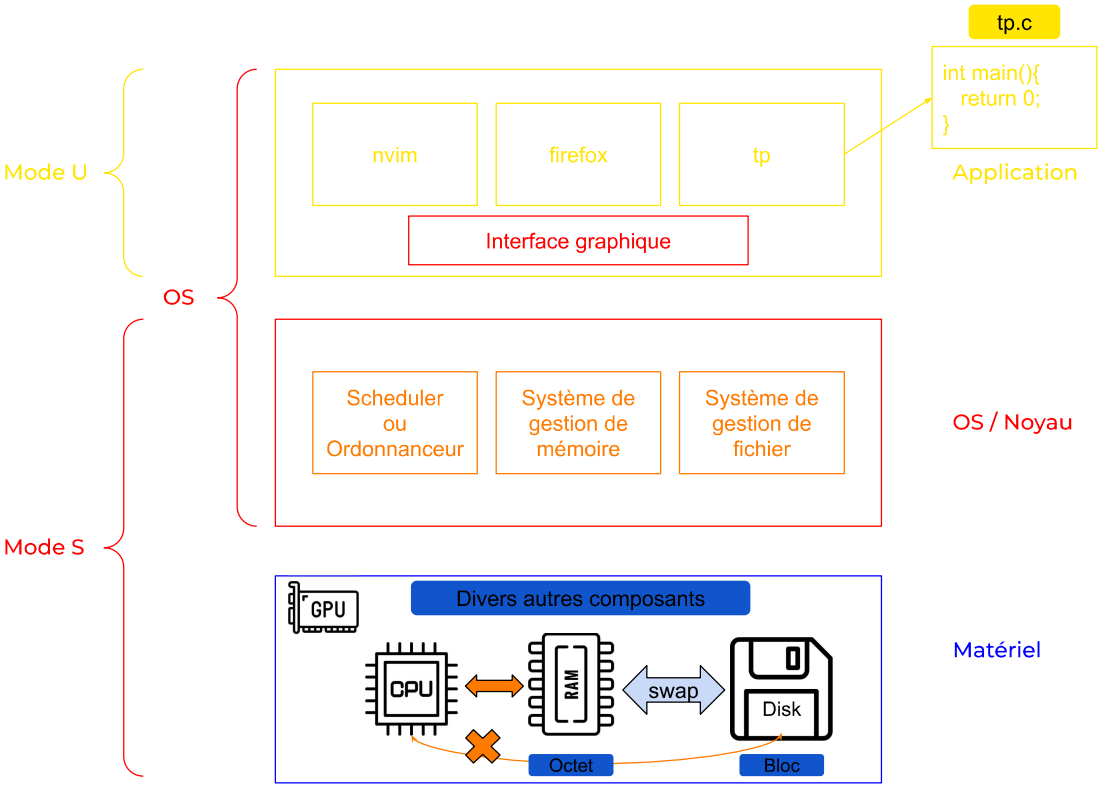
Programmation Shell et introduction au système

Cours 1 : Les trois composantes principales d'un ordinateur



➔ Qu'est-ce qu'un **système d'exploitation** ?

Un système d'exploitation est la couche logicielle, dont le rôle est de gérer tous les périphériques (processeurs, mémoire principale, disques, etc) et de fournir aux programmes utilisateur une interface simplifiée avec le matériel. C'est-à-dire, une couche qui cache partiellement, le matériel et fournit au développeur un jeu d'instructions plus pratique pour son travail. Il est, en général, la portion qui fonctionne en mode noyau (ou mode superviseur). Il remplit a priori deux tâches principales : l'extension de la machine et la gestion des ressources.

🚩Le kernel (noyau) n'est pas un système d'exploitation ! Le système d'exploitation est une interface entre l'utilisateur et la machine (c'est d'ailleurs le premier programme qui boot lors du démarrage). Le kernel quant à lui est une composante (centrale) du système d'exploitation ! C'est lui, le kernel, qui fait la traduction entre les commandes de l'utilisateur et le langage machine.

➔ La structure **matériel d'un ordinateur**

Le **CPU** (Central Processing Unit), est un microprocesseur installé sur la carte mère de l'ordinateur. Si la carte mère est le cœur du PC, le CPU est, quant à lui, considéré comme le "cerveau" de l'ordinateur. Il extrait des instructions de la mémoire, les décode et les exécute. Pour améliorer les performances du CPU : voir les structures pipeline et processeurs superscalaires. La plupart des processeurs utilisés dans les systèmes embarqués ont deux modes de fonctionnement : **le mode utilisateur et le mode superviseur / noyau** (concept

matériel !!!). En mode superviseur, le processeur peut exécuter n'importe quelle instruction de son jeu et utiliser toutes les caractéristiques du matériel sous-jacent. Le système d'exploitation tourne en mode superviseur, ce qui lui donne accès à l'ensemble du matériel. En revanche, les programmes utilisateur tournent en mode utilisateur, qui ne permet l'accès qu'à un sous-ensemble des instructions du CPU et des ressources de la machine. Pour accéder aux services offerts par le système d'exploitation, un programme utilisateur doit faire un **appel système** (ou autres) qui intervertit les modes utilisateur et superviseur. Ils ont un espace réservé dans le processeur notamment pour contrer les attaques, vols de données, bug…

La mémoire vive (**RAM**), par opposition à la mémoire morte (**ROM**), est la mémoire à court terme d'un ordinateur ayant un support de stockage par adresse en octet. Aucun de vos programmes, fichiers ou flux Netflix ne fonctionnerait sans la **RAM**, celle-ci correspondant à l'espace de traitement de votre ordinateur. La pile et le tas se situent dans cette partie de l'ordinateur.

On accède au disk par bloc. (voir cours sur le système de fichiers *discord.com/cours-devoirs/'système de fichiers'*)

Les **disques durs sont** du matériel informatique utilisé pour stocker du contenu numérique et des données sur des ordinateurs. Chaque ordinateur possède un **disque dur** interne, mais il existe également des **disques durs** externes qui peuvent être utilisés pour augmenter la capacité de stockage d'un ordinateur.

➔ Interaction entre **le matériel**

Le CPU peut accéder à la RAM contrairement au disque.
Si la RAM est saturée, le CPU peut y accéder, mais ne peut pas y ajouter des processus si cela venait à arriver, le CPU tuerait des processus. Un autre cas de figure, un **swap** peut éventuellement être effectué entre la RAM et le disque, mais c'est une action très lente.

