4.12 I/O SYSTEMS

I/O Hardware

- ➢ The role of the operating system in computer I/O is to manage and control I/O operations and I/O devices.
- ➢ A device communicates with a computer system by sending signals over a cable or even through the air.

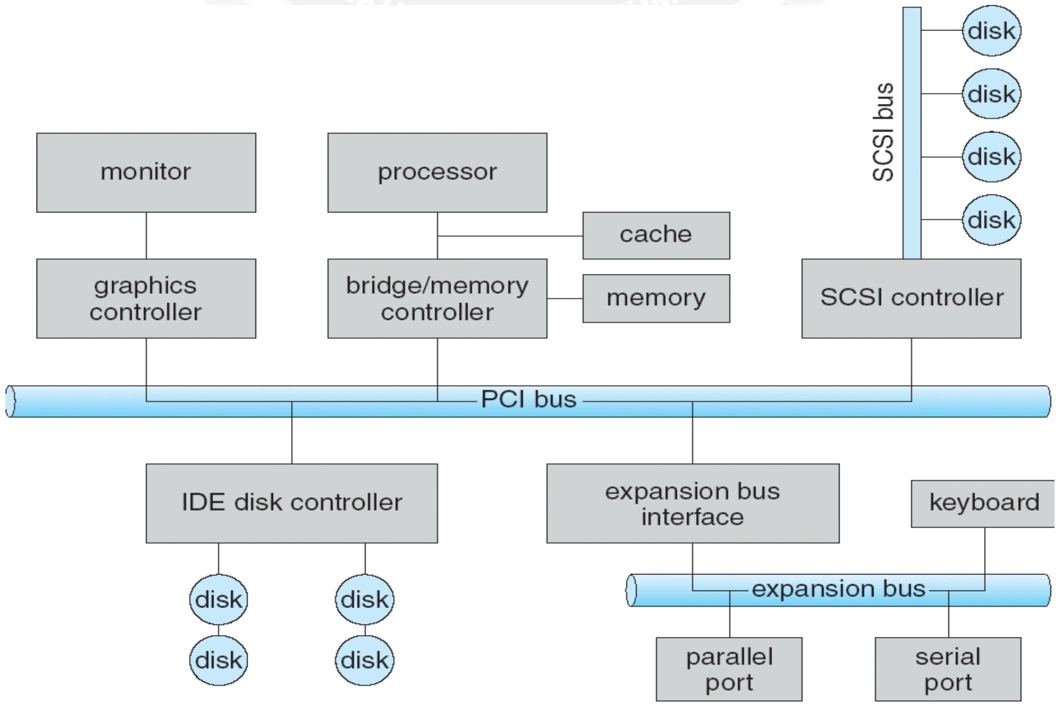
Port: The device communicates with the machine via a connection point (or port), for example, a serial port.

Bus: If one or more devices use a common set of wires, the connection is called a bus.

Daisy chain: Device 'A' has a cable that plugs into device 'B', and device 'B' has a cable that plugs into device 'C', and device 'C' plugs into a port on the computer, this arrangement is called a daisy chain. A daisy chain usually operates as a bus. Only one device is attached to computer directly.

PC bus structure:

- PCI bus- that connects the processor-memory subsystem to the fast devices
- Expansion bus that connects relatively slow devices such as the keyboard and serial and parallel ports
- A **controller or host adapter** is a collection of electronics that can operate a port, a bus, or a device.



- A serial-port controller is a simple device controller. It is a single chip in the computer that controls the signals on the wires of a serial port.
- SCSI bus controller - contains a processor, microcode, and some private memory.

How can the processor give commands and data to a controller to perform an I/O transfer?

- Direct I/O instructions -Use special I/O instructions that specify the transfer of a byte or word to an I/O port address. It triggers bus lines to select the proper device and to move bits into or out of a device register
- Memory-mapped I/O -The device-control registers are mapped into the address space of the processor. The CPU executes I/O requests using the standard data-transfer instructions to read and write the device-control registers.

An I/O port typically consists of four registers: status, control, data-in, and data-out registers.

- Status register- Read by the host to indicate states .
- Control register - Written by the host to start a command.
- data-in register - Read by the host to get input
- data-out register - Written by the host to send output

Polling

Interaction between the host and a controller

- The controller sets the busy bit when it is busy working, and clears the busy bit when it is ready to accept the next command.
- The host sets the command ready bit when a command is available for the controller to execute.

Coordination between the host & the controller is done by handshaking as follows:

- The host repeatedly reads the busy bit until that bit becomes clear.
- The host sets the write bit in the command register and writes a byte into the data-out register.
- The host sets the command-ready bit.
- When the controller notices that the command-ready bit is set, it sets the busy bit.
- The controller reads the command register and sees the write command. It reads the data-out register to get the byte, and does the I/O to the device.

- The controller clears the command-ready bit, clears the error bit in the status register to indicate that the device I/O succeeded, and clears the busy bit to indicate that it is finished.
- In step 1, the host is —**busy-waiting or polling**: It is in a loop, reading the status register over and over until the busy bit becomes clear.

