OPERATING SYSTEM
IT-42033

CHAPTER – 13
Q&A

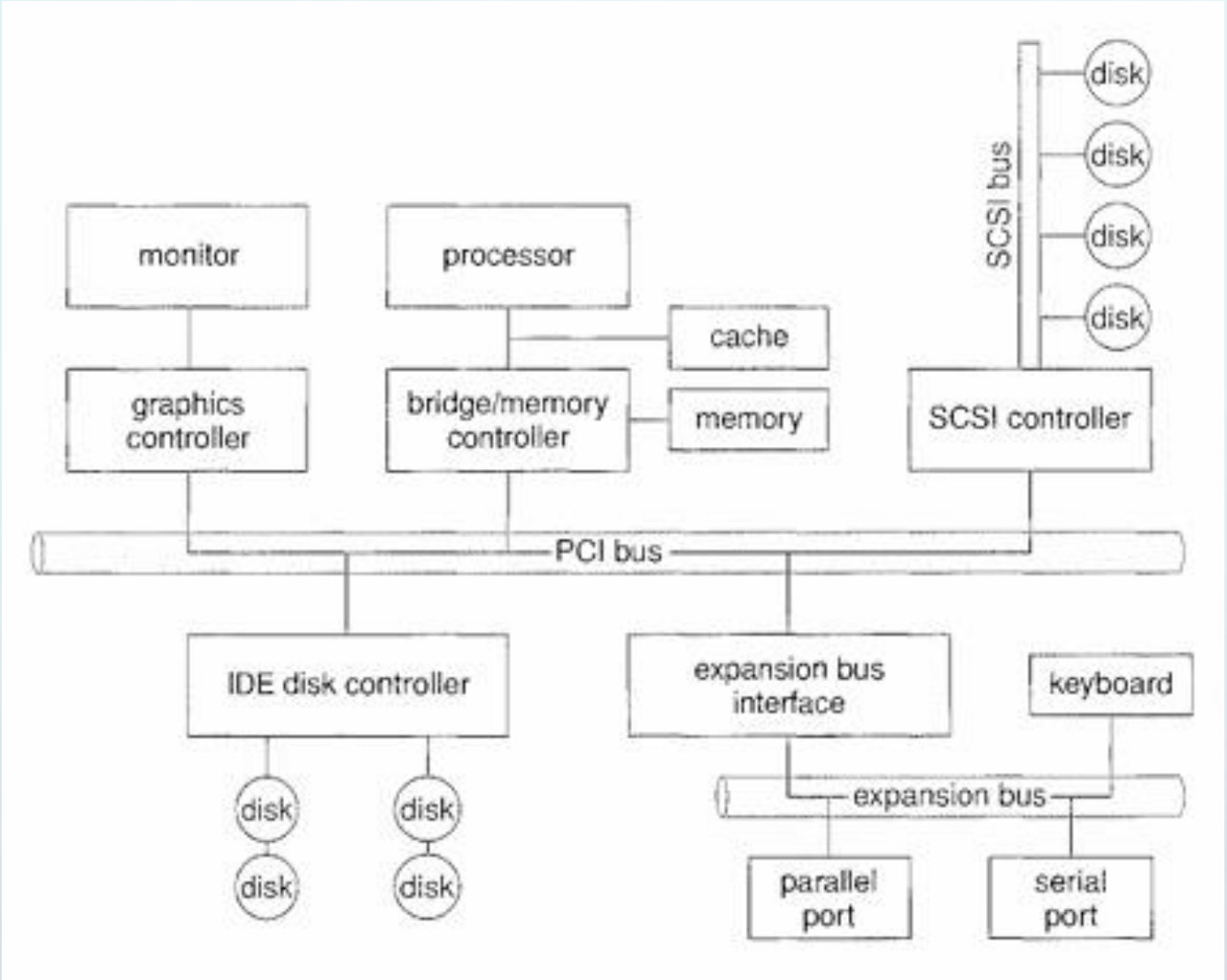
13.4

Polling can be more efficient than interrupt-driven I/O. This is the case when the I/O is frequent and of short duration. Even though a single serial port will perform I/O relatively infrequently and should use interrupts, a collection of serial ports such as those in a terminal concentrator can produce a lot of short I/O operations, and interrupting for each one could create a heavy load on the system. A well-timed polling loop could alleviate that load without wasting many resources through looping with no I/O needed.