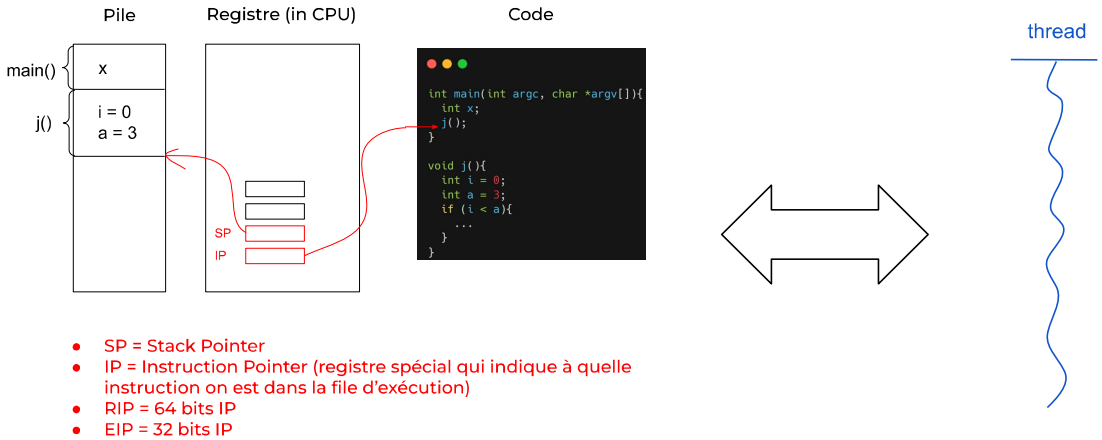
➔ **Composant du système d'exploitation**

L'ordonnanceur ou scheduler :
https://fr.wikipedia.org/wiki/Ordonnancement_dans_les_syst%C3%A8mes_d'exploitation

Cours 2 : Quel composant est responsable de mettre les instructions dans le processeur ? (thread / interruption)

On trouve linux dans tout ce qui est embarqué (serveurs, caméra bluetooth, android -> linux modifié, freebox, etc.)

➔ **Qu'est-ce qu'un file d'exécution ?**



- SP = Stack Pointer
- IP = Instruction Pointer (registre spécial qui indique à quelle instruction on est dans la file d'exécution)
- RIP = 64 bits IP
- EIP = 32 bits IP

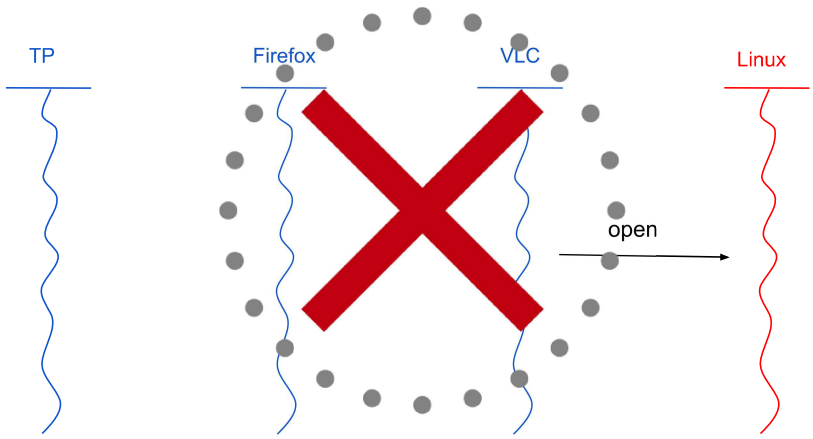
Un processus est un espace d'adressage. Un processus peut avoir plusieurs threads (fils/flots d'exécution).
Un fil d'exécution est un programme (dans la partie application) en cours d'exécution. Il est caractérisé par un stack pointer (donc une pile) dont l'adresse est stockée dans le registre RSP du CPU, et un instruction pointer (donc un code) dont l'adresse est stockée dans le registre RIP du CPU.

🚩A thread is an independent set of values for the processor registers (for a single core). Since this includes the Instruction Pointer (aka Program Counter), it controls what executes in what order. It also includes the Stack Pointer, which had better point to a unique area of memory for each thread or else they will interfere with each other.

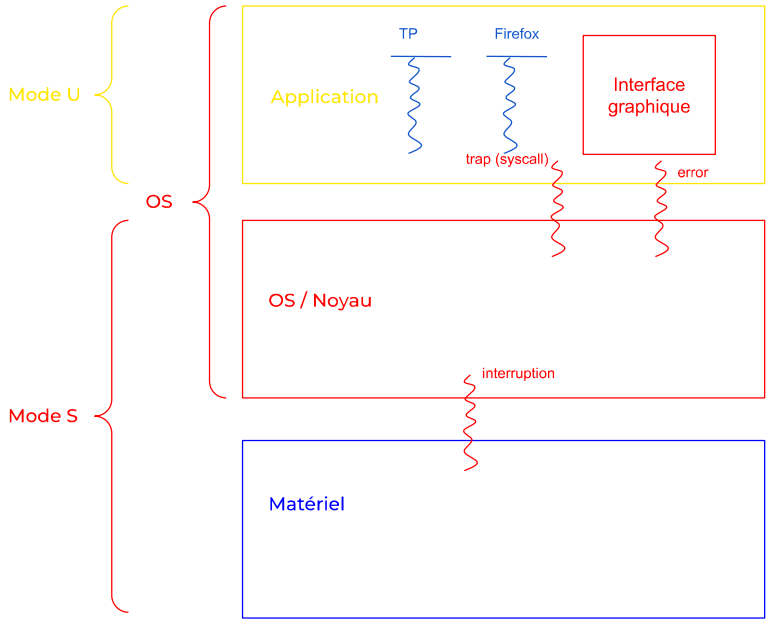
Threads are the software unit affected by control flow (function call, loop, goto), because those instructions operate on the Instruction Pointer, and that belongs to a particular thread. Threads are often scheduled according to some prioritization scheme (although it's possible to design a system with one thread per processor core, in which case every thread is always running and no scheduling is needed).

In fact the value of the Instruction Pointer and the instruction stored at that location is sufficient to determine a new value for the Instruction Pointer. For most instructions, this simply advances the IP by the size of the instruction, but control flow instructions change the IP in other, predictable ways. The sequence of values the IP takes on forms a path of execution weaving through the program code, giving rise to the name "thread".

Les applications exécutent ces fils d'exécutions, mais attention le système d'exploitation n'est pas un fil d'exécution. Le noyau lui-même est exécuté par les fils d'exécutions.

