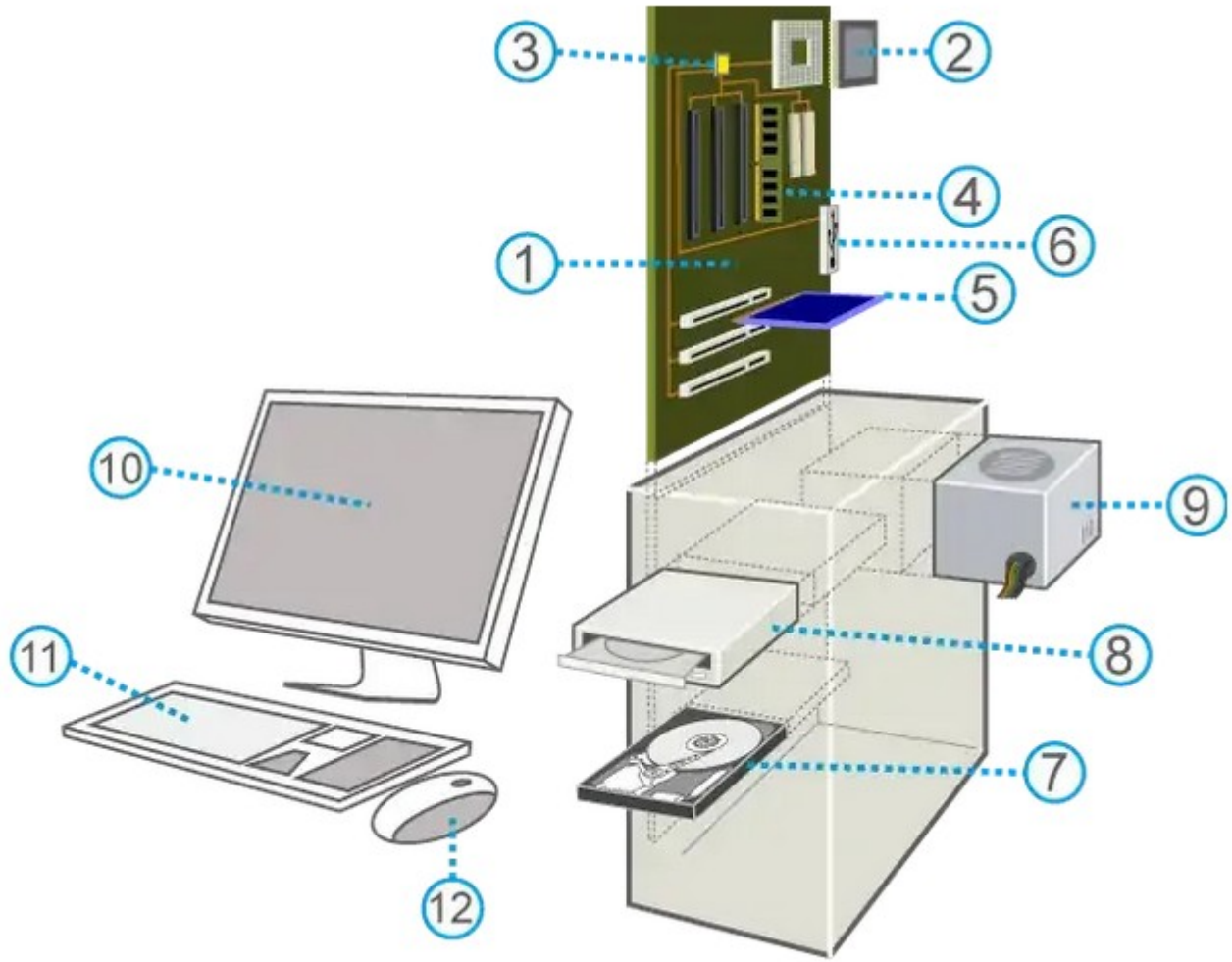
Architectures matérielles, systèmes d'exploitation et processus

Nicolas Blanc, Michel Courtieu, Lycée Anna de Noailles (Evian)

Sommaire

1 . Composants intégrés d'un système sur puce

1.1 Composants d'un ordinateur



1 :
2 :
3 :
4 :
5 :
6 :
7 :
8 :
9 :
10 :
11 :
12 :

1.2 Architecture de Von Neuman

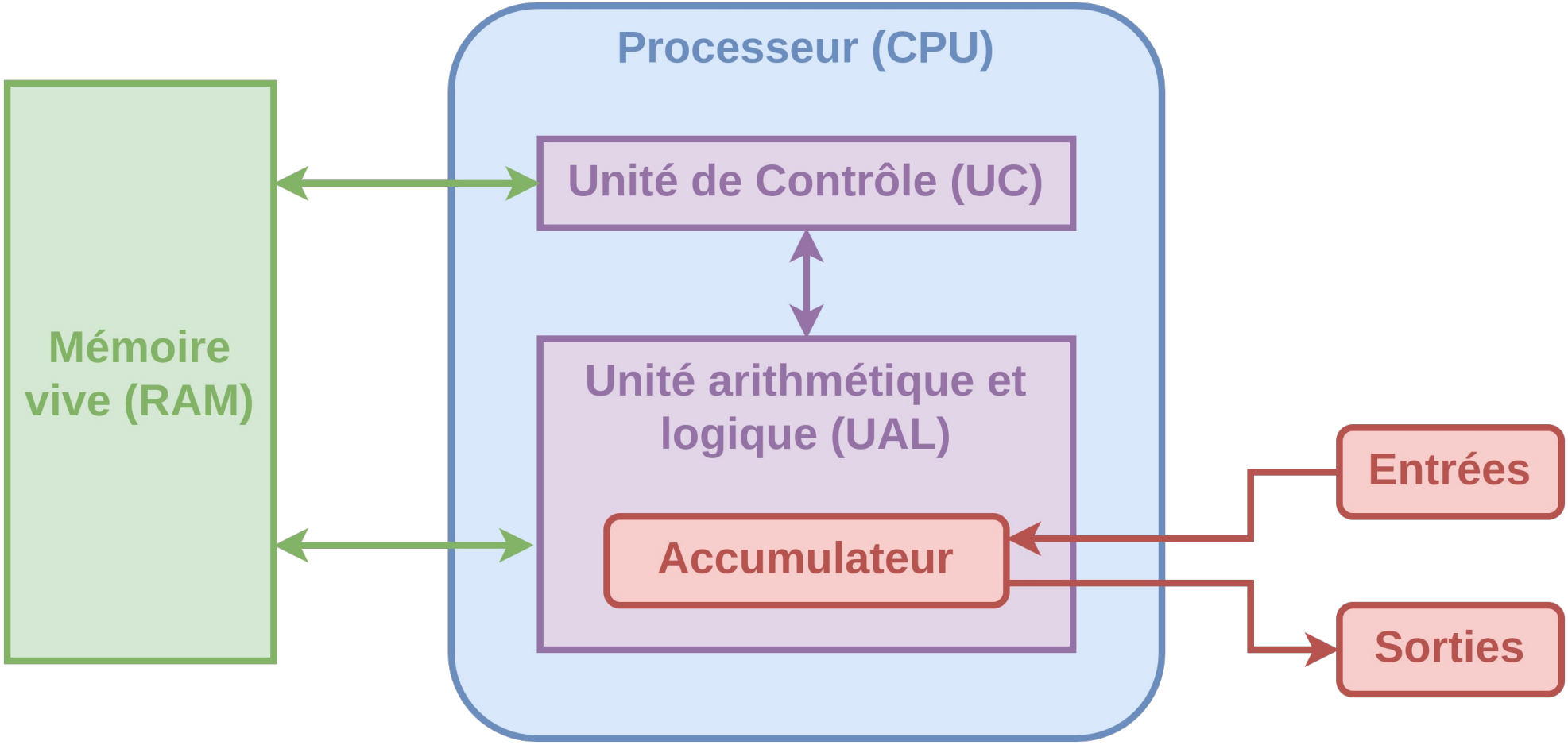
1.2.1 Les différentes parties de l'architecture de Von Neuman

L'architecture de *Von Neumann* est un modèle d'ordinateur.
Les ordinateurs actuels sont tous basés sur des versions améliorées de cette architecture.

L'architecture de *Von Neumann* décompose l'ordinateur en **4 parties distinctes** :

- **l'unité de contrôle** (ou **unité de commande**) chargée du séquençage des opérations ;
- **l'unité arithmétique et logique (UAL)** chargée d'effectuer les opérations ;
- **la mémoire** qui contient les programmes à exécuter et les données ;
- les **entrée-sortie**, qui permettent de communiquer avec l'extérieur.

Schéma général :

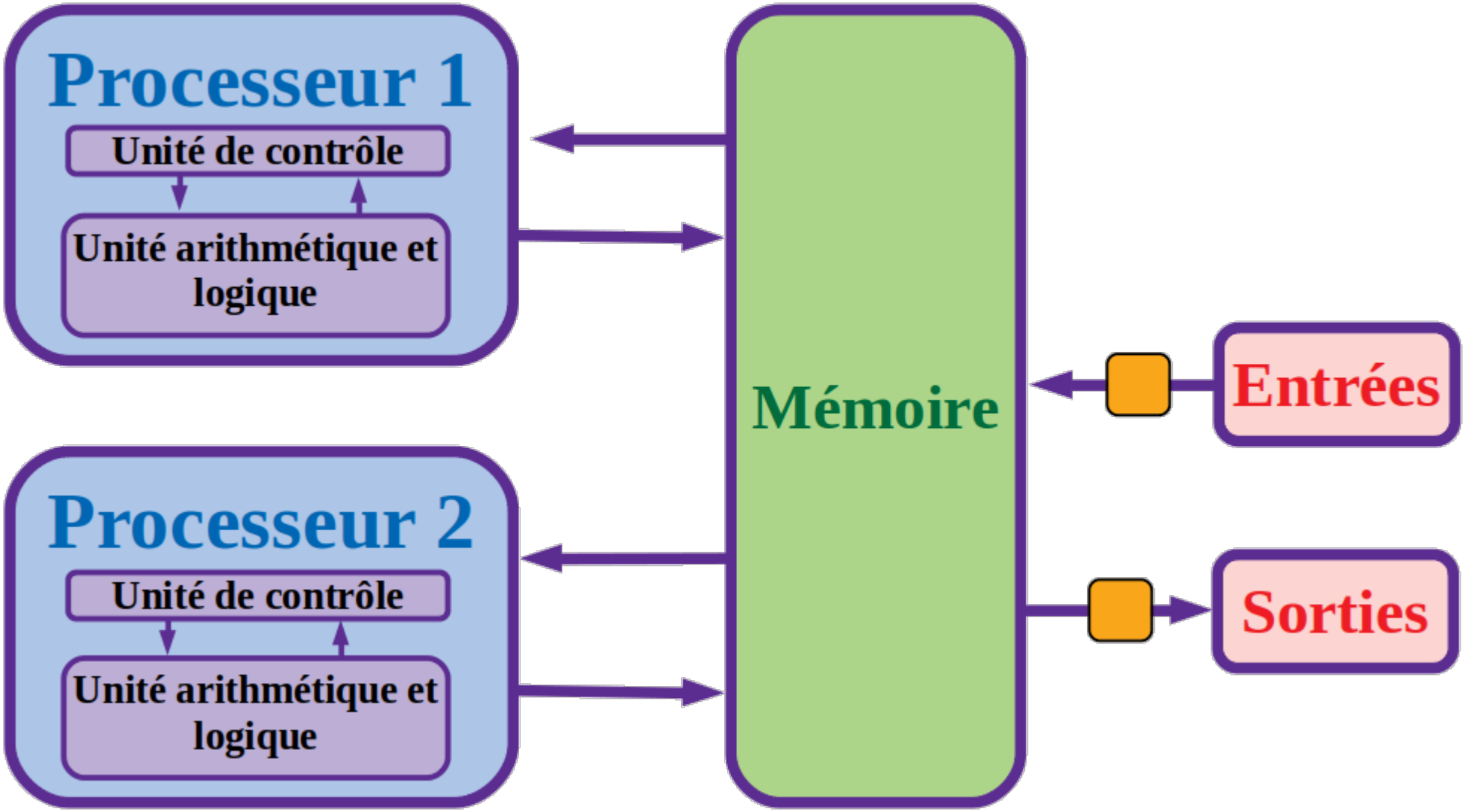


1.2.2 Amélioration de l'architecture de Von Neuman

Par rapport au schéma initial, on peut noter **deux évolutions principales :**

- Les **entrées-sorties** sont sous le contrôle de processeurs autonomes.
- Les ordinateurs comportent maintenant des **processeurs multiples** qui fonctionnent en parallèle. Cette organisation permet d'atteindre une puissance globale de calcul élevée sans augmenter la vitesse des processeurs individuels, limitée par les capacités d'évacuation de la chaleur dans des circuits de plus en plus denses.

