Architecture matérielle, systèmes d’exploitation

Table des matières

I) composants intégrés d’un système sur puce

A) Un peu d’histoire :

ENIAC (1945) : 30 tonnes, 18 000 tubes à vide, pas de CPU intégré → composants discrets
Années 1970 : Premiers microprocesseurs (Intel 4004) - CPU sur une puce
Années 1980 : Ajout de mémoires cache, contrôleurs intégrés
Années 2000 : SoC (System on Chip) - CPU + GPU + RAM + radios dans une seule puce
Smartphones modernes : SoC avec CPU multi-cœur, NPU, 5G, GPU haut de gamme (ex: Snapdragon, Apple Silicon)

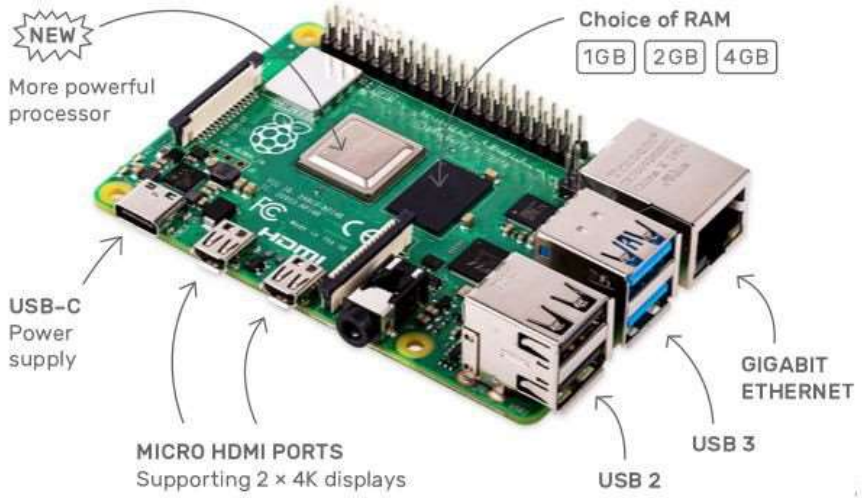
B) Le SoC (System on Chip):

Un **System on Chip (SoC)** est une puce électronique qui intègre tous les composants essentiels d'un système informatique complet. Contrairement aux architectures traditionnelles où chaque composant (CPU, GPU, mémoire, contrôleurs) est séparé, un SoC rassemble tout dans un seul circuit intégré.

Avantages d'un SoC :
- **Miniaturisation** : Idéal pour smartphones, tablettes, IoT (raspberry Pi, drones).
- **Économie d'énergie** : Moins de composants = moins de pertes électriques.
- **Performances optimisées** : Communication ultra-rapide entre CPU/GPU/mémoire.
- **Coût réduit** en production de masse.

C) Exemple avec le Raspberry pi :

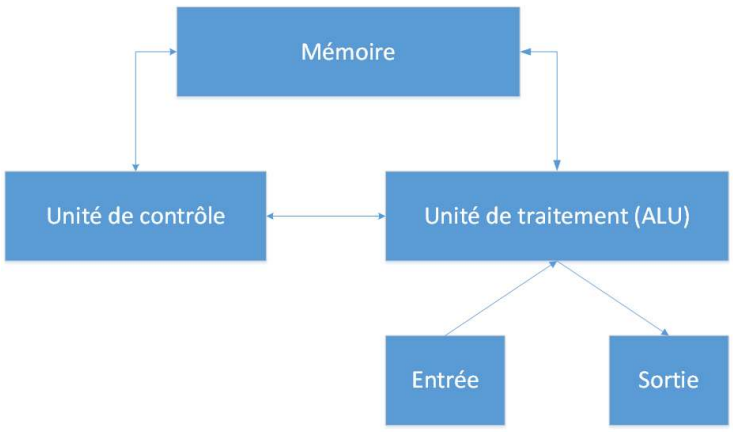
Le Raspberry pi est un **« single board computer »**, probablement le plus connu, et est un microordinateur ne se trouvant que sur une plaquette. Il s'agit donc d'un **SoC**.
Modélisation :



Nos téléphones, par exemple, utilisent tous un SoC ce qui leur permet d'avoir leur taille réduite.

D) Mémoire :

Il existe différents types de mémoire que voici :
- le **registre processeur** (comme des feuilles sur votre bureau)
- la mémoire **cache** dans un **processeur** (comme des feuilles dans un tiroir)
- la mémoire **RAM** (comme des feuilles d'archive localisées dans la pièce voisine)
- la mémoire de **stockage** (dans le SSD) (comme des feuilles archivées au sous-sol)
(RAM:Random Access Memory, SSD : Solid State Drive)



E) schéma de circuit du Raspberry pi :

II) systèmes d’exploitation : Gestion des processus

A) Définition :

Un processus est un programme en cours d’exécution sur un ordinateur.
Il possède quelques caractéristiques :

- **Exécution autonome** : Chaque processus s'exécute indépendamment des autres.

