Sysc 4001

Assignment 1

Ben Woodhead
101057746

Emad Alomari
101162157

Oct 13, 2023

Part 1

a) **Explain why these special cards are needed, what their function is, and how they interact with the Operating System.**

i) Early computers didn't have advanced resource management. The individual job requirements had to be defined in advance by the programmer. The Job Control Language (JCL) and other batch processing systems were designed to help facilitate the management of jobs. The programmer would define the memory requirements, programs needed, resources to be mapped into the execution of the user program and provide control flow including steps to skip based on status.

The JCL special cards were instructions for The Monitor operating system. The JCL cards added special characters to the instructions to avoid collisions with user program instructions. This precaution was extended by the introduction of hardware change that allowed the CPU to segregate user commands from operating system commands. The JCL instructions would run in kernel mode allowing for resource loading and would switch to user space before allowing user programs to run.

The Job Control Language has instructions such as start a job ($JOB), end a job ($END), loading a program ($LOAD), executing a program ($EXEC) and a data section ($DATA). The start job section would allow the programmer to define the memory requirement and needed resources such as databases to be available to the user program during execution.

Special Cards are a crucial part of the human-computer interfaces for computing systems because they make it possible to carry out tasks and manage resources in a computing environment. A list of instructions called $JOB must be carried out. It is a set of programmes, data, and instructions designed to work independently of human intervention. The $END function notifies the CPU that the program has ended. It informs the CPU of the resources that are now accessible for usage by the next program.

ii) **Explain what would happen if, in the middle of the program execution, we detect the card $END. What should the Operating System do in that case?**

The Monitor operating system would consider the job complete and it would clean up, release memory, release resources, and end the job. The system would continue to spool until it found the next $JOB start instruction. If the next job instruction wasn't immediate then the operating system would be able to identify

this as an error state. The operating system designer could create an error handler for this but it would be difficult to define a best practice with human error.

b) Explain how kernel and user modes are used to protect against access by other users.

Kernel mode is a hardware change to the CPU that prevents contentious commands from directly accessing the hardware. All hardware access such as memory allocation, IO requests, and printer management were restricted to kernel mode. If a user-mode program needed access to hardware, it would issue a system call to the operating system. The operating system would switch to kernel mode and request the required resources on the user programs behalf. This allows the kernel code to gate keep any request. The kernel mode restriction could notice an memory out of bounds error and stop cross application memory temporing.

User mode has limited access and is unable to carry out some tasks that Kernel mode can. while Kernel mode has complete access to all resources, including memory and hardware. It's crucial to keep user mode and kernel mode separate because you don't want a user application to create a new process. This security is implemented via hardware that only kernel mode can access.

c) Write examples of three privileged instructions, and three non privileged instructions.

1) Loading jobs and terminated jobs are privileged instructions. The user may request a job be loaded or terminated but the operating systems long-term scheduler will provide the service when suitable. The loading and terminating process requires requesting resources such as memory, files, databases, etc and knowledge of the appropriate amount of jobs concurrently running.

2) Read/Write IO operations require communications with hardware that is only accessible by the operating system, making them privileged instructions. A program that needs to read and write to a file must request the operating system to load the data and provide it through a system call interface. The operating system controls IO operations to make sure that only authorized processes can communicate with delicate hardware parts. Without this restriction, any program could read from and write to important storage media, opening the door to unauthorized access and data breaches.

3) Turning off interrupts is a privilege. A serious issue would happen if two processes attempted to access the same crucial resource simultaneously. Interruptions during these crucial sections can result in conflicts. The kernel can prevent this from happening by disabling interrupts. Letting any program to disable interrupts can create serious issues because any program has the capacity to interfere with the system's regular operation.

4) Reading the system time is non privileged. Since reading the system time is a passive action with no immediate effect on how the system works. The reason why it's non privileged is because The only thing it does is retrieve the time and date from a clock or system counter. No settings or resources within the system are changed by this action.

5) Reading the processor's, is regarded as a non privileged operation.For decision-making or action, programs frequently need to inspect their own condition. In order to perform an action, they might need to know the value stored in a register.It only impacts the program itself, reading the processor status is a local operation. No system-wide parameters or settings that can potentially impair other software or the operating system are altered by it.

6) Arithmetic and logical operations are non privileged instructions. Loading and storing data is also non privileged.

d) Some early computers protected the OS by placing it in ROM. Define the advantages and difficulties of such a scheme.

Read Only Memory (ROM) is memory you can only read from and not write. It can only be a part of the main memory. The Operating systems in ROM had both advantages and disadvantages.

Advantages:

Unauthorized modifications - Since ROM is read-only, it cannot be changed intentionally or unintentionally. For important systems, where messing with the OS could have negative effects, this is essential.

Energy efficiency - Since ROM does not need to spin up discs or activate other energy-intensive components to retrieve the OS, they may use less power while starting up.

Security - Due to its invulnerability to modification, ROM OS is immune to the majority of viruses and malware. In contrast to systems where the OS is located on writable storage, this offers a higher level of security.

Simplicity - The system did not require additional drivers and could run a fetch execute cycle directly against the ROM or copy the ROM into memory and continue. which order, thee

Disadvantages:

Upgradable - The system is difficult to upgrade with introduction of new features and bug fixes. The invention of electrically erasable programmable read only memory (EEPROM).