Schéma général de l'architecture améliorée de Von Neuman :



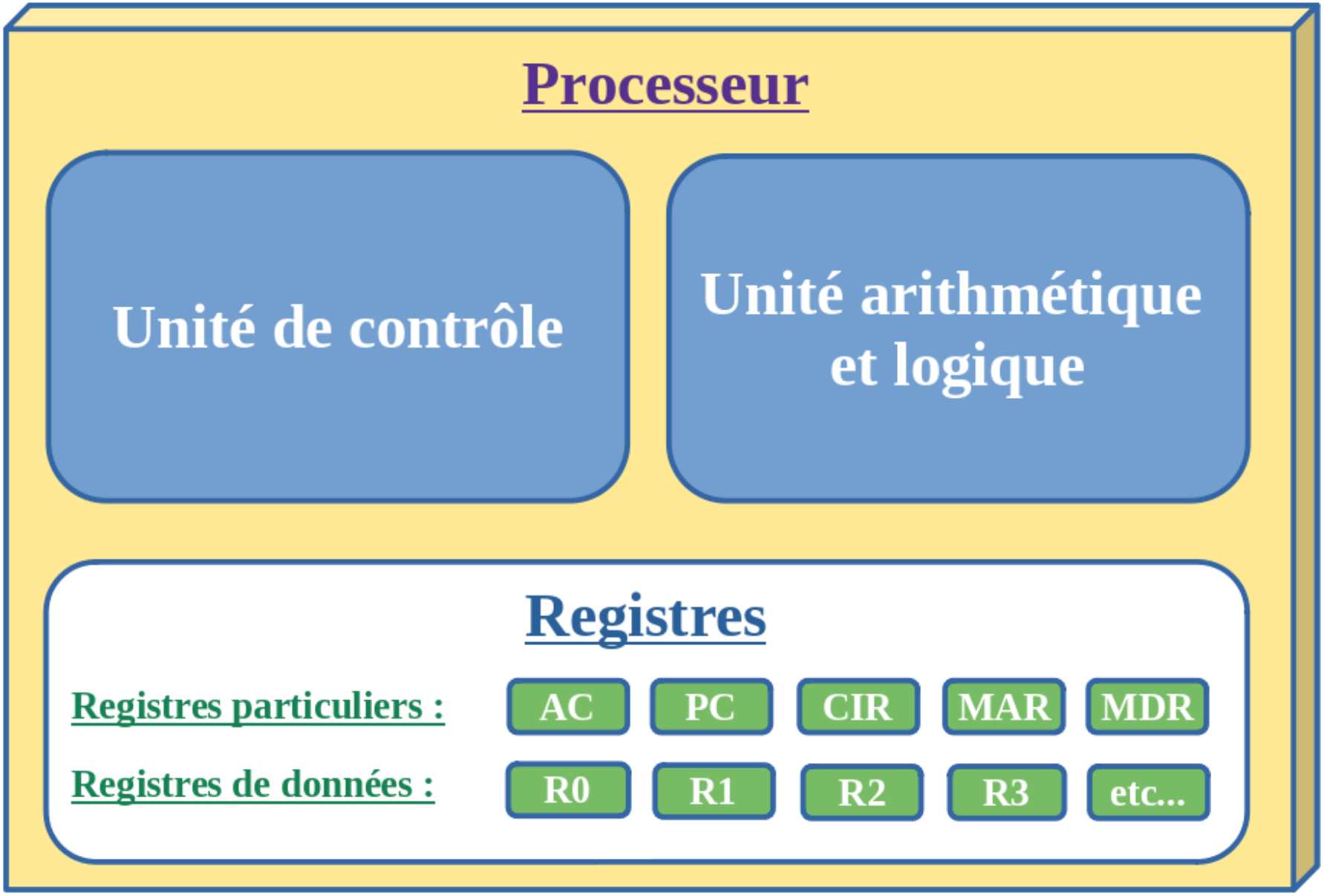
1.2.3 Le processeur (CPU)

Le **processeur** ou **CPU** (**Central Processing Unit**) est construit autour de deux éléments principaux :

- **l'unité de contrôle** ou unité de commande ;
- **l'unité arithmétique et logique** ;

Le processeur contient aussi de petite quantité de mémoire appelée **registres**, qui permettent de mémoriser de l'information transitoirement pour opérer dessus ou avec.

Schéma général :



Nicolas Blanc, Michel Courtieu, Lycée Anna de Noailles (Evian)

1.2.4 L'unité de contrôle

L'unité de contrôle permet de séquencer le déroulement des instructions.

- Elle effectue d'abord la recherche en mémoire de l'instruction et la stocke dans le **registre d'instruction (RI** ou **IR** ou **CIR)**.
- Comme chaque instruction est codée sous forme binaire, elle en assure le décodage avec le **décodeur d'instruction.**
- Elle réalise ensuite l'exécution de l'instruction.
- Puis elle effectue la préparation de l'instruction suivante, en mettant à jour un registre appelé **compteur de programme (PC)** qui contient ou permet de calculer l'adresse en mémoire de la prochaine instruction.

1.2.5 L'unité arithmétique et logique

C’est le cœur du processeur. Elle regroupe les circuits qui assurent les traitements nécessaires à l'exécution des instructions et utilise certains registres pour stocker temporairement des valeurs.

- **L’Unité Arithmétique et Logique** (**UAL**) est un circuit complexe qui assure les fonctions logiques (ET, OU, Comparaison, Décalage , etc…) ou arithmétique (Addition, soustraction).
- Le **registre d'état** **(STATUS)** est composé de quelques bits à considérer individuellement. Chacun de ces bits est un indicateur dont l'état dépend du résultat de la dernière opération effectuée par l’UAL.
- Les **accumulateurs** **(AC)** sont des registres de travail qui servent à stocker une opérande au début d'une opération arithmétique et le résultat à la fin de l'opération.

1.2.6 La mémoire vive (RAM)

La **mémoire vive** ou (**RAM**) sert au stockage temporaire de données.
Elle doit avoir un temps de cycle très court pour ne pas ralentir le processeur.
Les mémoires vives sont en général volatiles : elles perdent leurs informations en cas de coupure d'alimentation.
Les données stockées en mémoires sont de deux types différents :

- les **instructions** du programme à exécuter ;
- les **variables** du programme.

Une mémoire est un circuit à semi-conducteur permettant d'enregistrer, de conserver et de restituer ces informations (instructions et variables).
Ces informations peuvent être écrites ou lues.

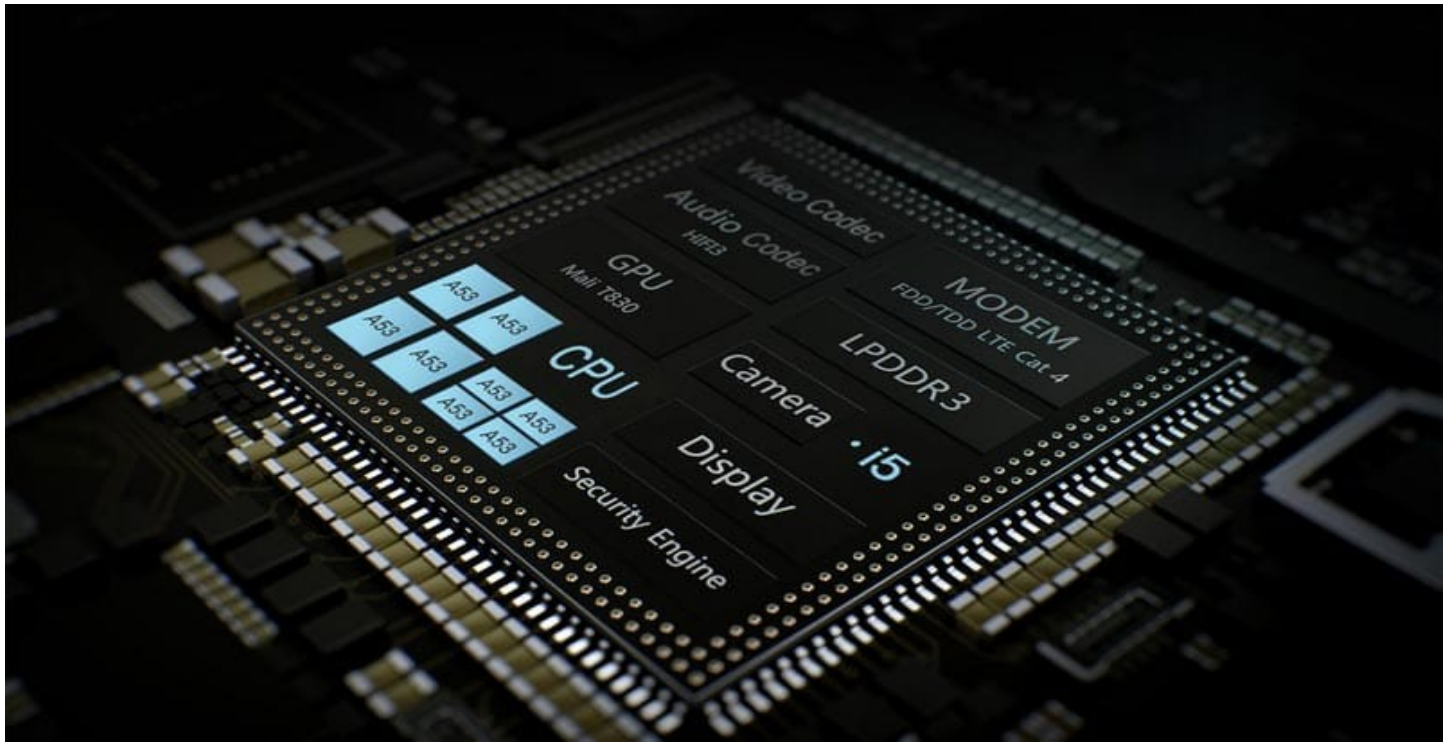
1.3 Système sur puce (SoC)

1.3.1 Généralités

Au sein d'un ordinateur (**PC : Personnal Computer)**, les choses sont assez simples. **Le processeur (CPU : Central Processing Unit)** se charge de réaliser les calculs les plus répandus, ceux qui permettent par exemple de faire tourner le système d'exploitation ou un navigateur web. On trouve aussi **la carte graphique (GPU : Graphics Processing Unit)** qui se charge d'afficher une image, qu'elle soit en 2D ou bien en 3D comme dans les jeux. **La carte-mère** relie entre eux tous les composants, le **CPU**, le **GPU**, mais également **la mémoire vive (RAM : Random Access Memory)** et d'autres cartes additionnelles (carte son, carte réseau, …) et petites puces et tous ses éléments sont interconnectés à l'aide de **bus**.

