LES PROCESSUS

1. RAPPELS DE PREMIERE

a. Rôle du système d'exploitation

Le programme qui tourne en permanence sur l'ordinateur est le système d'exploitation. Les rôles principaux du système d'exploitation sont les suivants :

- Fournir une «interface» entre l'ordinateur et l'utilisateur pour permettre à ce dernier de donner des ordres à la machine ou pour lui signaler les erreurs d'exécution ;
- Gérer les «ressources» de l'ordinateur, à savoir ses mémoires, son microprocesseur et ses périphériques ;
- Être indépendant du matériel, masquer les particularités de la machine en substituant aux ressources physiques des abstractions (par exemple, la notion de fichier est une notion abstraite, indépendante de la nature du support sur lequel les données de ce fichier sont réellement stockées) ;
- Contrôler les usagers en leur donnant des droits différents selon leur statut (associés par exemple à différents mots de passe).

b. Commandes Linux de base

`sudo`	*subsitute utilisator do*	Permet d'effectuer des commandes non autorisées. Le mot de passe est demandé.
`pwd`	*print working directory*	Affiche le répertoire de travail
`cd` *arg/*	*change directory*	Change de répertoire de travail ; *arg/* est ce nouveau répertoire. `cd` sans argument positionne dans le répertoire *home*.
`ls` *arg*	*list*	Liste le contenu *arg* si celui-ci est un répertoire. Sans argument, `ls` liste le contenu du répertoire de travail.
`ll` *arg* ou `ls -l` *arg*	*long list*	Affiche des informations détaillées sur le fichier *arg*
`ls -a` *arg*	*list all*	Affiche tous les fichiers de *arg*, même ceux commençant par `..`, si celui-ci est un répertoire. Les options de `ls` peuvent être combinées : `ls -al`.
`chmod` *autorisations arg*	*change mode*	Change les autorisations (droits) du fichier/dossier, cf. cours de 1ère.
`mkdir` *arg*	*make directory*	Crée le répertoire *arg*.
`rmdir` *arg*	*remove directory*	Supprime le répertoire s'il est vide.
`rm` *arg*	*remove*	Supprime le fichier *arg*.
`rm -r` *arg*	*remove recursively*	Supprime *arg* et, éventuellement, tous les fichiers qu'il contient.
`mv` *arg1 arg2*	*move*	Renomme *arg1* en *arg2* (si *arg2* n'est pas un nom de répertoire), ou déplace *arg1* dans le répertoire *arg2* (si *arg2* est un nom de répertoire).
`touch` *arg*	*touch*	Crée un fichier vide de nom *arg*, s'il n'existe pas, sinon met à jour, avec la date courante, sa date de dernière modification s'il existe déjà.
`kill` *arg*	*kill*	Arrête le processus numéro *arg*

2. Définition d'un processus

Un processus est un programme en cours d'exécution, qui est constitué :

- D'un ensemble d'instructions à exécuter, pouvant être dans la mémoire morte, mais le plus souvent chargé depuis la mémoire de masse vers la mémoire vive ;
- D'un espace d'adressage en mémoire vive pour stocker la pile, les données de travail, etc. ;
- de ressources permettant des entrées-sorties de données, comme des ports réseau.

Un même logiciel peut être associé à plusieurs processus, c'est par exemple le cas de Firefox qui lance un nombre de processus dépendant du nombre d'onglets actifs (mais qui n'est pas égal à ce nombre, ni au nombre de fenêtres)

Les processus peuvent être créés :

