Tutorial 8 - IO

Gidon Rosalki

2025-05-21

Notice: If you find any mistakes, please open an issue at `https://github.com/robomarvin1501/notes_operating_systems`

1 Introduction

IO can be categorised into 3 groups, depending on what they provide:

- Input only: keyboard, mice, remote controller.
- Output only: most of the displays, headphones.
- Both: disk, network card.

The OS has control of these devices, handling commands, interrupts, and errors.
The OS supplies abstractions for these devices, by hiding implementation details, and error handling from users. The OS should allow utilising IO devices, and the CPU at the same time. It should also allow parallel or concurrent access to devices and protect an access if necessary.

The OS manages the IO with respect to a few things:

- Scheduling: The OS can schedule operations, such as disk accesses, which can affect the device structure, and affect performance.
- Synchronisation: Accessing devices can cause trouble if done concurrently, so the OS handles this too
- Error handling: An IO request might fail, due to either transient or permanent reasons (failed memory, cosmic ray, etc.)
- Buffering: Might be useful to postpone or delay requests, to minimise the number of accesses in expensive cases

2 Hardware

2.1 Bus

One way to connect the IO devices is using bus. This is a communication system that covers multiple hardware devices, and also connects to the CPU, and thus allows data transfer between the CPU, and the IO devices.

This has the advantages of being easy to connect new devices, and hardware migration is easy if two computers use the same bus standard. It is also wonderfully cheap. Unfortunately, it is not amazingly scalable, and is a bottleneck, since a bus can only be of a certain size. Additionally, it should keep working for all the devices, but this generality can hurt it being specifically better for certain things.

Devices are connected to a computer using ports. The serial ports allow a transfer of one bit at a time, where parallel would allow multiple bits at a time, through multiple data lines. USB (Universal Serial Bus) is a defined standard for all connections (with many different types).

2.2 Controllers

Most IO devices have controllers, which are often designed as chips or cards, allowing the device to be plugged into them. The interface between the controller and the device is usually standardised (even if it is not officially). For example, in hard disks, the controller should know how to insert/retrieve data at a given address, and convert an address to a specific location on the disk (be that cylinder, section, plate, etc.). Sometimes there might be hardware issues, that the controller will handle transparently to the user.

3 Software

3.1 Drivers

The controller is the **hardware** performing as an interface for the device. However, as the OS, how do we know how to talk with the controllers? The OS uses abstract interfaces that each device manufacturer should implement. They implement the interface in a program named a **driver**.

Drivers are programs that allow the OS to communicate with the controllers. They are considered trusted, and run in kernel mode. Since there are lots of devices out there, far more than OSs, the drivers match themselves to the OS, and support a known common standard interface. So instead of the OS having a different driver for SATA disks, USB disks, and SCSI disks, it has a standard disk driver, and the devices match themselves to that.

4 CPU and IO

4.1 Reminder

Polling and interrupts are both ways to wait for an event from an IO device:

- Interrupts: The device sends a message when it finishes, and the CPU must stop and deal with it
- Polling: The CPU keeps checking relevant registers of the device control, to see if it has finished

Interrupts are the common approach here, since this prevents busy waiting, and allows the CPU to be busy on other stuff.

It is important to remember the process for interrupts:

1. The CPU sets relevant controller registers, according to what operation needs to be done
2. The CPU sends the controller a command
3. The device executes the command
4. The CPU keeps working, until an interrupt is received, at which point it will handle it
5. The CPU will check for correctness, since something being incorrect will require different branches
6. Save the result (if necessary, on the main memory)

However, the intervention of the OS in step 6 is unnecessary, thanks to Direct Memory Access (DMA), meaning that the device can write to a dedicated memory location directly, without the OS or CPU.

5 Storage

5.1 Hard Disk Drives (HDD)

These are non volatile memory storage devices (meaning they contine to store data after power loss). They were introduced by IBM in 1956. The basic transfer units are called **sectors**, and a rotating magnetic arm navigates to tracks on the spinning platters, in order to read sectors. They are incredibly sensitive, scratches, dust, and bumps are liable to destroy the data.

