Mohamed Kheider Biskra University
Faculty of Natural and Life Sciences
Computer science department

Course materials

Module: Operating Systems I

Chapter 4: Input/Output Management

Level: 2LMD
Module Leader: Dr. D. Boukhlouf

2024/2023 Academic Year

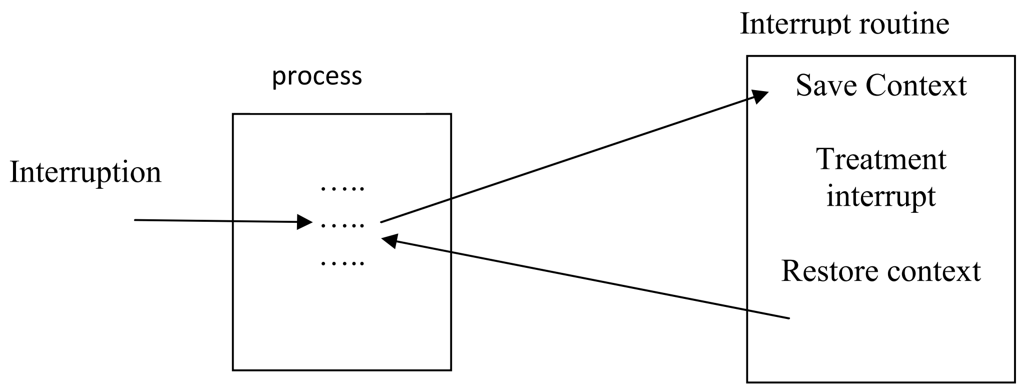
1. Introduction

One of the main functions of an operating system is to control all I/O devices on the computer. It must issue commands to devices, intercept interruptions and manage errors. It also provides a simple interface between the peripherals and the rest of the system. As far as possible, the interface must be the same for all devices (device independence).

2. Interruption mechanisms:

2.1 Definition and Principle:

An interrupt is a mechanism that allows to interrupt the execution of a process following an external or internal event and to pass the control to a routine called "interrupt routine" (dealing with interrupt or interrupt handler).
Interruptions can come from a variety of sources, but they are generally classified into three main types:
External (process independent): operator interventions, failures, etc.
CPU Internal Clamp Diversion: Overflow, dividing by zero, invalid memory accesses, Page Default (causes that cause a disk backup of the memory image) etc.
System calls: such as input-output requests.
When the interruption occurs the processor, after completion of the current instruction, gives control to the interrupt routine associate with the event. It first saves the context of the interrupted process before processing. At the end of the process, the context of the interrupted process is restored, allowing it to continue its execution properly at the place where it was interrupted.



2.2 Interrupt Vector:

When an interrupt signal arrives, it changes the status of an indicator (flag or flag) that is regularly tested by the CPU. Once the signal is detected, the cause of the interruption must be determined. For this we use an indicator, for the different causes of interruption. This indicator is used to access an interrupt vector that links to each interrupt type the address of the corresponding interrupt routine. An interrupt vector therefore has the following structure:

Interrupt N°	Interrupt routine address
0	Adr0
1	Adr1
2	Adr2
3	Adr3
...	...
N	adrN

Example: IBM PC Case:

The IBM PC interrupt vector can have 256 entries each corresponding to a cause of interruption. Interruptions can fall into three categories: diversions, hardware interruptions and software interruptions.
There are ***6 diversions involving:***

- The division by zero
- Step by step operation (an interrupt is generated for each instruction)
- Problems that may appear when accessing memory (page default, for example)
- Reaching a breakpoint
- The digital overflow
- Failure to recognise an instruction

There are ***16 physical interrupts***. They are called IRQ (Interrupt ReQuest) 0 to 15. Some are pre-filled, but many are only assigned to peripheral events when the machine is configured. Here are some of them (Linux system):