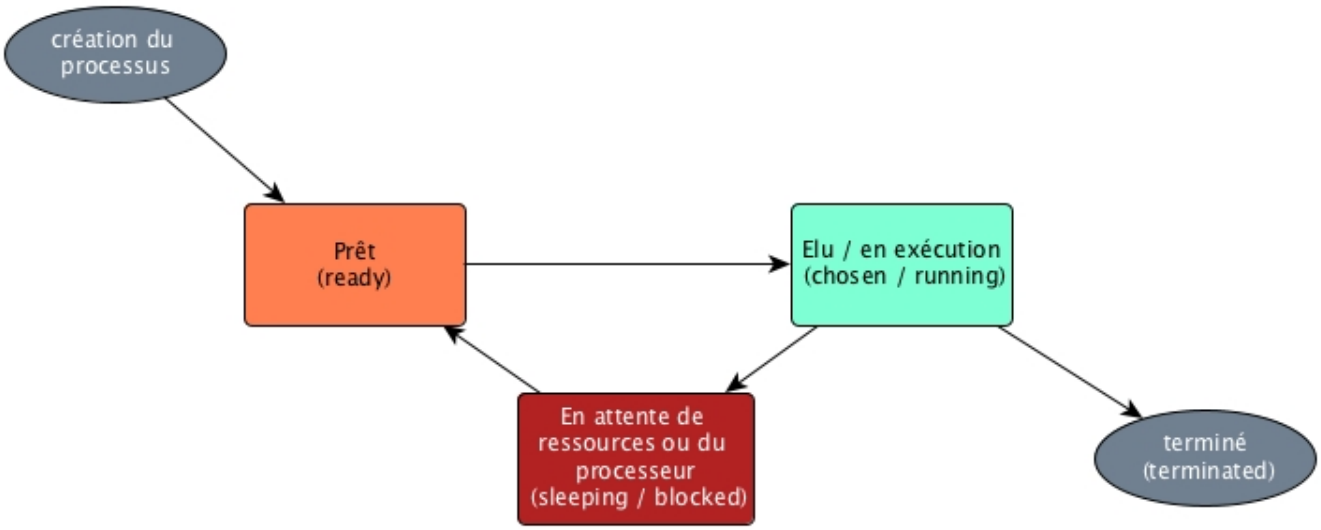
- automatiquement au démarrage, comme le système. Les processus système qui tournent en permanence sont des démons (un daimôn, δαίμων en grec, peut être assimilé au destin, mais aussi à un esprit permettant la communication entre mortels et immortels). Le tout premier processus lancé au boot est le processus chargé d'aller chercher le système sur le disque dur et de le lancer. Ce processus est en mémoire morte, le système étant ensuite chargé en mémoire vive.
- par l'utilisateur, en lançant un programme.
- par d'autres processus. Par exemple, le processus système en engendre plusieurs autres (comme l'interface graphique). On dit que le processus fils est engendré par le processus parent.

Le PID est un entier qui identifie le processus. Cet entier est créé en même temps que le processus est lancé.