5.2 Solid State Drives (SSDs)

SSDs use the memory storage technique, also called flash memory. The data is stored in cells, and an electric charge inside those cells. Changing the charge, means changing the bits. They are built as a 3D structure of cells, with lots and lots of cells.

The SSDs are the common memory storage technique today, since they are so much faster than HDDs, and are more durable, thanks to having no mechanically moving components. Due to the increased durability, we tend to use SSDs in telephones and laptops. However, they are more expensive per gigabyte, but in places where quantity of data is not the most important thing, but rather continued fast operation, this is a reasonable trade off.

5.3 Performance

Compared to the CPU, the disk latency is very high. Seek time is in the order of magnitude of milliseconds. We can increase the performance by allowing reading to take place in parallel. We store the data spread across several disks, which is cheap to do, since disks are individually relatively cheap. In 1998, RAID was introduced to improve disk performance and reliability. RAID stands for Redundant Array of Independent Disks.

5.3.1 RAID 0

In RAID 0, we have several disks, with data spread across all of them. Each portion (on one disk), is called a strip, and we can read a file stripped across the disks from all the disks at once, and two processes writing different data can write simultaneously. If one disk fails, we can still access a partial amount of the data.

5.3.2 RAID 1

In RAID 1, we mirror each disk to a second disk. This way, if one disk fails, we still have the data duplicated in a second location. Also, we can now read from two locations at once, and 2 different processes can read the same data, at the same time.

5.3.3 RAID 4

However, this can be seen as a bit much, to copy **all** the data. In RAID 4, all the disks, except for one, contain the data. The remaining disk is called the ”parity disk”. It contains the XOR of the data from all the other disks. This way, if one disk fails, we can use the parity data to restore the lost data to a new, replacement disk.

5.4 Partitions

We can divide the disk into separate, independent portions, called partitions. Each partition, can host a different file system. This is an example of virtualisation, even though each one of the partition is treated as an independent disk, and each one is considered as a separate storage device, they are all part of the same device. The partitions are listed in the disks partition table, which is also stored on the disk. The partition info contains the sector in which it starts, and the length.

6 User Interface Devices

6.1 Keyboard

Each key is associated with a number, generally called the **scan code**. This is **not** the ASCII code! The scan code relates to a *physical* key on the keyboard, rather than a letter, so the same scan code is sent when typing ”a”, and when typing ” ”.

Although the keyboard and monitor are different independent devices, we have become very used to having the character we type appear on the screen as we type them. This is called echoing, which is quite complicated, since the user might type as an output is printed. However, this will be glossed over in this course.

Each time a key is pressed, or released, an interrupt is generate (or poll message in modern computers, PS2, not the console, was the last standard to use interrupts). The information about the key is extracted by the keyboard driver, using the IO port associated with the keyboard. Everything else is the software’s responsibility, including lower/upper case, or the language.

6.2 Mouse

Mice used to use interrupts, but these days generally use polling. A mouse message is generated when a mouse buttwon was clicked, or when the mouse was moved ”significantly”. For example, this significant distance might be at least 0.1mm. Sometimes, this unit is called a mickey. This message includes Δx, Δy, and button status. The OS then interacts with the GUI to notify it. The GUI manager keeps track of what window is currently active, and processes the click/cursor move events.

7 Booting

Sector 0 of the disk is called the Master Boot Record. This sector contains a short code for booting the ocmputer, and the partition table of the disk at the end of the sector. This is also called GPT (GUID Partition Table). At least one partition should be active, then the BIOS reads this partition. The BIOS (Basic Input-Output System) is firmware. It comes with the motherboard, and has nothing to do with the OS.

7.1 Boot loaders

The MBR is small, only one sector, so the boot loading code should be very short. Long MBRs are often split into pieces, where one fits into the boot sector, and knows how to load (bootstrap) the rest of the sectors. The boot loader is responsible for loading the kernel code into memory, as well as configuring it, and starting the kernel. The kernel is often stored in files, but neither the CPU or BIOS can understand the abstractions of files, so the boot loader has to somehow access those files.

If more than one partition is enabled (different OSs in different partitions), then you can choose the partition you want to execute. In the BIOS, you can change the default one, or change their priorities, such as what happens when one is disabled.