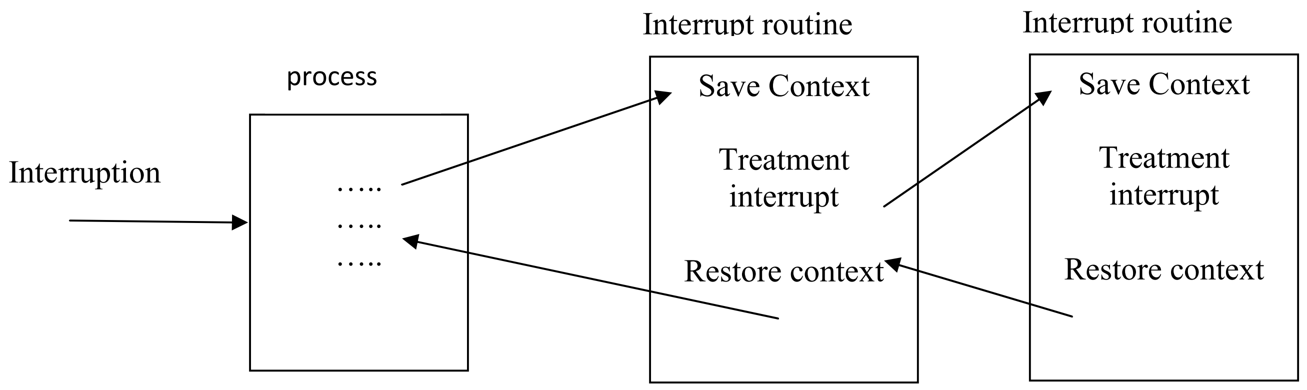
IRQ	Usage
0	system timer (cannot be changed)
1	keyboard controller (cannot be changed)
2	cascaded signals from IRQs 8–15
3	second RS-232 serial port (COM2: in Windows)
4	first RS-232 serial port (COM1: in Windows)
5	parallel port 2 and 3 or sound card
6	floppy disk controller
7	first parallel port
8	real-time clock
9	open interrupt
10	open interrupt
11	open interrupt
12	PS/2 mouse
13	math coprocessor
14	primary ATA channel
15	secondary ATA channel

Software interruptions use the same mechanism, except that they are generated by specific instructions that allow you to call BIOS or operating system functions without knowing their implementation addresses, which may vary from one version to another.

During the occurrence of an interruption, or diversion, the processor receives an index that points to the interrupt vector. This index is generated by the processor in the case of diversions.
It is provided by a motherboard circuit called an interrupt controller in the case of a hardware interruption and by the instruction in the case of a software interruption. In view of this index, the processor accesses the corresponding input of the interrupt table.

2.3 Prioritization and masking of interruptions:

A hierarchy of interruption priorities can be considered. Thus, when several interruptions occur at the same time, the highest priority is taken care of first. This also implies that the routine itself of an interruption can be interrupted by a higher priority interruption (see figure: Cascading interruptions).

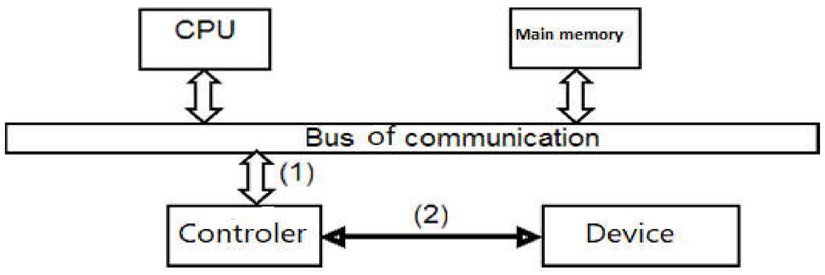


Maskable Interrupt: A supervisor process can hide an interruption. To hide an interruption is to delay its effect temporarily. She can be exposed later.
Non Maskable Interrupt: An interrupt that cannot be disabled or ignored by the instructions of CPU are called as Non-Maskable Interrupt. A Non-maskable interrupt is often used when response time is critical or when an interrupt should never be disable during normal system operation.

3. Physical Aspects of Inputs/Outputs

3.1 Device Controller

Each device is connected to the computer via a special electronic card called a device controller. The following figure illustrates a "classical" architecture of material.



An I/O device actually contains two parts: a device (keyboard, screen, disk, etc.) and a device controller. The controller serves as the interface between the device and the processor: it receives requests from the processor and transforms them into commands for the device, and vice versa, it sends requests from the device to the processor. It also handles transfers between processors and the device. The main requests a

processor sends to a device are read and write. Depending on the device these queries can correspond to simple or complex commands.
There are two main categories of devices considered: character-based devices and block-based devices.
• In a block device, information can only be accessed by blocks and each block has an address (example: the disk).
• In a character-based device, you can of course access character-by-character information (for example: the keyboard) but you cannot specify an address or search for information: you simply receive or send a flow of characters.
A controller generally has:

- a buffer zone,
- control bits: state bit (ready, busy), and command bit (read, write).

