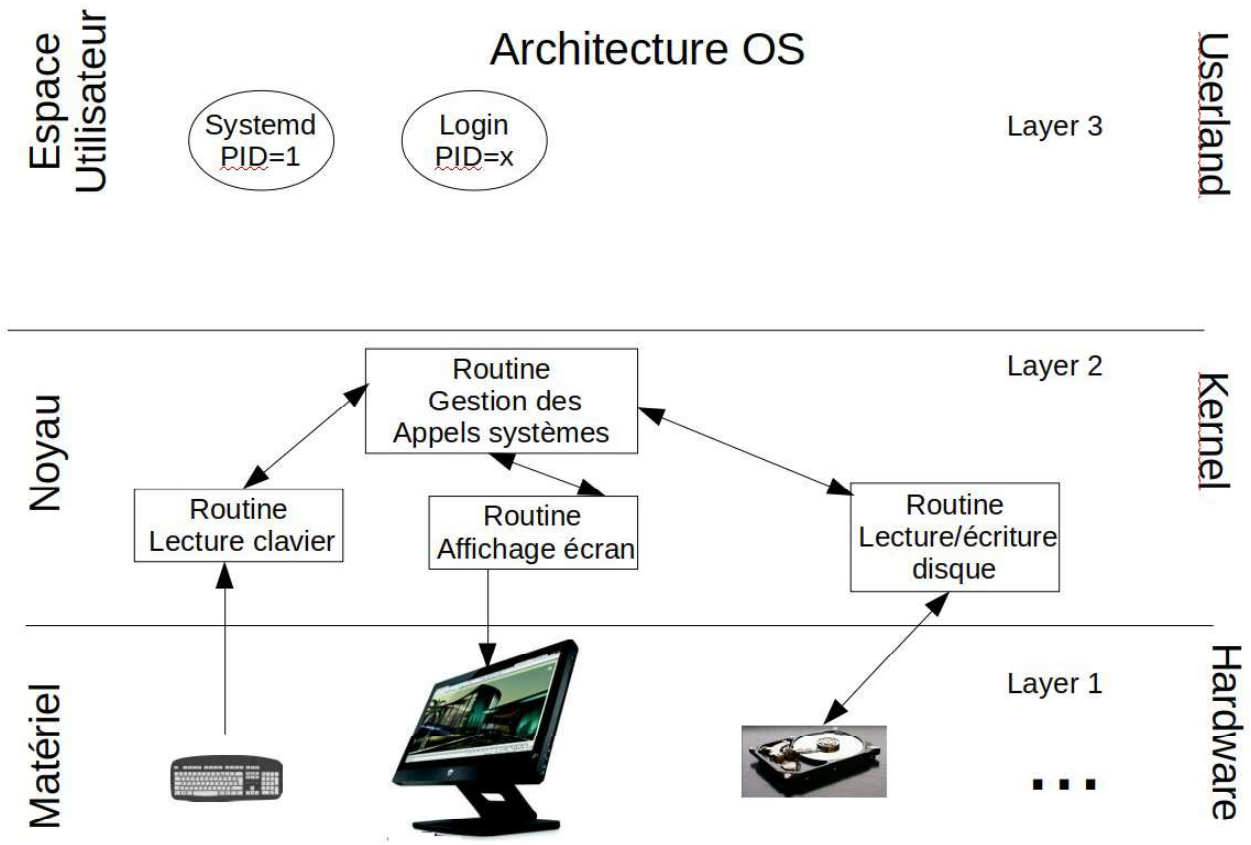
ARCHITECTURES MATÉRIELLES ET REPRÉSENTATION DES DONNÉES

NSI 02

SYSTÈME D’EXPLOITATION : ZOOM SUR LES PROCESSUS

1. Introduction et rappels de première



En première, pour comprendre comment fonctionne un système d'exploitation, nous avions découpé (schématiquement!) un ordinateur en 3 couches :

- la couche « matériel » : elle est constituée de tous les périphériques et ressources matérielles à disposition de la machine et/ou de l'utilisateur

- la couche « noyau » : c'est LA partie fondamentale du système d'exploitation qui est l'intermédiaire entre le matériel et les requêtes envoyées de l'espace utilisateur. Le code des programmes tournant dans cette couche est appelé **routine**.