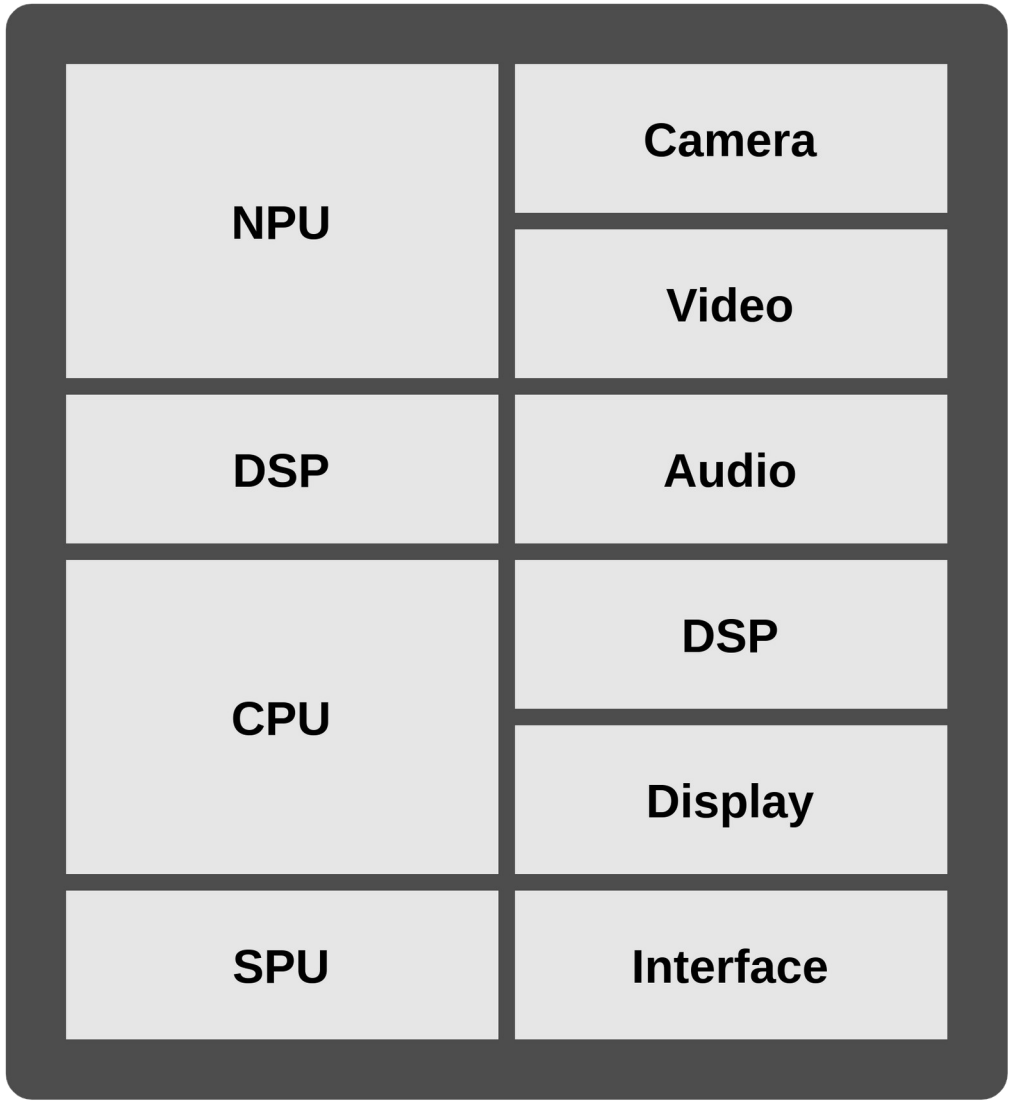
Mais depuis le début de l'ère **des smartphones**, on assiste à l'émergence de systèmes tout-en-un. Ainsi, presque tout le contenu d'un ordinateur se retrouve finalement dans une seule puce sur le smartphone : le **System on Chip (SoC)**, ou **système sur puce** en français.
C'est un circuit intégré essentiel au fonctionnement des objets connectés et des smartphones.

Exemple de SoC :



1.3.2 Composants d'un système sur puce (SoC)

Schéma :



Description :

- **NPU** (**N**eutral **P**rocessing **U**nit) : gère l'**intelligence artificiel**le.
- **DSP** (**D**igital **S**ignal **P**rocessor) : le processeur de signaux numérique : gère la compression et l'extraction des différents signaux numériques (musique, vidéo, ...) .
- **CPU** (**C**entral **P**rocessing **U**nit) : le **processeur**.
- **SPU** (**S**ecure **P**rocessing **U**nit) : gère les données sensibles : biométrie, SIM. bancaire, etc …
- **GPU** (**G**raphic **P**rocessing **U**nit) : la **carte graphique**.

1.3.3 Avantages et inconvénients d'un système sur puce (SoC)

Avantages :

- Une **augmentation de la vitesse de communication**. La diminution des distances permet de gagner en rapidité car plus la distance augmente, plus les hautes fréquences ont tendance à provoquer des pertes d'informations.

- La **diminution de la taille des composants** signifie également une **baisse de la consommation** globale du système, notamment pour **optimiser la batterie**.

- La diminution de la consommation signifie aussi une **diminution de l'échauffement** et donc aucune nécessité d'équiper le système d'un ventilateur : **un SoC est donc silencieux**.

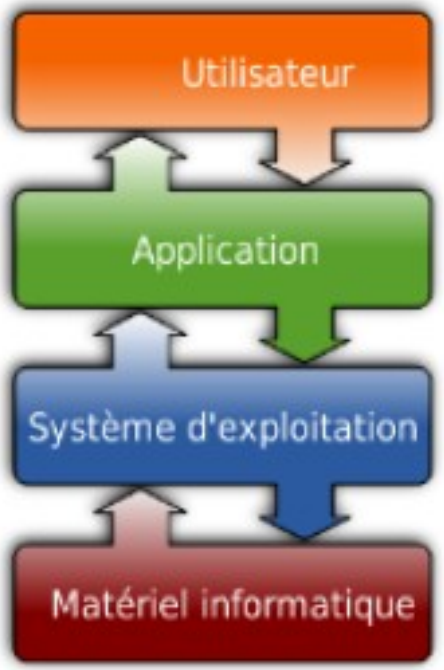
Inconvénients :

- Un ordinateur équipé d'une **carte mère** permet de faire évoluer les composants individuellement, l'extrême intégration du SoC présente en revanche l'inconvénient de n'autoriser **aucune mise à jour du matériel**.

2 . Système d'exploitation

2.1 Généralités

En informatique, un **système d'exploitation** (OS — de l'anglais **Operating System**) est un ensemble de programmes dont le but est de gérer les ressources matérielles et logicielles d'un ordinateur.



2.2 Rôles du système d'exploitation

- **Gestion du processeur :** Le système d'exploitation est chargé de gérer l'allocation du processeur entre les différents programmes grâce à l'**ordonnanceur**.
- **Gestion de la mémoire vive :** Le système d'exploitation est chargé de gérer l'espace mémoire alloué à chaque application grâce au **gestionnaire d'allocation mémoire**.
- **Gestion des entrées/sorties :** Le système d'exploitation permet de contrôler les périphériques matériels (carte graphique, disques durs, clavier, etc...) par l'intermédiaire des **pilotes de périphériques** (**drivers** en Anglais).
- **Gestion des protocoles réseaux :** Le système d'exploitation est chargé de l'implémentation des protocoles réseaux comme TCP/IP grâce à la **pile réseau**.
- **Gestion des fichiers :** Le système d'exploitation gère la lecture et l'écriture dans **le système de fichiers** et les droits d'accès aux fichiers par les utilisateurs et les applications.
- **Gestion des droits :** Le système d'exploitation est chargé de la sécurité liée à l'exécution des programmes en garantissant que les ressources ne sont utilisées que par les programmes et utilisateurs possédant les droits adéquats.

2.3 Composants logiciels d'un système d'exploitation

Parmi les différents composants logiciels présents dans les systèmes d'exploitation moderne on trouve :

- l'**ordonnanceur** qui décide quel programme s'exécute à un instant donné sur le processeur ;
- le **gestionnaire de mémoire**, qui répartit la mémoire vive entre les différents programmes en cours d'exécution ;
- le **système de fichiers**, qui définit la manière de stocker les fichiers sur les supports physiques (disque, clés USB, disques optiques, etc.) ;
- la **pile réseau** qui implémente entre autres des protocoles tels que TCP/IP • les pilotes de périphériques (ou drivers en anglais), dont le but est de gérer les périphériques matériel (carte graphique, disques durs, clavier, etc.).

2.4 Familles de systèmes d'exploitation

Les deux familles de systèmes d’exploitation les plus populaires sont **Unix** (dont macOS, GNU/Linux, iOS et Android) et **Windows**. Cette dernière détient un quasi-monopole sur les ordinateurs personnels avec près de 90 % de part de marché depuis 15 ans.

3 . Commandes Linux de base

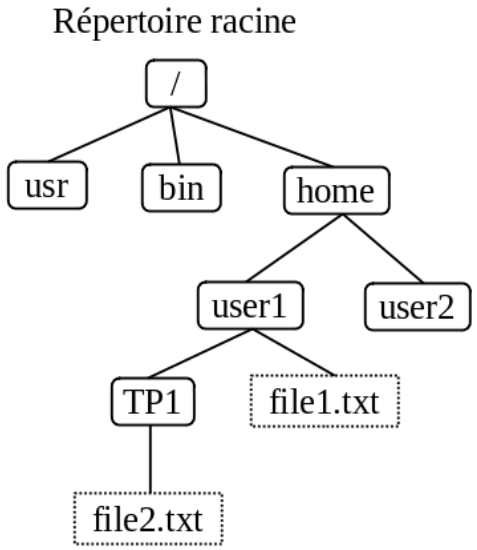
3.1 Commande pwd

La commande **pwd (Print Working Directory)** utilisée dans un **terminal** affiche le nom du répertoire de travail en cours (**répertoire courant**).

Exemple :

Terminal ×

```
user1@pc1:~$ pwd
/home/user1
```

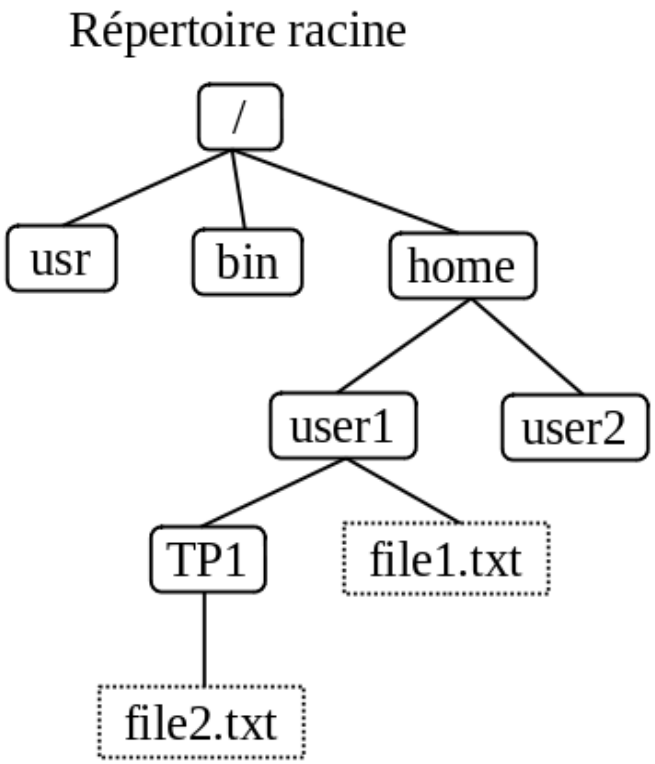


EXERCICE 1

Que va afficher la commande suivante dans un terminal ?