- **Espace mémoire édié** : Un processus possède sa propre zone mémoire (isolée des autres processus).

- **Ressources allouées** (fichiers ouverts, connexions réseau, etc.).

- Géré par le **système d'exploitation (OS)** qui planifie son exécution.

B) création d’un processus :

Un processus peut se créer :

- Lors du **démarrage** du système
- Par l’appel d’un **autre processus**
- Par l’action d’un **utilisateur** (lancement d’une application par exemple)

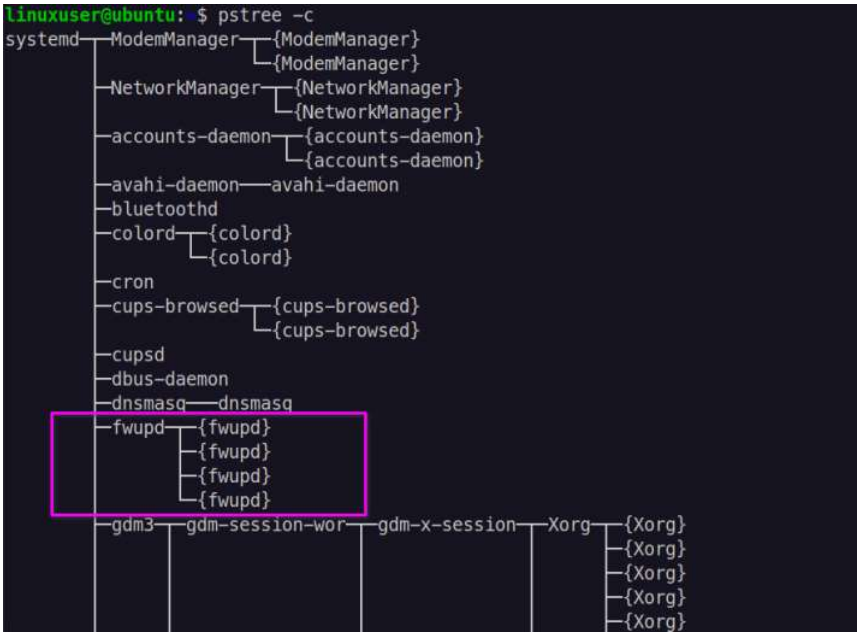
Sur Linux, un processus est créé par le clonage d’un autre à travers l’appel système : **fork().**
Le processus appelant fork est appelé le processus père.
Logiquement, le processus étant crée est appelé le **processus fil / fils.**

Un processus a forcément un identifiant. Son **PID**. (Process Identifier).
Le processus ayant créé ce même processus est le **PPID** (Parent Process Identifier) de ce processus.

C) Arborescence et « ordre » :

le système de création (fork) des processus mène à une structure particulière. On appelle arborescence des processus cette structure . Ceci est dû aux parents qui créent des fils qui eux-mêmes engendrent des fils et ainsi de suite.

Sur Linux le parent de tous les processus est appelé **init** et est créé lors du démarrage. L'instruction **pstree** permet de visualiser cet arbre et il ressemblera à cela :



Où on reconnaît parfaitement la structure d'un arbre.

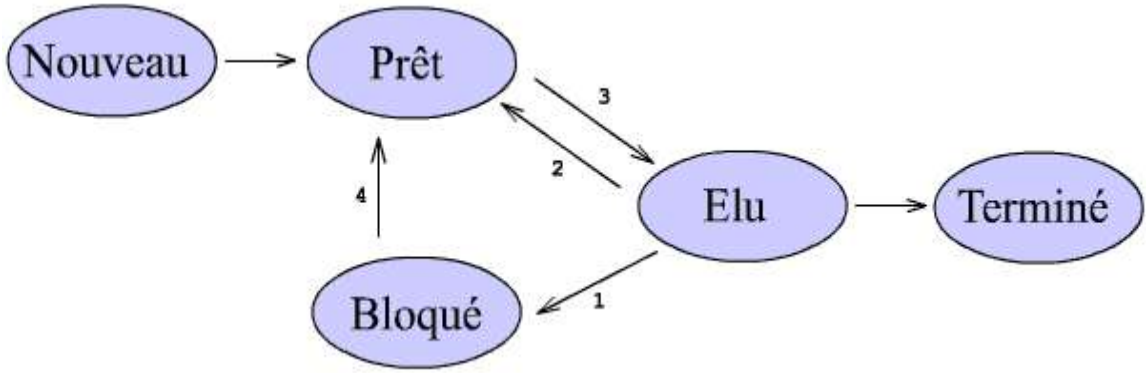
Il y a une liste de commandes qui permettent d'effectuer des actions en rapport avec les processus.