13.13

Consider a system which performs 50% I/O and 50% computes. Doubling the CPU performance on this system would increase total system performance by only 50%. Doubling both system aspects would increase performance by 100%. Generally, it is important to re move the current system bottleneck, and to increase overall system performance, rather than blindly increasing the performance of individual system components.

Q. Draw a typical PC bus structure.



A typical PC bus structure

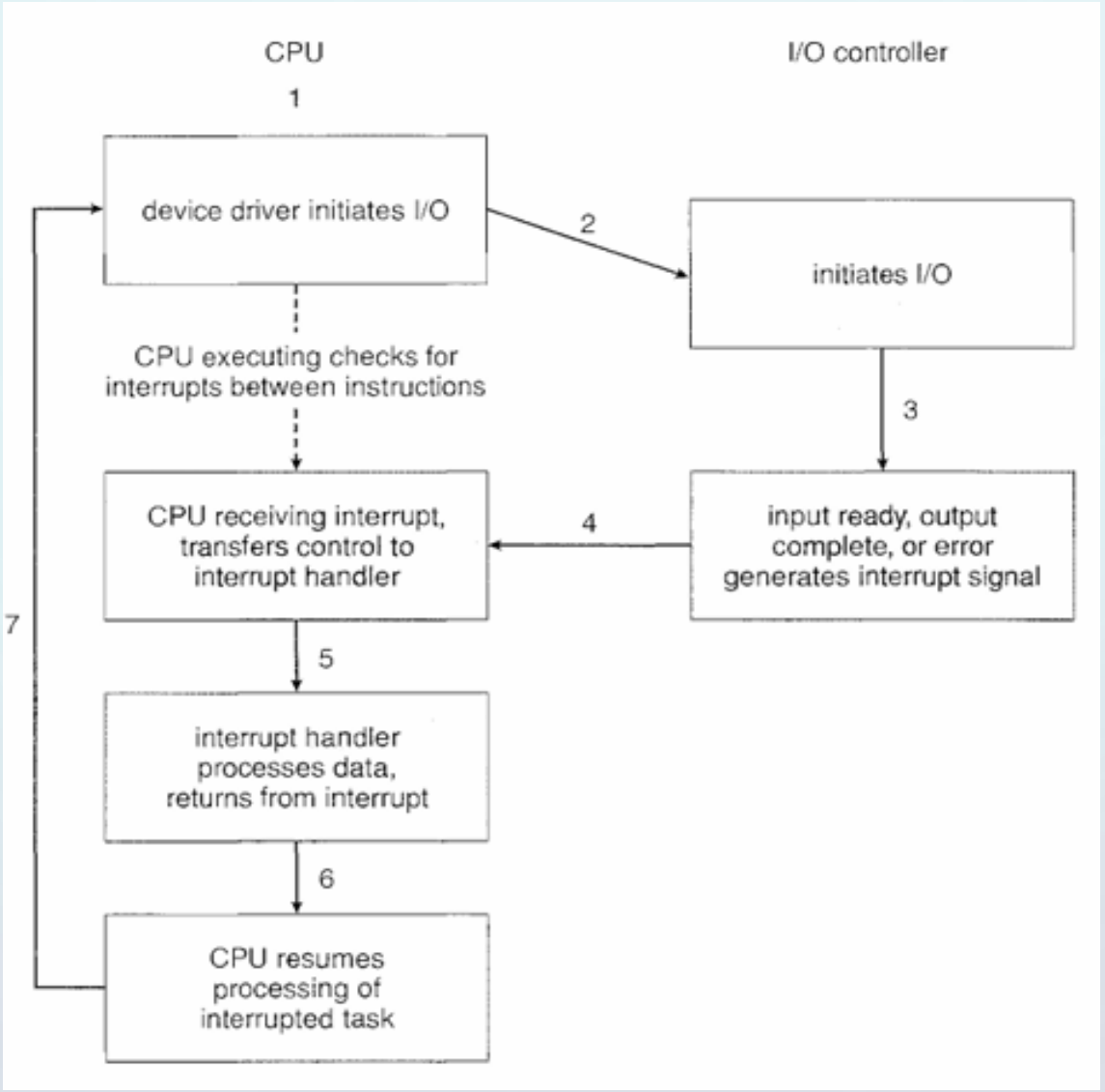
Q. Explain how the busy bit works in the handshaking process and draw the interrupt-driven I/O cycle.