Terminal ×

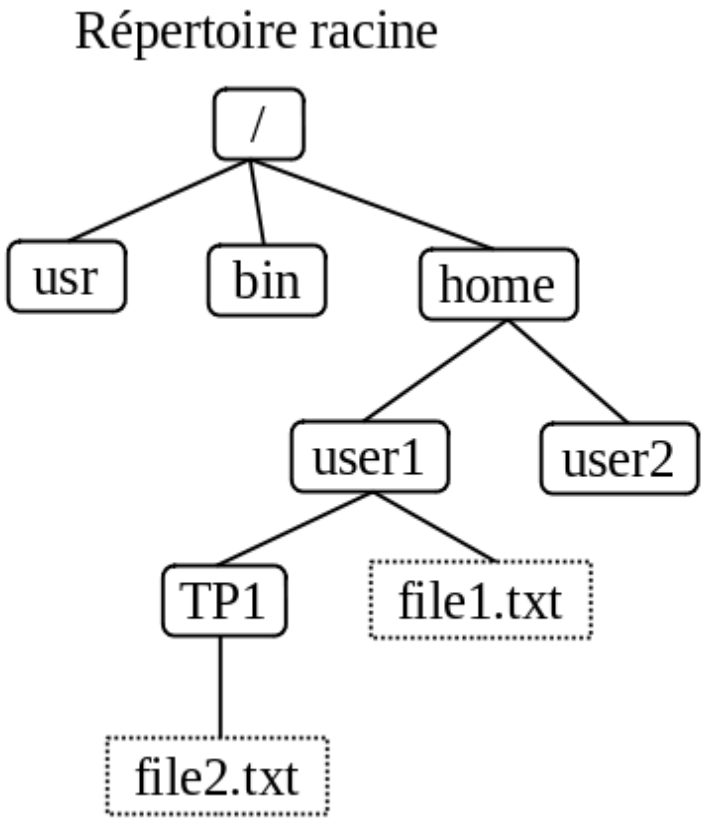
```
user1@pc1:~/TP1$  pwd
```



CORRECTION

Terminal ×

```
user1@pc1:~/TP1$  pwd
/home/user1/TP1
```



3.2 Commande ls

La commande **ls (List)** utilisée dans un **terminal** affiche le contenu d'un répertoire.

Exemple :

Terminal ×

```
user1@pc1:~$ ls
TP1                file1.txt
user1@pc1:~$ ls TP1
file2.txt
```

Options :

ls -a : Affiche tout le contenu d'un répertoire y compris les fichiers commençant par un .

ls -l : Affiche tout le contenu d'un répertoire et affiche d'autres informations sur les fichiers et sous répertoire en plus de leur nom.

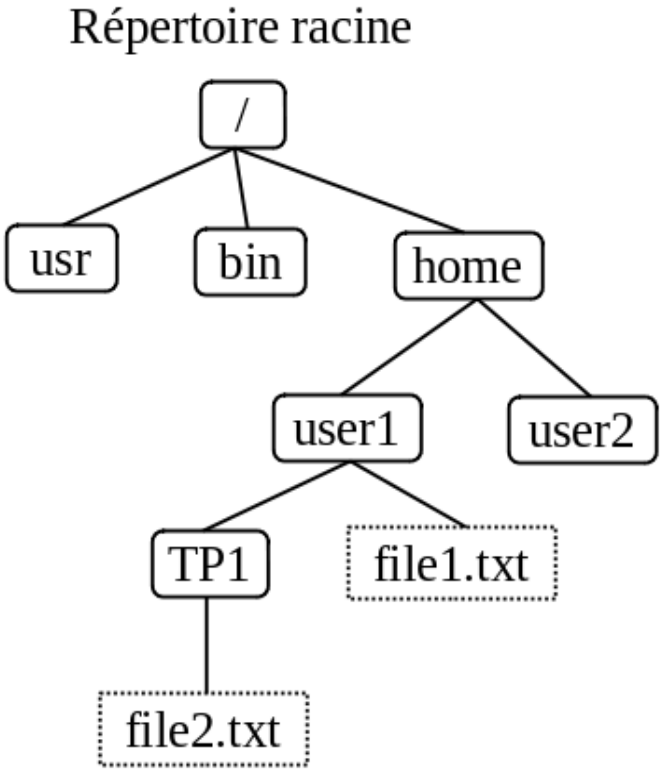
Syntaxe : ls ou ls *chemin*

EXERCICE 2

Que va afficher la commande suivante dans un terminal ?

Terminal ×

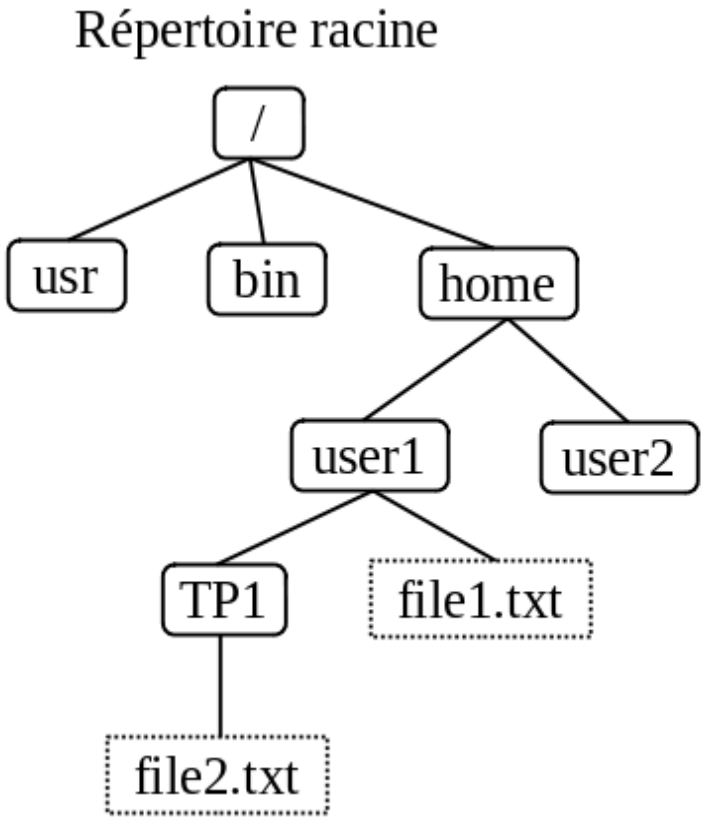
```
user1@pc1:~/TP1$  ls
```



CORRECTION

Terminal ×

```
user1@pc1:~/TP1$  ls
file2.txt
```



3.3 Commande cd

La commande **cd (Change Directory)** permet de changer de répertoire de travail.

Exemple :

Terminal ×

```
user1@pc1:~$ cd TP1
user1@pc1:~/TP1$ cd
user1@pc1:~$ cd ..
user1@pc1:/home$
```

Syntaxe : cd *nom_repertoire*

Remarques :

- si l'on tape **cd** seul. On revient dans son répertoire utilisateur.
- si l'on tape **cd ..** , on remonte d'un niveau dans l'arborescence.

3.4 Commande cp

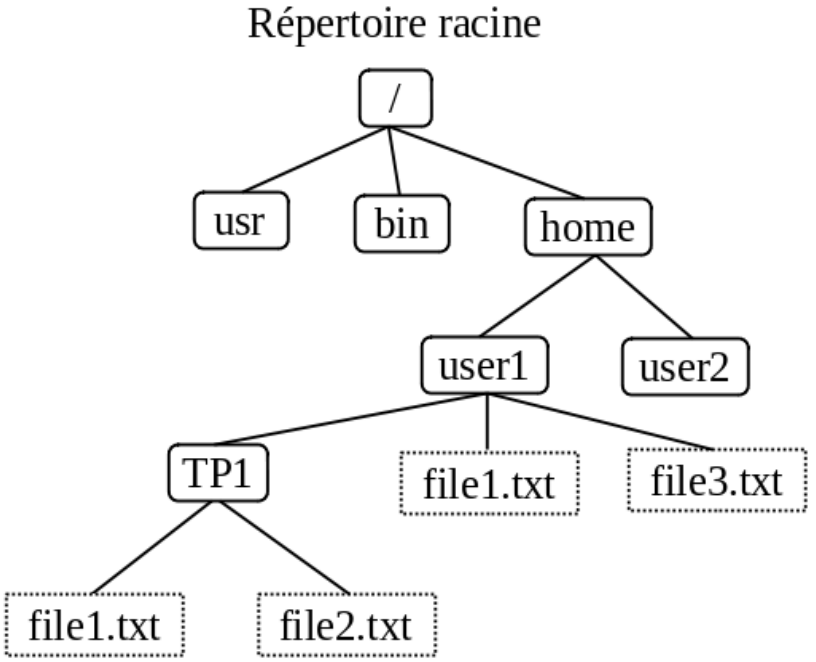
La commande **cp (Copy)** permet de copier des fichiers.

Exemple :

Terminal ×

```
user1@pc1:~$ cp file1.txt /home/user1/TP1
user1@pc1:~$ cp /home/user1/TP1/file2.txt /home/user1/file3.txt
user1@pc1:~$ ls
TP1                file1.txt                file3.txt
```

Syntaxe : cp *source* *destination*



3.5 Commande mv

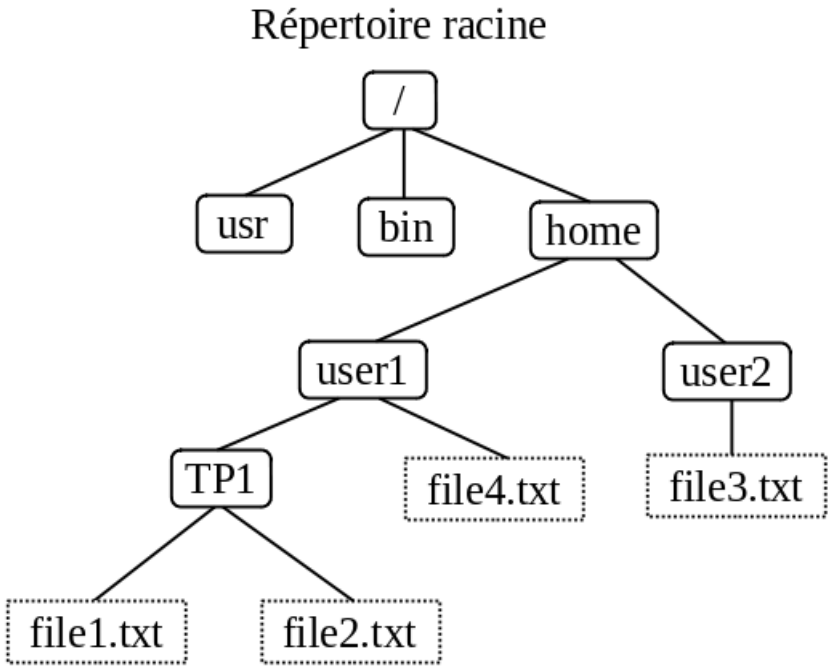
La commande **mv (Move)** permet de déplacer ou de renommer des fichiers.

Exemple :

Terminal ×

```
user1@pc1:~$ mv file3.txt /home/user2/file3.txt
user1@pc1:~$ mv file1.txt file4.txt
user1@pc1:~$ ls
TP1                file4.txt
```

Syntaxe : mv *source destination*



3.6 Commande rm

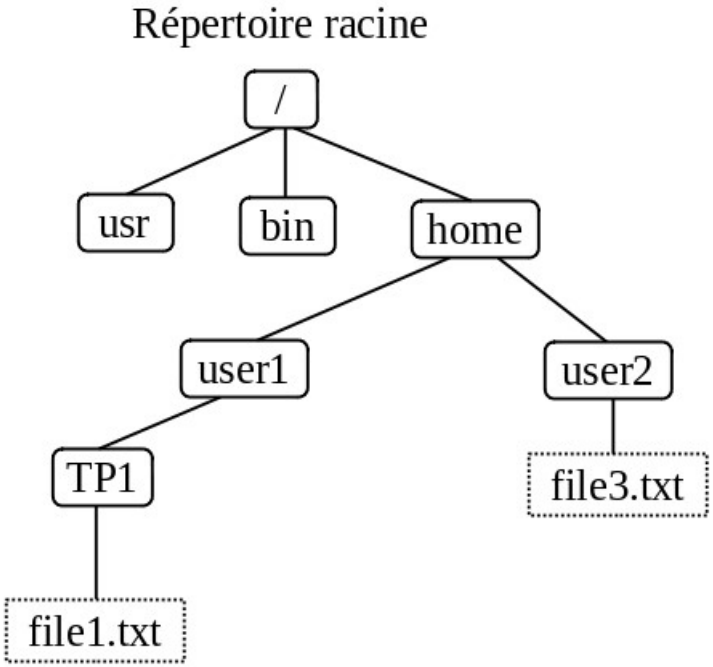
La commande **rm (Remove)** permet de supprimer des fichiers.