- la couche « espace utilisateur » : on y trouve toutes les applications lancées et exploitées par l'utilisateur. Lorsqu'un programme est lancé, le système d'exploitation crée un (ou plusieurs) **processus** qui va (vont) demander au noyau de faire une action ou de renvoyer un résultat. Ces demandes sont des **appels système**.

Dans ce chapitre, nous allons étudier la manière dont le noyau attribue du temps processeur à chacun des processus afin qu'il exécute son code. La distribution du temps processeur à chacun des processus par le noyau s'appelle l'**ordonnancement** des processus (***scheduling*** en anglais).

2. Processus : notion d’états et ordonnancement

2.1. Naissance d’un processus

Au démarrage d’un ordinateur, le premier élément du système d’exploitation qui se lance en mémoire est le noyau, avec toutes ses routines. Une fois opérationnel, le noyau crée le premier processus dans la couche «userland». Sous Linux, ce premier processus s’appelle « Systemd » : il est chargé de lancer dans la couche «userland» les premières applications prévues lors du démarrage du système d’exploitation (par exemple, celle demandant un identifiant et un mot de passe pour se connecter à une session).

Pour identifier un processus, le noyau utilise une valeur appelée PID (*Process Identifieur*). Sous Linux, le noyau affecte au processus « Systemd » le PID 1.

Un processus est enfermé par le noyau dans une zone mémoire qui lui est propre : il ne peut donc pas interférer avec un autre processus. Ainsi, un processus qui dysfonctionne ne peut pas perturber le fonctionnement d’un autre processus.

Dans la zone mémoire d’une processus, on trouve :

- le code à exécuter par ce processus ;
- les données qu’il manipule.

Une fois mis en place, le processus « Systemd » demande au noyau de lancer d’autres processus. Ces derniers seront les « processus enfants » (parfois appelés « processus fils ») de « Systemd ». On dit aussi que « Systemd » est le « processus père » de ces nouveaux processus.

Une fois l’ensemble de ces processus en action, ceux-ci peuvent demander au noyau la création d’autres processus qui seront les « processus enfants » du processus créateur et les « petits-enfants » du processus « Systemd ». Cette opération peut se répéter plusieurs fois : on obtient ainsi un arbre des processus.

Remarques :

- sous Windows, le tout premier processus est appelé « idle » et le PID est de 0
- le PPID d’un processus est le PID de son processus père

2.2. Les différents états d’un processus

Le noyau peut mettre un processus dans différents états :

- **Prêt** (*ready*) : est dans la liste des processus prêts à être exécutés par le processeur

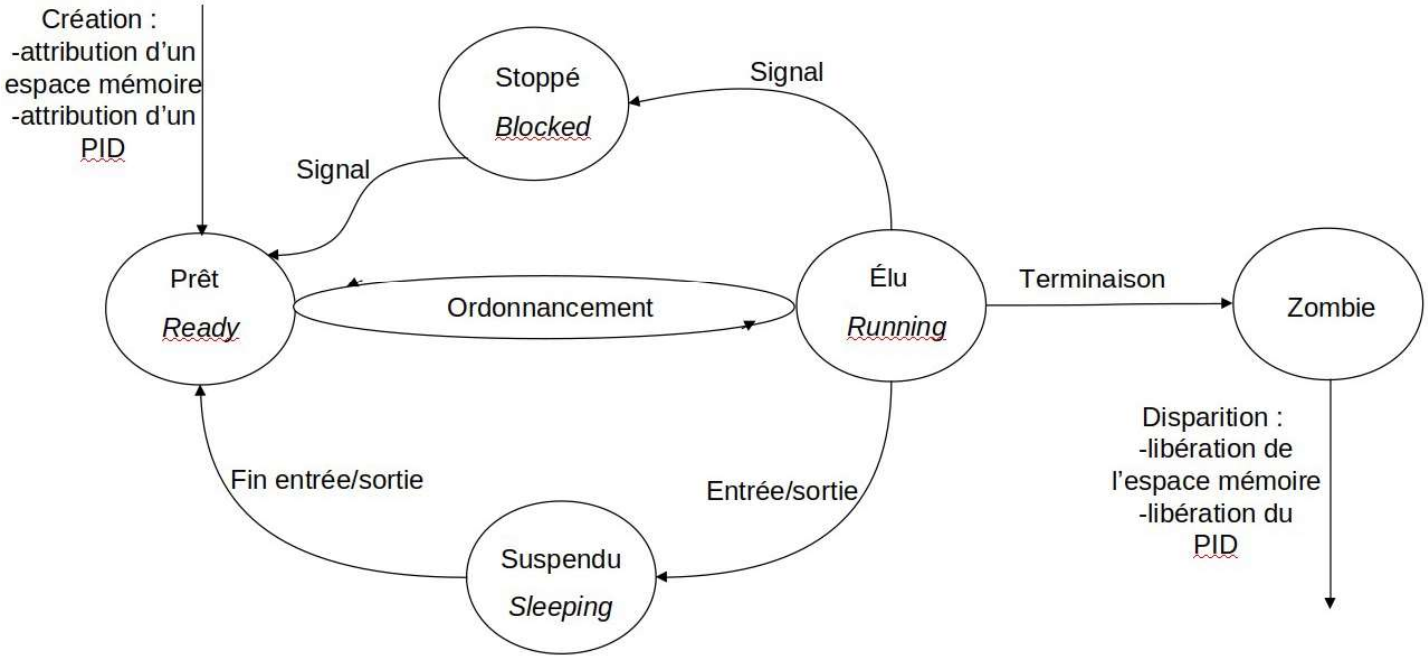
- **Élu** (*running*) : le code du processus est exécuté par l’un des cœurs du processeur

