Terminale	Gestion des processus	NSI

Objectifs de la séance :

- *Connaître la notion de processus - thread*
- *Comprendre l'ordonnancement des processus – threads*
- *Comprendre l'interblocage (deadlock).*

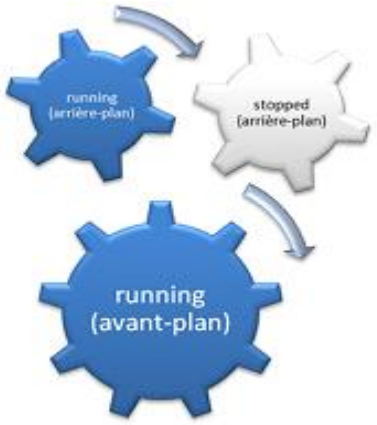
Matériel et Logiciel nécessaire :

- *PC avec accès à un système Unix pour démonstration prof.*
- *Un éditeur python pour démonstration prof.*

Programme et documents:

- *interblocageV2.py*
- *Guide de gestion des processus.pdf*

Durée estimée : 2h



Un processus (en anglais, process) est un programme en cours d'exécution par un ordinateur. Un ordinateur équipé d'un système d'exploitation multitâche est capable d'exécuter plusieurs processus de façon apparemment simultanée. S'il y a plusieurs processeurs, l'exécution des processus est distribuée de façon équitable sur ces processeurs.

Sources : Renaud Lachaize, Benjamin Monmège, Didier Müller, David Roche, wikipedia

1. Définitions

1.1. Programme

Un programme est une suite figée d'instructions, un ensemble statique.

1.2. Processus

Un processus est l'instance d'exécution d'un programme dans un certain contexte pour un ensemble particulier de données.

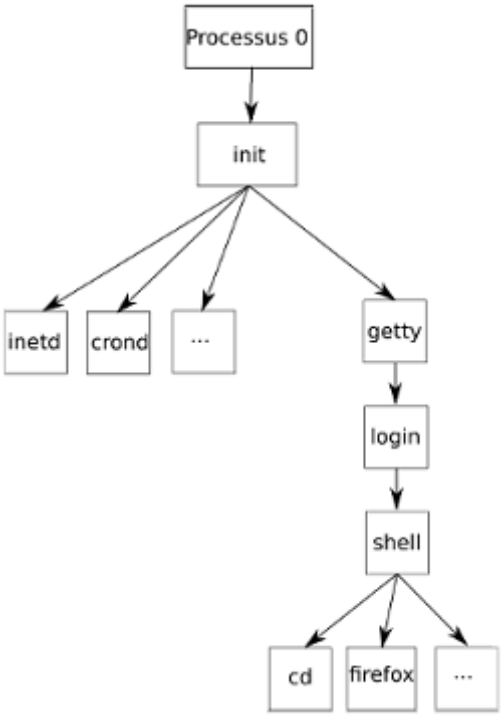
On peut caractériser un processus par les tables qui le décrivent dans l'espace mémoire réservé au système d'exploitation. Ces tables définissent les contextes :

- matériel (partie dynamique du contexte : contenu des registres...)
- logiciel (partie statique du contexte : droits d'accès aux ressources) dans lesquels travaille ce processus.

2. 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.

Sous un système d'exploitation comme GNU/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"...



2.1. PID et PPID

Chaque processus possède un identifiant appelé PID (Process Identification), ce PID est un nombre. Le premier processus créé au démarrage du système à pour PID 0, le second 1, le troisième 2... Le système d'exploitation utilise un compteur qui est incrémenté de 1 à chaque création de processus, le système utilise ce compteur pour attribuer les PID aux processus.

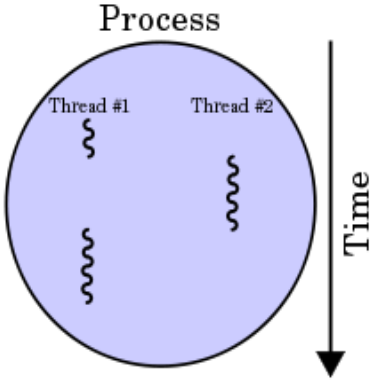
Chaque processus possède aussi un PPID (Parent Process Identification). Ce PPID permet de connaître le processus parent d'un processus (par exemple le processus "init" vu ci-dessus à un PID de 1 et un PPID de 0). À noter que le processus 0 ne possède pas de PPID (c'est le seul dans cette situation).

2.2. Thread

Au démarrage, un processus correspond à un espace de mémoire virtuelle et à un flot d'exécution qui représente un instantané du déroulement du programme.

Si on le souhaite, on peut au cours de l'exécution d'un processus créer des activités supplémentaires qui font partie du même espace de mémoire virtuelle. C'est ce qu'on appelle des threads (en français, fils d'exécution ou flots d'exécution).

Les différents flots d'exécution vont partager toutes les ressources du processus et ils auront accès à l'ensemble de la même mémoire virtuelle et des ressources qui auront été mises à disposition du processus, notamment aux fichiers.

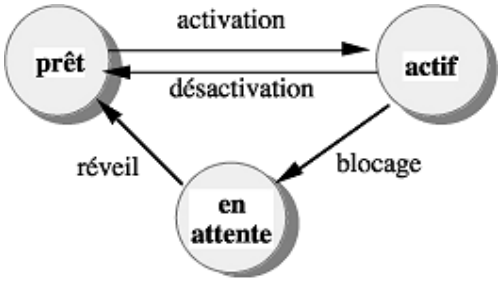


3. États d'un processus

Dans un système multitâche, plusieurs processus peuvent se trouver simultanément en cours d'exécution : ils se partagent l'accès aux ressources de la machine.

