IFT-2245 | General notes

Tarik Hireche : 202 301 89

18. January 2026

Rapport écrit par Tarik Hireche

1 The foundation : Hardware

Before even jumping into OS theory, one should understand what is it built upon, right?

By default, a computer is mainly a CPU, the central processing unit. It is the brains of the computer, it sequentially fetches, decodes and executes instructions… until it shuts down.

The CPU lives in it’s own bubble (RAM + the CPU itself).

1.1 The problem : the outside world

1.1.1 a full hands on computer needs to interact with external stuff, the peripherals

We’re talking about your keyboard, your hard disk drive, SSD, network etc.

1.1.2 The speed choc: The lambo against the snail

The CPU is blazingly fast, we’re talking nanoseconds fast. In fact, it is so fast that it has to wait for its inside and outside neighbours.

Its outside neighbours (the peripherals), are extremely slow compared to the CPU (milliseconds or even seconds).

Therefore, if the CPU needs to read a specific data on your HDD to continue its computing process, there’s a huge, HUGE temporal gap between the two.

1.1.3 A solution but a bad one: Polling

Before the invention of the concept of interruptions, we literally used a super loop:

An always true loop that gets triggered when the CPU orders the disk to read a specific data. After the order, the CPU would literally ask in a loop :

- „Is it over?“ -> Nope.
- „Is it over?“ -> Nope.
- „Is it over?“ -> Nope.
- „Is it over?“ -> Nope.

1.1.4 It is absolutely catastrophic. Why?

Because, the most precious resource in a computer is the CPU’s time. With active polling, the whole brain is freaking stuck waiting for the snail to finish its walk. Isn’t that a huge waste of potential? **The whole system is frozen, waiting**

1.1.5 A solution that revolutionized computers: Interruption mecanisms

In order to free our blazing fast lambo of this constant surveillance, engineers added a physical component, **a literal wire, that goes from the CPU**, to the device controler.

So **interruption** is **NOT a software concept**, it is **material, physically implemented**. It can be seen as an electric bell at the CPU's door, that rings to get its attention when required.

1.2 How do interrupts work?

1. The launch:

- The CPU goes and orders the disk controler: „I need this file, search it for me while I do something else while you do it“.

2. The freedom! (yay):

- The CPU can finally explore its full potential, that is, being blazingly fast.
- So it goes back to doing some CPU stuff; computing for another process, executing a thread, displaying an graphical interface, etc
- **It does NOT surveil the disk**

3. The event:

- The snail finally finished its freaking walk. So the snails manager, the disks controler **sends an electrical current that flows through the interrupt wire**.

4. The lambo's reaction:

- The CPU receives the electrical signal, some coulombs flowing being pushed by voltage in the wire.

Let's get more into interruptions:

- An interruption causes a transfer of control from the currently executing program to a specific routine that's **designed to handle that event**.
- This process is managed using an **interrupt vector table (IVT)**.

1.3 How the tranfer of control works

1. **Interruption**: An event occurs (Ex: A timer expires or a keyboard key is pressed), sending a signal to the CPU.
2. **Context saving**: Before switching tasks, the CPU **saves the current execution state**, typically the program counter and status registers onto a stack so it can come back to it and pop it to retrieve the precedent state, allowing it to continue its work on it.
3. **Vector identification**: The system identifies a unique **interrupt vector**, it's an index or a number that is associated with that specific triggered event.
4. **Look up and jump**: It then looks at the **IVT**, the interrupt vector table, specifically, it finds the memory address that is associated to the specific **interrupt service routine** that handles that event.
5. **Execution**: The PC (program counter) is updated with that new adress, and the cpu begins executing the service routine.
6. **Return**: Once the interrupt service routine is executed and returns, it generally end with an instruction that looks like **IRET** or **RETI** that restores the saved context in the stack, thus returning control to the original program.

- A key takeaway is that the **whole** OS is guided by interruptions.
- The OS preserves the anterior state before an interruption to come back to it later.
- **Interrupt Vector**: A unique code or number that identifies a specific interrupt handler. In some contexts, it refers to the actual memory address (pointer) of the service routine.
- **Interrupt Vector Table (IVT)**: A data structure—often an array or table of pointers —stored in a reserved block of memory. It maps each interrupt vector to the starting address of its corresponding ISR.
- **Routine/Service Routine (ISR)**: The specialized function that contains the code to process the interrupt.
- **Table of peripherals state**: Contains the type, address and state of each peripherals. So the CPU maintains this table, as we could have simultaneaous interruptions.

