which is a software code. Then the ISR saves extra CPU registers and information about the state. Then, the ISR executes the interrupt handling code. After that, the ISR sends an end of interrupt signal to the interrupt controller to signal the end of the interrupt code. The ISR will follow up by restoring the saved registers. Then it executes the interrupt return instruction given by the software. The CPU also restores the PC and PSW from the stack and then the CPU can continue with the execution of the program that was previously interrupted.

b)

A system call is a program interface, which allows user programs to request services from the operating system. For example system calls such as read(), write(), exit(). Read and write are used to read or write files, so file input and output, and exit is used to end a program. System calls are also implemented in a similar fashion as interrupts. A trap is an interrupt generated by software and it is caused by a request made by the user. When a user program has to make a system call, it will first execute an instruction that can cause a trap. The hardware side of the CPU will then save the PC and switch from user mode to kernel mode. So the mode bit is used in the hardware to separate when the system runs a user code and when it runs a kernel code. When a user runs the mode bit which is set to the user and when kernel code is executing, the mode bit will then switch to the kernel one. After switching to kernel mode, the system call will

be started with all the privileges and the access to the ressources which are normally protected. The kernel will then complete the operations, which were requested like accessing a file, allocate memory or handling an I/O device. After completing the system call, the kernel will send the result and start the return instruction. The hardware will then restore the program counter which was previously saved and then switch the mode back from kernel to user mode. This in turn, will allow the user program to continue where it left off. It is working this way to make sure that the users won't access privileged instructions and this protects the system from any unwanted behaviors that may happen due to faulty user programs.

C)
i) Check if printer is OK

To check if the printer is ready to do a job, the driver will first poll the device until it is ready. The name of this is programmed I/O. The driver will then send a command to the controller and it will constantly check the status of the printer. The driver will then call check_if_printer_OK() to make sure that the printer is not busy and that it has no current errors, such as being out of paper. Then the driver will poll the device and constantly check its status until it receives a ready signal. If the status shows that the printer is ready, the drive will then proceed. If the printer is unfortunately busy, the driver will keep on polling or it will queue the request. In the case of an error condition like out of paper, the driver will tell the user/notify him. This way of polling makes constant use of the CPU time, but it also makes sure that the printer is ready before any attempts of printing.
ii)
LF = Line Feed
CR = Carriage Return

For the print(LF,CR) statement, the printer must carry out the following steps. First, the driver checks if the printer is ready by checking the status register's busy bit. Then the CR command is sent to the data register of the printer. When the printer receives the command and activates the motor to move the print head back to the leftmost position (just like when you use the left arrow key on the keyboard to go back to the leftmost side of the document). The printer also sets a busy flag during the operation. Once the carriage return operation is done, the printer will clear the busy flag (not busy anymore). After that, the driver checks if the printer is ready again before sending the LF command to the data register of the printer. Then the printer will then activate the paper feed motor to start advancing the paper one line on the vertical (y-axis) of the paper (just like pressing enter on the keyboard and going to a new line). In addition, the printer will set again the busy flag during the operation and clear it once the operation is complete. The driver will also poll the status register or wait for an interrupt to confirm that both of the operations have been completed correctly. The driver also has the possibility of checking for states such as paper jam or out of paper as cases of error handling.

d)
In batch operating systems, off-line operation consists of using separate computers to handle slow I/O operations in a way that is independent from the main computer. The process works