3.2 Direct access to memory (DMA)

A "DMA coupler" (DMA: Direct Memory Access) is a particular type of controller that is capable of transferring data directly from memory to a device (or vice versa) without that data passing through the CPU. So the use of the DMA relieves the CPU of an important part of the control and execution work of the I/O. The programming of a DMA coupler is done, as for a controller by writing data in "registers". This time we need at least three components:

- the address of the start of the memory area to be transferred,
- the length of the memory area to be transferred,
- the command word/status.

3.3 Drivers

Each device controller has specific controls. Since an OS designer cannot include all controls for all devices in their software, there are different software layers to interface between the OS and the devices. From the OS's point of view, devices only need to be read or written, which is the highest level interface. Then, the way to access a device differs depending on whether it is a device in blocks (several independently addressable blocks) or characters (character flow); there is therefore a low-level interface for both types of devices. Finally, the lower-level software layer is device-specific: it translates general device commands by blocks or characters into commands specific to the target device. This last software layer is called a driver; it is usually provided by the device manufacturer and is inserted into the OS.

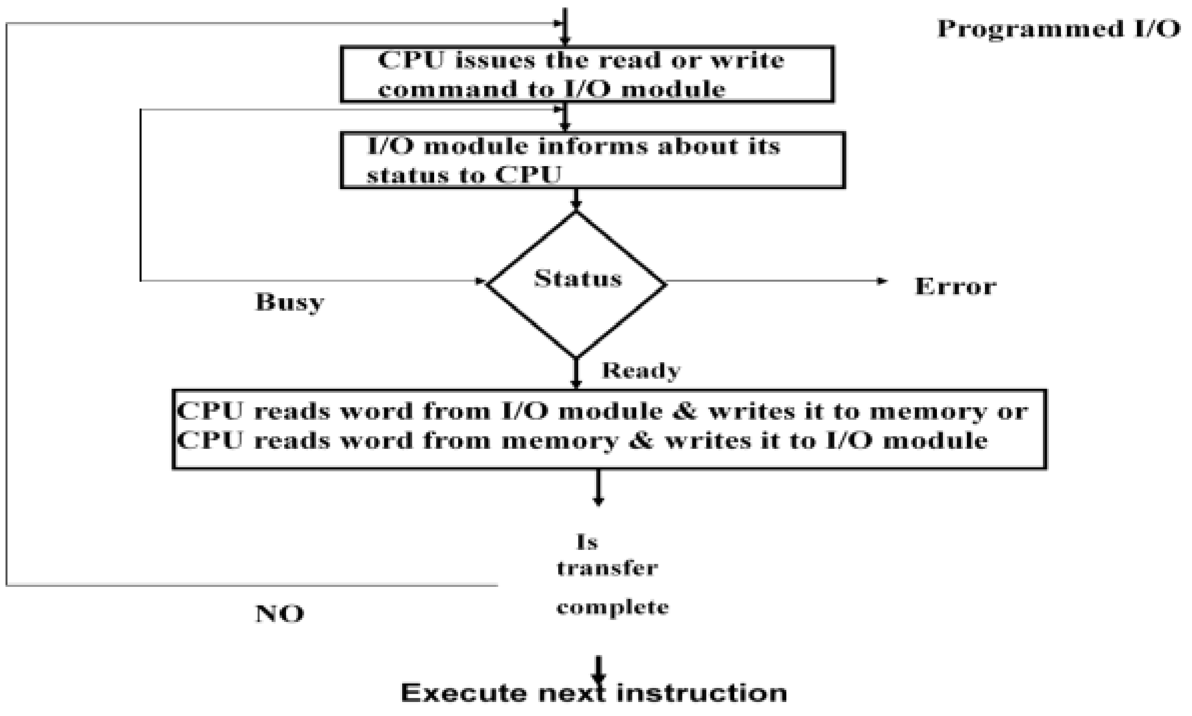
5.3 The different physical input/output modes:

The basic principle used for the exchange of data between two physical constituents is represented by the following figure. Apart from the actual data, two additional links are necessary, to enable the transmitter of the data to signal the actual presence of this data on the corresponding wires, and to enable the receiver to signal that it has read the data.
Several input-output modes have been proposed in computer systems:

- programmed I/O,
- interrupt initiated I/O,
- I/O with DMA access.
- Input/Output Processor

5.3.1 Programmed input-output (with scanning)

The programmed I/O was the simplest type of I/O technique for the exchanges of data or any types of communication between the processor and the external devices. With programmed I/O, data are exchanged between the processor and the I/O module. The processor executes a program that gives it direct control of the I/O operation, including sensing device status, sending a read or write command, and transferring the data. When the processor issues a command to the I/O module, it must wait until the I/O operation is complete. If the processor is faster than the I/O module, this is wasteful of processor time.