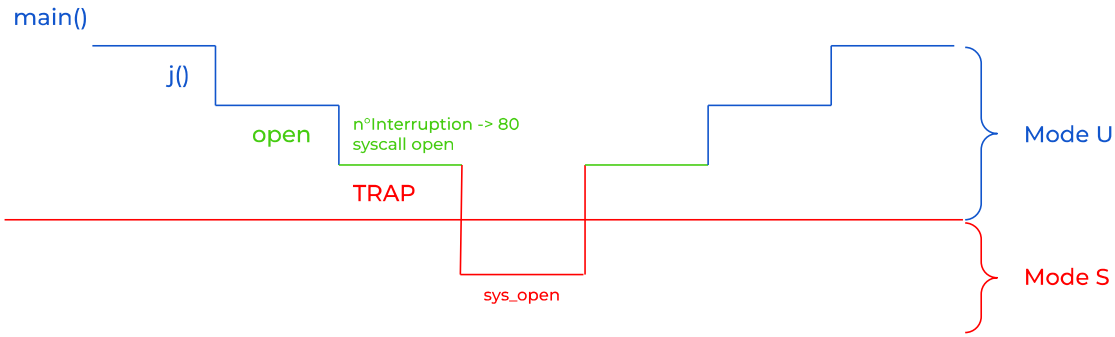


➔ **kernel ≠ fil d'exécution**



Le kernel est un handler d'interruption, trois moyens d'avoir une interruption :

1. faire une erreur logicielle (division par 0, stackoverflow, …)
2. en faisant un appel système, syscall, càd le programme s'interrompt et demande à l'OS d'effectuer une certaine tâche
    - a Trap is an exception that switches to kernel mode by invoking a kernel sub-routine (any system call). Usually a trap creates any kind of control transfer to the operating system. Whereas SYSCALL is a synchronous and planned user process to kernel mode



3. interruption matérielle (déclenchée par un périphérique, l'horloge par exemple)

La seule manière de passer entre le mode U et le mode S est une interruption ("commande" par le matériel (?)).

Qu'est-ce que le mode S et le mode U : (ce sont des concepts matériels)

User Mode: When a Program is booted up on an Operating system let's say windows, then it launches the program in user mode. And when a user-mode program requests to run, a process and virtual address space (address space for that process) is created for it by windows. User-mode programs are less privileged than user-mode applications and are not allowed to access the system resources directly. For instance, if an application under user-mode wants to access system resources, it will have to first go through the Operating system kernel by using syscalls.

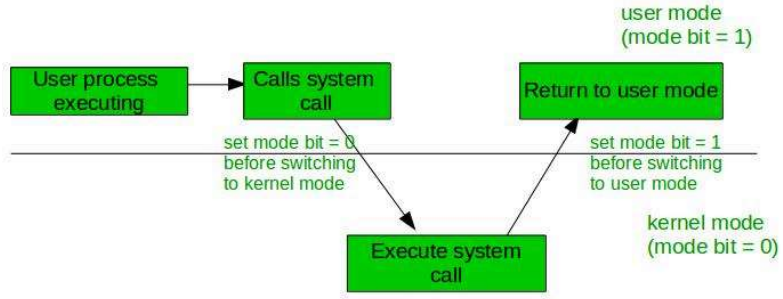
Kernel Mode: The kernel is the core program on which all the other operating system components rely, it is used to access the hardware components and schedule which processes should run on a computer system and when, and it also manages the application software and hardware interaction. Hence it is the most privileged program, unlike other programs it can directly interact with the hardware. When programs running under user mode need hardware access for example webcam, then first it has to go through the kernel by using a syscall, and to carry out these requests the CPU switches from user mode to kernel mode at the time of execution. After finally completing the execution of the process the CPU again switches back to the user mode.

Utilité du mode S et du mode U :

An error in one program can adversely affect many processes, it might modify data of another program, or also can affect the operating system. For example, if a process is stuck in the infinite loop then this infinite loop could affect the correct operation of other processes. So to ensure the proper execution of the operating system, there are two modes of operation:

User mode –
When the computer system is run by user applications like creating a text document or using any application program, then the system is in user mode. When the user application requests for a service from the operating system or an interrupt occurs or system call, then there will be a transition from user to kernel mode to fulfill the requests.
Note: To switch from kernel mode to user mode, the mode bit should be 1.

Given below image describes what happen when an interrupt occurs:



Kernel Mode –
When the system boots, hardware starts in kernel mode and when the operating system is loaded, it starts the user application in user mode. To provide protection to the hardware, we

have privileged instructions which execute only in kernel mode. If the user attempts to run privileged instruction in user mode then it will treat instruction as illegal and traps the OS. Some of the privileged instructions are:

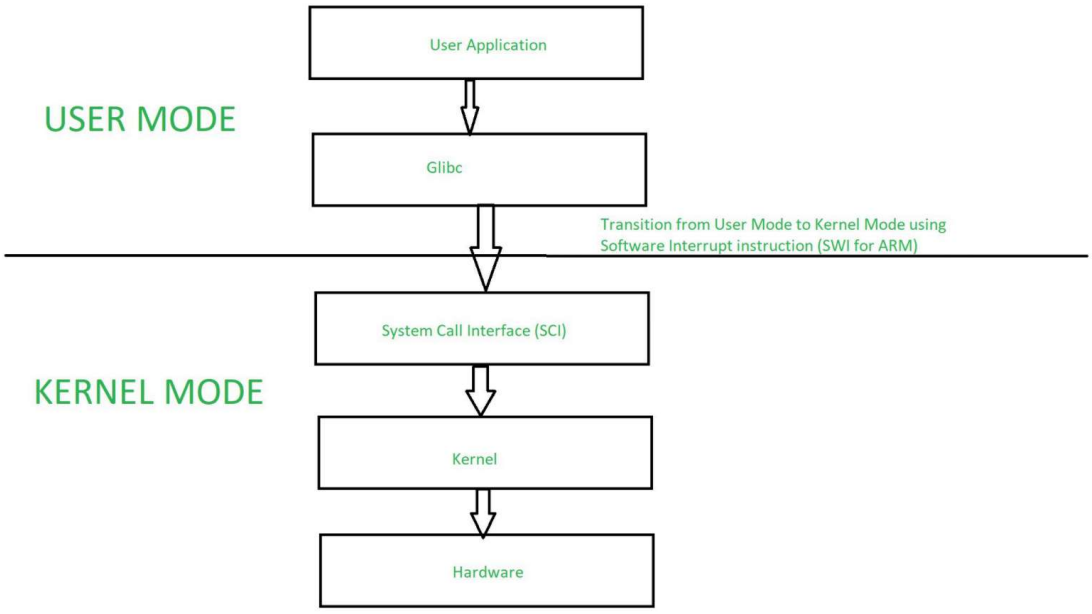
- Handling Interrupts
- To switch from user mode to kernel mode.
- Input-Output management.
- Note: To switch from user mode to kernel mode bit should be 0.

