I/O Systems

2 main jobs of a computer are I/O and processing. In some cases, the main job is I/O just like browsing a web page or editing a file where out immediate interest is to read / enter info rather than any computations

The control of devices connected to the computer is a major concern of the OS because I/O devices vary widely in their function & speed.

Various methods are ∴ needed to control them. & these methods form the I/O subsystem of the kernel. This separates rest of the kernel from complexities of managing the I/O devices

A) I/O Hardware

- Device drivers present a uniform device access interface to the I/O subsystem.
- Device communicates with a computer system by sending signals over a cable / through air
- Device communicates with the machine via a connection point / port

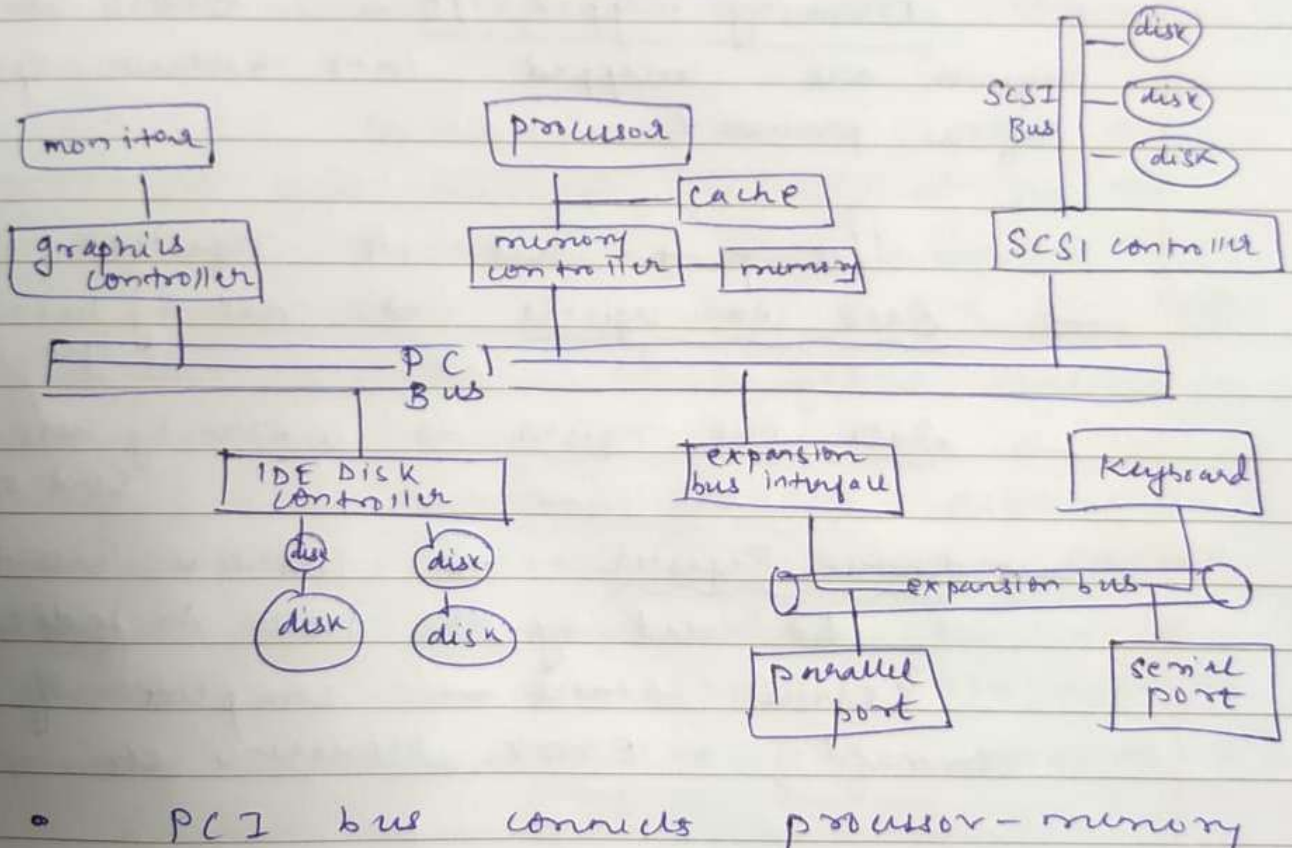
- If devices share a common set of wires, the connection is called a bus

• Bus is basically a set of wires and rigidly defined protocol to specify a set of messages that can be sent on the wires.

• When a device A has a cable that plugs into device B, device B similarly into device C, device C plugs into a port on the computer, this arrangement is called a daisy chain

• Buses vary in signalling methods, speed, throughput, method of connection etc.