with the first phase where the card readers on small computers read the punch cards containing the programs and data. Then, it follows by writing them to the magnetic tape. Following that, the main computer reads from the magnetic tape instead of directly from card readers and it allows it to process the jobs in sequence. Finally the last step is that the results are written to another magnetic tape, which is then printed offline by a separate computer. The advantages are that first the main computer is no longer constrained by the speed of the card readers and line printers but only limited by the speed of the magnetic tapes which are faster than those items. So, in other words, there is improved CPU utilization. In addition, the possibility of using multiple I/O devices (multiple readers to magnetic tapes and magnetic tapes to printer systems) for one CPU is there and a grand advantage because if the CPU can process input twice as fast as the reader can read cards, then making use of two readers working at the same time can make enough tape to keep the CPU busy. It is also cheaper, because cheaper computers will handle I/O while the expensive one will solely focus on computation. There is also increased throughput due to the fact that jobs can be packed together on tape for efficient processing which is also sequential. The disadvantages are first there is a longer delay in getting a job run. The job has to first be read onto the tape, then it has to wait until enough additional jobs are also read onto the tape to fill the tape. Then, the tape must be unloaded, carried to the CPU, and mounted on a free tape drive. This leads to users having to wait longer before seeing results due to the batch processing. Also, it necessitates multiple computers and good coordination between them. Finally, users can't interact with programs during their execution.

e)

i)

The $ used on the cards is used to help differentiate between data and program cards. Specifically, it is used as a way to identify a control card. Once detected, the control-card interpreter will read and carry out the instructions on the cards at the point of execution. If a driver fails to check for the $ symbol at the beginning of cards, the operating system would in turn incorrectly interpret Job Control Language commands as normal data. This in turns will cause the program to process system commands as input data, commands like $END, $LOAD, $RUN or more would be seen and treated as typical text and the job would either crash or lead to incorrect results/ouputs. A way to prevent this is for the OS to routinely/always check is a card starts with $ before sending it to user programs. The check is directly implemented within the OS and not left to the programmers to implement. So the OS will check if the cards read and handle the Job Control Language cards specifically.

ii)

In that case the Operating System should end the program that is currently running. Then it will return the control to the monitor. After that, it should process the $END card as any other JCL command and then continue to the next job in the batch. This will happen because as

mentioned earlier, the OS always checks for JCL commands to make sure that Job Control language always goes first over data from user programs.

f) Chosen instructions are Set Timer Counter and I/O Instructions (Found in Ch1, Silberschatz, Galvin)

Four Privileged Instructions

Enable/Disable Interrupt:
when user write the code user shouldn't have the privilege to control interrupts because it could monopolize the CPU or Operation system cannot handle critical events like timers or Input/output operations also humans are tend to make more errors than machine it could corrupt the entire system.

set processor status register or flags register
when changing the CPU status word it should be done in kernel mode and bits in status flags register control CPU so giving user this privilege can break protection and security so setting flag registers is considered privilege instruction.

Set Timer Counter:

The function of the instruction is to set the timer counter which will determine when the timer interrupts occur. It is a privileged instruction because as seen in the notes/book : "The Operating system sets the counter (privileged instruction)". If user programs had the power to set the timer, they could essentially monopolize the CPU by preventing interrupts caused by the timer or setting very long intervals. This will break the time-sharing concept seen in multiprogramming.

I/O Instructions:

The function is to direct communication with the I/O device registers. It is privileged because as seen in the notes/book device drivers "run in kernel mode". If user programs had direct access to I/O devices, they could essentially interfere with other processes' I/O or even have access to devices that are restricted/blocked. This would basically break the security and stability of the system.

g)

i)

The routines involved are Interrupt Processing, which handles the hardware interrupts from the devices. There are also the Device Drivers which control I/O devices, such as tape drives, printers, card readers and more. After that, there is the Job Sequencing, which manages flow operations (job flow) and transitions between the components of the system. Finally, there is the Control Language Interpreter, which parses through and also executes the JCL commands such as $LOAD, $RUN and more.

ii)

For $LOAD TAPE1:

First, the Control Language Interpreter reads the card and recognizes $LOAD TAPE1. Then, the Control goes to Job Sequencing which will activate the Loader routine. After that, the Device Drivers are called to access TAPE1. Then the Interrupt Processing handles the completion of the tape I/O. Finally, the control goes back through the Job Sequencing and back to Control Language Interpreter.