Processus et switch entre le mode S et le mode U :

A process can access I/O Hardware registers to program it, can execute OS kernel code and access kernel data in Kernel mode. Anything related to Process management, I/O hardware management, and Memory management requires processes to execute in Kernel mode.

This is important to know that a process in Kernel mode gets power to access any device and memory, and at the same time any crash in kernel mode brings down the whole system. But any crash in user mode brings down the faulty process only.

The kernel provides System Call Interface (SCI), which are the entry points for the kernel. System Calls are the only way through which a process can go into kernel mode from user mode. Below diagram explains user mode to kernel mode transition in detail.



https://www.geeksforgeeks.org/difference-between-user-level-thread-and-kernel-level-thread/

Cours 3 : Thread et processus

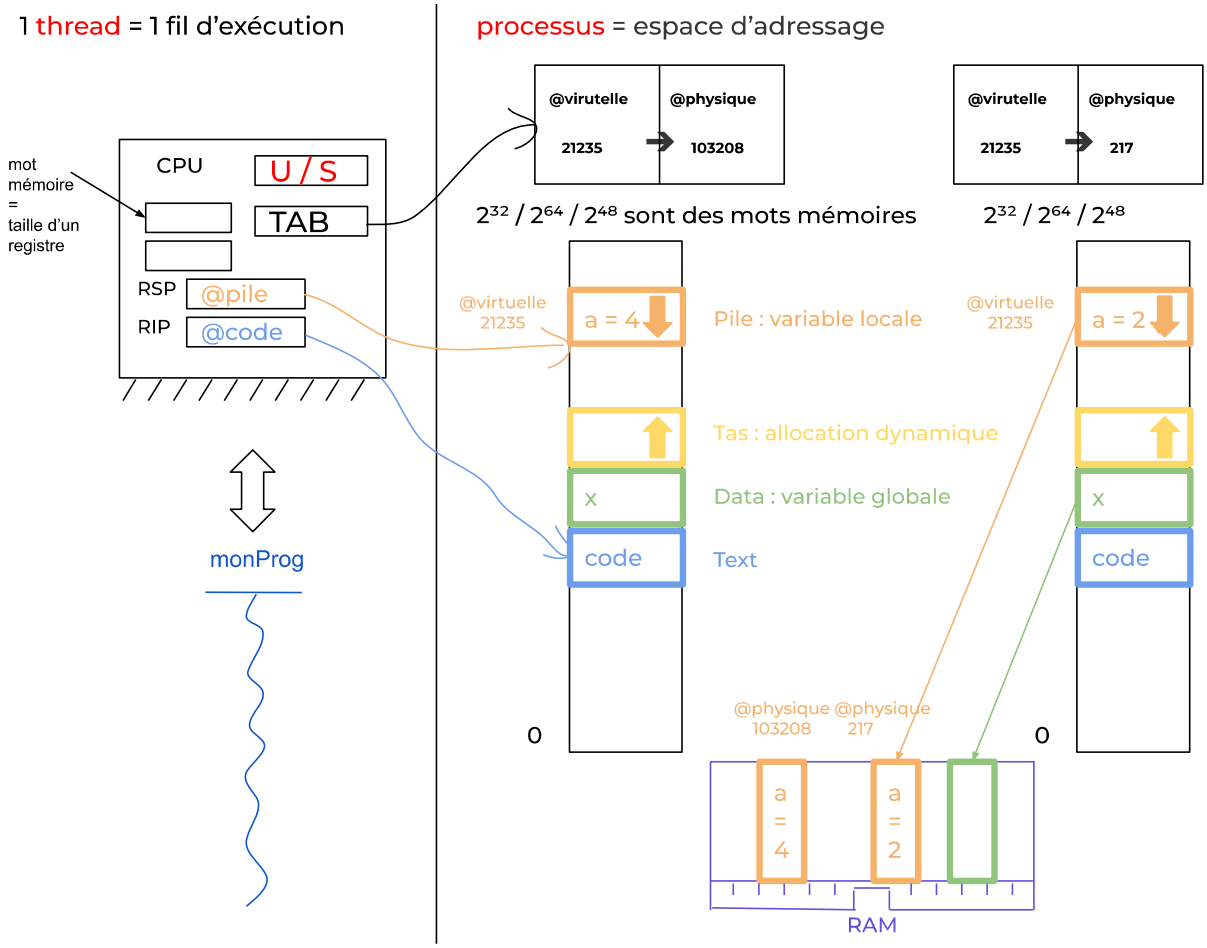
Comment voir les appels systèmes ?

- Dans un helloWorld.c, le seul appel système est au moment printf(« Hello World »);
- strace ./HelloWorld permet de voir tous les appels systèmes.

Résumé du cours 2 :

- ★ Qu'est-ce qu'un fil d'exécution ?
  - pointeur d'instruction (là où on est dans le code)
  - registre
  - stack pointer
- ★ kernel ≠ fil d'exécution
  - un peu comme lib, c'est un handler d'interruption (relie un signal électrique à un code)
    - événement matériel
    - événement logiciel (les erreurs)
    - appel système
  - quand il y a une intérruption, l'OS prend la main pour lui dire de se suicider
    - le programme devient mode S
    - appel $\mathrm{exit}(-1)$
  - par les interruptions : processus devient super-processus mais subit le code écrit par nous-même

Le code des piles issues du schéma sont en bas.



Dans le schéma du processus, il y a une barrière OS invisible entre la pile et le tas en cas de stackoverflow.
La table, au sein du processeur, traduit les adresses physiques et les adresses virtuelles. Pour chaque processus, on a une table d'équivalence entre les adresses. Ainsi les deux processus ayant pour adresse 21235 sont "dans deux univers parallèles" donc l'un n'a rien à voir avec l'autre en quelque sorte. Néanmoins, il y a des défauts de sécurité apparemment?

```
int x;

int main(int argc, char *argv){
   int a = atoi(argv[1]); //ascii to integer
   while (1){
      printf("%p %d\n", &a, a);
      sleep(1);
   }
}

>> gcc monProg.c -o monProg
```

./monProg 4	./monProg 2
21235 : 4 21235 : 4 21235 : 4 . . .	21235 : 2 21235 : 2 21235 : 2 . . .