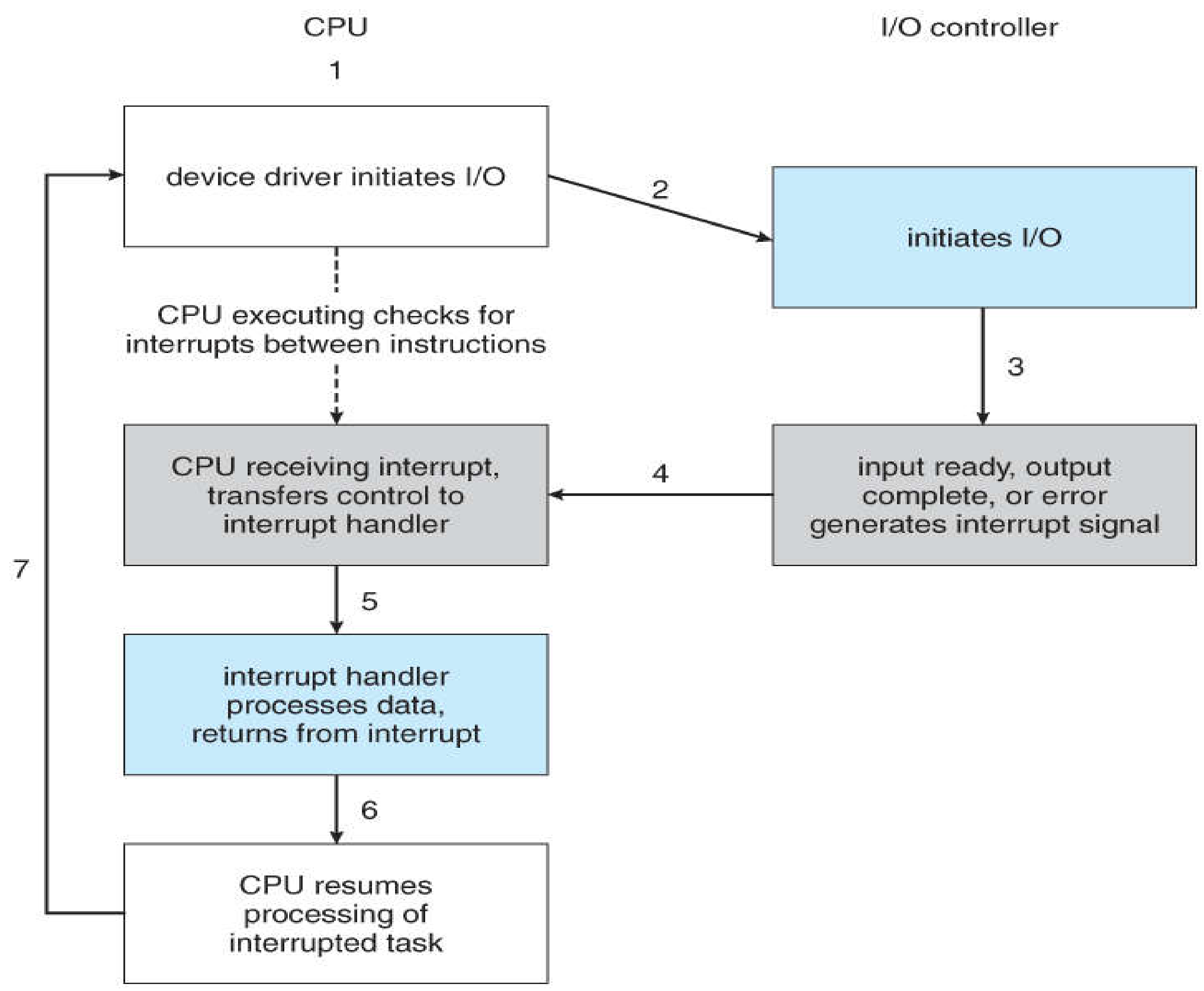

Programmed I/O
CPU issues the read or write command to I/O module
I/O module informs about its status to CPU
Status
Busy
Error
Ready
CPU reads word from I/O module & writes it to memory or CPU reads word from memory & writes it to I/O module
Is transfer complete
NO
Execute next instruction

Critical : With this mode, throughput is often limited to 50 KB/s (Kilobytes per second). Moreover, if the device is slow, the processor is monopolized for the duration of the exchange. In this case, you only read a few bytes at a time to avoid this monopolization. This form of exchange was the only one possible in the first generations of machines. In summary:

- the CPU is slowed to the speed of the device,
- the CPU executes an active standby loop instead of which it could perform calculations on behalf of other programs.