For $RUN:

First, the Control Language Interpreter reads and recognizes $RUN. After that, the Job Sequencing gives control to the loaded user program. Finally, during the execution, the Device Drivers and Interrupt Processing handle any I/O activity. When done, the control will be returned to the Job sequencing.

iii)

Step 1 : $Load Tape1:

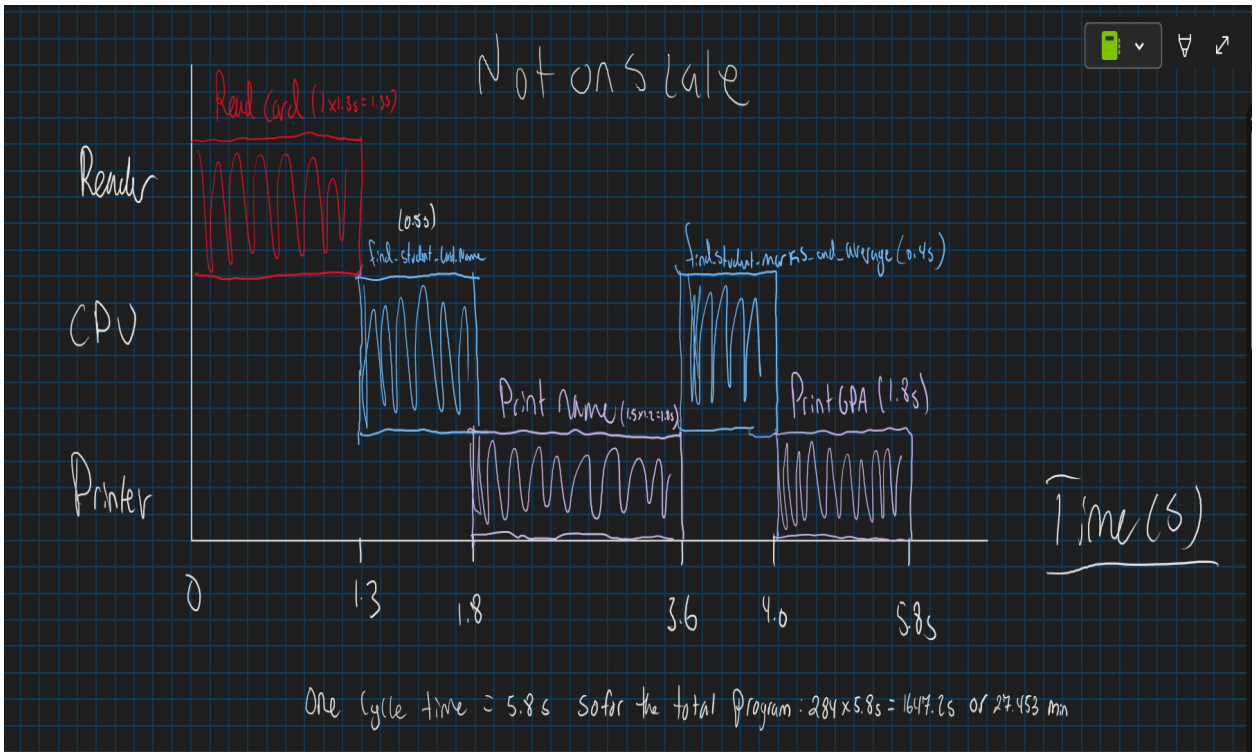
1. The Control Language Interpreter reads cards from the input devices and finds the $ which indicates JCL command. It will then parse the LOAD command with TAPE1 as parameters. Finally, it will call upon the Job Sequencing with LOAD as a request,
2. The Job Sequencing will receive the LOAD request. Then it will find and activate the loader routine. Finally the Job Sequencing will manage how the memory will be allocated for the starting program.
3. The Device Drivers which is the tape driver in this case will receive as a request to read from TAPE1 and then, it will send hardware commands to the tape controller. After that, it will start the read tape operation and finally send the data to the specified memory areas which are user areas.
4. The Interrupt Processing will then receive an interrupt when the tape read operation is completed and in turn, it will save the current state of the processor and execute the interrupt handler. After that, it will tell the Job Sequencing once it is done and restore the saved processor state.
5. The Job Sequencing will then send a confirmation signal/message to say that the load operation was successful and it will give back the control to the Control Language Interpreter.