Ce n'est pas réellement la même adresse pour des mesures de sécurité.
En même temps à l'inf, ce sont deux variables à la même adresse.

Donc un processus est un univers virtuel dans lequel il s'exécute, un espace d'adressage équivaut à une traduction physique.

Qu'est-ce qu'un processus ?
An executing instance of a program is called a process.
Some operating systems use the term 'task' to refer to a program that is being executed.
A process is always stored in the main memory also termed as the primary memory or random access memory.
Therefore, a process is termed as an active entity. It disappears if the machine is rebooted.
Several processes may be associated with the same program.
On a multiprocessor system, multiple processes can be executed in parallel.
On a uniprocessor system, though true parallelism is not achieved, a process scheduling algorithm is applied and the processor is scheduled to execute each process one at a time yielding an illusion of concurrency.
Example: Executing multiple instances of the 'Calculator' program. Each of the instances are termed as a process.

Cours 4 : Structure d'un processus

Résumé du cours 3 :

- ★ Thread : état de registre du CPU + pile
  - ○ fil d'exécution relier à un exécutable
- ★ processus : espace d'adressage (monde qui n'appartient qu'à lui)
  - ○ adresse virtuelle
  - ○ MMU ? (dans le CPU)
  - ○ tuer un processus => tuer un fil d'exécution

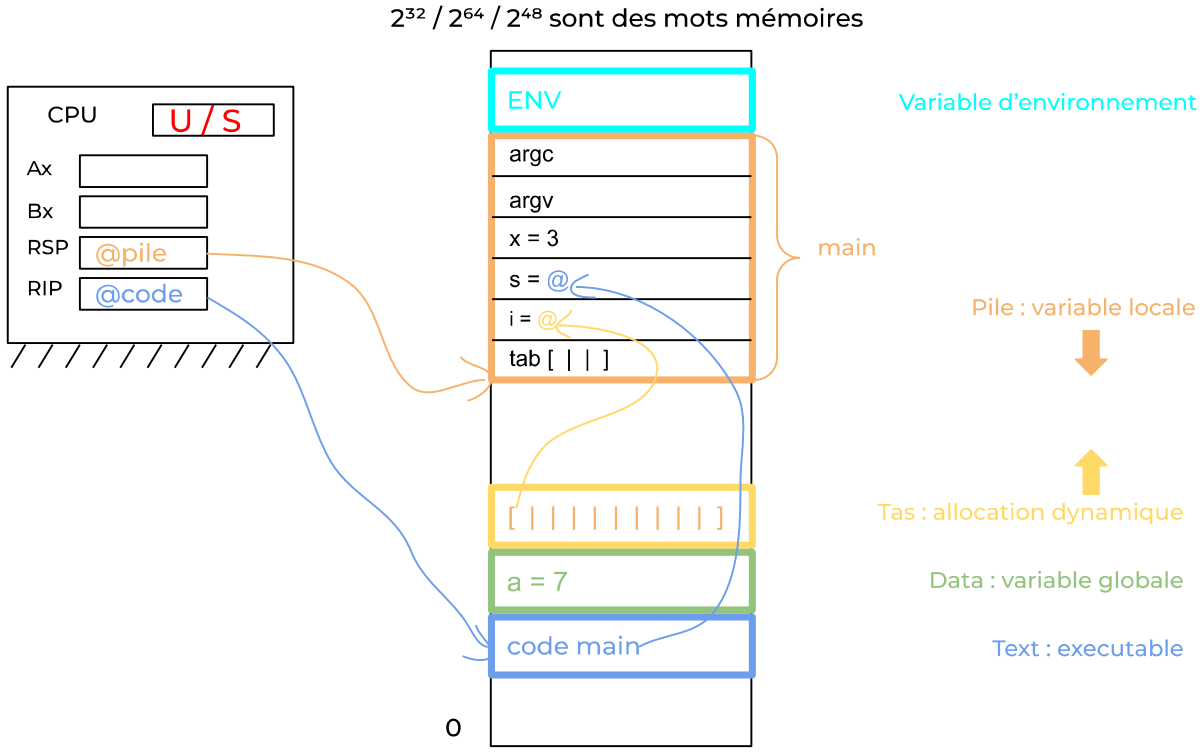
Introduction à exec() :

ELF : format executable

```
int a = 7;

int main(int argc, char *argv){
   int x = 3;
   char *s = "toto";
   int *i = (int *)malloc(sizeof(int)*10);
   int tab[3];
}

>> gcc monProg.c -o monProg
```



La vie d'un processus est le temps de la pile, quand on exécute l'exécutable l'OS utilise exec() et remplie la pile.

```
#!/bin/bash
a=4
b="toto"
export x=3

>>./monScript.sh
```

