

Carleton University

Assignment 1: Group Submission **SYSC 4001A - L3 - GROUP 20: Operating Systems**

Student 1: Sahil Todeti (101259541)

Student 2: Teereza Allaham (101289630)

Due: October 6, 2025

Part I – 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]

Answer-

Definition: The interrupt is a mechanism which sends the request to the CPU by the external device to hold the current execution process temporarily.

When the CPU gets a signal from an external device, it recognizes the signal and pauses the current execution process to deal with it; this part of the process is done by the hardware. The CPU saves its current state, so that it doesn't lose its ongoing process. The mode then switches to kernel mode from user mode, as it's a privileged operation; this part is also done by the hardware. The CPU then transfers this control signal to the service routine maintained by the Operating System; this is the part where it is treated by the hardware as software interrupt. The Service Routine then reads and performs necessary actions to solve this interrupt errors and preventing any further interruptions. After solving the errors, it sends the signal back to the CPU. The mode switches back to user mode from kernel mode, and the CPU resumes its paused execution, restores its previously saved state from the stack, including program counter and registers; the process shifts back to the hardware from software.

b) [0.1 marks] Explain, in detail, what a System Call is, and 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]

Answer-

System Calls are an interface used for services that are available in the Operating System. It's a low-level programming tool that makes it accessible for users to perform services that only the operating system can accomplish. System Calls can safely transfer control from user mode to kernel mode, allowing the Operating system to act.

System Calls are used for multiple actions, for example:

- File Management - open(): opens a file, allowing content to be used
- Device Management - read(): read data from an input device.
- Process Control - wait(): waits till the call before it exits or completes execution

System Calls use software interrupts to ask Operating System for help while hardware interrupts ensures the safety and control between switches of user programs and Operating System. Both use a type of interrupt for different reasons.

System Calls are used for requests and I/O interrupts are used for interference in urgent events.

The Interrupt hardware mechanism is used to implement System Calls by letting user programs trigger a special kind of interrupt, known as a software interrupt, to switch the CPU into kernel mode.

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.).

Answer-

i. To check if the printer is OK and Ready to work, the following must be checked:

- Hardware Signal: This signal indicates that the printer is responsive to Operating system signal
- Power Check: This check ensures that the printer is receiving power
- Connection Check: This check ensures that the printer is connected to the Operating System
- Self-Diagnostic: To make sure all parts of the printer are functioning and ready to work
- Ink Availability: Makes sure enough ink is available to complete the interaction
- Paper Availability: Ensures enough paper is present in the printer to complete printing
- Paper Jam check: Ensures no stuck paper are present in the printing tray
- Error Check: Scan for any current error states or warning messages from the checks above
- Ready Signal: Confirms that the printer is in a Ready state to perform the operating system commands

Each of these checks is required for the printer to safely, reliably, and accurately receive and execute output commands from connected devices.

ii. When the printer receives the command from a device, it performs either LF or CR
LF - Line Feed - this performs the operation of ending a line and moving to the next line.

CR - Carriage Return - this performs the operation of the print head moves to the beginning of the current line

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

Answer-

Off-line operation (tape batching) is a late 1950s / early 1960s method. Slow card readers and line printers were replaced by magnetic-tape units operated off-line. Operators first copied card decks to tape on a separate device. When a tape was full, it was mounted on the main computer. Programs read “cards” from that tape and wrote output to another tape, which was later printed off-line. This method freed the CPU from device speeds and let multiple reader to tape and tape to printer setups keep one CPU busy. The trade off was a longer turnaround, a job had to be staged to tape, the tape filled, rewound, carried, and mounted before execution that was acceptable for batch workloads.

Advantages:

- Higher CPU utilization: CPU reads/writes from fast tape instead of waiting on slow card readers/printers.
- Parallel staging: Multiple reader to tape and tape to printer units can keep one CPU fed.
- Simpler mainframe I/O path: Slow peripherals handled off-line by auxiliary devices/operators.

Disadvantages:

- Longer turnaround per job: Jobs wait to be staged to a full tape, rewound, carried, and mounted.
- Operational risk: More hardware, tape handling, and potential deck/tape mixups.
- Sequential limits: Can’t write new cards while reading earlier ones, no random access.

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?

Answer-

i - If \$ is forgotten, "\$" cards get treated as ordinary data. The compiler/loader never gets invoked as intended and it might cause input/syntax error.

In order to prevent that error the card reader path must be owned by the OS (privileged), which intercepts records starting with "\$" and strips them out before any user program sees input.

ii - The Operating System recognizes "\$END" as end-of-job control, not user data. It should terminate the current job. Stop execution → close files/devices → write accounting → queue/print spooled output → release resources → and return to the job sequencer for the next job.

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).

