CH5 Gestion des processus

1 Notion de Processus

ACT 1 Jouons avec les processus

A retenir :
Un **processus** est l'ensemble des informations correspondant à l'exécution d'une suite d'instructions par un processeur :

- la suite d'instructions elle-même, c'est à dire le *programme* qui s'exécute
- les données du programme
- les ressources (entrés-sorties, mémoire, contenu des registres processeurs, etc.) que le programme utilise

A chaque processus est associé un identifiant *unique*, son **PID** (***P**rocess **ID**entifier*) affecté par le noyau.
Le PID est un entier de 0 à 2^{15} , ou de 0 à 2^{32} sur les systèmes 64 bits.

Les processus peuvent être créés à tout moment :

- par le système lui-même (***processus système***). Le propriétaire de ces processus est le super-utilisateur *root*.
- par l'utilisateur de la machine (***processus utilisateur***)

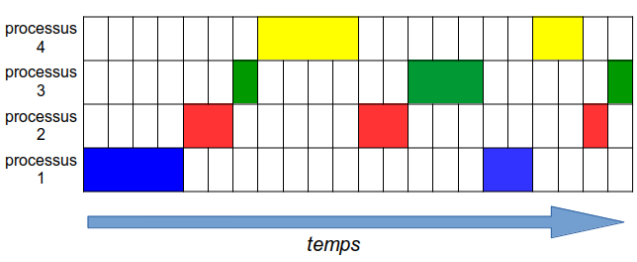
Exemple :
Le premier processus lancé au démarrage du système, et qui reste actif jusqu'à son extinction, appelé init*, reçoit le numéro 1. La tache de ce processus est de lancer les autres processus.*

Lorsqu'il a été lancé par un autre processus, appelé ***processus-père***, le ***processus-fils*** aura donc aussi un numéro de ***PPID*** (*Parent Process IDentifier*), c'est à dire le PID du processus père.
Seul le processus *init* n'a pas de processus-père, il est à la racine de la hiérarchie de tous les autres processus.

2 Ordonnancement des tâches

A retenir :
Sur un processeur à un seul noyau ne peut s'exécuter qu'**un seul processus à la fois**.
Le noyau d'un système d'exploitation gère alors l'**ordonnancement** des processus, c'est à dire le temps-processeur et l'accès aux ressources, qu'il va leur *allouer à tour de rôle*.

L'ensemble des processus peut donc être vu comme des « sous-machines » indépendantes et ne faisant fonctionner qu'un seul programme (*mono-tache*), que le noyau feraient fonctionner à tour de rôle en passant très rapidement de l'une à l'autre pour exécuter tel ou tel travail, ce qui, pour l'utilisateur, donne l'illusion d'un système **multi-tâches**, c'est à dire exécutant plusieurs processus simultanément



Remarque : *Les processeurs à plusieurs noyaux permettent réellement d'exécuter plusieurs tâches en parallèle*).

Le ***contexte*** d'un processus, c'est à dire les zones mémoire contenant son code et ses données, les valeurs sauvegardées des registres processeurs, de la pile exécution, etc... permet de reprendre où elle en était l'exécution d'un processus qui a été interrompue par celle d'un autre.

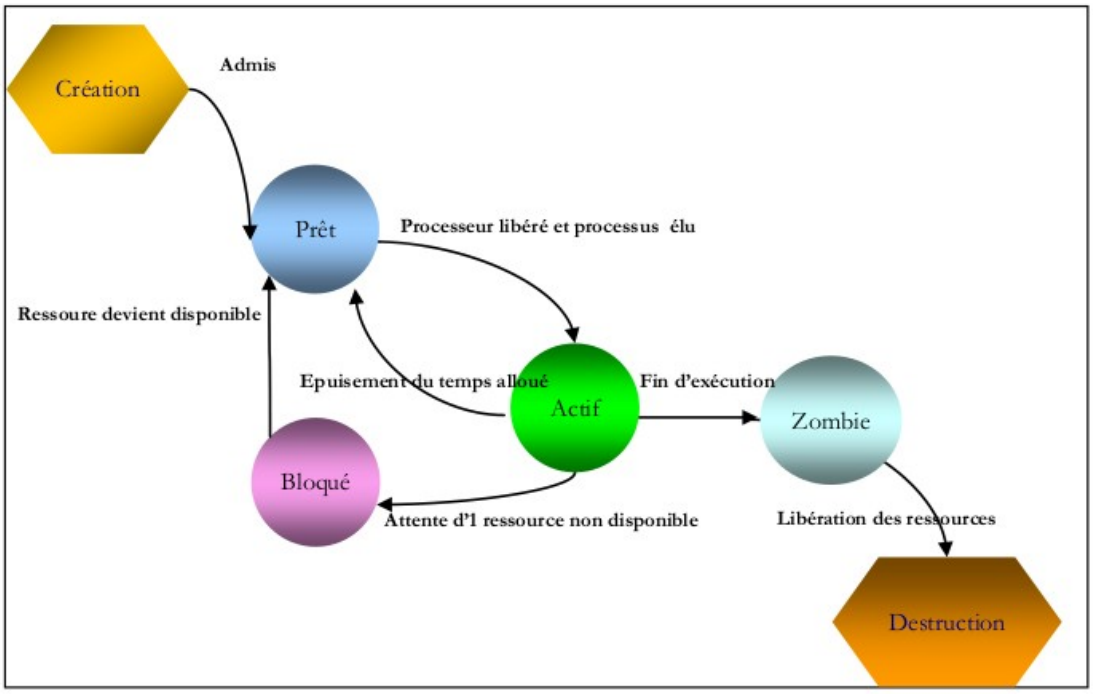
Différents algorithmes sont utilisés pour gérer l'ordonnancement des processus, et c'est toujours un domaine très actif de recherche :
Par exemple :