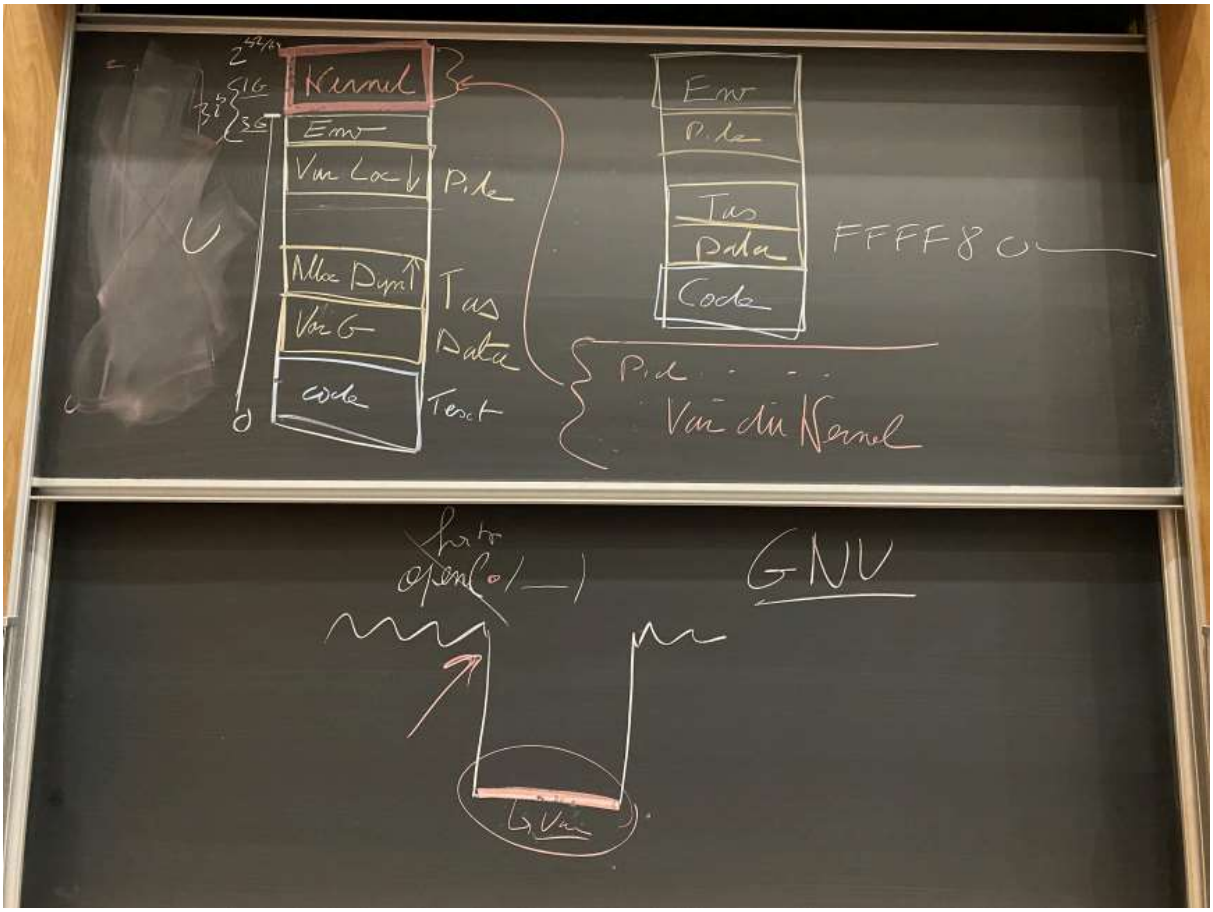
La variable est a. Le $ agit, grosso modo, comme un pointeur en C. On écrit pas : $a=3 mais on écrit a=3.

Lancer ./monScript.sh et lancer ./monProg fonctionne de la même manière, "même appel système" :

- Il n'y a pas de main dans le monScript.sh mais chemin /bin/bash après le shebang est pris comme tel.
- Si l'OS trouve le shebang en parcourant les premiers octets du fichier, il va regarder la chaîne de caractères ensuite le "main du bash" (/bin/bash).

Cours 5 : Comment un processus naît-il ?

On va utiliser un schéma avec le kernel en bas du processus mais en réalité il est au dessus:

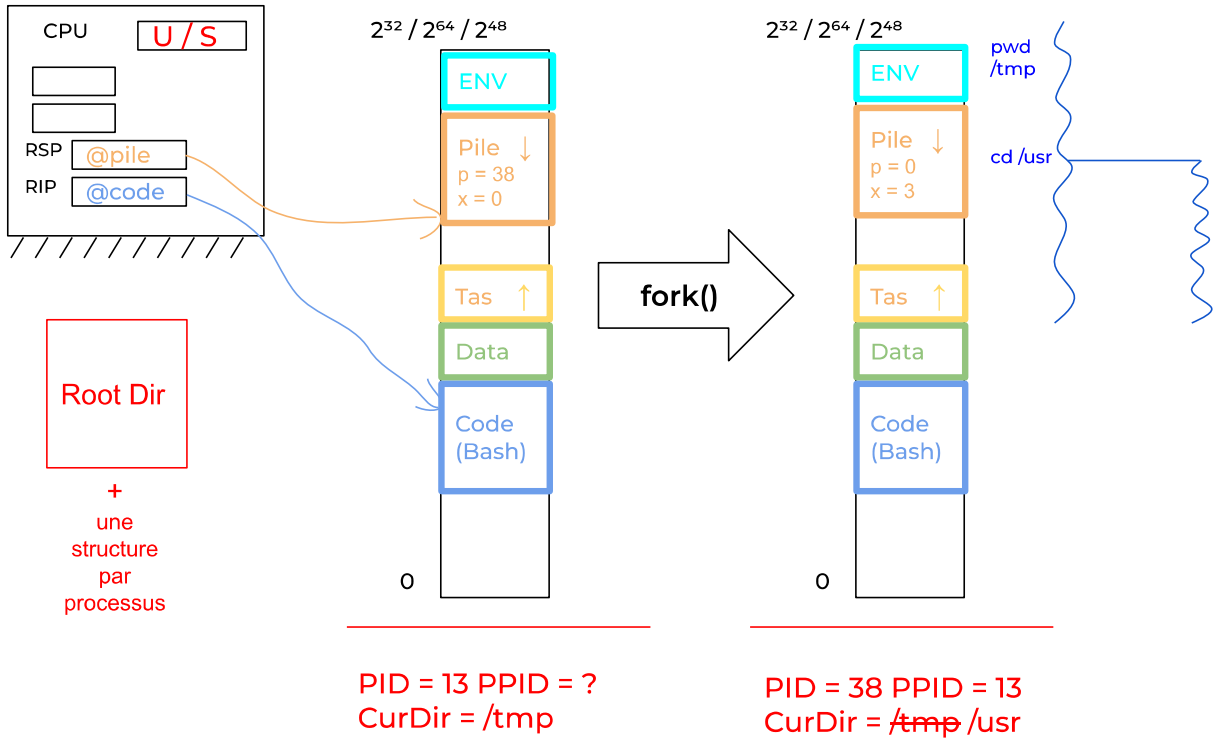


```c
#include <sys/types.h>
#include <stdio.h>

int main(int argc, char *argv){
   int p;
   int x = 0;
   printf("%d ", x);
   p = fork();
   if (p < 0) {
      printf("fork failed");
      return 1;
  } else if (p == 0) {
      //printf ("child process successfully created!");
      //printf ("child_PID = %d,parent_PID = %d", getpid(), getppid( ) );
      x++;
  } else {
      wait(NULL);
      printf ("parent process successfully created!");
      printf ("child_PID = %d, parent_PID = %d", getpid( ), getppid( ) );
  }
   printf("%d ", x);
   return 0;
}

>> gcc monProg.c -o monProg
```

Le schéma ci-contre décrit le cas p == 0 :



```
>> gcc monProg.c -o monProg
>> ./monProg
0 3 3
```