Answer-

HLT (Halt Instruction): This instruction can only run when the system is in kernel mode. It stops the CPU and halts it until the next interrupt occurs. It is a privileged instruction because this instruction halts the entire CPU preventing it from executing any further instructions, it only runs in Kernel mode **[Student 1]**

Timer Management: This is considered a privileged instruction because when the user modifies or sets the timer, it directly affects the CPU's execution. For example, when a user sets or changes the timer value, the system cycle gets disrupted which blocks the interrupts. **[Student 1]**

I/O instructions: Read or write data from INPUT or OUTPUT devices that are connected via I/O to the hardware of the operating system. The I/O instructions are privileged as they have direct access to hardware which can cause damage to the hardware or leaks of data. **[Student 2]**

Load (LDR) / Store (STR): Is used to access and manage memory by controlling registers. They are essential for moving data between memory and CPU. They are privileged because they can change how the memory is mapped and protected, which may lead to crashing the system or having a less secure system. **[Student 2]**

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:

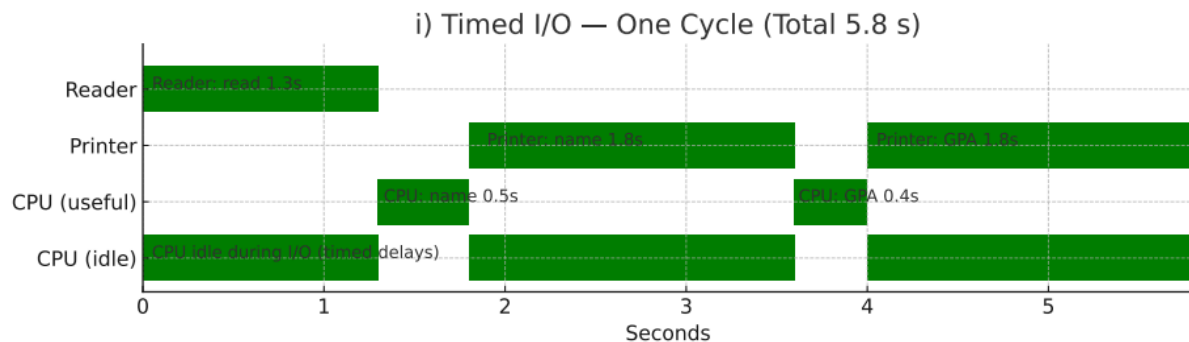
- i. A clear identification and description of the routines involved, with direct reference to the figure.**
- ii. A detailed explanation of the execution order and how the routines interact.**
- iii. A step-by-step breakdown of what each routine performs during its execution.**

Answer-

- i -
 - Job Control Interpreter (JCI / Monitor): reads control cards (\$) and orchestrates privileged actions.
 - I/O Subsystem & Tape Driver: executes physical I/O on TAPE1, raising interrupts/completions to the Operating System
 - Loader: brings the executable from device to memory, performs relocation and entry point setup.
 - Execution/Dispatcher: creates the process control block, sets the CPU's PC to the entry point, switches to user mode, and runs the program.
- ii -
 1. Monitor parses \$LOAD TAPE1 which calls Loader with source device TAPE1.
 2. Loader requests Tape Driver to read the executable, segment by segment, into the User Area → does relocation/verification → returns to Monitor.
 3. Monitor parses \$RUN which calls Execution/Dispatcher to build PCB, set the entry point, and dispatch the job.
 4. The program runs any device I/O it requests later and also goes through the I/O subsystem and drivers, with control returning to the Monitor on completion/termination.
- iii -
 1. Monitor → recognize \$LOAD and reads the next card.
 2. Call Loader → creates a load request targeting TAPE1.
 3. I/O Subsystem → positions tape, reads file header, code/data blocks.
 4. Loader → Verifies format, resolves relocations, records the entry address, sets up program image then returns status to Monitor.
 5. Monitor → notes that a load image is present and ready to run follows up with scanning other cards.
 6. Monitor → reads \$RUN, validates that a load image exists
 7. Execution → builds PCB, initializes registers/stack, I/O descriptors as needed, sets PC .
 8. User Program executes → exit or \$END, control returns to the Monitor for the next job.

h) [0.3 marks] Consider the following program: Loop 284 times { $x = \text{read_card}()$; $\text{name} = \text{find_student_Last_Name}(x)$; // 0.5s $\text{print}(\text{name}, \text{printer})$; $\text{GPA} = \text{find_student_marks_and_average}(x)$; // 0.4s $\text{print}(\text{GPA}, \text{printer})$; } 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

