# 【OS】Day38

| | |
|---|---|
| Class | Operating System: Three Easy Pieces |
| Date | @February 17, 2022 |

## 【Ch36】I/O Devices

### 36.1 System Architecture

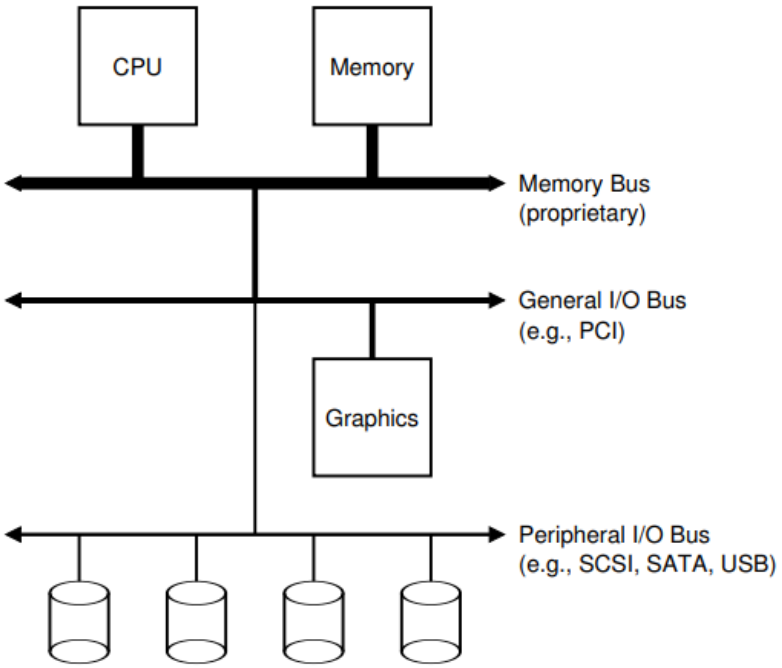



Figure 36.1: **Prototypical System Architecture**

The picture shows a single CPU attached to the main memory of the system via some kind of memory bus or interconnect.

Some devices are connected to the system via a general I/O bus, which is many modern systems would be PCI. Graphics and some other higher-performance I/O devices might be found here.

Finally, even lower down are one or more of what we call a peripheral bus, such as SATA, SCSI, or USB. These connect slow devices to the system, including disks, mice, and keyboards.

Modern systems increasingly use specialized chipsets and faster point-to-point interconnects to improve performance.

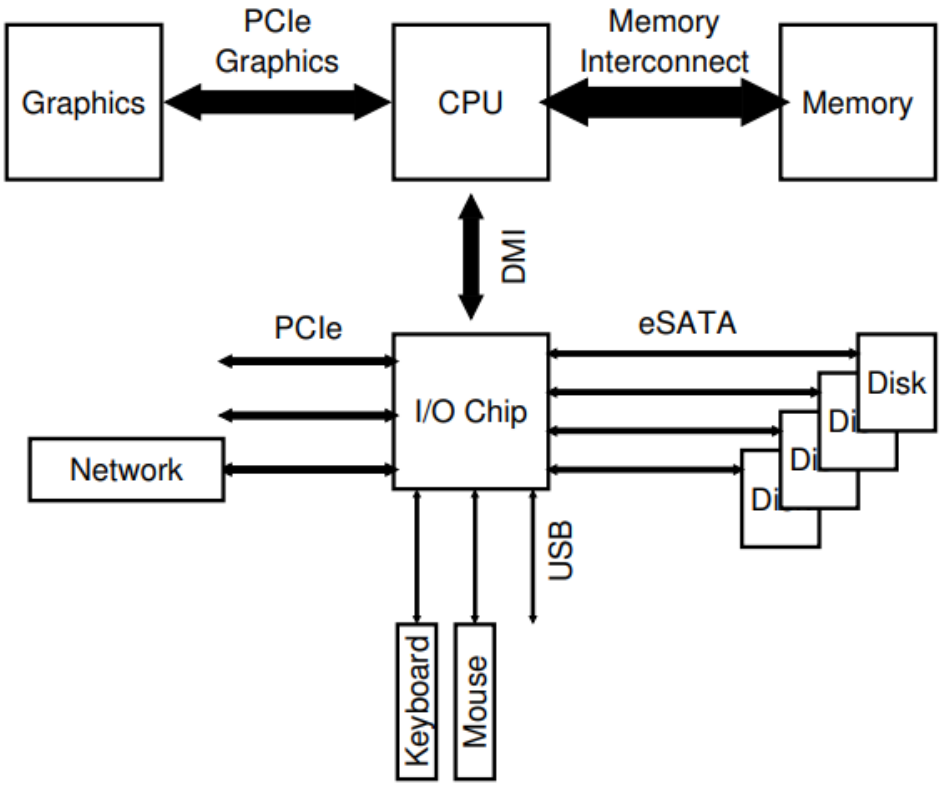



Figure 36.2: **Modern System Architecture**

The figure above has a high-performance connection to the graphics card(for gaming) and memory.

The CPU connects to an I/O chip via DMI(Direct Media Interface), and the rest of the devices connect to this chip via a number of different interconnects.