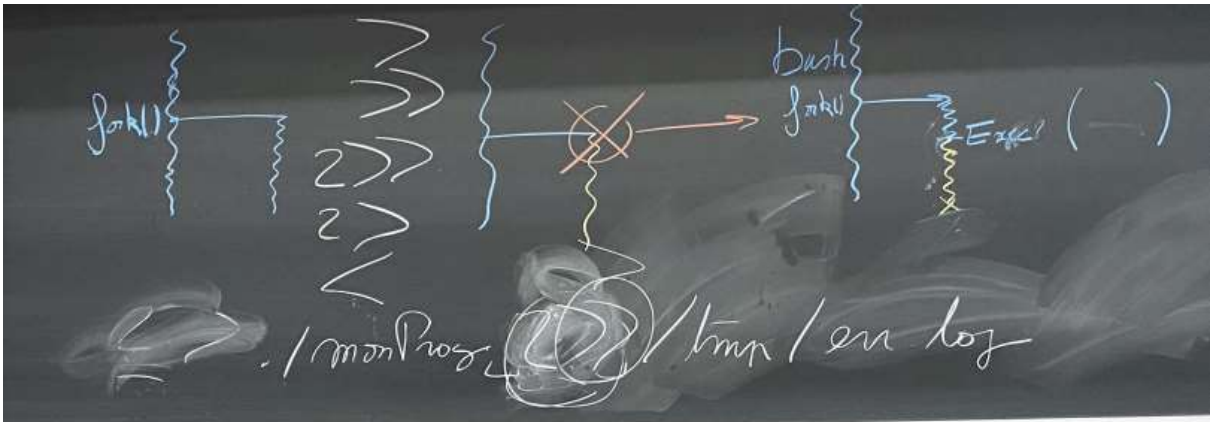
Tous les processus en exécution dans notre machine, mise à part le processus 0, sont dû à l'appel système fork().

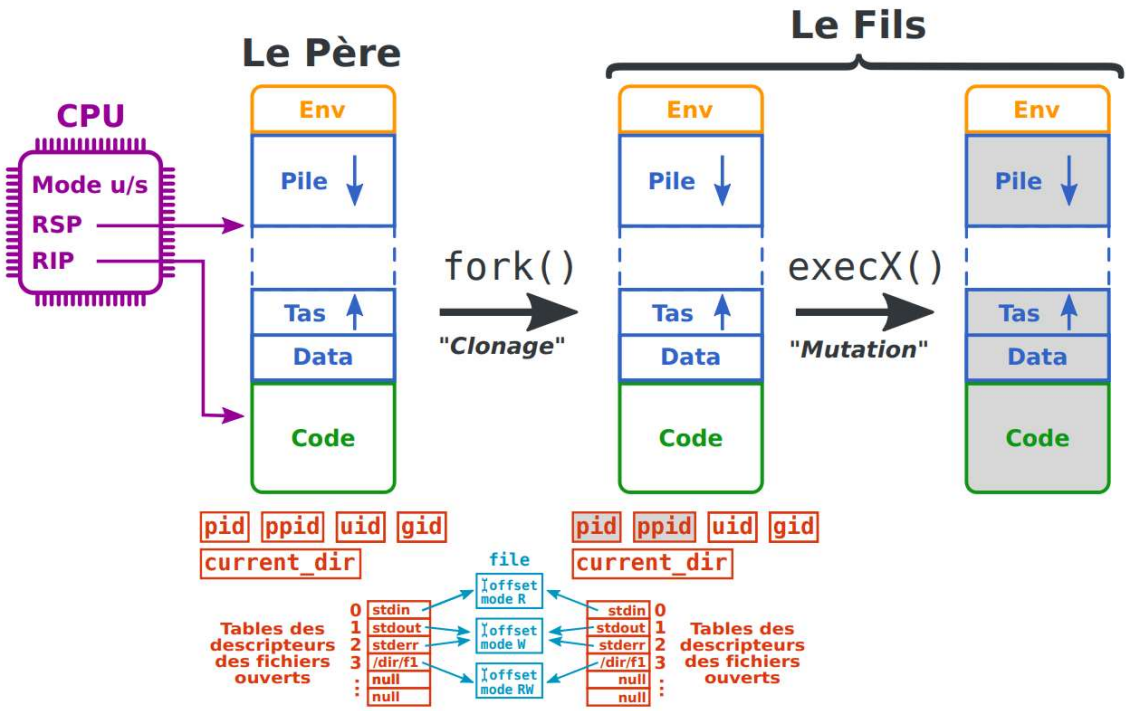
🚩 Comment créé un nouveau processus ? Par clonage :
A new process can be created by the fork() system call. The new process consists of a copy of the address space of the original process. fork() creates a new process from an existing process. Existing process is called the parent process and the process created newly is called the child process. The function is called from the parent process. Both the parent and the child processes continue execution at the instruction after the fork(), the return code for the fork() is zero for the new process, whereas the process identifier of the child is returned to the parent.
Fork() system call is situated in the <sys/types.h> library.

Cours 6 : fork() et exec()

- https://www.google.com/amp/s/www.softprayog.in/programming/creating-processes-with-fork-and-exec-in-linux/amp
- https://stackoverflow.com/questions/1653340/differences-between-fork-and-exec
- https://www.tutorialspoint.com/how-to-create-a-process-in-linux#:~:text=A%20new%20process%20can%20be,newly%20is%20called%20child%20process.

exec() permet de lancer un programme






```
...
if (fork() == 0){                    // si le fils a bien été créé
    execl("/bin/firefox", " ");  // " " = argv
    exit(-1);
}
...
```

En faisant exec(), on perd toutes les variables dans ce processus pour lancer un autre programme sauf les variables d'environnements. Un processus s'arrête quand on vide la pile.

par exemple quand on fait une nouvelle fenetre d'une invite de commande (~ fork) on garde notre PATH car exec() conserve les variable d'environnement (= heritage)
→ utiliser export est très utile pour garder des valeur entre processus

On met une barrière de sécurité avec le exit(-1) car en cas d'erreur, le fils va lui aussi exécuter le code suivant, ce qu'on ne souhaite pas à priori puisque la suite du code est réservé pour le processus père en théorie.

- ★ fork() ne modifie que le PID et PPID
- ★ exec() ne modifie que la pile, le tas et la data.
- ★ Un user identifier ou UID permet d'identifier un utilisateur sur les systèmes d'exploitation Unix et Linux. Cette technique est utilisée principalement pour les droits d'accès à des ressources ou à des domaines et donc pour la sécurité du système.

★ Le ./bashrc permet d'enregistrer en dur les modifications des variables d'environnements.