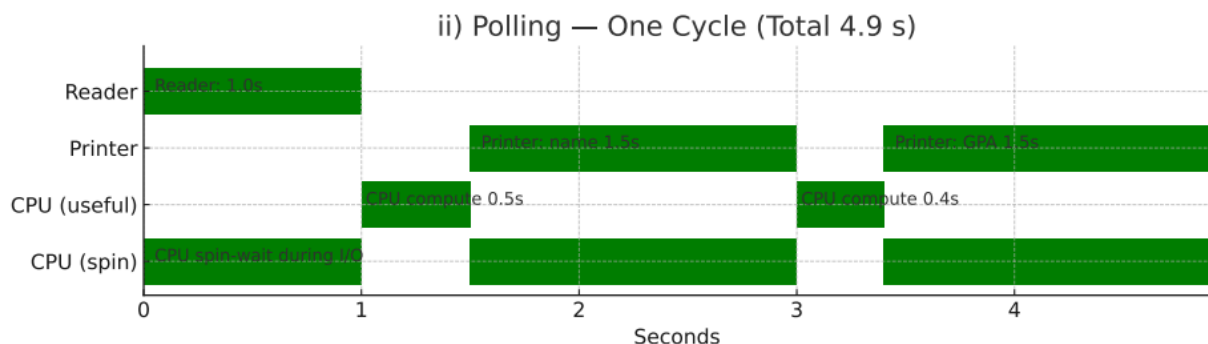


→ Per cycle: $1.3 + 0.5 + 1.8 + 0.4 + 1.8 = 5.8$ s

→ Total: $5.8 \times 284 = 1,647.2$ s

→ Note: CPU sleeps on fixed delays with inflated device times.

ii) Polling

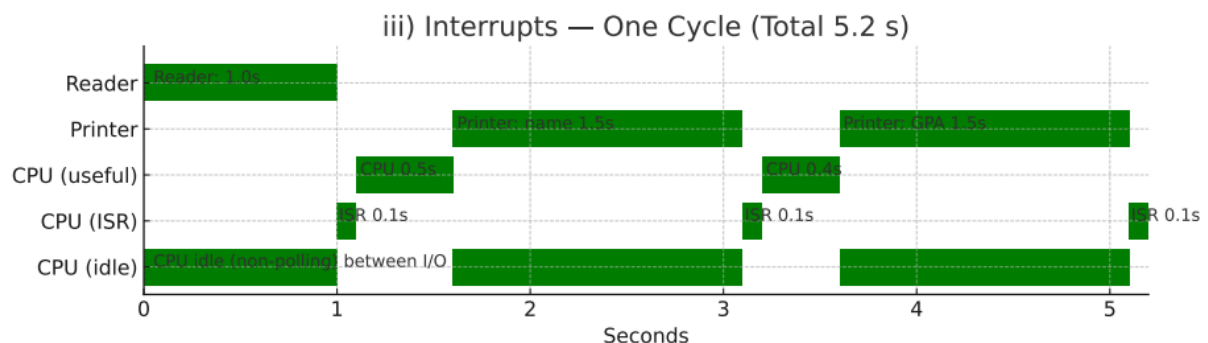


→ Per cycle: $1.0 + 0.5 + 1.5 + 0.4 + 1.5 = 4.9$ s

→ Total: $4.9 \times 284 = 1,391.6$ s

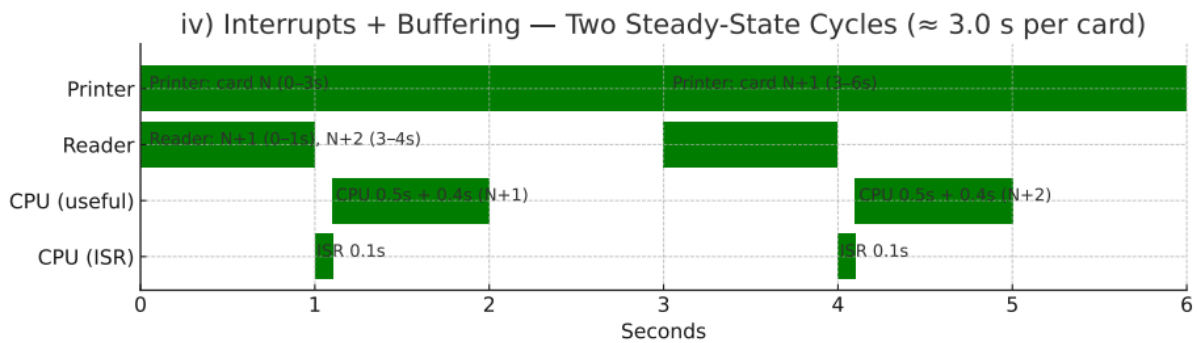
→ Note: Faster than timed I/O, but CPU time is wasted spin-waiting during I/O.

iii) Interrupts



- Per cycle: base 4.9 s + 3 ISRs × 0.1 = 5.2 s
- Total: 5.2 × 284 = 1,476.8 s
- Note: Small ISR overhead still serializes each step.

iv) Interrupts + Buffering (Consider the buffer is big enough to hold one input or one output)



- Total: 3.0 × 284 = 852.0 s
- Note: Overlap lets read, CPU and ISRs fit inside the 3.0 s print window.

https://github.com/TeerezaAllaham/SYSC4001_L3-G20_Assignment01.git

Source: A. Silberschatz, P. B. Galvin, and G. Gagne, Operating System Concepts, 10th ed. Hoboken, NJ, USA: John Wiley & Sons, Inc., 2018.