Interrupts

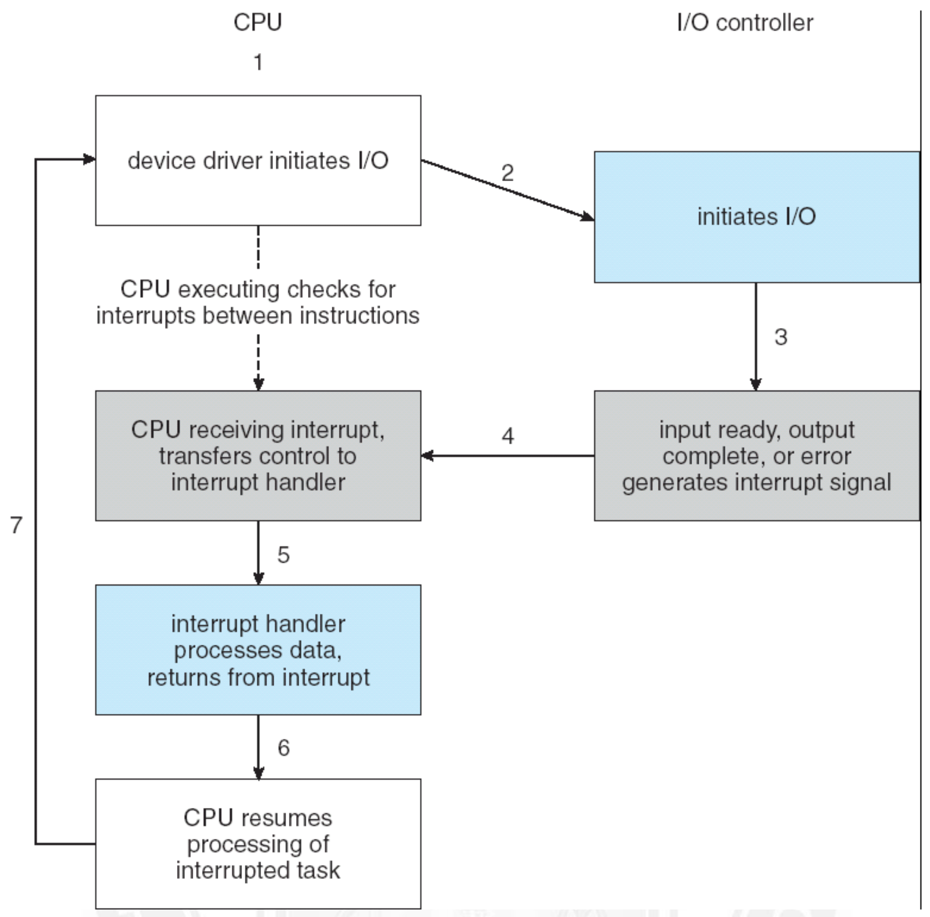
The CPU hardware has a wire called the —interrupt-request line.

The basic interrupt mechanism works as follows;

- The CPU has an ***interrupt-request line*** that is sensed after every instruction. A device's controller ***raises*** an interrupt by asserting a signal on the interrupt request line.
- The CPU then performs a state save, and transfers control to the ***interrupt handler*** routine at a fixed address in memory.
- The interrupt handler determines the cause of the interrupt, performs the necessary processing, performs a state restore, and executes a ***return from interrupt*** instruction to return control to the CPU.

Two interrupt request lines:

- **Nonmaskable interrupt**: which is reserved for events such as unrecoverable memory errors
- **Maskable interrupt**: Used by device controllers to request service. CPU can temporarily ignore this interrupts during critical processing.


CPU
I/O controller
1
device driver initiates I/O
2
initiates I/O
CPU executing checks for interrupts between instructions
3
CPU receiving interrupt, transfers control to interrupt handler
4
input ready, output complete, or error generates interrupt signal
7
5
interrupt handler processes data, returns from interrupt
6
CPU resumes processing of interrupted task

Interrupt-driven I/O cycle.