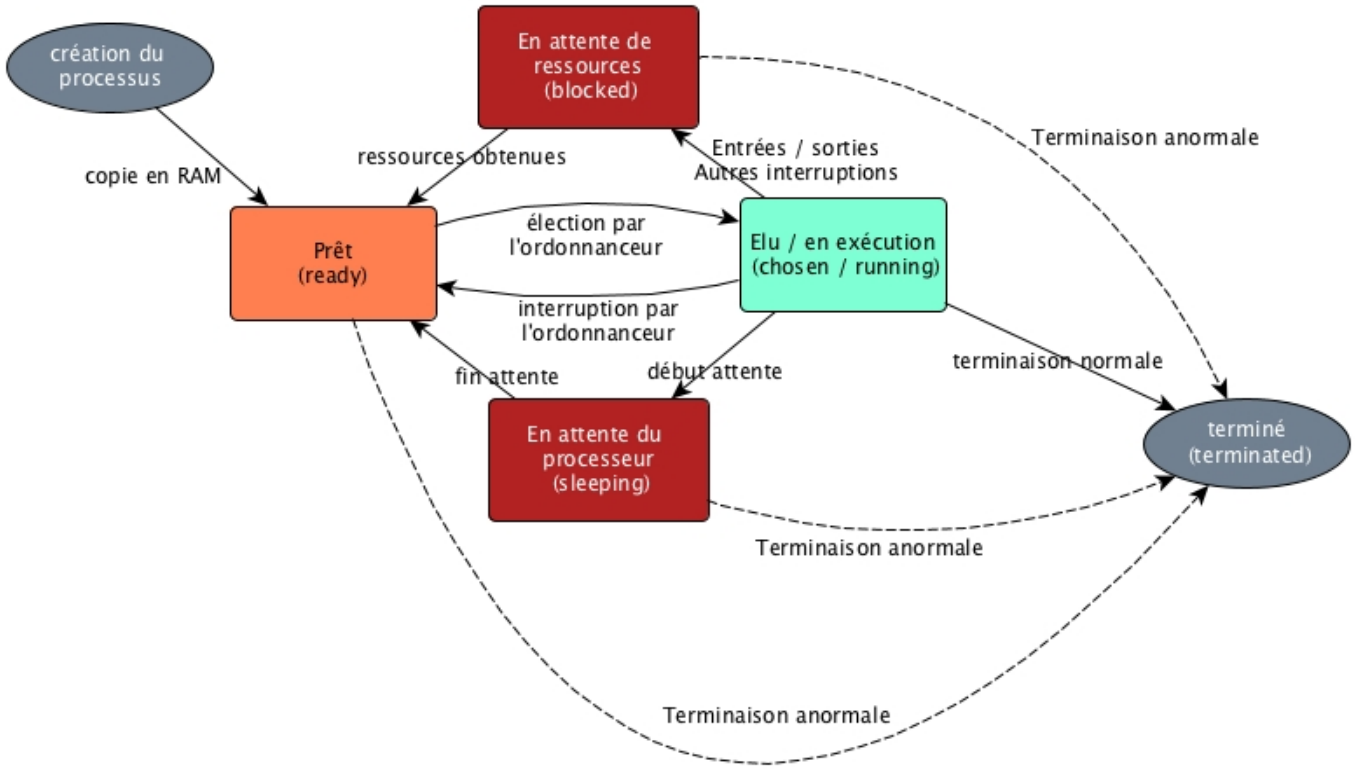
3. Etats des processus

Du temps des dinosaures, les processeurs étaient monotâche : ils ne pouvaient gérer qu'un seul processus à la fois. Dans les années 1960, les processeurs multitâches se sont généralisés. L'utilisateur à l'impression que plusieurs processus s'exécutent simultanément. Ce n'est pas le cas en général, les processus sont exécutés par petits segments en rotation très rapide dans le processeur. Certains logiciels, sur des processeurs multicœurs, sont réellement exécutés en parallèle.
Les processus passent donc d'un état à l'autre :

En version simple (suffisant pour la terminale):



En version plus complète :



- La création du processus se fait avec fork() (sous Linux) ;
- Le processus est prêt, il attend en RAM que le processeur soit libre ;
- Quand le processeur est libre, le processus est élu (running) ;
- Dans l'état élu, le processus peut passer la main et être « mis en attente » en attendant à nouveau d'être prêt. Ses ressources sont sauvées momentanément afin de pouvoir reprendre exactement au même point ;
- Le processus peut aussi attendre des ressources, il est alors bloqué ;
- Le processus peut être mis en sommeil (comme une veille de surveillance de réception de mail, un anti-virus etc.) ;
- A la fin du processus, il est « mort ».
- D'autres états existent, mais ils sont plus rares. Par exemple, si un processus père est tué avant le processus fils, alors le fils est Zombie.

4. ORDONNANCEMENT

Dans les systèmes d'exploitation, l'ordonnanceur désigne le composant du noyau du système d'exploitation choisissant l'ordre d'exécution des processus sur les processeurs d'un ordinateur. En anglais, l'ordonnanceur est appelé scheduler.
Suivant le type de machine, l'ordonnanceur utilise des stratégies différentes :

- Sur une machine en temps réel (aéronautique, industrie, nucléaire, salles de marchés, etc.) il est indispensable que certaines tâches soient réalisées en un temps donné.
- Sur les ordinateurs « classiques », l'ordonnancement se fait en temps partagé.
- Actuellement les deux types de machines utilisent un système de priorité, qui, comme son nom l'indique, favorise certains processus.

D'autres algorithmes d'ordonnancement existent :

- Le tourniquet : chaque processus dispose du même temps de processeur à tour de rôle
- FIFO : une simple file, le premier processus lancé est le premier exécuté jusqu'à ce qu'il soit terminé.
- Shortest Job First : le processus le plus court est exécuté en premier.

5. Commandes sur les processus

Les commandes Linux vues en TP permettent de connaître les processus tournant sur la machine :

- `ps -ef` (avec éventuellement les options`-ef` ou `-elf`), vue statique
- `pstree`, vue arborescente
- `kill PID` permet de tuer un processus (version extrême lorsque tout résiste `kill -9 PID`)
- `top`, vue dynamique. `V` donne une vue arborescente et `k` permet de tuer un processus avec son `PID`

6. Interblocage

Le fonctionnement multitâche impose de nombreuses contraintes en plus de l'ordonnancement. L'une d'entre elles concerne l'allocation des ressources. Considérons deux processus utilisant le clavier et l'écran. Le premier processus a affiché une fenêtre de saisie, et attend que l'utilisateur remplisse un champ pour libérer l'écran. Le deuxième processus bloque le clavier car l'utilisateur s'en sert, et attend que le texte saisi s'affiche à l'écran pour libérer le clavier : les deux processus sont en situation d'interblocage (deadlock).

Edward Coffman a théorisé le principe de l'interblocage avec 4 conditions

- exclusion mutuelle : au moins une ressource est conservée en accès exclusif par un processus ;
- rétention de ressources : un processus détient au moins une ressource, et a besoin d'une autre ressource, détenue par un autre processus, pour continuer ;
- non préemption : le processus qui utilise une ressource est le seul à pouvoir la libérer ;
- attente circulaire : chaque processus attend une ressource, détenue par un autre processus, pour pouvoir continuer.

