Devices and Disk Drives

Devices

We won't go into a whole lot of detail in this section. We'll mostly focus on some important vocabulary.

Devices are things like keyboards, mice, hard drives, monitors, etc. Data is sent back and forth from the devices to the computer via a data line called a *bus*. Systems have several kinds of buses with names like SCSI, PCI, SATA, and USB. You know USB (Universal Serial Bus) well as it's used to connect all sorts of external devices. You may also have run across SATA when connecting disk drives.

Buses are of varying speeds, with the faster ones being used for shuttling information from the CPU to RAM and the GPU. High speed buses need to be short partly because they are expensive and partly because the speed of light actually acts as a practical speed limit. For instance, things at the CPU level are measured in nanoseconds, and light/electricity cannot travel more than one foot in that amount of time.

Most devices have *firmware*, which is software running on the device itself to help it do what it does.

Device communication

The OS communicates back and forth with the devices. Typical devices have various registers, such as status registers that the OS can use to get info about the device's state, and command registers that the OS can use to tell the device to do things. Devices also often have a data buffer. Each device has its own particular interface consisting of these things. Computer users and programmers don't want to have to deal with all the possible different interfaces, so the OS takes care of it. The specific OS code that handles this is called a *device driver*. The majority of an OS's code consists of device drivers. Usually they are separate from the kernel. They are often written by the device manufacturer. But the manufacturer probably won't make drivers for every niche OS out there, so sometimes they are written by others.

The two key concepts are *polling* and *interrupts*. To make an analogy with threads, polling is like spin locks and interrupts are like futexes. With polling, the OS sends a command to a device and sits in a while loop continually checking the device's status register waiting for a change. This is okay if only done for a short while, but otherwise it's slow and wasteful.

With interrupts, when a device finishes something, it sends a signal along a bus. That signal is picked up by an interrupt controller chip that sits on the motherboard. That controller may send the interrupt right away or it may send it later if there are other more pressing interrupts. That interrupt causes a trap, just like with context switches (when the context switch timer goes off, that actually generates and interrupt). The running process is stopped and predefined interrupt handling code from the OS's trap table is run.

For the OS to send info to a device, there are two common options. The first option is special CPU instructions, only accessible in kernel mode, whose job is to send and receive data from devices. The second option is called *memory mapping*. This is where the device's registers are mapped to specific memory addresses. Reading and writing to the device corresponds to reading and writing to those memory locations.

Often there is a lot of data to be sent to a device, and it would be a waste of the CPU's time to be doing the writing itself. When the CPU does the writing itself, that's called *programmed I/O*. The alternative is called *direct memory access*. This is where there is a specific device that sits on the motherboard whose job it is to do the writing. The OS points it to the data in memory and then the device does the copying. This frees up the CPU to do other things.

Disk drives

There are two main types we are concerned with here: Hard disk drives (HDDs) and Solid state drives SSDs). HDDs date back to the 1950s. They are mechanical devices that look and work a little like a record player, except that the data is stored via magnetism. SSDs are purely electronic devices that use flash memory to store data. They started becoming widely available in the 2000s.

Pros and cons SSDs have a lot of upsides over HDDs and not many downsides except for two big ones: price and capacity. Though prices have been coming down, SSDs are still around 2 to 3 times as expensive per gigabyte than HDDs. And if you need a large drive, like 10 TB, then it will probably have to be an HDD. Huge SSDs do exist, but they are neither cheap nor common.

The big advantage of SSDs over HDDs is speed. To read from a random location on an SSD is about 50 to 100 times faster than an HDD. The slowness has to do with the fact that the HDD is a mechanical device, which we'll cover more in a bit. Reading bytes sequentially on an SSD is also faster, but only about 5 to 10 times faster. One practical consequence of this is systems where the OS is stored on an SSD boot up considerably faster than with an HDD. Sometimes computers will have one of each kind of drive. The SSD is used to hold things like the OS that users want quick access to, and the HDD is used for bulk storage of things like photos or videos.

Because HDDs are physical devices, they are more susceptible environmental factors like temperature and pressure (they don't work quite right at high altitudes), being dropped, and being exposed to magnets. SSDs, being electronic, are much less affected by these kinds of things.