On the right, one or more hard drives connect to the system via the eSATA interface.

Below the I/O chip are a number of USB(Universal Series Bus) connections, which in this depiction enable a keyboard and mouse to be attached to the computer. On may modern systems, USB is used for low performance devices such as these.

Finally on the left of I/O chip, other higher performance devices can be connected to the system via PCIe(Peripheral Component Interconnect Express).

## 36.2 A Canonical Device

Let us look at a canonical device.

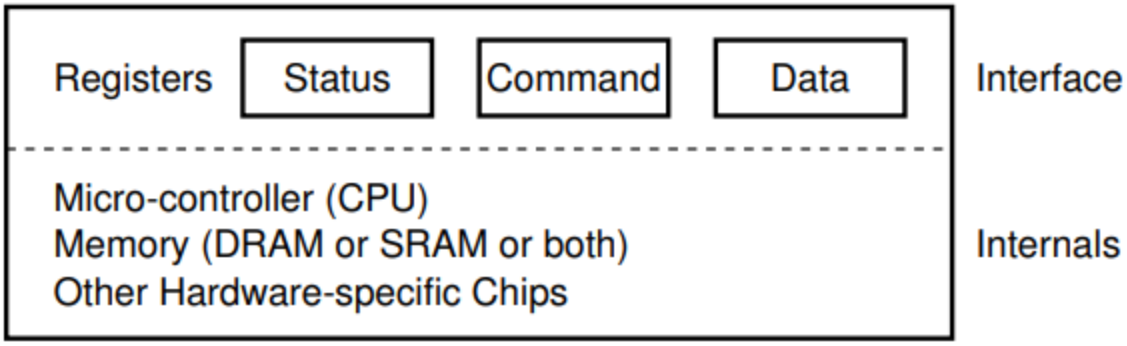



Figure 36.3: **A Canonical Device**

A device has two important components.

The first is the hardware interface it presents to the rest of the system. Hardware must also present some kind of interface that allows the system software to control its operation.

The second part is its internal structure. This part of the device is implementation specific and is responsible for implementing the abstraction the device presents to the system.

## 36.3 The Canonical Protocol

In the device, the simplified device interface is comprised of three registers: a status register, which can be read to see the current status of the device; a command register,

to tell the device to perform a certain task; and a data register to pass data to the device, or get data from device.

Let us know describe a typical interaction that the OS might have with the device in order to get the device to do something on its behalf. The protocol is as follows:

```
While (STATUS == BUSY)
    ; // wait until device is not busy
Write data to DATA register
Write command to COMMAND register
    (starts the device and executes the command)
While (STATUS == BUSY)
    ; // wait until device is done with your request
```

The protocol has four steps:

1. In the first, the OS waits until the device is ready to perceive a command by repeatedly reading the status register; we call this polling the device.
2. Second, the OS sends some data down to the data register. When the main CPU is involved with the data movement, we refer to it as programmed I/O(PIO).
3. Third, the OS writes a command to the command register; doing so implicitly lets the device know that both the data is present and that it should begin working on the command.
4. Finally, the OS waits for the device to finish by again polling it in a loop, waiting to see if it is finished.

## 36.4 Lowering CPU Overhead With Interrupts

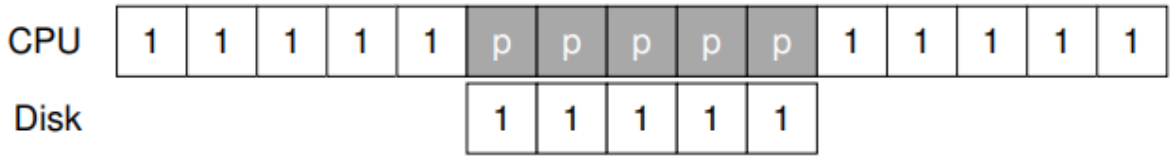
The invention that many engineers came upon years ago to improve this interaction is the interrupt.

Instead of polling the device repeatedly, the OS can issue a request, put the calling process to sleep, and context switch to another task.

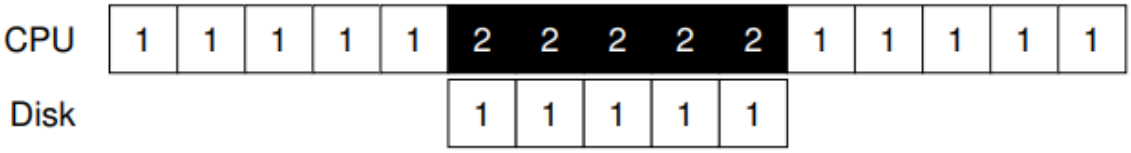
When the device is finally finished with the operation, it will raise a hardware interrupt, causing the CPU to jump into the OS at a predetermined interrupt service routine(ISR)

or more simply an interrupt handler.

Interrupts thus allow for overlap for computation and I/O, which is key for improved utilization. This timeline shows the problem:




If instead use interrupts, the OS can do something else while waiting for the disk:




However, interrupts are not always the best solution. If the I/O operations is very fast, it might be better to just let the CPU poll a while. It may be best to use a hybrid that polls for a while and then, if the device is not yet finished, uses interrupts.