PC Bus Architecture



• PCI bus connects processor-memory subsystem to fast devices & an expansion bus connects relatively slow devices like keyboard, serial & USB ports.

• SCSI => Small computer System Interface is also present.

$\boxed{\text{Controller}}$ is a collection of electronics that can operate ~~a~~ a port, a bus / a device.

Eg: Serial port controller is a single chip in the computer that controls the signals on the wire of a serial port

$\boxed{\text{SATA}}$ or Serial Advanced Technology Attachment has a microcode & processor to perform lots of task.

Processor gives commands & data to a controller by some registers & signals

<u>Memory mapped I/O</u> — Device control register are mapped into address space of the processor.

I/O port has 4 registers

→ <u>Data in register</u> → read by host to get I/P

→ <u>Data out register</u> → written by host to send O/P

→ <u>Status Register</u> → Contains bits to be read by the host to indicate several states — completion of command, error occurrence etc

→ <u>Control Register</u> → Written by host to start a command or change the mode of a device

Polling => Same a busy waiting of synchronization.
It is a loop, reading the status register over & over until the busy bit becomes clear.
Too much time taking.

Interrupt => hardware controller to notify CPU when a device is ready for service.
It prevents the CPU to poll repeatedly for an I/O completion.
CPU hardware has a wire called interrupt request line which the CPU senses after executing every instruction
When CPU detects that a controller has asserted a signal on the interrupt request line, CPU performs a state save & jumps to the interrupt handler routine at a fixed address in memory
Interrupt handler determines the cause of interrupt, performs desired processing, performs state restore & executes a return from interrupt to where CPU was executing prior to interrupt

1) Device controller raises an interrupt by asserting a signal on interrupt service line
2) CPU catches the interrupt & dispatches it to interrupt handler
3) handler clears interrupt by servicing the device.

→ BASIC INTERRUPT HANDLING

MODERN INTERRUPT HANDLING

It needs
- Ability to defer interrupt handling during critical processing
- Efficient way to dispatch to the proper interrupt handler for a device without polling all devices to see who raised the interrupt
- multi-level interrupts for OS to distinguish betn high & low priority interrupt & responds as per degree of urgency.

This additional features are provided by CPU & interrupt controller hardware.

TYPES OF INTERRUPT REQUEST LINES

- Non maskable — extremely urgent events like unrecoverable memory errors ∴ CPU must handle them asap
- maskable — it can be ~~turned off~~ delayed/ turned off by CPU before executing critical instructions that must not be interrupted.

Interrupt Vector :- Has memory address of specialized interrupt handlers.

Interrupt chaining :- Each element in the interrupt vector points to the head of a list of interrupt handler.

Interrupts also help to handle exceptions

Direct memory Access (DMA)

To enhance CPU utilization, it is better not to involve CPU in I/O operations ∴ CPU delegates another controller to do this operation & in the meantime it itself performs some other crucial task.

This special controller is the DMA controller which does the necessary data transfer bet^n the I/O device & memory

DMA needs 4 parameter

→ Source address	:	from where to read data
→ target address	:	Where to write data
→ byte count	:	how much data in bytes
→ operation	:	Whether to read / write

6 steps in DMA transfer (Figure 13.5 in book)

• To initiate a DMA transfer, the host writes a DMA command block in the memory.

• This block has a pointer to the source of transfer, target address, byter count & operation (basically the 4 parameters)

• CPU writes the address of this command block to DMA controller & goes on with other work.

• Communication / Handshaking bet^n DMA controller & device controller is via DMA request & DMA acknowledgement

- Device controller places a signal on DMA request line when a data is available for transfer.
- This signal causes the DMA controller to seize the memory bus, place desired address on memory address wire & place a signal on DMA acknowledge wire.
- When device controller receives the DMA acknowledgement signal, it transfers the word of data to memory & removes the DMA request signal.
- When entire transfer is done, DMA controller interrupts the CPU.

<u>Cycle stealing :-</u> When DMA controller seizes the memory bus, CPU is momentarily prevented from accessing main memory although it can access cache memory.

B) <u>Application I/O Interface</u>

- A processor / CPU is connected to its peripheral device through an interface
- Basically it acts as a translator or Intermediate
- CPU is pretty fast & I/O are slow so this balancing / synchronizing is via an interface. It takes i/p from I/O serially & sends it to CPU parallely

- It also helps in signal conversions and data format matching of different I/O devices & CPU.
- Each peripheral devices has different responsibilities & working. So again the interface provides some commonality amidst these.