Les voici :
- **h** : afficher l'aide
- **M** : trie la liste par ordre décroissant en mémoire. Permet de casser ceux trop gourmands
- **P** : trie la liste par ordre décroissant d'occupation processeur
- **i** : filtre les processus inactifs. Ne montre que ceux qui travaillent réellement
- **k** : permet de tuer un processus, à condition d'en être propriétaire. (ne tuez JAMAIS init)
- **V** : permet d'avoir la vue arborescente sur les processus.
- **q** : permet de quitter top
- **top** : affiche tous les processus

Pour terminer un processus :
- **SIGTERM()** : demande la terminaison d'un processus en se terminant proprement avec les ressources dont il a besoin.
- **SIGKILL()** : terminaison violente d'un processus à n'utiliser que sur les processus qui ne répondent pas à SIGTERM.

C) Ordonnancement des processus par l'OS :

Un ordinateur normal possède un processeur (à 1 coeur) qui ne peut accomplir qu'une tâche à la fois. Il faut donc départager les processus par temps qu'ils passent en exécution dans le processeur. C'est la travail de l'**ordonnanceur** . Il doit déterminer le prochain processus qui doit être exécuté. On a donc différents états dans lequel un processus peut se trouver :
-**prêt** : le processus attend son tour pour s'exécuter
-**en exécution** : le processus a accès au processeur pour exécuter ses instructions
-**en attente** : le processus attend un évènement (saisie clavier, libération d'une donnée...)
-**arrêté** : soit il est fini, soit il a été terminé par sigkill/sigterm et il libère ses ressources
-**zombie** : il a fini, il attend que son père l'élimine. Cela peut durer si le père vit longtemps.
Les 3 premiers états sont les plus importants à retenir et on obtient un schéma comme celui qui suit :



Sur Linux il est possible de donner des ordres à l'ordonnanceur en fixant des priorités à des processus dont on est propriétaire : on change leur valeur de priorité (entre +-20).

Les utilisateurs autres que le **super-utilisateur root** ne peuvent que diminuer la priorité de leurs processus. Et encore, ils sont restreints entre 0 et 19. **Seul root** peut modifier l'intégralité de l'échelle, pour n'importe quel processus actif.

On peut donc agir à 2 niveaux :
- fixer une priorité à une nouvelle tâche dès son démarrage avec la commande **nice**
- modifier le priorité d'un processus en cours grâce à la commande **renice** .

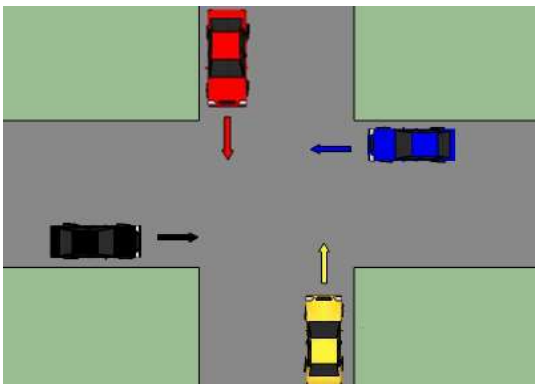
Voici une illustration de renice :

```
top - 11:20:12 up 49 days,  9:11,  3 users,  load a
Tasks: 185 total,   1 running, 175 sleeping,   8 st
%Cpu(s):  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 w
MiB Mem :  15936.8 total,   9743.5 free,    787.9 u
MiB Swap:   4096.0 total,   4096.0 free,      0.0 u
Renice PID 1008034 to value 0
    PID USER      PR  NI    VIRT    RES    SHR S  %
  52678 mysql     20   0 2869128 439016  37708 S
1004281 root      20   0       0      0      0 I
1008034 root      21   1   10476   4036   3436 R
      1 root      20   0  167808  13512   8460 S
      2 root      20   0       0      0      0 S
      3 root       0 -20       0      0      0 I
      4 root       0 -20       0      0      0 I
      5 root       0 -20       0      0      0 I
      6 root       0 -20       0      0      0 I
      8 root       0 -20       0      0      0 I
     10 root       0 -20       0      0      0 I
```

Ci-dessus, on a entré : une priorité de 0 le programme ayant le PID de 1008034 (le PID étant unique).
Sur l'image ci-dessus, les colonnes **PR** et **NI** montrent le niveau de priorité de chaque processus.
Leur relation est simple : **PR = NI + 20**
Donc une priorité PR de 0 correspond à un niveau de priorité **maximal.**

D) Interblocage :

Uns situation d'interblocage peut se produire. Voici une schématisation intuitive de la chose :



Cette situation d'interblocage a été théorisée par **Edward Coffman** qui a énoncé **quatre conditions** (appelées « de coffman ») menant à un **interblocage** :
-**exclusion mutuelle** : au moins une des ressources du système doit être en accès exclusif
-**Rétention des ressources** : un processus détient au moins une ressource et en requiert une autre
-**Non préemption** : seul le détenteur d'une ressource peut la libérer
-**Attente circulaire** : chaque détenteur d'une ressource attend la libération d'une ressource qu'un processus tiers possède (dans le schéma, chaque voiture attend la libération de la route de droite par une autre voiture.).

Il existe cependant des moyen de prévenir l'arrivée de ces situations. Ces méthodes ne seront pas abordées ici, car elles sortent du programme de terminale.