- **Suspendu** (*sleeping*) : le processus n’utilise plus de temps processeur. Il attend que le noyau le passe à l’état prêt sous condition d’un retour de donnée par le noyau (par exemple, une chaîne de caractères tapée au clavier)

- **Stoppé** (*bloked*) : le processus n’utilise plus de temps processeur. Il passe à cet état souvent à la demande de l’utilisateur, qui envoie une requête vers le noyau lui demandant de bloquer un processus (par exemple : appui des touches Ctrl+Z pendant l’exécution d’un processus en arrière plan).

- **Zombie** : le processus a terminé d’exécuter son code. Il n’aura donc plus de temps processeur. Il attend que son processus père récupère sa valeur de retour puis disparaît de la mémoire vive et libère son numéro de PID.

On peut résumer les transitions entre des différents états par le schéma suivant :



Exemple :

```
(base) marion@marion-Inspiron-7501:~$ find / -name "*.odt" 2>/dev/null  >/dev/null
^Z
[1]+  Arrêté                 find / -name "*.odt" 2> /dev/null > /dev/null
(base) marion@marion-Inspiron-7501:~$ ps aux | grep find
marion   25991  9.2  0.0  30000 11356 pts/0    T    15:18   0:00 find / -name *.odt
marion   25998  0.0  0.0  15668  1056 pts/0    R+   15:18   0:00 grep --color=auto find
(base) marion@marion-Inspiron-7501:~$ bg
[1]+ find / -name "*.odt" 2> /dev/null > /dev/null &
(base) marion@marion-Inspiron-7501:~$ ps aux | grep find
marion   25991 14.4  0.0  30000 11356 pts/0    R    15:18   0:02 find / -name *.odt
marion   26002  0.0  0.0  15668  1116 pts/0    R+   15:19   0:00 grep --color=auto find
```

2.3. Ordonnancement des processus par le noyau

Lorsque plusieurs processus doivent s'exécuter sur un processeur, on pourrait naïvement penser que le premier processus s'exécute complètement, puis le second, puis le 3e etc. Le risque est que si un processus est long, l'utilisateur va avoir l'impression que l'ordinateur reste « bloqué » un certain temps.

Pour éviter cela, le noyau joue le chef d'orchestre de l'ensemble des processus :

- il attribue à chaque processus un état qu'il actualise si besoin
- il donne, dans un certain ordre qu'il choisit, un temps d'exécution (appelé « quantum de temps ») et un processeur aux processus qui sont dans un état « prêt ».

Une fois qu'un processus « prêt » passe à l'état d'« élu », plusieurs cas de figure sont possibles :

- le processus finit l'exécution de son code dans le quantum de temps. Il passe ensuite à l'état « zombie » en attendant que son processus père viennent récupérer son résultat, puis disparaît.
- le processus manque de temps. Le noyau l'arrête pour laisser la place aux autres et le remet dans l'état « prêt » : il fait de nouveau la queue !
- au cours de son exécution, avant la fin du quantum de temps, il peut avoir besoin d'une donnée non disponible immédiatement (attente d'une valeur tapée au clavier ou arrivée d'un signal « stop » venant du noyau par exemple). Dans ce cas de figure, il passe soit à l'état suspendu, soit à l'état bloqué.