Exemple :

Terminal ×

```
user1@pc1:~$ rm file4.txt
user1@pc1:~$ rm TP1/file2.txt
user1@pc1:~$ ls
TP1
```

Syntaxe : rm *chemin/fichier*



3.7 Commande mkdir

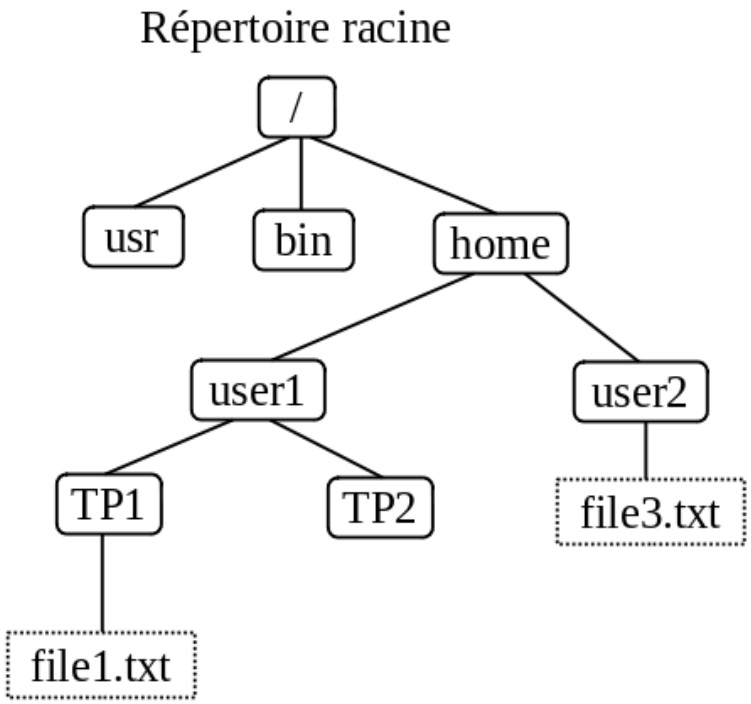
La commande **mkdir (Make directory)** permet de créer des répertoires.

Exemple :

Terminal ×

```
user1@pc1:~$ mkdir TP2
user1@pc1:~$ ls
TP1             TP2
```

Syntaxe : mkdir *répertoire*



3.8 Commande rmdir

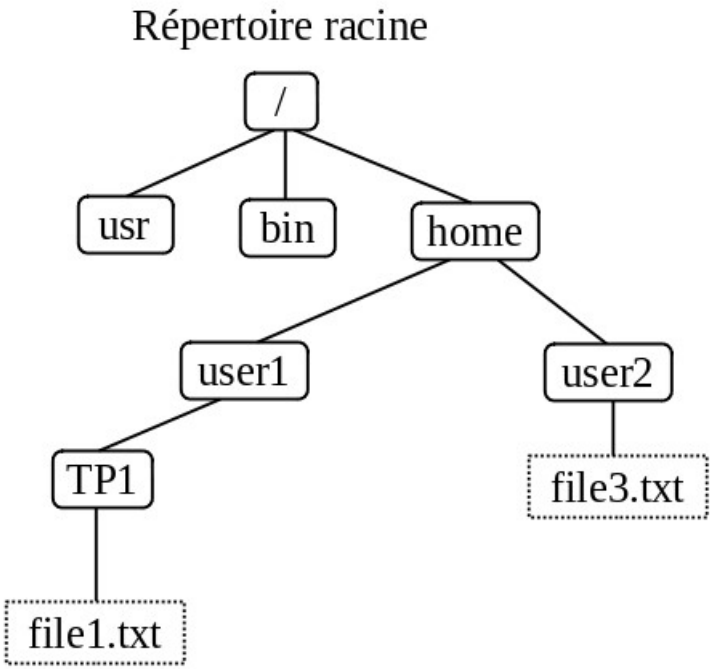
La commande **rmdir (Remove directory)** permet de supprimer des répertoires vides.

Exemple :

Terminal ×

```
user1@pc1:~$ rmdir TP2
user1@pc1:~$ ls
TP1
```

Syntaxe : rmdir *répertoire*



3.9 Commande touch

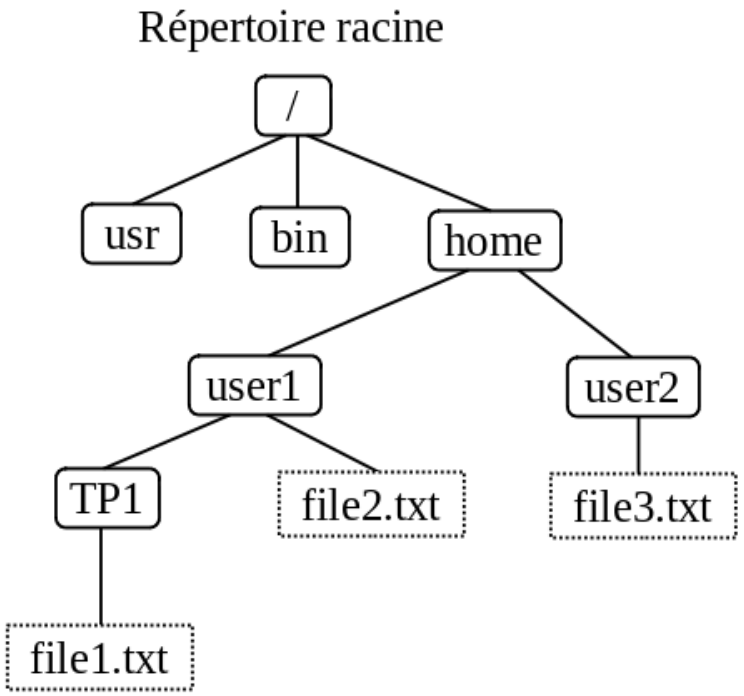
La commande **touch** permet notamment de créer un fichier vide.

Exemple :

Terminal ×

```
user1@pc1:~$  touch file2.txt
user1@pc1:~$  ls
TP1                 file2.txt
```

Syntaxe : touch *fichier*



3.10 Utilisation de joker

Un **joker** ou **métacaractère** est un type de caractère informatique utilisé lors de la recherche d'un mot ou d'une expression incomplète sur un réseau informatisé.
***** désigne une chaîne de caractères quelconques.

Exemple :

- ls b* : *liste tous les fichiers du répertoire courant commençant par b.*
- mv *.jpg repertoire_img : *déplace tous les fichiers du répertoire courant se terminant par .jpg dans le dossier repertoire_img.*
- rm *.bak : efface tous les fichiers du répertoire courant se terminant par* .bak

4 . Gestion des fichiers

4.1 Éditer un fichier avec Nano

Nano est un éditeur de texte accessible dans la console.
Pour le lancer, il suffit de taper "**nano file1.txt**" dans la console, pour ouvrir le fichier **file1.txt**. Certains raccourcis sont notés en bas de la fenêtre.

- Le symbole **^** correspond à la touche **contrôle**.
- Le symbole **M** correspond à la touche **alt**.

Exemple de raccourcis :

^G : accéder à l'aide

^O : enregistrer le fichier

^X : quitter

SHIFT appuyé + flèches permet de sélectionner

^K : couper

M6 : copier

^U : coller

Remarques :

- Nano gère la coloration syntaxique de nombreux langages et peut donc être utiliser pour écrire du code.
- Pour faciliter le codage, lorsqu'on tape "**nano -l**", les numéros de ligne apparaissent.
- Il existe deux autres éditeurs de texte célèbres, ayant plus de fonctionnalités : emacs et vim ceux-ci sont beaucoup plus riches mais moins facile d'accès.

Syntaxe : nano *fichier* ou nano -l *fichier*

4.2 Archive tar et fichier compressé

La commande **tar** permet de créer une archive au format **.tar** c'est à dire un fichier avec l'extension .tar qui contient tout le contenu d'un dossier.
Par défaut une archive n'est pas compressée, en rajoutant une option on peut créer une archive compressé qui prend moins de place en mémoire.

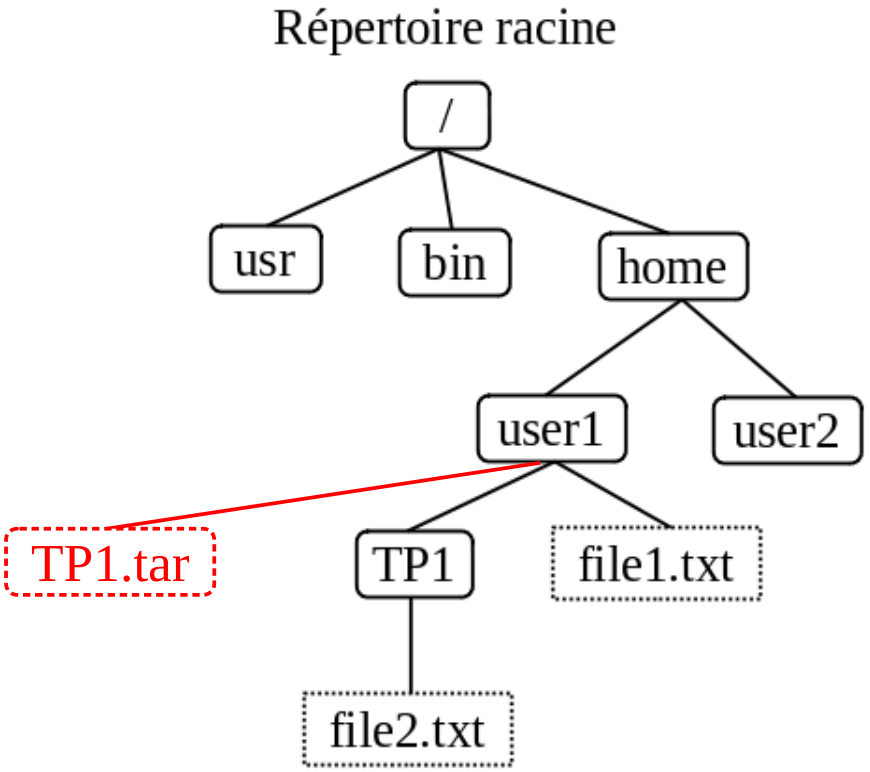
Options importantes de la commande tar :

-c : crée l'archive
-v : affiche la progression
-f : permet de spécifier le nom du fichier d'archive
-z : compresse avec gzip et crée une archive compressée au format .tar.gz
-t : affiche le contenu d'une archive
-x : permet d'extraire une archive