Exemple :
Il existe en France des carrefours en croix avec quatre stops (ou sans aucun), comme ci-contre dans le dessin de Félé pour Le Courrier de l'Eure. Il y a bien une situation d'interblocage dans ce dessin. Identifier les conditions de Coffman.

Exercice

Ex Un : Le Dîner des philosophes

Source Wikipedia

La situation est la suivante :

- cinq philosophes (initialement mais il peut y en avoir beaucoup plus) se trouvent autour d'une table ;
- chacun des philosophes a devant lui un plat de spaghettis ;
- à gauche de chaque plat de spaghettis se trouve une fourchette.

Un philosophe n'a que trois états possibles :

- penser pendant un temps indéterminé ;
- être affamé (pendant un temps déterminé et fini sinon il y a famine) ;
- manger pendant un temps déterminé et fini.

Des contraintes extérieures s'imposent à cette situation :

- quand un philosophe a faim, il va se mettre dans l'état « affamé » et attendre que les fourchettes soient libres ;

- pour manger, un philosophe a besoin de deux fourchettes : celle qui se trouve à gauche de sa propre assiette, et celle qui se trouve à droite (c'est-à-dire les deux fourchettes qui entourent sa propre assiette) ;
- si un philosophe n'arrive pas à s'emparer d'une fourchette, il reste affamé pendant un temps déterminé, en attendant de renouveler sa tentative.

Le problème consiste à trouver un ordonnancement des philosophes tel qu'ils puissent tous manger, chacun à leur tour.

1. Décrire une situation d'interblocage, en détaillant les conditions de Coffman.
2. Que faire si un philosophe meurt de faim alors qu'il a une fourchette en main (i.e. un processus se crashe alors qu'il utilise une ressource) ? La question est assez rhétorique, elle est là juste pour que vous réalisiez le problème dans ce cas.
3. On propose une solution, basée sur la règle suivante : « un philosophe ayant une seule fourchette la repose après 10 minutes, et attend 10 minutes avant de la reprendre ». Cette règle permet-elle d'éviter l'interblocage ? Justifier.
4. Une autre solution est basée sur la hiérarchisation des ressources. Les fourchettes sont numérotées de 1 à 5, pas forcément dans l'ordre de leur emplacement sur la table. Les philosophes connaissent les numéros des fourchettes dont ils ont besoin pour manger. Un philosophe prendra d'abord la fourchette de numéro le plus bas, avant de prendre celle de numéro le plus haut. Cette méthode permet-elle d'éviter l'interblocage ? Justifier
5. On reprend la méthode précédente. On rajoute du parmesan à table, de numéro 0. Les philosophes ont maintenant besoin de 3 ressources : les deux fourchettes et le parmesan. Supposons que le parmesan soit libre, et qu'un philosophe ait les fourchettes 1 et 4. Que doit-il faire pour manger ? Conclure sur un des défauts de cette méthode.
6. Une méthode générale est proposée, pour un nombre quelconque de philosophes nécessitant un nombre quelconque de ressources.
   - Les fourchettes sont soit propres, soit sales.
   - Pour chaque paire de philosophes pouvant accéder à la même fourchette, on commence par la donner à celui qui est en premier dans l'ordre alphabétique.
   - Un philosophe qui veut manger doit obtenir les fourchettes de ses deux voisins. Pour chaque fourchette qui lui manque, il émet poliment une requête.
   - Lorsqu'un philosophe qui a une fourchette en main entend une requête pour celle-ci :
     - soit la fourchette est propre et il la garde ;
     - soit la fourchette est sale, alors il la nettoie et il la donne.
   - Après qu'un philosophe a fini de manger, ses deux fourchettes sont devenues sales. Si un autre philosophe avait émis une requête pour obtenir une de ses fourchettes, il la nettoie et la donne.

   Montrer qu'il reste une situation d'interblocage possible, au démarrage. Préciser la condition à rajouter pour que cette situation ne puisse pas parvenir. Expliquer qu'alors ces règles permettent d'éviter l'interblocage (on pourra se contenter de deux philosophes). Une rédaction correcte est exigée.
7. Trouver une solution simple pour éviter l'interblocage, dans le cas où le nombre de philosophes est pair (on les numérotera et raisonnera sur la parité).

Exercices 176, 178 et éventuellement 177 p 380 du livre.