USB

Basic Objective behind the design of this USB is Unified Low,Medium and High Speed into a Common Bus.

USB is a peripheral BUS it is not a controller bus , USB controller is also reffered as a Host controller.

BUS: Buses where designed based on the demand of the devices ,
some device are Low Speed some device are high and Medium speed.
- Low Speed device – UART
- Medium Speed devce – Parallel
- High Speed device – PCI [Peripheral Component Interconnect]

In PCI :
- No hot plug
- No -autoconfiguration
- Limited System resources [IO addr,, and Non-Shared irq lines]
- Limited standrad connectors

To Over come thouse drwbacks we Use USB

=>Address : We have limited IO address space and limited virtual address, since the device come its OWN registers , we need to allocate some system resources for the device to work.
=> IRQ line : we have limited no.of irq lines we need to engage the one of the IRQ line for device [like shared irq].

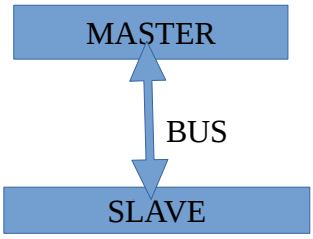
=> Physical level: the connections are limited [More device of same type can not plu-in a same point of time]

“USB is the solution for all the above problems”
USB is One Bus One Connector type
- we can connect the Low , medium and High speed device to same bus.

USB device does’t take the address from the processor and irq also.
[thre is no such thing called allocating the IO address, allocating memory address or IRQ line to USB device]

Key Benifits:
=> USB device are Hot Pluggable
- uesr need to configur it.
=> USB device don’t consume the system resources
=> 127 devcie connected [including the HUB]
=> Low , Medium and High speed devices is connected to same USB port.
[all HID [Human Interface Devices] are Low speed:- keyboard,touchpad]
=>No IRQ --- No ISR rotiune/*handlar*/function.

USB deviec has 3 Entities:--



1.Master – Slave Bus
2.Slave device
3.USB Host Controller [Master]

This 3 entities make up the entair USB system.

[Which you are connecting that Interface is internally connected to a BUS , which is connected to Controller and Controller internals connected to a Processor .]
[Host Controller job is to drive all command (control commands) , all data adress to BUS , **Slave** Controller will responds to the requests which are sent by the Master]

Two kinds of Specifications for Implimention of Controller
1- Host controller
2- On The Go controller [It can be a Master it also can perform the services of a Slave depending on the connection Eg: Mobile Phones]
[this two are act as Master]
3- Device controller [Slave]
[This 3 Different spcies contain for building up a circuit called CONTROLLER]

USB Device Classes:
1.Audio
2.Communication [Modem]
3.HUB
4.HID
5.Printer
6.Mass Storage Classes
Vendor Specific Classes:
- Scanner, Video,Ether-net , Serial connectors

Q.when ever the Device changes the state , How will Software known about it and what are the ways?
There are Two ways:
1.polling
2.interrupt.

Q.How the polling work?
Keep looking into some registers to watchout for State Change (status registers).
Q.How does Interrupt will work?
Devcie will trigger a signal when ever it under goes in State Change .

Q.How will we write driver to know what the H/w is doing . [when USB device dont have I/O addr , IRQ. How will driver Shut Down State change.
[**Network logic]
->USB driver are more like Networking

```
 if((actual_len != 0) &&(actual_len != 12))
                {
                   if(fflag == 1){ current_pos = current_pos+12; actual_len = actual_len-12;}
                   if(actual_len <= temp_len){
                      len = vfs_write(fp, (char *)(tail->dp_urb->transfer_buffer)
+current_pos,actual_len, &fp->f_pos);
                      if (len < 0) {
                         printk(KERN_ERR "vfs_write failed\n");
                      }
                      total_len += len;
                      current_pos = current_pos + len;
                      fflag=0;
                   }
                   else {
                      while(actual_len != 0) {
                         len = vfs_write(fp,(char*)(tail->dp_urb->transfer_buffer)
+current_pos,temp_len, &fp->f_pos);
                         if (len < 0) {
                            printk(KERN_ERR "vfs_write failed\n");
                         }
                      total_len += len;
                         actual_len = actual_len - len;
                         if(actual_len < temp_len)
                            temp_len = actual_len;
                         current_pos = current_pos + len;
                      }
                      fflag=0;
                   }
                }
                   else if(actual_len == 12)
                      fflag=1;
             }
             set_fs(old_fs);
```