Un processus peut prendre 3 états :

- État actif ou élu (running) : le processus utilise le processeur.
- État prêt ou éligible (ready) : le processus pourrait utiliser le processeur s'il était libre (et si c'était son tour).
- État en attente ou bloqué : le processus attend une ressource (ex : fin d'une entrée-sortie).



Tout processus qui se bloque en attente d'un événement, passe dans l'état bloqué tant que l'événement attendu n'est pas arrivé. Lors de l'occurrence de cet événement, le processus passe dans l'état prêt. Il sera alors susceptible de se voir attribuer le processeur pour continuer ses activités.

Le changement d'état d'un processus peut être provoqué par :

- un autre processus (qui lui a envoyé un signal, par exemple)
- le processus lui-même (appel à une fonction d'entrée-sortie bloquante,
- une interruption (fin de quantum, terminaison d'entrée-sortie, ...)

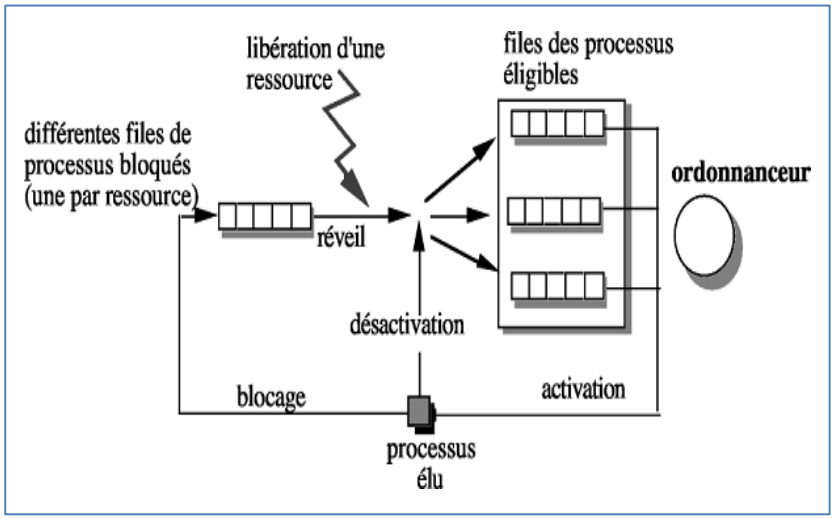
La sortie de l'état actif pour passer à l'état prêt se produit dans le cas des ordonnancements préemptifs lorsqu'un processus plus prioritaire que le processus actif courant devient prêt.

4. Ordonnancement

Le système d'exploitation gère les transitions entre les états. Pour ce faire il maintient une liste de processus éligibles et une liste de processus en attente. Il doit également avoir une politique d'activation des processus éligibles (ou politique d'allocation du processeur aux processus).

Les processus sont classés dans la file des processus éligibles par l'ordonnanceur (en anglais scheduler). C'est le distributeur (en anglais dispatcher) qui se chargera de leur activation au moment voulu.

Afin d'éviter qu'un processus accapare l'unité centrale, le système d'exploitation déclenche un temporisateur à chaque fois qu'il alloue l'unité centrale à un processus. Quand ce temporisateur expire, si le processus occupe toujours l'unité centrale (il peut avoir été interrompu par un événement extérieur ou s'être mis en attente), le système d'exploitation va le faire passer dans l'état prêt et activer un autre processus de la liste des processus éligibles. La désactivation d'un processus peut se faire sur expiration du temporisateur ou sur réquisition (en anglais preemption) du processeur central par un autre processus.



D'autres événements sont signalés au système par des interruptions c'est le cas des fins d'entrées-sorties. Le système est ainsi prévenu que le ou les processus qui étaient en attente sur cette entrée-sortie peuvent éventuellement changer d'état (être réveillés).

Le contexte doit être sauvegardé quand le processus est désactivé ou bloqué et doit être restauré lorsqu'il est réactivé.

Par contexte, on entend toutes les informations relatives à l'exécution d'un processus, telles que :

- l'état des registres (compteur ordinal, registre d'état, registres généraux),
- les registres décrivant l'espace virtuel du processus et son espace physique d'implantation,
- les pointeurs de piles,
- les ressources accédées (ex : droits d'accès, fichiers ouverts),
- autres informations (ex : valeur d'horloge).

4.1. Interblocage

Un interblocage (deadlock en anglais) est un phénomène qui peut survenir en programmation concurrente. L'interblocage se produit lorsque des processus concurrents s'attendent mutuellement. Un processus peut aussi s'attendre lui-même. Les processus bloqués dans cet état le sont définitivement, il s'agit donc d'une situation catastrophique.

Un exemple concret d'interblocage peut se produire lorsque deux processus P1 et P2 essayent d'acquérir deux ressources R1 et R2 dans un ordre différent :

1. P1 acquiert M1.
2. P2 acquiert M2.
3. P1 attend pour acquérir M2 (qui est détenu par P2).
4. P2 attend pour acquérir M1 (qui est détenu par P1).

Dans cette situation, les deux processus sont définitivement bloqués.

4.2. Famine

La famine est un problème qui se produit lorsqu'un algorithme ne garantit pas que les processus qui souhaitent entrer dans une section critique y entrent dans le même ordre que les demandes (c'est-à-dire un comportement FIFO).

Une section critique est une portion de code dans laquelle il doit être garanti qu'il n'y aura jamais plus d'un thread simultanément. Il est nécessaire d'utiliser des sections critiques lorsqu'il y a accès à des ressources partagées par plusieurs threads.

Dans le cas contraire, le thread attendra indéfiniment une ressource qu'il ne pourra jamais obtenir.