Size Restrictions - ROMs are limited inside and more costly then cards, tapes or hard drives. This would cause a limit on the amount of features available from the operating system.

e) What is the purpose of interrupts? How do they work?

An interrupt is a hardware signal that tells the CPU that external hardware requires the CPUs attention. The interrupt line would be held either high or low depending on the implementation by the external hardware. As an example, a hard drive that has completed an IO operation would send a signal (5 volts on a line as an example) to the CPU saying that the data is available to be picked up.

f) What are the differences between a trap and an interrupt? How do traps work?

A trap or system call is a mechanic to allow user programs to request access to restricted resources. The system call exploited the previously created interrupt system that had already been introduced for external hardware. An interrupt is a signal on a line from an external device to the CPU. On the other hand, a software interrupt sets the interrupt value directly on the CPU. To avoid confusion the software interrupts are often called traps.

Both traps and interrupts are techniques used in computing to deal with situations that demand the operating system's attention. An interrupt is a signal that a hardware component sends to the CPU to momentarily halt the current program's execution and switch to the Interrupt Service Routine (ISR). Traps are started by the program itself to ask the operating system for a certain service to be provided by the ISR. Traps are used to switch between user mode and kernel mode,

enabling the user program to ask the operating system for services like reading from a file.

How traps operate: When a program executes a trap instruction, the CPU has a flag set to interrupt that will be seen during the fetch execute cycle. The ISR will enter Kernel mode and run the appropriate ISR. The CPU then records the status of the program so that it can pick up where it left off at a later time. The CPU hands off control to the ISR, which is located in a predetermined area of memory. This is the location of the operating system's trap code. The operation is carried out by the trap handler. When the trap handler has finished its job, it restores the program's saved state so that it can resume running from the point where the trap was started.

g) **Explain briefly how a Time-Sharing Operating System works. How does it manage to handle multiple interactive users with a single CPU?**

A time sharing operating system gives each process a predefined max running time called a cpu time slice or time quantum. A hardware timeout counter will provide a CPU interrupt that will run the short-term scheduler ISR providing an IO request hadn't already caused a process switch. An example of Time Sharing Operating systems are real time operating systems where execution needs to be deterministic.

h) **[4] (Kernel structure) Let us suppose we have a process in a multitasking OS**.

i) What are the events needed to make a new process to move from the "New" to the "Ready" state?

The long term scheduler is an ISR that will respond to a timeout from an external clock. The long term scheduler will admit jobs based on a policy of how many jobs should run concurrently. As an example, a single job at a time queue will have nothing to run during IO while having too many jobs would consume all the resources. Once a job has been loaded and all the required resources are provided then an **admit event** if fired. The admit ISR would promote the job from the new queue to read queue and create the process control block. In a more interactive environment a job could be created by a system call such as fork or exec from the user at a terminal or GUI.

ii) How about events to move the process from "Running" to "Terminated"?

An exit event will be produced when the application makes an exit system call. The exit system ISR will run and promote the job to the terminated queue. The

process will wait in the terminated queue until all the resources have been released. At that point memory will be cleaned up, the process control block will be deleted and room for another job will be created.The long term scheduler can then promote another job from the new queue to the ready queue appropriate. In a more interactive state, a system call to kill a process could be created externally and force the application to quit.

k) **DMA is used for high-speed I/O devices to reduce overhead:**

i) How does the CPU interface with the device to coordinate the transfer?

DMA is used by the CPU to manage data transfers with high-speed I/O devices.The DMA controller is initially set up by the CPU with the necessary transfer parameters. The DMA controller momentarily assumes control of the system bus when the I/O device signals that it is ready for a data transfer, enabling it to interface with memory directly for data transfer. Bypassing the CPU in this way, reducing the processing overhead associated with managing the transfer. The DMA controller may inform the CPU when the transfer is finished.

ii) How does the CPU know when the DMA operations are complete?

The DMA uses an interrupt to notify the CPU when the DMA operations are finished. The DMA controller initiates a hardware interrupt signal after the DMA operations are finished. This halts the CPU's ongoing operations and instructs it to carry out a specific Interrupt Service Routine linked to DMA completion. After acknowledgment, the ISR handles the completion event, enabling the CPU to get back to work on its usual responsibilities. While DMA activities are occurring, the CPU may handle other tasks effectively thanks to its asynchronous notification system.

iii) The CPU can execute other programs while DMA is transferring data. Does this process interfere with the execution of the user programs? If so, explain how.

Yes, there are various ways in which the concurrent execution of CPU and DMA operations can interfere with the operation of user programs. The DMA controller and CPU both have access to system memory. Conflict may arise if the CPU and DMA attempt to access the same region at the same time. When the DMA controller generates an interrupt to inform the CPU that a data transfer has been completed, that can also cause interference. The CPU must terminate the present task it is working on and run the related Interrupt Service Routine. This may cause user programs to be delayed. The priority management is yet another area where this could interfere. There may be competition between the CPU and DMA for system bus access. To ensure fair access, the operating system system must manage priorities, which causes delays.