Exemple 1 : création d'une archive .tar

Terminal ×

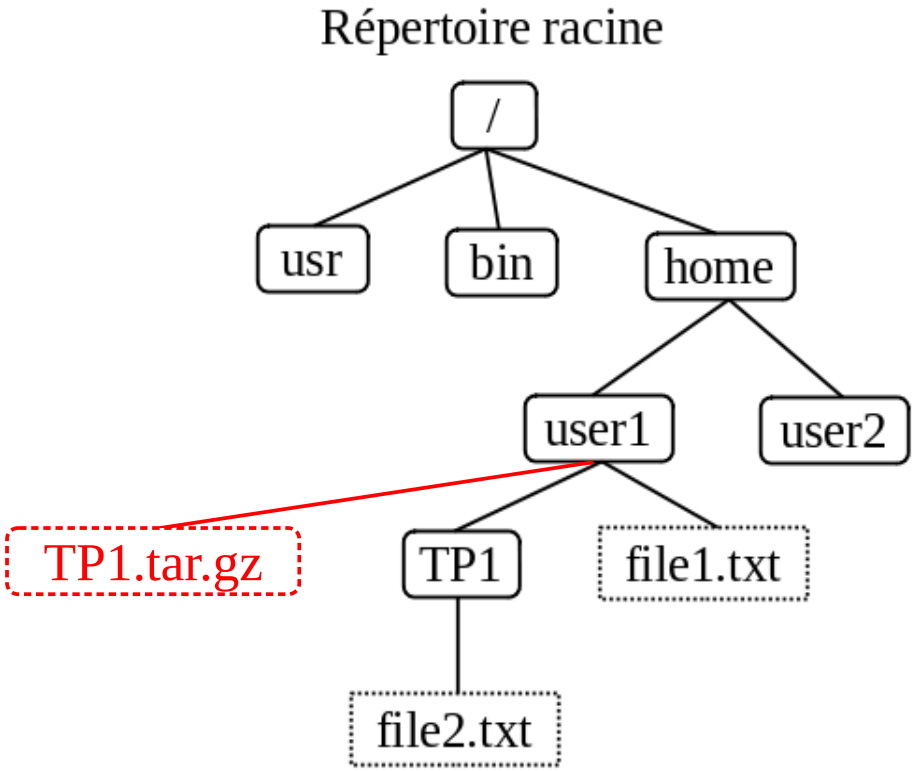
```
user1@pc1:~$ tar -cvf TP1.tar ~/TP1
tar: Suppression de « / » au début des noms des membres
/home/user1/TP1
/home/user1/TP1/file1.txt
```



Exemple 2 : création d'une archive compressée .tar.gz

Terminal ×

```
user1@pc1:~$ tar -czvf TP1.tar.gz ~/TP1
tar: Suppression de « / » au début des noms des membres
/home/user1/TP1
/home/user1/TP1/file1.txt
```



Exemple 3 : extraction d'une archive .tar dans le répertoire courant

Terminal ×

```
user1@pc1:~$ tar -xvf TP1.tar
/home/user1/TP1
/home/user1/TP1/file1.txt
```

Exemple 4 : extraction d'une archive .tar.gz dans le répertoire courant

Terminal ×

```
user1@pc1:~$ tar -xvf TP1.tar.gz
/home/user1/TP1
/home/user1/TP1/file1.txt
```

Exemple 5 : extraction d'une archive .tar dans un autre répertoire

Terminal ×

```
user1@pc1:~$ tar -xvf TP1.tar -C ~/Rep1/
/home/user1/TP1
/home/user1/TP1/file1.txt
```

Exemple 6 : lister une archive .tar dans la console

Terminal ×

```
user1@pc1:~$ tar -tvf TP1.tar
/home/user1/TP1
/home/user1/TP1/file1.txt
```

4.3 Les droits des fichiers

A chaque fichier est associé des droits suivant le type d'utilisateurs.

Il y a 3 types d'utilisateurs :

- **u** : (**user**) le propriétaire du fichier
- **g** : (**groupe**) le groupe d'utilisateurs du fichier
- **o** : (**other**) les autres

Il y a 3 types de permissions :

- **r** : (**read**) permission en lecture
- **w** : (**write**) permission en écriture
- **x** : permission en exécution

Exemple : Quand on exécute dans un terminal la commande **ls -al** on voit apparaître à côté du fichier : `-rwxr-x--x`

- le premier tiret "**-**" indique un fichier, ce serait "**d**" pour un répertoire ;
- le "**rwx**" indique les droits du propriétaire : il a ici accès au fichier en lecture, écriture et exécution ;
- le "**r-x**" indique les droits du groupe : il a ici accès au fichier en lecture et en exécution ;
- le "**--x**" indique les droits des autres : il ont accès au fichier qu'en exécution.

4.4 Modifier les droits des fichiers

La commande **chmod** permet de modifier les autorisations d'accès à un fichier.

Syntaxe : chmod *mode* *fichier*

Le mode se décompose en 3 parties :

- le type d'utilisateur : **u** , **g** , **o** ou **a** (pour all)
- ajout ou suppression : + ou -
- droits accordés : r,w,x

Exemples :

```
chmod o+x file1   (ajout des droits d'exécution aux autres)
chmod g-w file1   (suppression des droits d'écriture au groupe)
chmod a+rwx file1   (ajout de tous les droits à tous les utilisateurs)
```

Notation octale : Il existe une notation octale avec trois chiffres des droits

- premier chiffre : **propriétaire**
- deuxième chiffre : **groupe**
- troisième chiffre : **autres**

Le chiffre **1** signifie droit en exécution (**x**), le chiffre **2** signifie droit en écriture (**w**) et le chiffre **4** signifie droit en lecture (**r**).
On peut alors combiner par addition ces chiffres pour accorder plusieurs droits :
1 + 2 = 3 signifie droit en exécution et écriture, c'est encore plus significatif si on écrit ces chiffres en binaire.

Exemple : `chmod 751 file1` s'interprète ainsi :

7 [4 + 2 + 1 = $(111)_2$] : le propriétaire a tous les droits
5 [4 + 0 + 1 = $(101)_2$] : le groupe a accès en lecture et en exécution
1 [0 + 0 + 1 = (001)2] : les autres ont accès qu'en exécution

Remarque : Il existe les commandes **chown** et **chgroup** pour changer les propriétaires et les groupes des fichiers.

EXERCICE 1

On considère la commande suivante tapée dans la console.

Terminal ×

```
user1@pc1:~$  chmod  734  file1.txt
```

Interpréter la commande ci-dessus en indiquant les droits de chaque type d'utilisateurs.

- droits du propriétaires : .....
- droits du groupe : .....
- droits des autres : .....

CORRECTION

On considère la commande suivante tapée dans la console.

Terminal ×

```
user1@pc1:~$  chmod  734  file1.txt
```

Interpréter la commande ci-dessus en indiquant les droits de chaque type d'utilisateurs.

- droits du propriétaires : 7 = 4 + 2 + 1 : tous les droits
- droits du groupe : 3 = 2 + 1 : droit en écriture et en exécution
- droits des autres : 4 : droit en lecture

EXERCICE 2

Ecrire la commande pour établir les droits d'accès suivants concernant le fichier file2.txt :

- droits du propriétaires : lecture et écriture
- droits du groupe : lecture et exécution
- droits des autres : exécution

Terminal ×

```
user1@pc1:~$ .....
```


CORRECTION

Ecrire la commande pour établir les droits d'accès suivants concernant le fichier file2.txt :

- droits du propriétaires : lecture et écriture
- droits du groupe : lecture et exécution
- droits des autres : exécution

Terminal ×

```
user1@pc1:~$ chmod 651 file1.txt
```


5 . Gestion des processus

5.1 Vocabulaire

5.1.1 Les programmes

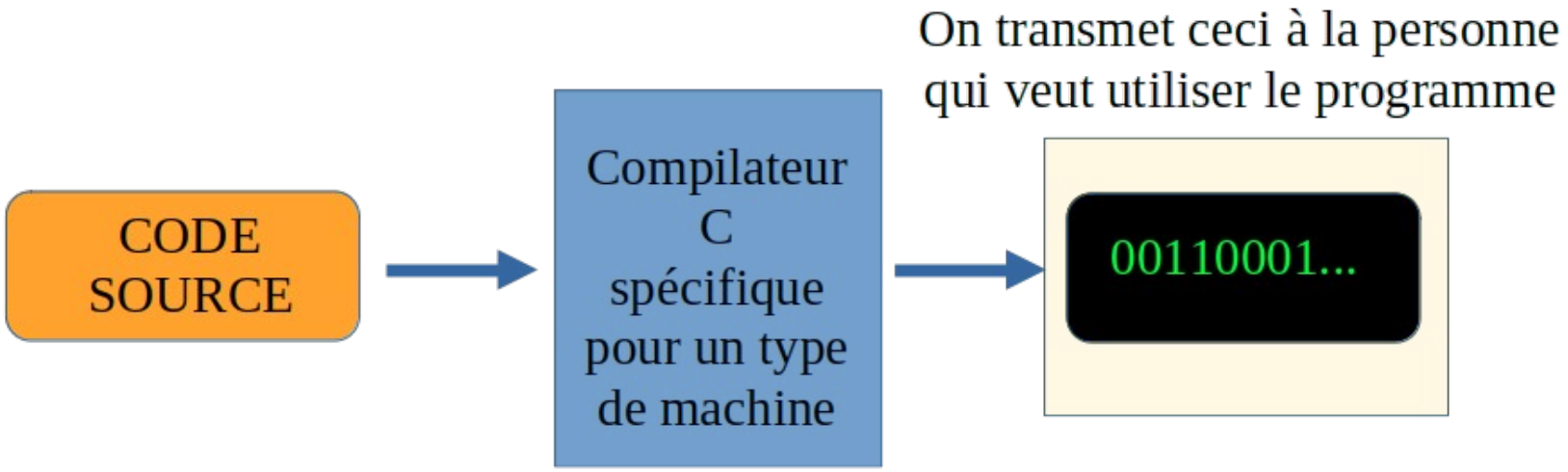
Définition : Un **programme** est un ensemble d'**instructions** permettant de faire réaliser certaines tâches à un système informatique. Il s'agit donc d'une implémentation en machine d'un **algorithme** (algorithme qui peut lui être vu comme l'idée abstraite de l'action à accomplir).

En réalité, derrière ce mot "**programme**" se trouve deux notions :

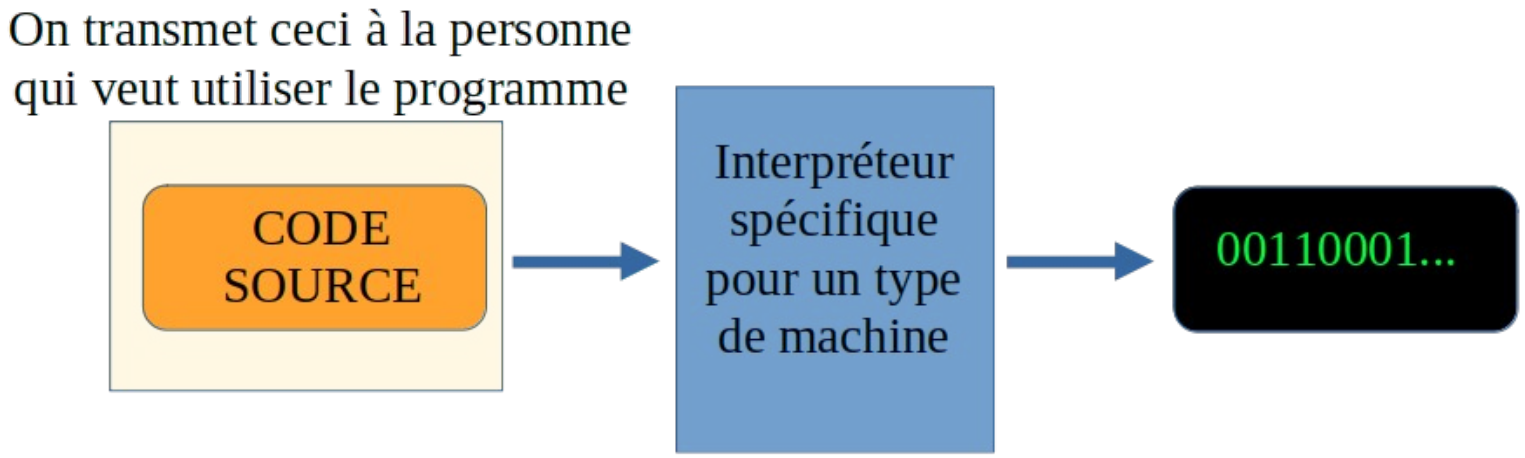
1. Un **programme binaire** (ou en **langage machine**) : Il s'agit d'une **suite de bits** directement compréhensible par le **processeur**.
   Exemple : `10110000 10000000`
   On peut traduire mot à mot ce langage binaire en utilisant quelques mots clés, cette traduction s'appelle le **langage assembleur**.
   Exemple : `10110000 10000000` se traduit en assembleur par :
   `MOV AL # 128` ce qui signifie écrire la valeur 128 dans le registre nommé **AL**

2. Un **programme source** (ou **code source**) : Il s'agit des instructions fournies dans un **langage de programmation** (Python par exemple). Ce code source est compréhensible par un humain, mais pas par le processeur. Il doit donc être traduit en **langage machine**.
Il y a deux grandes méthodes pour traduire un code source en langage machine.