Step 2: $RUN:

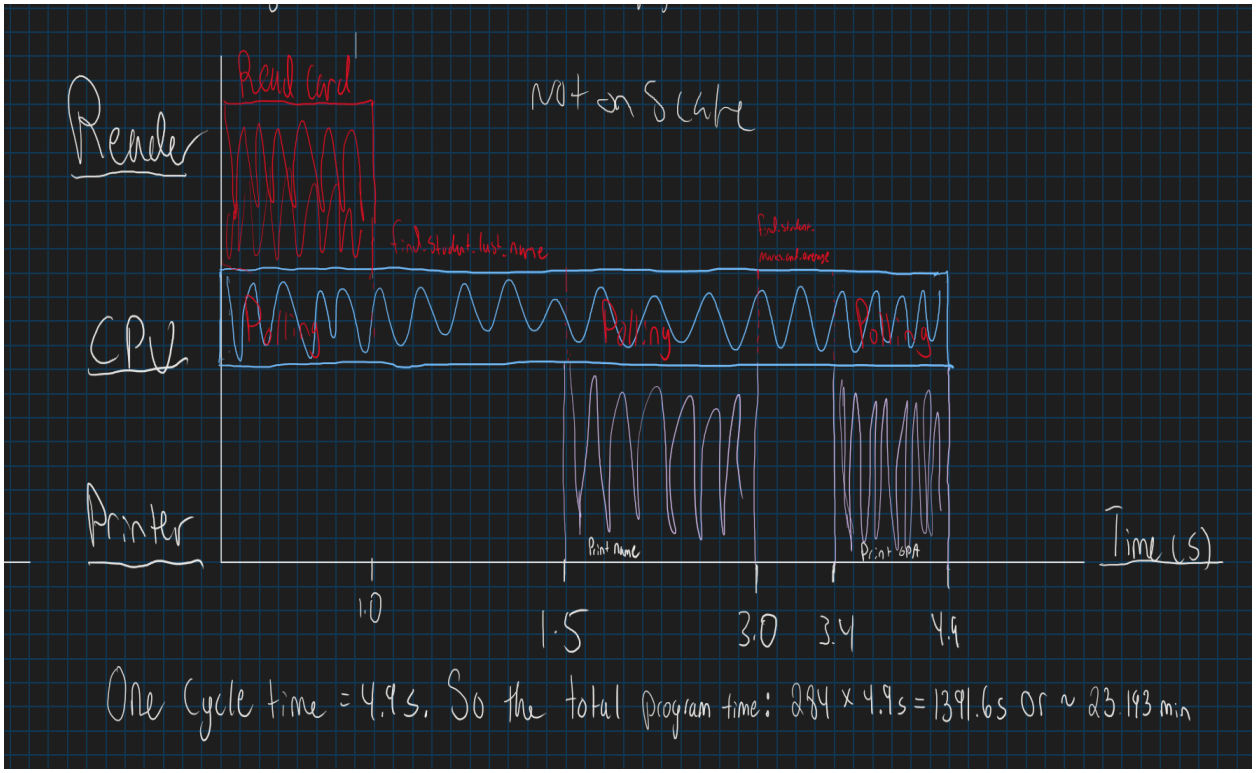
1. The Control Language Interpreter reads the next card and recognizes the $RUN command and then it will send a request to the Job Sequencing to execute the loaded program.
2. The Job Sequencing will then set and prepare the Program Counter to the start of the User Area. After that, it will switch the processor to user mode and transfer the control to the user program.
3. During the execution of the user program, if the I/O is needed, the Device Drivers will handle the operations for the device. The Interrupt Processing will manage how I/O is completed and finally, the Job Sequencing will manage the returns to the user program.
4. When the task is done, the control will be given back to the Job Sequencing. The Job Sequencing will then return to the Control Language Interpreter and the system will then be ready for the next job.