1.4 I/O system

- One of the OS's objective, is to abstract the material from its user (you don't need to know that your ram is 3200 MHz DDR4 to open google chrome).
- **But how does the I/O system handle the peripherals without slowing down the CPU?**

It uses 3 key mechanisms:

1. Buffering:
   - The problem: the network **sends data by small irregular packets**, but your **HDD wants to write data by big blocks at once**.
   - The solution: A waiting zone (the buffer). We accumulate the data until it's big enough to satisfy the required size for the HDD to write.
2. Caching:
   - The problem: The HDD is slow. We don't want to read the same thing 50 times when we already read it.
   - The solution: We **keep a copy of recently used data in RAM** for faster access.
3. Spooling:
   - The concept: It's basically a queue for peripherals that can only do one thing at a time, example: a printer.
   - Key takeaway: If three persons launch an impression, the OS wont mix up the data of the individual impression requests, it will put them in a **spool** and sends them one by one.

The pattern in all of theses techniques: Boosts the CPU's speed by making it less dependant on peripherals.

1.5 Storage

First of all, let's talk about the physical structure of a disk.

- **Tracks**: The rings that make the disk, kind of like how onions are made of circles glued together that make a disk if we were to slice it at a specific height $z = k, k \in \mathbb{R}$. They are concentric circles on each platter surfacem, kind of like this: (((())))

- **Sectors**: On the tracks that make the cylinders of a disk, we have **sectors**. They are **the smallest addressable unit of data**.
- **Cylinders**: a literal physical cylinders composed of superposed concentric rings (the **tracks**), so the **Cylinder #4 is the track #4 of the platter #1 + the track #4 of the platter #2 + track #4 of the platter #3, etc.** So **the number of the cylinder is the number of the tracks superposed on all of the platters.**
- an arm moves mechanically to the right sector to read/write data.

1.6 Storage fragmentation

- When creating a file and putting lots of data, how does the CPU deal with injecting that data when adjacent data around the file itself are unrelated?
    - This is where the **file system** intervenes! The file system uses an **allocation table**, allocation of what? **data**! it's like a book's table of contents.
    - **If the file is too big to stay in one piece, the OS cuts it** and puts in its allocation table **the physical places where the file exists so it can individually find the individual data pieces and construct the file**, Example: „The start of the file is **at sector 10** then the next piece is at **sector 300**, then **sector 561**".
    - **This is what fragmentation is**. For the CPU, it's transparent because it's abstracted from it, the cpu orders the hdd for data through amulti layered process. **The CPU does not see the fragments of data, but the OS and Disk Controller do**. But from the HDD's POV, it's **painfully slow** to read data, it's arm has to jump from sector 10 to sector 300,etc.
- Let's look into disks algorithms that optimizes time:

xyz

Pourquoi on a l'impression que certains programmes doivent se „reveiller" lorsqu'on n'a pas utise un programme depuis un bout de temps par exemple, le programme est ouvert mais on ne l'utilise plus du tout, on ouvre un autre programme mais la ram est pleine, alors on a ce qu'on appelle la virtualisation memoire, on utilise un espace dans le HDD qu'on appelle SWAP, la cle pour comprendre ici est la ram & HDD -> On prend literallement une image de la ram et onon the HDD there is something called a swap, the os will take the memory spaces that you didn't use for a long time and put them in there, when alt

En user mode, lorsqu'on execute un programme, le processus (dans le programme) demarre une minuterie (qui dure quelques ms), apres la fin de cette minuterie, le noyau prend le controle pour s'assurer que tout va bien (ex: eviter une boucle infinie), si le code du processus est flawed, une minuterie sauve l'os, elle lui redonne le controle pour faire fonctionner le systeme.

Ex:

User process executing -> calls system call -> execute system call (mode bit = 0) -> execute system all, done (mode bit =1, usermode) -> return from system call It is invisible to the user, everytime your screen gets updated there is literally some code from the kernel mode to interact with the physical components (screen in this case) fun fact, the majority of computers in the world are running on linux because there are something that windows can simply not do. As a matter of fact, all 500 super computers in the world run on linux.

1.7 OS and the services it offers

The OS offers services:

1. User interface: GUI,batch,command line
2. Environment of execution for programmes
3. I/O services
4. Folder and file gestion
5. Error detections
6. Accounting (audit en francais), garde des logs de tout ce qui se passe sur l'ordi. (generally disabled, always enabled in computer for finance: to monitor a theft for example)