- **les langages compilés :** On transmet le **code source** à un **compilateur** qui traduit ce **code source en langage machine** puis on transmet à l'utilisateur ce code machine (fichier .exe sous windows).



• **les langages interprétés :** On transmet le **code source** à un **interpréteur** qui traduit ce code source en langage machine à la volée. On transmet donc à l'utilisateur le code source.



5.1.2 Les processus

Définition : Un **processus** (ou une tâche) est l'exécution d'un **programme** par le **processeur** à partir d'une zone de mémoire vive (**RAM**). On dit qu'un **processus** est une **instance de programme**. Le programme est donc le moule permettant de générer un ou plusieurs processus.

Remarques : Quand on veut exécuter un **programme**, c'est à dire une suite d'**octets** stockés en **mémoire de masse** (ou **mémoire morte**), on va devoir :

1. Réserver de la place en mémoire vive (RAM) (on parle de **virtualisation de la mémoire**).
2. Copier le code du programme depuis la mémoire de masse en mémoire vive.
3. Commencer à gérer l'exécution du code du **processus**.

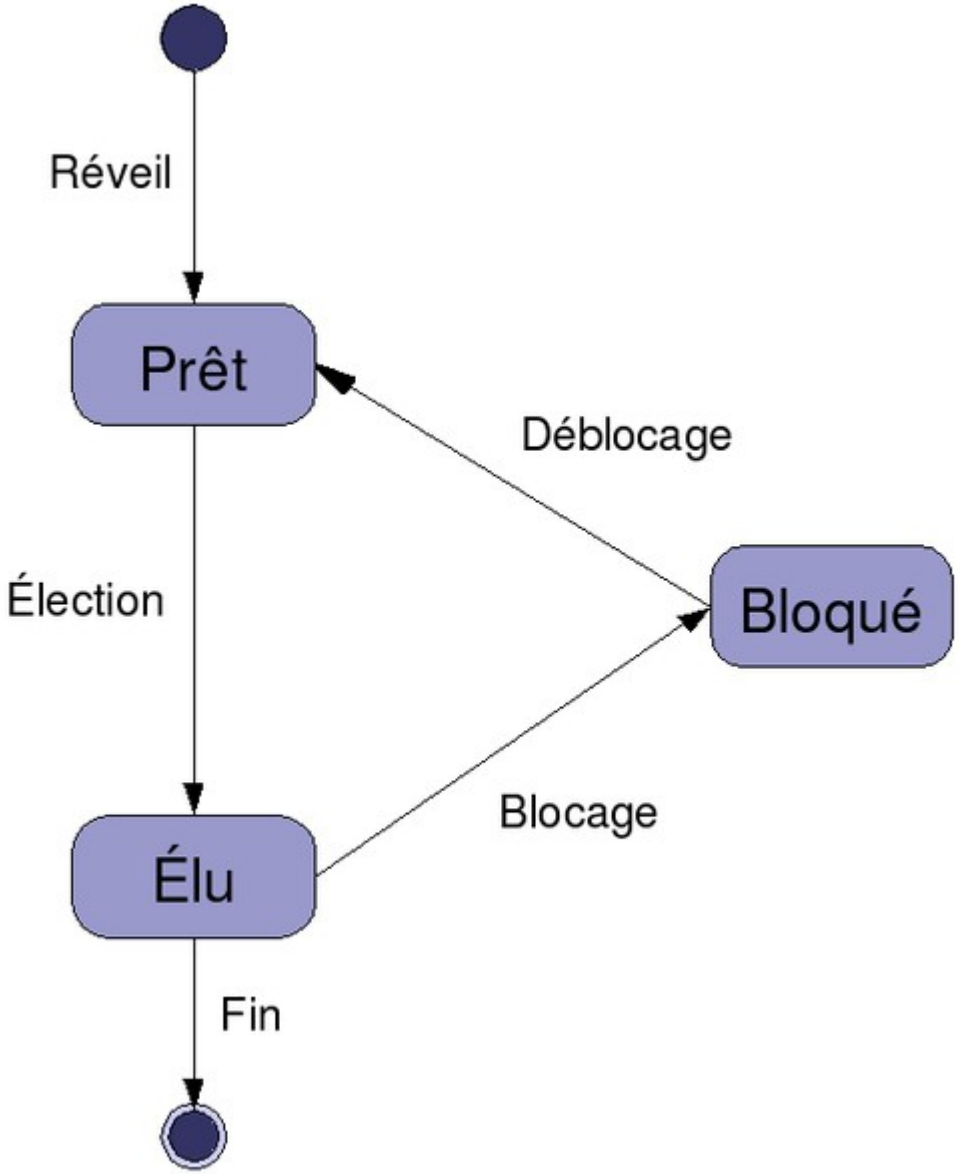
5.2 États des processus

Un **processus** a besoin d'accéder au **processeur** pour être exécuté, mais il a aussi besoin d'accéder à d'autres ressources comme :

- La **mémoire vive** (**RAM**) ;
- La **mémoire de masse** (**ROM** ou **mémoire morte**) ;
- La lecture ou l'écriture d'un **fichier** ;
- Les **périphériques** d'entrée et de sortie : souris, clavier, écran, imprimante, …

Ces ressources externes au processeur étant beaucoup moins rapide que le processeur, elles bloquent les processus lors de leur exécution.
Ainsi, lors de la vie d'un processus, celui-ci peut passer par quatre états :

- **PRÊT** : le processus est lancé et attend l'accès au processeur ;
- **ÉLU** : le processus a obtenu l'accès au processeur : il peut s'exécuter ;
- **BLOQUÉ** : le processus est en cours d'exécution, mais il attend une ressource en mémoire par exemple, il quitte le processeur pour libérer les ressources.
- **TERMINÉ** : le processus est terminé.



5.3 Ordonnancement

A un instant donné, il y a en général plus de **processus** à exécuter que de **processeurs**.
Lors de leur lancement, les **processus** sont placés dans une **file d'attente**.
L'**ordonnanceur** est le composant du **noyau du système d'exploitation** qui choisi **l'ordre d'exécution des processus** sur les **processeurs** d'un ordinateur.
Il existe plusieurs **algorithmes d'ordonnancement** :

- la **méthode du tourniquet** (**Round Robin**) : l'ordonnanceur traite la file d'attente comme une file circulaire et alloue successivement un temps processeur à chacun des processus de la **file** ;
- la **méthode FIFO** (First In First Out) ;
- la **méthode SJF** (Shortest Job First) on affecte le processeur au processus possédant le temps d'exécution le plus court.
- la **méthode à priorité** L'utilisateur donne des priorités aux différents processus et ils sont activés en fonction de cette priorité et de leur ordre de déclenchement.

5.4 Gestionnaire d'interruptions

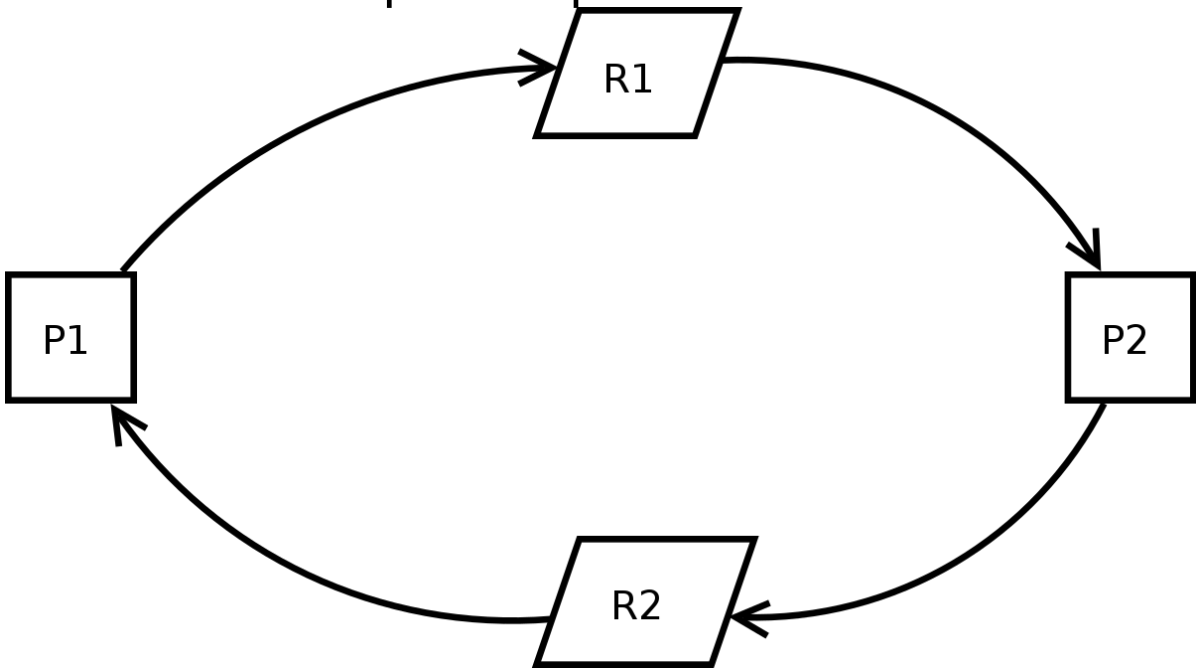
Les **systèmes d'exploitation** intègrent un **système de gestion d'interruption** : un processus peut signaler qu'il doit effectuer des actions si un certain événement apparaît (clic de la souris ou appuie sur une touche du clavier, ...)

Le système d'exploitation peut alors interrompre le fonctionnement du **processeur** et provoquer une **nouvelle élection** d'un processus particulier qui est à l'état bloqué.

Un exemple d'interruptions est **l'interruption d'horloge** qui permet de provoquer périodiquement une **nouvelle élection**.

5.5 Interblocage

Un **interblocage** (**deadlock** en Anglais) est un phénomène qui peut se produire en programmation concurrente lorsque des processus s'attendent mutuellement.



- le processus P1 acquiert la ressource R1.
- le processus P2 acquiert la ressource R2.
- le processus P1 attend pour acquérir R2 (qui est détenu par P2).
- le processus P2 attend pour acquérir R1 (qui est détenu par P1).

Dans cette situation, les deux processus P1 et P2 sont définitivement bloqués.

Il existe plusieurs façons de gérer les **interblocages** :

- les **ignorer** ce qui était fait initialement par UNIX qui supposait que la fréquence des interblocages était faible et que la perte de données encourue à chaque fois est tolérable.
- les **détecter** : Un **algorithme** est utilisé pour suivre l'allocation des ressources et les états des processus, il annule et redémarre un ou plusieurs processus afin de supprimer le blocage détecté.
- les **éviter** : des **algorithmes** sont utilisés pour supprimer une des quatre conditions nécessaires à la possibilité de l'interblocage.