The interaction between a host and a controller involves a simple handshaking process. For an example, 2 bits are used to manage the communication between them. The controller uses a busy bit to indicate whether it is working or ready to accept a command. When the controller is busy, it sets the busy bit to 1. When it is ready, it clears the busy bit to 0. The host signals its readiness to send a command by setting the command-ready bit in its command register.

The host first checks the busy bit repeatedly until it becomes clear. Then, it sets the write bit in the command register and writes data into the data-out register. After that, the host sets the command-ready bit. The controller, upon noticing the command-ready bit is set, sets its busy bit and processes the command. It reads the command and data, performs the necessary I/O, and then clears the command-ready bit and the busy bit to indicate completion.

This polling method works well if the controller and device respond quickly. However, if the wait is long, the host may need to switch to another task. Polling can be efficient, but if the device is rarely ready, it can waste CPU resources. In such cases, using interrupts allows the hardware to notify the CPU when the device is ready, making the process more efficient.

Q. Explain how the busy bit works in the handshaking process and draw the interrupt-driven I/O cycle. (Cont …)



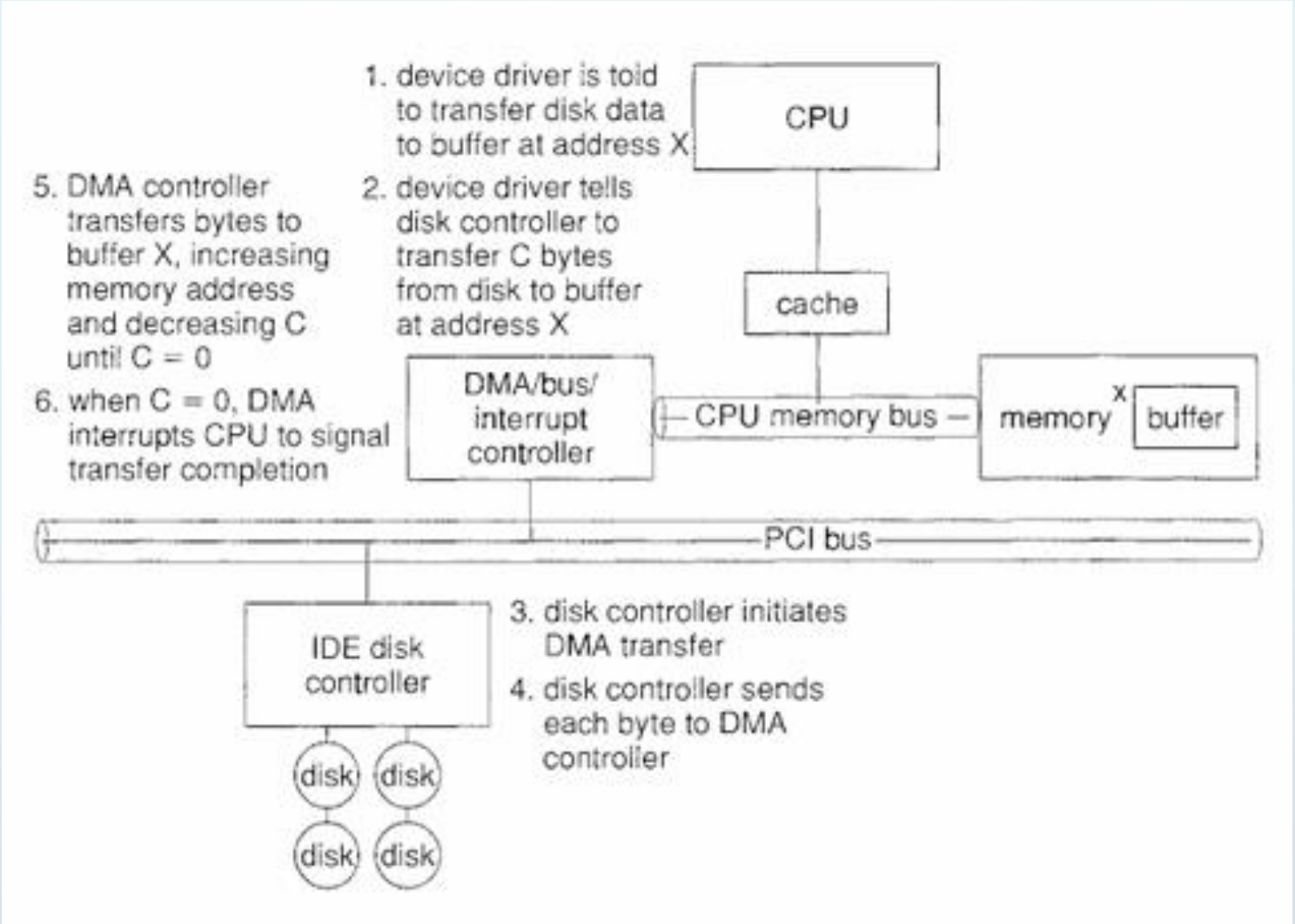
Interrupt-driven I/O cycle

Q. How does the use of a DMA controller improve system performance compared to using the CPU for data transfers?

Direct Memory Access (DMA) is a method used for transferring large amounts of data without burdening the main CPU. Instead of the CPU managing each byte transfer, a special-purpose processor called a DMA controller handles the data transfer. To start a DMA transfer, the host writes a command block in memory that includes the source and destination addresses and the number of bytes to transfer. The CPU then informs the DMA controller of this command block and continues with other tasks.