h)

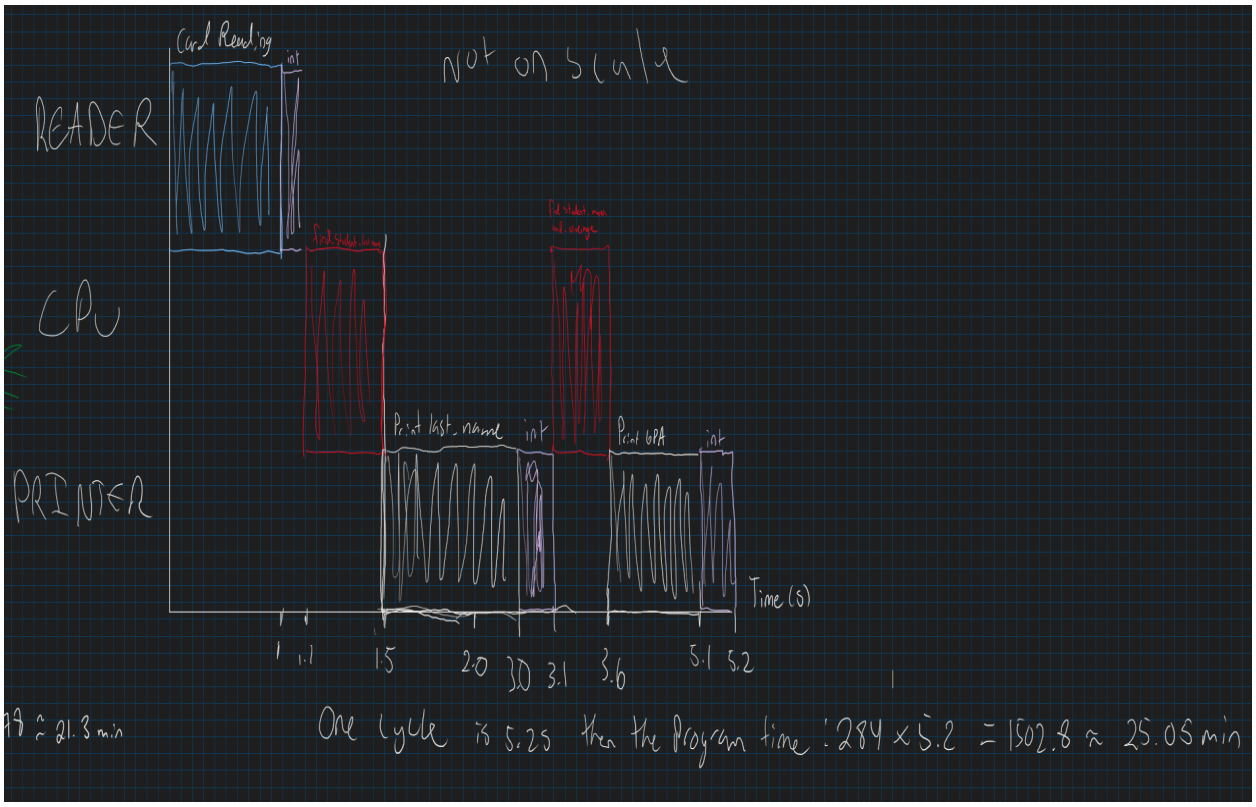
i)