EXERCICE 3

Dans un bureau d'architectes, on dispose de certaines ressources qui ne peuvent être utilisées simultanément par plus d'un processus, comme l'**imprimante**, la **table traçante**, le **modem**. Chaque programme, lorsqu'il s'exécute, demande l'allocation des ressources qui lui sont nécessaires. Lorsqu'il a fini de s'exécuter, il libère ses ressources.

Programme 1	Programme 2	Programme 3
demander (table traçante)	demander (modem)	demander (imprimante)
demander (modem)	demander (imprimante)	demander (table traçante)
exécution	exécution	exécution
libérer (modem)	libérer (imprimante)	libérer (table traçante)
libérer (table traçante)	libérer (modem)	libérer (imprimante)

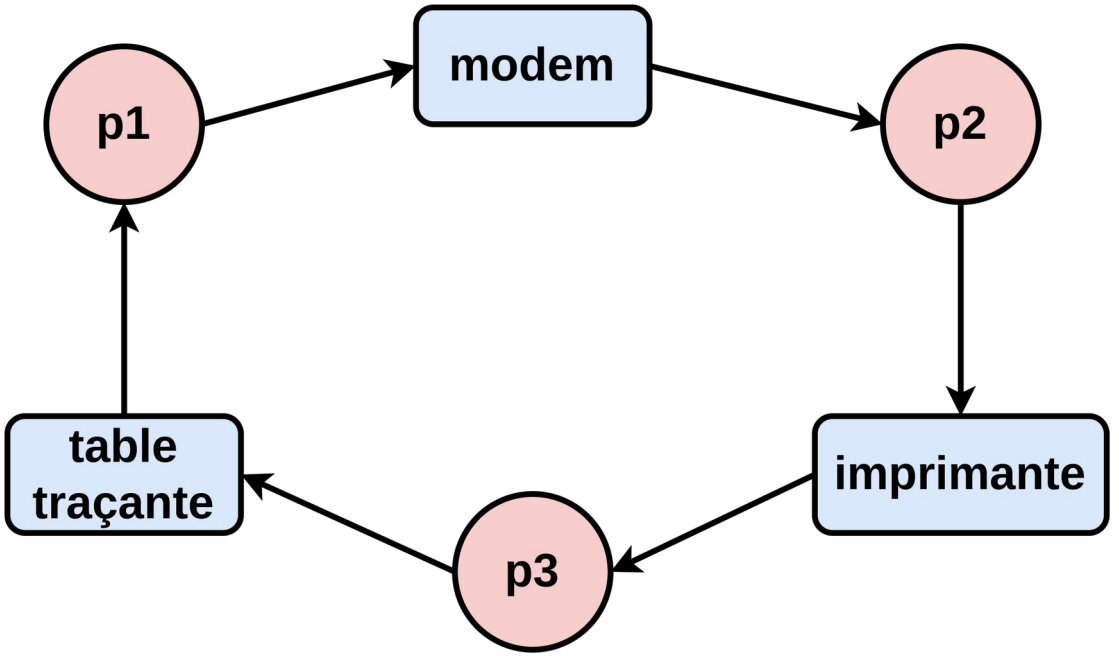
On appelle p1, p2 et p3 les processus associés respectivement aux programmes 1, 2 et 3.

a. Justifier à l'aide d'une figure qu'une situation d'interblocage peut se produire.

b. Modifier l'ordre des instructions du programme 3 pour qu'une telle situation ne puisse pas se produire.

CORRECTION

a. Figure d'interblocage :



b. Modification du programme 3 pour éviter l'interblocage :

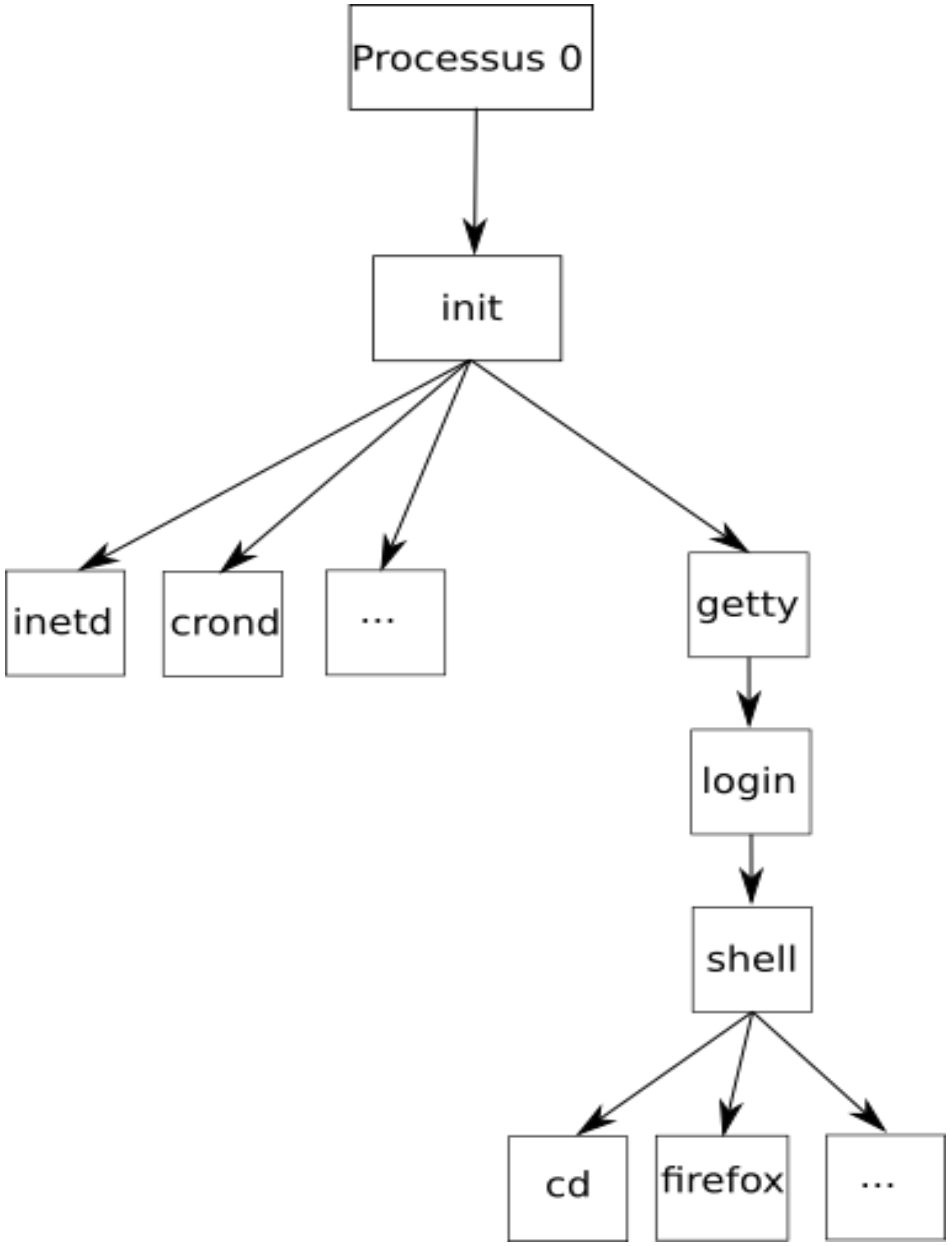
```
Programme 3
demander (table traçante)
demander (imprimante)
exécution
libérer (imprimante)
libérer (table traçante)
```

5.6 Création d'un processus

Un **processus** peut créer un ou plusieurs processus à l'aide d'une commande système ("fork" sous les systèmes de type Unix). Imaginons un processus A qui crée un processus B. On dira que **A est le père de B** et que **B est le fils de A**. B peut, à son tour créé un processus C (B sera le père de C et C le fils de B). On peut modéliser ces relations père/fils par une structure arborescente.
Si un processus est créé à partir d'un autre processus, comment est créé le tout premier processus ?

Sous un système d'exploitation comme **Linux**, au moment du démarrage de l'ordinateur un tout premier processus (appelé **processus 0** ou encore **Swapper**) est créé à partir de "rien" (il n'est le fils d'aucun processus). Ensuite, ce processus 0 crée un processus souvent appelé "**init**" ("**init**" est donc le fils du processus 0). À partir de "**init**", les processus nécessaires au bon fonctionnement du système sont créés (par exemple les processus "**crond**", "**inetd**", "**getty**",...). Puis d'autres processus sont créés à partir des fils de "init"...

On peut résumer tout cela avec le schéma ci-après :



5.7 Commandes linux

5.7.1 La commande ps

La commande **ps** permet d'obtenir la liste des **processus actifs** dans un terminal Linux.

Exemple :

Terminal ×

```
user1@pc1:~$ ps
PID    TTY      TIME      CMD
8716   pts/0    00:00:00  bash
17810  pts/0    00:00:00  ps
```

Options :

- **ps -a** : Affiche tous les processus
- **ps -u** : Affiche le noms des utilisateurs qui ont lancé le processus
- **ps -x** : Affiche les processus qui n'ont pas été lancé par la ligne de de commande
- **ps -l** : Affiche en plus d'autres informations sur les processus
- **ps -p pid** : Affiche le processus dont l'identifiant est pid

ps -eo pid,ppid,stat,command : Affiche des informations ciblées pour chaque processus

Informations sur les processus :

PID : (**Process Identifier**) : identifiant entier du processus. Le premier processus créé au démarrage du système à pour PID 0, le second 1, le troisième 2...

PPID : (**Parent Process Identifier**) : identifiant du processus parent.

STAT : état du processus.

- **R** : (Running ou Runnable) : **Prêt** ou **En exécution**
- **S** : (Sleeping) : **En attente**
- **T** : (Terminated) : **Terminé**
- **Z** : (Zombie) : **Terminé mais toujours chargé en mémoire**
- …

COMMAND : le nom du processus

5.7.2 Autres commandes

- La commande **top** permet d'obtenir **en temps réel** la liste des **processus en cours**.
- La commande **pstree** permet d'obtenir la hiérarchie des processus sous forme d'une arborescence.
- La commande **kill** permet d'arrêter les processus dont le **PID** est passé en argument : **kill 1142** arrête le processus dont le PID est 1142.

EXERCICE 4

Avec une ligne de commande, dans un terminal linux, on obtient l'affichage suivant :

UID	PID	PPID	C	STIME	TTY	TIME	CMD
root	1	0	0	10:01	?	00:00:02	/sbin/init splash
root	4	2	0	10:01	?	00:00:00	[kworker/0:0H]
root	6	2	0	10:01	?	00:00:00	[mm_percpu_wq]
....	....	.....	....	.....	.....	.....	.....
bob	3383	1	0	10:25	?	00:00:00	sh -c /usr/bin/google-chrome-sta

1. Parmi les commandes suivantes, quelle est celle qui a permis d'obtenir cet affichage :

a. ls -l **b.** cd .. **c.** ps -aef **d.** chmod 741 processus.txt

2. Quel est l'identifiant du processus qui a lancé google-chrome ?

3. Quel est l'identifiant du processus parent du processus qui a lancé google-chrome ?

CORRECTION

Avec une ligne de commande, dans un terminal linux, on obtient l'affichage suivant :

UID	PID	PPID	C	STIME	TTY	TIME	CMD
root	1	0	0	10:01	?	00:00:02	/sbin/init splash
root	4	2	0	10:01	?	00:00:00	[kworker/0:0H]
root	6	2	0	10:01	?	00:00:00	[mm_percpu_wq]
....	....	.....	....	.....	.....	.....	.....
bob	3383	1	0	10:25	?	00:00:00	sh -c /usr/bin/google-chrome-sta

1. Parmi les commandes suivantes, quelle est celle qui a permis d'obtenir cet affichage :

a. ls -l **b.** cd .. **c.** **ps -aef** **d.** chmod 741 processus.txt

2. Quel est l'identifiant du processus qui a lancé google-chrome ? **3383**

3. Quel est l'identifiant du processus parent du processus qui a lancé google-chrome ? **1**