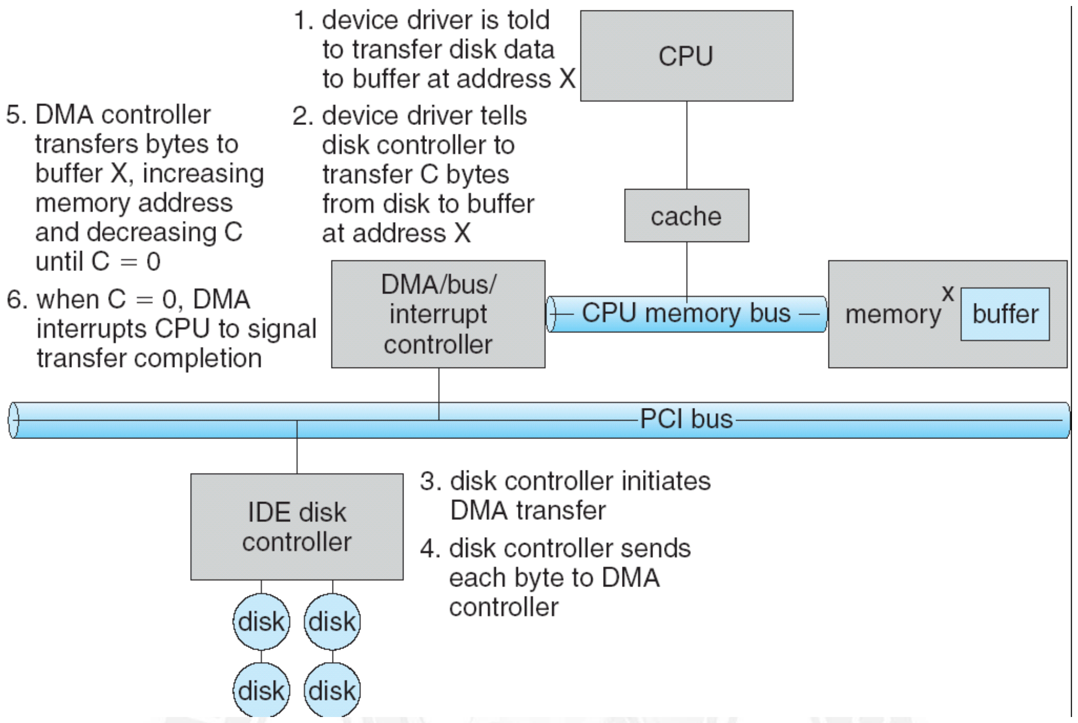
Direct Memory Access (DMA)

In general it is tough for the CPU to do the large transfers between the memory buffer & disk; because it is already equipped with some other tasks, then this will create overhead. So a special-purpose processor called a direct memory- access **(DMA)** controller is used.

Steps:

1. DMA controller transfers data without the intervention of CPU.DMA-request is initiated by device controller for getting DMA access.

2. Then DMA controller seizes bus and places memory address then send DMA-ACK. 3.Then DMA controller transfers data.


1. device driver is told to transfer disk data to buffer at address X
2. device driver tells disk controller to transfer C bytes from disk to buffer at address X
3. disk controller initiates DMA transfer
4. disk controller sends each byte to DMA controller
5. DMA controller transfers bytes to buffer X, increasing memory address and decreasing C until C = 0
6. when C = 0, DMA interrupts CPU to signal transfer completion
CPU
cache
DMA/bus/ interrupt controller
CPU memory bus
memory
X
buffer
PCI bus
IDE disk controller
disk
disk
disk
disk

Application I/O Interface

- I/O system calls encapsulate device behaviours in generic classes
- Device-driver layer hides differences among I/O controllers from kernel

Devices vary in many dimensions

1. Character-stream or block
2. Sequential or random-access
3. Sharable or dedicated
4. Speed of operation
5. read-write, read only, or write only

Types	Description	Example
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.	Terminal, Disk
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.	Modem, CD-ROM
Sharable or dedicated	A sharable device can be used concurrently by several processes or threads; a dedicated device cannot.	Tape, Keyboard
Speed of operation	Latency, seek time, transfer rate, delay between operations	
read-write, read only, or write only	Some devices perform both input and output, but others support only one data direction.	CD-ROM, Graphics controller, Disk

1. Block and Character Devices

- ***Block devices*** are accessed a block at a time. Operations supported include read( ), write( ), and seek( ).
- ***Character devices*** are accessed one byte at a time. Supported operations include get( ) and put( ).

2. Network Devices

- Because the performance and addressing characteristics of network I/O differ significantly from those of disk I/O, most operating systems provide a network I/O interface that is different from the read0 -write() -seek() interface used for disks.

➢ Windows NT provides one interface to the network interface card, and a second interface to the network protocols.

➢ In UNIX, we find half-duplex pipes, full-duplex FIFOs, full-duplex STREAMS, message queues and sockets.

3. Clocks and Timers

Most computers have hardware clocks and timers that provide three basic functions:

1. Give the current time

2. Give the elapsed time

3. Set a timer to trigger operation X at time T

These functions are used by the operating system & also by time sensitive applications.

Programmable interval timer: The hardware to measure elapsed time and to trigger operations is called a programmable interval timer.

Counter: The hardware clock is constructed from a high frequency counter. In some computers, the value of this counter can be read from a device register.

4. Blocking and Non-blocking I/O (or) synchronous & asynchronous:

Blocking I/O: When an application issues a blocking system call;

- The execution of the application is suspended.
- The application is moved from the operating system's run queue to a wait queue.
- After the system call completes, the application is moved back to the run queue, where it is eligible to resume execution, at which time it will receive the values returned by the system call.

Non-blocking I/O: Some user-level processes need non-blocking I/O.

Examples:

1. User interface that receives keyboard and mouse input while processing and displaying data on the screen.

2. Video application that reads frames from a file on disk while simultaneously decompressing and displaying the output on the display