5.3.2 Interrupt initiated I/O

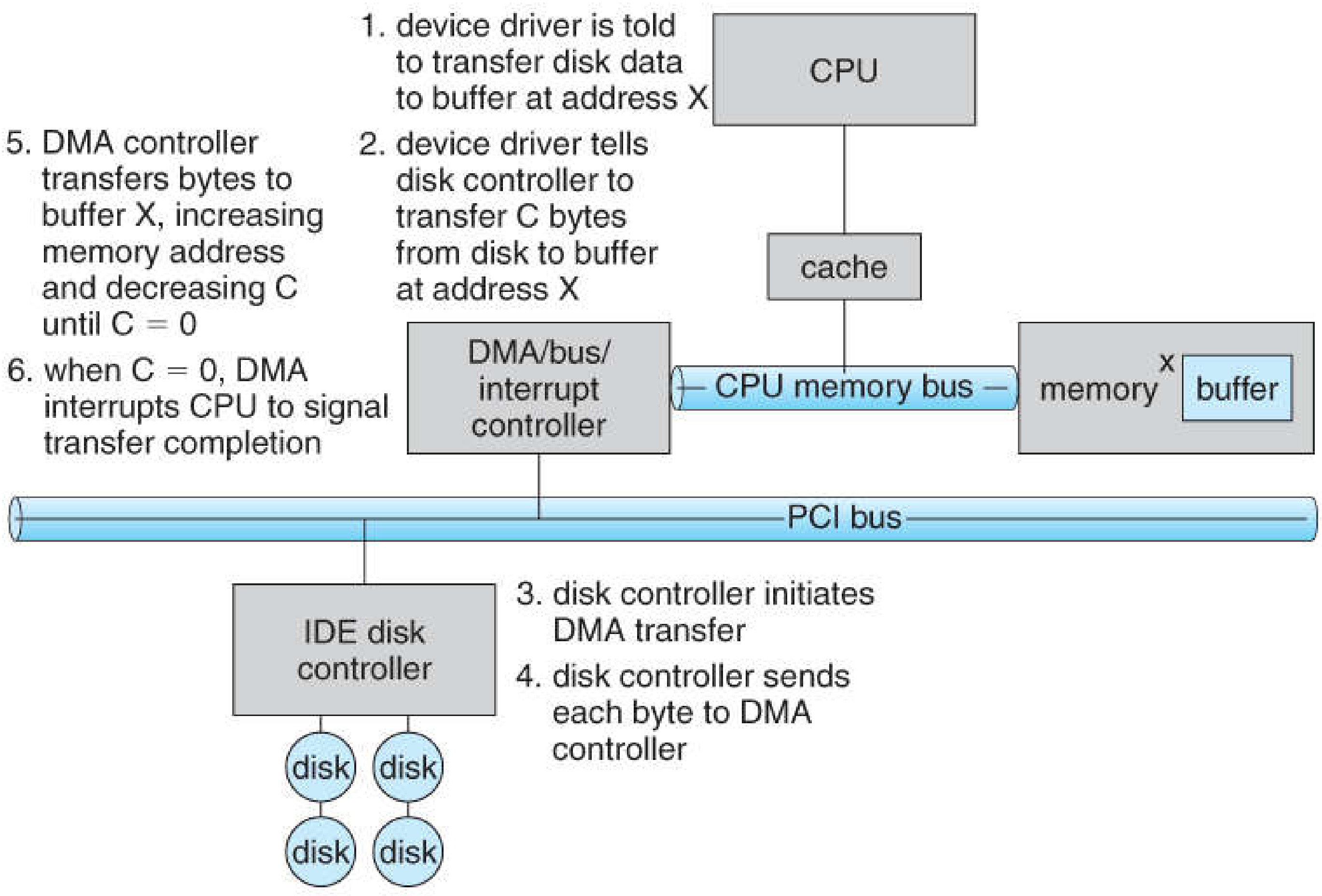
To avoid monopolization of the processor throughout the I/O, a first improvement was made thanks to the interrupt mechanism. The interrupt facilities and special commands inform the interface for issuing an interrupt request signal as soon as the data is available from any device. In the meantime, the CPU can execute other programs, and the interface will keep monitoring the i/O device. Whenever it determines that the device is ready for transferring data interface initiates an interrupt request signal to the CPU. As soon as the CPU detects an external interrupt signal, it stops the program it was already executing, branches to the service program to process the I/O transfer, and returns to the program it was initially running.