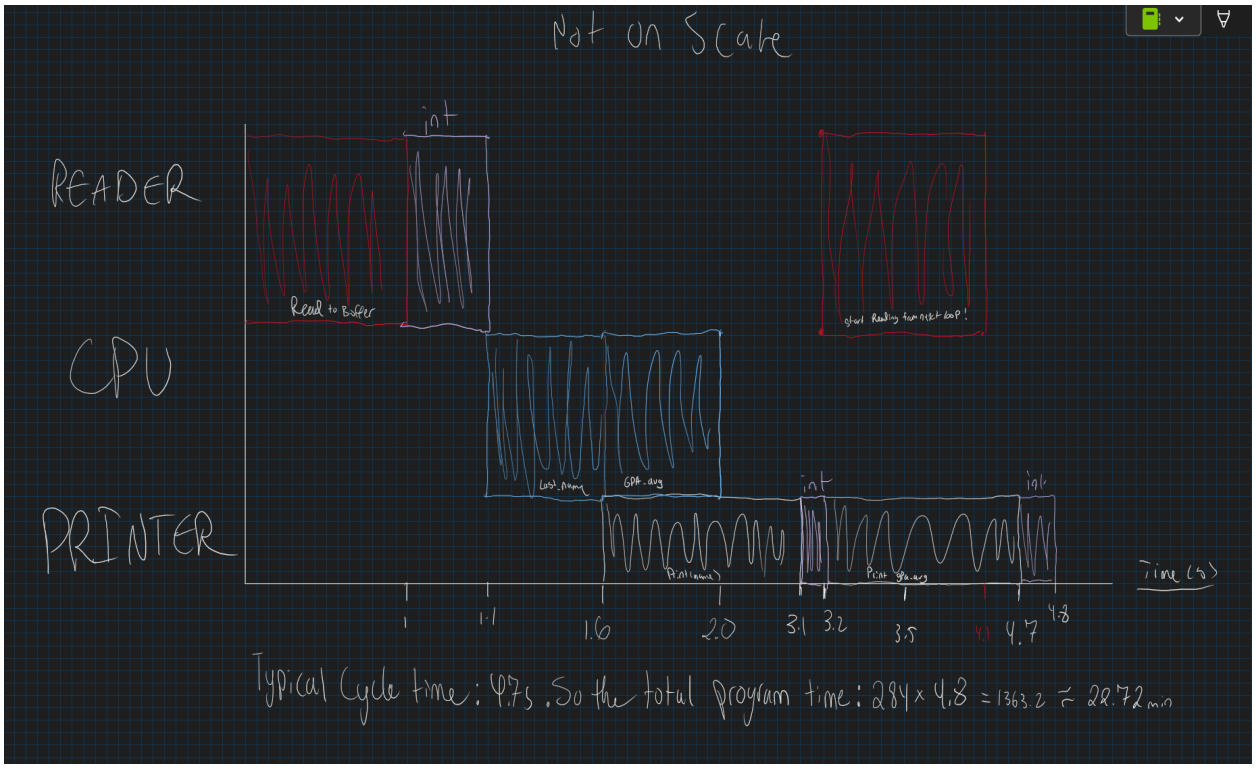
ii)


Reader
Read Card
not on scale
Find student last name
Find student marks and average
CPU
Polling
Polling
Polling
Printer
Print name
Print GPA
Time (s)
1.0
1.5
3.0
3.4
4.9
One Cycle time = 4.9s. So the total program time: 284 x 4.9s = 1391.6s or ~ 23.193 min

iii)


Card Reading
int
not on scale
READER
CPU
PRINTER
Print last_name
int
Print GPA
int
Time (s)
1
1.1
1.5
2.0
3.0
3.1
3.6
5.1
5.2
One cycle is 5.25 then the Program time: 284 × 5.2 = 1502.8 ≈ 25.05 min

iv)



To conclude, the Timed I/O was the slowest and this was due to timing errors and the CPU waiting for tasks. Also, tasks had to be done sequentially, which adds to the overall time of the program. In addition, polling is an improvement due to the fact that it eliminates timing errors but the CPU will still have to wait for tasks and the CPU constantly checks for an event, which can be expensive. Interrupts are a larger improvement in contrast to polling due to the fact that it allows the CPU to do work during the I/O, but it is still a tiny bit slower than polling. Finally, the fastest and best performance was through Interrupts and Buffering due to the fact that operations can be done simultaneously. So, it allows printing at the same time as the CPU is conducting calculations due to the buffers.