Voici quelques politiques d'ordonnancement :

- le Tourniquet (*Rand Robin*) : chaque processus dispose du même temps de processeur à tour de rôle
- FIFO (*First In, Firt Out*) : le premier processus lancé est le premier exécuté jusqu'à ce qu'il soit terminé.
- sporadique : les processus sont ordonnés en fonction de la deadline connue à l'avance pour chacun d'entre eux

Exemple : la commande « chrt » permet d'obtenir des information sur les politiques d'ordonnancement :

```
(base) marion@marion-Inspiron-7501:~$ chrt -m
propriété min/max SCHED_OTHER   : 0/0
propriété min/max SCHED_FIFO    : 1/99
propriété min/max SCHED_RR      : 1/99
propriété min/max SCHED_BATCH   : 0/0
propriété min/max SCHED_IDLE    : 0/0
propriété min/max SCHED_DEADLINE        : 0/0
```

On voit ici entre autre les politiques du Tourniquet (SCHED_RR), FIFO (SCHED_FIFO) et Sporadique (SCHED_DEADLINE) ...

3. Situation d’interblocage

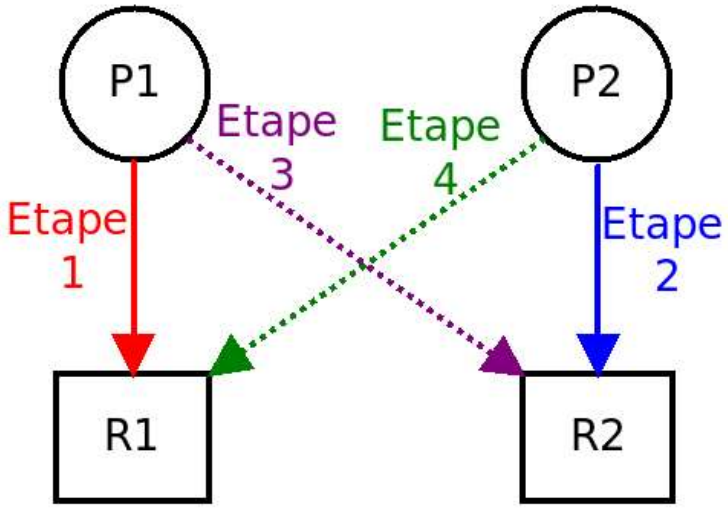
Soient 2 processus P1 et P2 et 2 ressources R1 (par exemple une zone mémoire) et R2 (par exemple un fichier).

Initialement, les 2 ressources sont "libres" (utilisées par aucun processus).

1. Le processus P1 demande au noyau à avoir un accès exclusif à la ressource R1. Il obtient satisfaction puisque R1 est libre. A partir de ce moment là, le noyau empêchera les autres processus d’utiliser R1 tant que le processus P1 n’a pas libéré la ressource.
2. De même, le processus P2 demande au noyau à avoir un accès exclusif à la ressource R2. Il obtient satisfaction puisque R2 est libre. A partir de ce moment là, le noyau empêchera les autres processus d’utiliser R2 tant que le processus P2 n’a pas libéré la ressource.
3. P1 demande ensuite à utiliser de manière exclusive la ressource R2 : le noyau l’en empêche et le met en attente (état « suspendu »). Le noyau sortira P1 de l’état suspendu lorsque la ressource R2 sera libérée.
4. De même, P2 demande à utiliser de manière exclusive la ressource R1 : le noyau l’en empêche et le met en attente (état « suspendu »). Le noyau sortira P2 de l’état suspendu lorsque la ressource R1 sera libérée.

Les deux processus étant suspendus et attendant respectivement que l’autre se termine, ils resteront dans cet état de manière infinie.

Illustration de la situation :



Cette situation est qualifiée d'interblocage (*deadlock* en anglais).

Remarque : nous avons vu l’exemple d’un interblocage avec 2 processus et 2 ressources, mais dans nos machines où des dizaines de processus ont accès à des dizaines de ressources en même temps, l’interblocage peut concerner un grand nombre de processus et un grand nombre de ressources.

Pour modéliser le partage de plusieurs ressources entre plusieurs processus, il existe un modèle mathématique appelé réseau de Petri (*Petri net* en anglais). Ce modèle permet notamment de mettre en évidence les situations d’interblocage et de les analyser afin de trouver des solutions.