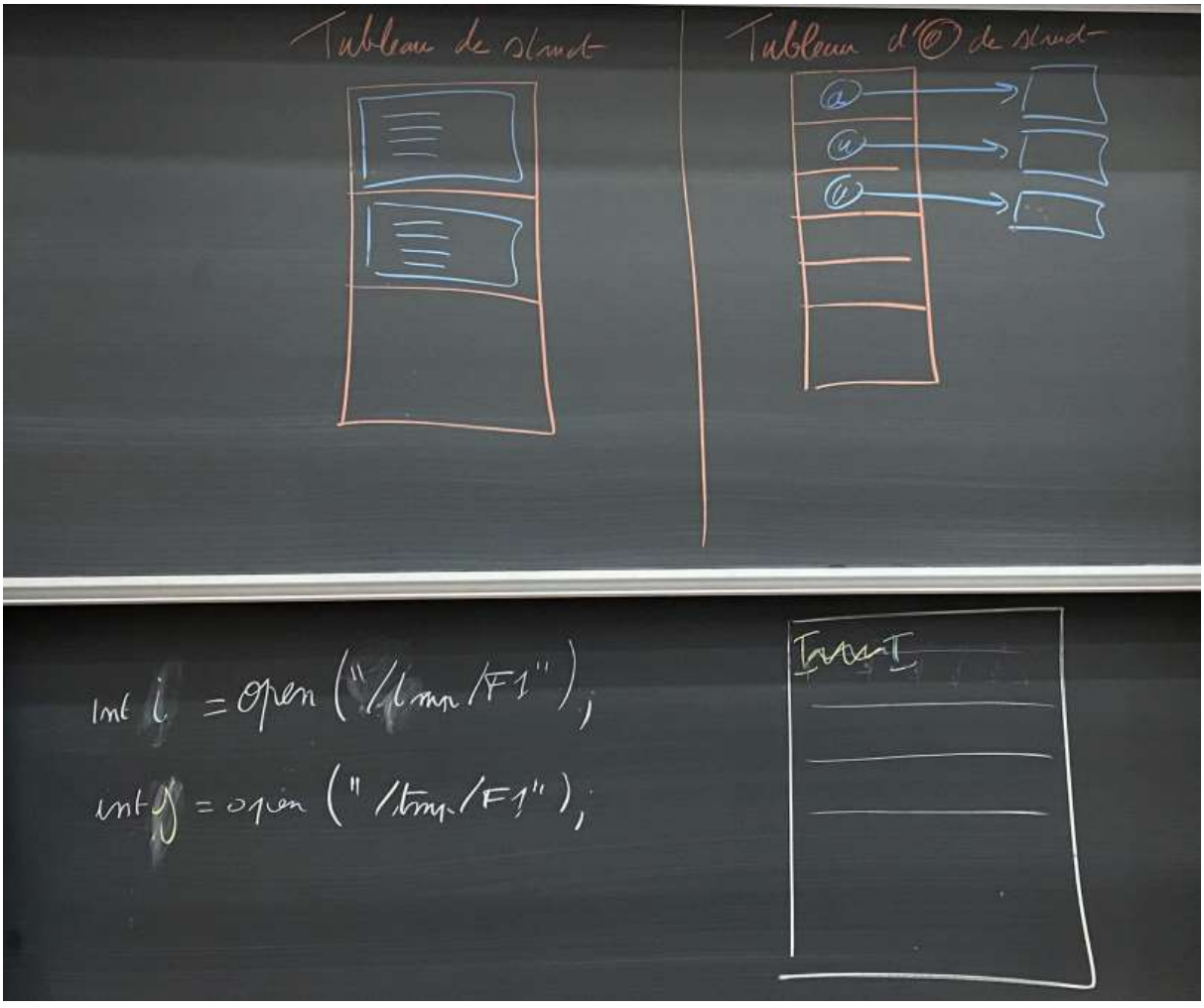
```
...
if (fork() == 0){
    execl("/bin/firefox", " ");
    printf("x");
    exit(-1);
}
...
```

Explication des flux d'entrés et flux de sorties:

Le concept d'entrée / sortie est universel.

La table des fichiers ouverts est un tableau de pointeur (voir photo ci-dessous), c'est-à-dire que chacune des cases possédera un pointeur vers une structure.

Le open() est forcément un carré bleu dans la table des fichiers ouverts. On peut faire plusieurs open(File) à la fois sur le même fichier, il lira entièrement le même fichier et à chaque ouverture on aura un curseur dédié à celui-ci, ce qui est très pratique. La fonction open() est gérée par l'OS en mode lecture-écriture et l'offset (l'emplacement du curseur dans le fichier). On pourra éventuellement appeler la table des fichiers ouverts la table des descripteurs.



——————définitions——————

Un patch ou correctif est **une section de code que l'on ajoute à un logiciel, pour y apporter des modifications** : correction d'un bug, traduction, crack.

Un **registre** est un emplacement de mémoire interne à un processeur. Les registres se situent au sommet de la hiérarchie mémoire : il s'agit de la mémoire la plus rapide d'un ordinateur, mais dont le coût de fabrication est le plus élevé, car la place dans un microprocesseur est limitée. https://fr.wikipedia.org/wiki/Registre_de_processeur

——————————————————————————

——————pas au programme——————

Le système de gestion de fichier : il gère la partie vide du disque.
https://fr.wikipedia.org/wiki/Syst%C3%A8me_de_fichiers /
http://langevin.univ-tln.fr/cours/UPS/extra/chapitre3_sgf.pdf
Le système de gestion de mémoire : https://www.courstechinfo.be/OS/GestMem.html

——————————————————————————

—----------autres ressources—----------

Introduction au système d’exploitation :
http://www.gaudry.be/systeme-exploitation-introduction.html
Mode U et Mode S :
https://learn.microsoft.com/fr-fr/windows-hardware/drivers/gettingstarted/user-mode-and-kernel-mode
https://itigic.com/fr/communication-between-cpu-and-ram-how-does-it-occur/
https://www.google.com/search?q=comment+l%27information+communique+dans+un+syst%C3%A8me&oq=comment+l%27information+communique+dans+un+syst%C3%A8me+&aqs=chrome..69i57.11621j0j1&sourceid=chrome&ie=UTF-8
https://www.imedias.pro/cours-en-ligne/informatique/ordinateur/composants-ordinateur/#:~:text=Le(s)%20bus%2C%20syst%C3%A8me,diff%C3%A9rents%20composants%20d'un%20ordinateur.
https://www.quora.com/How-does-the-CPU-communicate-with-the-hard-drive
https://askcodez.com/comment-est-processeur-de-communiquer-avec-les-peripheriques.html

—----------—----------—----------—----------