- **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.
- **FIFO**: First In First Out
- **SJF** Shortest job first (SJF, ou SJN -Shortest Job Next-)

A retenir :
Un processus passe donc par plusieurs ***états*** pendant la durée où il tourne sur le système :

- ***éligible*** (prêt) : le processus est prêt à être exécuté
- ***élu*** (actif) : le processus s'exécute sur le processeur
- ***bloqué*** : en attente pendant l'exécution d'un autre processus ou d'une ressource non disponible

Un processus donné peut utiliser plus ou moins de temps-processeur ou de ressources par rapport aux autres selon la **priorité** qui lui est affecté.



Exemple :
Sous Linux, la priorité est une valeur (appelée `nice` *) comprise entre 19 (la plus petite priorité) et -20 (la plus grande priorité).*
Par défaut, un processus est lancé avec une valeur `nice` *de 0 : un utilisateur normal peut augmenter la valeur* `nice` *d'un processus, mais seul le super-utilisateur* `root` *peut la diminuer*

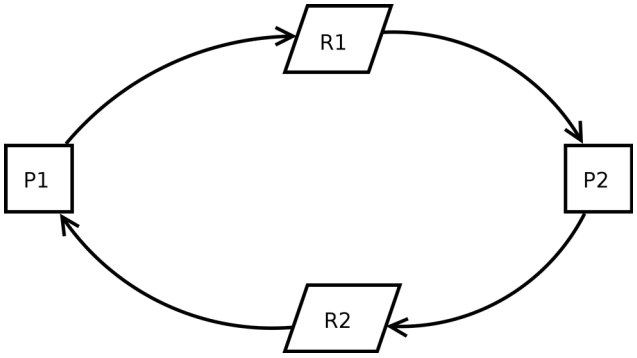
3 Un cas critique : l'interblocage

ACT2 Débloquez les tous !

Les processus sont notés Pi et les ressources nécessaires Ri.

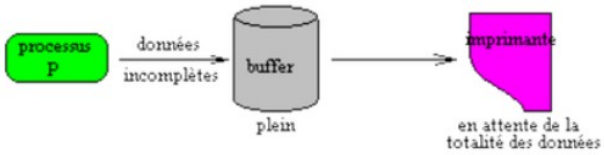
- P1 acquiert R1.
- P2 acquiert R2.
- P1 attend pour acquérir R2 (qui est détenu par P2).
- P2 attend pour acquérir R1 (qui est détenu par P1).

Dans cette situation, les deux processus sont définitivement bloqués.



A retenir :
Un **interblocage** (ou étreinte fatale, *deadlock* en anglais pour impasse) est un phénomène qui peut se produire en programmation concurrente lorsque des processus s'attendent mutuellement.

Exemple :
Le spooling massif : P veut imprimer de grandes quantités données, il les envoie donc à l'imprimante. L'imprimante utilise un spool (buffer qui créé une file de tâches), c'est à dire qu'elle attend que toutes les données à imprimer soient transférées sur le buffer avant de lancer le travail d'impression.



Le problème survient si le buffer est plein avant que P n'ai fini d'envoyer toutes ses données : P attend que le buffer se vide, et l'imprimante attend que P ait terminer le transfert...

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 conditions nécessaires à la possibilité de l'interblocage

Coffman a prouvé en 1971 qu'il y a quatre conditions nécessaires pour qu'un blocage puisse avoir lieu. Voir l'article Wikipédia Conditions de Coffman pour plus de détails.