Both SSDs and HDDs wear out over time. Neither seems to have a particularly strong advantage over the other in this regard.

Mechanics of an HDD

An HDD is a mechanical device that looks a lot like a record player. The main parts are the *platters*, the *controller arm*, and the *read head*. The platters are round disks that store the data. They are coated with a magnetic material that is used to store the information. Just like a record player, the platters are continually spinning.

The arm moves into position over the disks, and the read head, which is attached the arm, is used to read and write data on the platters. There are often several platters and read heads. The platters are broken up into pieces called *sectors*. Sectors are typically 512 bytes in size, but can be larger. Writes to a sector are atomic, meaning either the whole sector is written or none of it is.

To read and write on an HDD, we first have to position the read head to the right distance over the platter and then wait for the appropriate area of the disk to rotate around. This is called a *disk seek*, and it is the slowest part of an HDD. It typically takes a little under 10 ms to position the read head and then about 4 ms on average to wait for the right area to rotate around. (The platters typically rotate at anywhere from 4000 to 15000 RPM. At 7200 RPM, a half rotation takes 4 ms.) However, once the read head is in position, then reading things sequentially on the disk from there is considerably faster.

Disk scheduling algorithms

Because moving the read head on an HDD is so time-consuming, various algorithms have been developed to optimize the motion. In particular, if we have several disk writes queued up to sectors all over the disk, we can choose the order to do them so as to limit the amount of repositioning time. Here are some of the algorithms.

1. FIFO — We've seen FIFO algorithms several times before in other contexts. Here, the FIFO repositioning algorithm makes no effort to reorder things. It just takes them as they come. It's simple to implement, but inefficient.
2. Shortest seek time first — This algorithm always chooses the next thing to write to be the one that requires the arm to move the least amount.
3. Elevator (SCAN) — In this algorithm, the arm continually moves from the inside of the disk to the outside and back, sort of like an elevator that goes from the ground floor to the top and back again. The algorithm services the disk writes at each level.

4. C-SCAN — This is a variation of the Elevator algorithm in which the arm moves from one side to the other and when it reaches the end it jumps back to the start. This is sort of like an elevator that goes from the ground floor to the top floor, opening the doors at each floor, and when it reaches the top it goes back to the ground floor without making any stops.
5. Shortest positioning time first — This algorithm takes into account the fact that a disk seek involves both positioning the arm and waiting for the sector to rotate around. Sometimes, a disk write might require moving the arm very little but having to wait for almost a complete rotation. This could be less efficient than moving the arm a little further for another disk write where the rotation wait is much shorter. Knowing exactly the rotation position is usually behind the OS's capability, so this is done in the hard disk firmware.

For example, suppose we have the following sequence of disk writes to be done: [45, 60, 15, 95, 52]. The numbers indicate the positions on the disk of the writes in terms of arm distance from the center. If the arm is currently positioned at 50, what sequence of writes will FIFO, Shortest seek time first, the Elevator algorithm, and C-SCAN do?

1. FIFO — This is simple as it doesn't do any reordering. It just does 45, 60, 15, 95, and 52, as given.
2. Shortest seek time first — The arm is currently at 50, and 52 is the closest of all the values, so it does that first. So now the arm is at 52. Of the remaining values, 45 is the closest, so it does that. With the arm now at 45, we can go to 60, 15, or 95. The closest is 60. From 60, the closest is 95, and then 15 is last. So the order is [52, 45, 60, 95, 15].
3. Elevator — Let's assume the elevator is currently moving outward, from lower to higher numbers. Being at 50, the next one we meet is 52. After that we meet 60 and then 95. After this, it will turn around and start going back the other way toward lower numbers. It will first meet 45, and then 15. So the final order is [52, 60, 95, 45, 15].
4. C-SCAN — We'll assume the elevator only moves from lower to higher values. Starting at 50, it goes to 52, 60, and then 95. At this point it goes all the way back to 0 and starts scanning up again, reaching 15 and then 45. The final order is [52, 60, 95, 15, 45].