The DMA controller communicates with the device controller using two wires: DMA-request and DMA-acknowledge. When the device has data ready, it sends a signal on the DMA-request wire. The DMA controller then takes control of the memory bus, performs the transfer, and sends a DMA-acknowledge signal back to the device. Once the transfer is complete, the DMA controller interrupts the CPU to signal that the operation is finished. This process allows the CPU to focus on other tasks, improving overall system performance.

Q. How does the use of a DMA controller improve system performance compared to using the CPU for data transfers? (Cont …)



Steps in a DMA transfer.

Q. What is the difference between a character-stream device and a block device?

Character-stream or block: A character-stream device transfers bytes one by one, whereas a block device transfers a block of bytes as a unit.

Sequential or random access: A sequential device transfers data in a fixed order determined by the device, whereas the user of a random-access device can instruct the device to seek to any of the available data storage locations.

Synchronous or asynchronous: A synchronous device performs data transfers with predictable response times. An asynchronous device exhibits irregular or unpredictable response times.

Sharable or dedicated: A sharable device can be used concurrently by several processes or threads; a dedicated device cannot.

Speed of operation: Device speeds range from a few bytes per second to a few gigabytes per second.

Read-write, read only, or write only: Some devices perform both input and output, but others support only one data transfer direction.

Q. What are the key characteristics of I/O devices in a table format?

aspect	variation	example
data-transfer mode	character block	terminal disk
access method	sequential random	modem CD-ROM
transfer schedule	synchronous asynchronous	tape keyboard
sharing	dedicated sharable	tape keyboard
device speed	latency seek time transfer rate delay between operations	
I/O direction	read only write only read–write	CD-ROM graphics controller disk