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

5.3.3 Direct Memory Input-Output

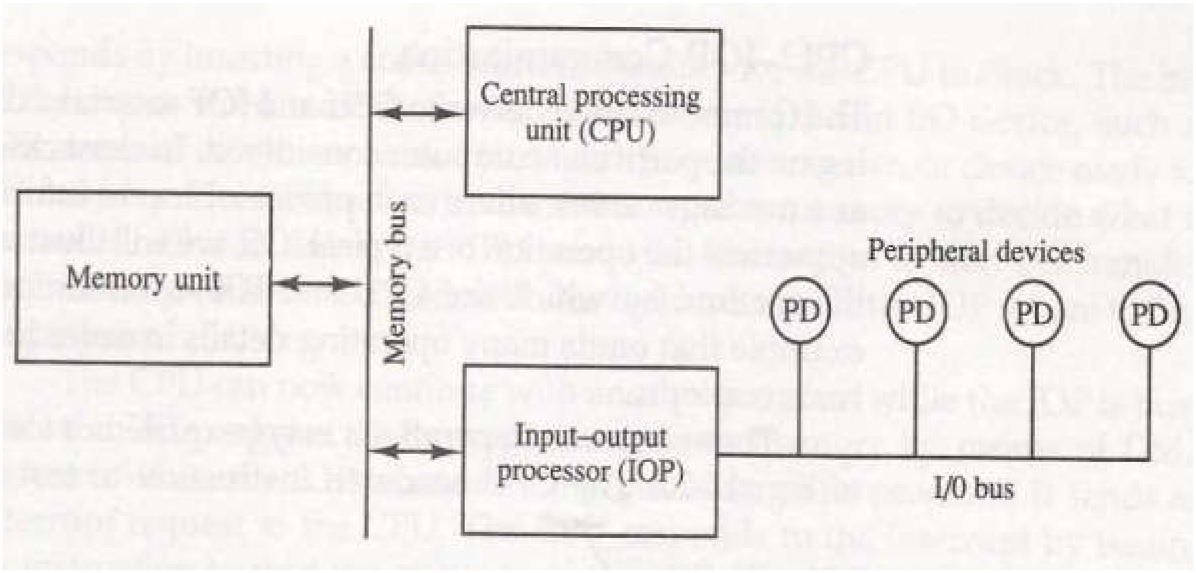
To increase the potential output throughput of the I/O, and decrease the CPU monopolization discussed above, another solution was to shift some functionality into the device that connects the device to the bus, so that it can store data from the device directly in memory in the case of a read, or directly extract that data from memory in the case of a write. This is called direct access to memory.

Steps in a DMA transfer:


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
CPU memory bus
memory
x
buffer
DMA/bus/ interrupt controller
PCI bus
IDE disk controller
disk
disk
disk
disk

5.3.4 Input/output Processor

It is a processor with direct memory access capability that communicates with IO devices. IOP is similar to CPU except that it is designed to handle the details of IO operation. Unlike DMA which is initialized by CPU, IOP can fetch and execute its own instructions. IOP instructions are specially designed to handle IO operation.



5.4 Synchronous - asynchronous inputs/outputs

Depending on the transfer mode used, the input-output mode can be synchronous or asynchronous:
- An I/O operation is said to be **synchronous** if the processor must wait until the end of the I/O operation to continue its processing; it is also said that in this case the transfer is blocking.
- An I/O operation is called **asynchronous** if the processor can start the operation and perform other tasks in parallel. It will be informed of the end of the I/O operation by an interrupt.

5.5 Device drivers

A driver is a program that manages a device. Each controller has one or more control registers. The device drivers send these commands and check their path. The disk driver, for example, is the only part of the OS that knows the registers of a given disk controller and their use. It is the only one who knows the sectors, the tracks, the cylinders, the heads, the movement of the arm, the positioning time of the heads and all the other mechanisms that allow the disc to function properly.
In general, a device driver must handle higher-level queries emanating from the software (independent of hardware) located above it.
The first step for a pilot is to translate an I/O request into concrete terms. A disk driver must, for example, determine where the required block is and check if the disk motor is rotating, if the arm is positioned properly, etc. In summary, it must determine the operations that the controller must perform and the order in which they must be performed. After this step, the pilot completes the controller logs to have these controls executed.