Characteristics of I/O devices

Q. What is the purpose of the select() function in the socket interface for network I/O?

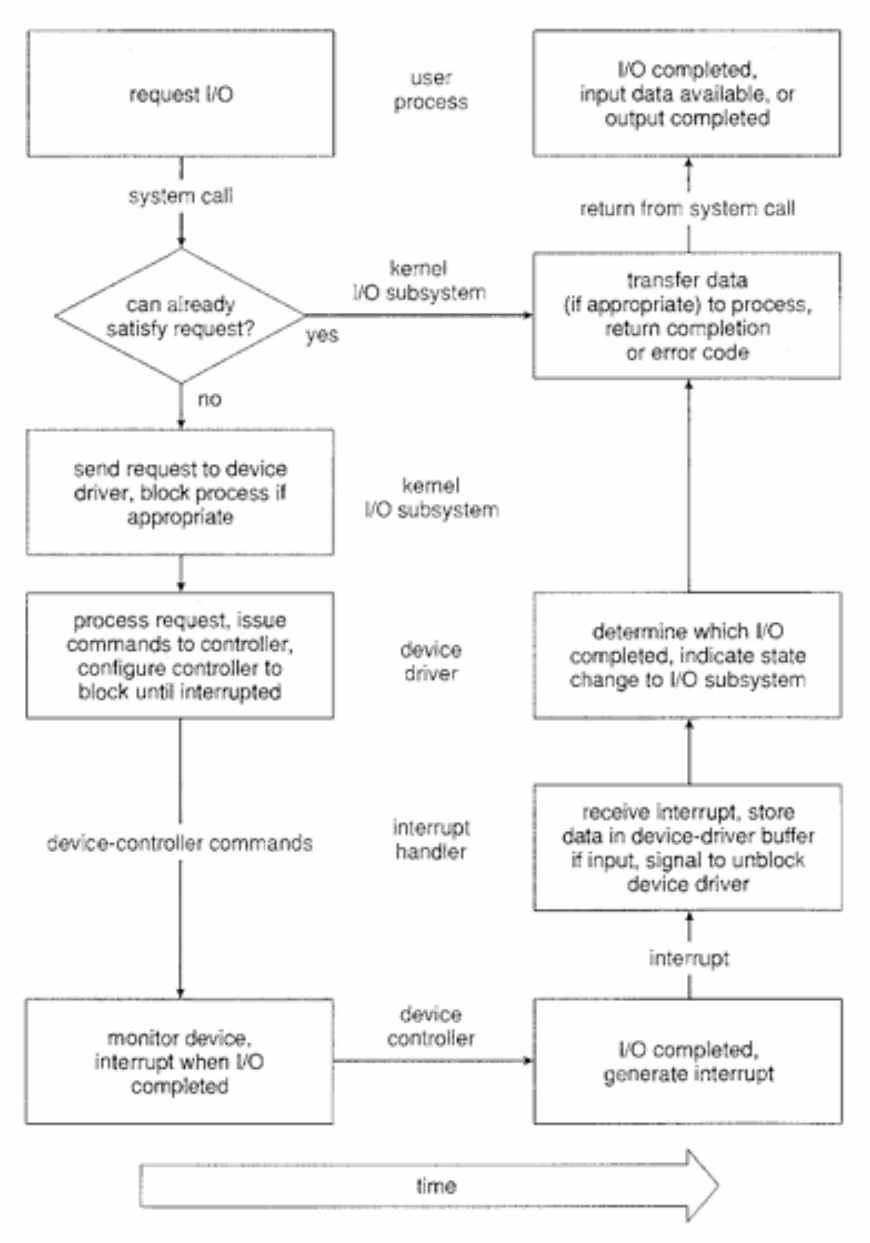
Network I/O differs significantly from disk I/O, leading most operating systems to provide a distinct network I/O interface. This interface, commonly known as the socket interface, allows applications to create sockets, connect to remote addresses, listen for incoming connections, and send or receive packets. The socket interface includes a function called *select( )*, which helps manage multiple sockets by indicating which ones have data ready to be processed, thus avoiding the need for polling. Various inter-process and network communication methods exist, with UNIX offering a range of options like pipes, FIFOs, and message queues, while Windows NT provides separate interfaces for network interface cards and protocols.

Q. How does I/O scheduling improve overall system performance and response times for applications?

I/O scheduling is the process of determining the best order to execute a set of I/O requests. The order in which applications issue system calls is often not optimal. By rearranging the order of service, the operating system can improve overall system performance, ensure fair access to devices among processes, and reduce the average waiting time for I/O completion. For example, if a disk arm is near the beginning of a disk and three applications request data from different locations, the operating system can serve the requests in a more efficient order to minimize the distance the disk arm must travel.

The operating system maintains a wait queue for each device, where requests are placed when applications issue blocking I/O system calls. The I/O scheduler rearranges this queue to enhance efficiency and response times. Additionally, the system may prioritize certain requests, such as those from the virtual memory subsystem, to ensure timely processing. Asynchronous I/O requires the operating system to track multiple requests simultaneously, often using a table that contains information about each device's state and the type of requests being processed.

Q. Illustrate the life cycle of an I/O request



The life cycle of an I/O request

References

Images: Internet

Source: Operating System Concepts (8th Edition)