TP n°5

Architecture des systèmes informatiques
INFO3 - S5

Les processus

L'objectif de ce TP est de découvrir et d'approfondir la notion de processus dans un SE, et plus particulièrement sous UNIX.

1 Les processus sous UNIX

Gestion des processus Les systèmes d'exploitation modernes permettent de donner l'impression qu'un très grand nombre d'applications peuvent s'exécuter simultanément sur un système physique. Cette impression est donnée en exécutant chaque application sur un faible quantum de temps, de manière invisible pour l'application.

Pour gérer les applications (de son lancement à sa mort), UNIX, et plus particulièrement Linux, utilise les notions de *session*, *groupe de processus*, *processus* et *threads*. Une session contient des groupes de processus, un groupe de processus contient des processus et un processus contient des threads. Tous ces objets possèdent des nombres qui permettent de les identifier ; nous avons par conséquent des ID de thread (tid), des ID de processus (pid), des ID de groupe de processus (pgid) et des ID de session (sid).

Les signaux Pour gérer les processus et permettre une communication basique entre le SE et les processus, UNIX utilise des *signaux*. Les signaux informent un processus d'un événement de façon asynchrone. La liste des signaux, ainsi que les différentes fonctions de manipulation des signaux sont définies dans le fichier `/usr/include/bits/signum.h` sous Linux. Vous pouvez également accéder à une description de ces signaux dans la page `signal` de la section 7 du *man*.

Le comportement des signaux classiques d'UNIX est malheureusement différent d'une version à l'autre. On emploie donc de préférence les mécanismes définis par la norme `POSIX`, qui offrent de plus la possibilité de masquer des signaux.

Pour chaque processus, le système conserve une table avec une entrée par signal. Dans cette table, il y a un indicateur (sur un bit) qui indique la présence d'un signal pendant, c'est-à-dire d'un signal qui a été envoyé à un processus mais non encore pris en compte. Lors de la délivrance du signal au processus, lorsque celui-ci passe de l'état *actif noyau* à l'état *actif utilisateur*, une fonction particulière est exécutée appelée le gestionnaire du signal (*handler*). Le *handler* par défaut dépend du signal, il est désigné aux fonctions par le symbole générique `SIG_DFL`. Mais il est impossible d'associer un autre gestionnaire que `SIG_DFL` aux signaux `SIGKILL`, `SIGSTOP`, et `SIGCONT`.

Question 1. Donnez la liste des principaux signaux existants sur votre système, ainsi que leur signification.

Question 2. Établir la liste des évènements responsables du lancement de chaque signal.

Subdivision des processus Une session peut avoir un terminal de contrôle (tty) et au plus un de ses groupes de processus peut être *au premier plan*. Si un signal est émis sur le terminal, ce signal est

propagé à tous les membres du groupe de processus se trouvant premier plan de la session associé au terminal.

QUESTION 1. Préciser ce qu'est un processus en répondant à la question suivante : quelle est la différence entre un processus et un programme ?

QUESTION 2. Que partagent plusieurs threads d'un même processus ?

2 Commandes utilisateurs

2.1 Lister les processus

A l'aide de la commande `ps`, afficher la liste de tous les processus s'exécutant sur votre machine comprenant les informations suivantes (pas nécessairement en une seule commande) : `USER`, `PID`, `PGID`, `SID`, `TTY`, `%CPU`, `%MEM`, `C`, `STAT`, `START`, `TIME`, `STIME`, `COMMAND`.

QUESTION 1. Donner une description de chaque colonne retournée.

QUESTION 2. Quel est le processus ayant le plus utilisé le processeur sur votre machine ?

QUESTION 3. Quel a été le premier processus lancé après le démarrage du système ?

QUESTION 4. À quelle heure votre machine a-t-elle démarré ?

QUESTION 5. Pouvez-vous établir le nombre approximatif de processus créés depuis le démarrage de votre machine ?

QUESTION 6. Quel est le processus ayant le plus utilisé le processeur sur votre machine ?

Processus parents Sous UNIX, chaque processus (excepté le premier) est créé par un autre processus, son processus père. Tous les processus sont ainsi identifiés par leur `PID`, mais aussi par le `PPID` du processus qui l'a créé (Parent `PID`).

QUESTION 1. Trouver une option de la commande `ps` permettant d'afficher le `PPID` d'un processus.

QUESTION 2. Trouver une option de la commande `ps` permettant d'afficher le `PGID` et le `SID` d'un processus.

QUESTION 3. Donner la liste ordonnée de tous les processus ancêtres de la commande `ps` en cours d'exécution.

QUESTION 4. Reprendre la question précédente avec la commande `pstree`.

QUESTION 5. Essayez la commande `top`, qui affiche les mêmes informations que `ps` mais en rafraichissant périodiquement l'affichage.

- La touche `?` permet d'afficher un résumé de l'aide de `top`. Afficher dans `top` la liste des processus triés par occupation mémoire (*resident memory*) décroissante.
- Quel est le plus *gros* processus sur votre machine ? À quoi correspond-il ?

2.2 Commandes de gestion des processus

Changer la priorité d'un processus Les processus tournent avec un certain degré de priorité, un processus plus prioritaire aura tendance à s'accaparer plus souvent les ressources du système pour arriver le plus vite possible au terme de son exécution. C'est le rôle du système d'exploitation de gérer ces priorités (via l'ordonnanceur).

Pour modifier la priorité d'un processus, vous disposez de la commande `nice`. Plus le nombre est grand, plus la priorité est faible. Par exemple une valeur de 0 donne la priorité la plus haute ; 20 donne la priorité la plus faible. La fourchette de valeur dépends de l'UNIX utilisé.

QUESTION 1. Compilez le programme `testtime` fourni sur *madoc* qui effectue une boucle infinie et mesure le temps passé dans la boucle.

QUESTION 2. Faites en sorte que l'exécution dure quelques secondes et exécuter plusieurs fois le programme simultanément (il faut plus de terminaux que de cœur d'exécution de votre machine). Que remarquez-vous ?

QUESTION 3. Modifiez la priorité d'un processus et recommencer l'opération. Décrivez le résultat.

Généralement on utilise `nice` sur des commandes qui prennent du temps afin de ne pas handicaper le reste des commandes exécutées sur la machine. Sur des commandes courantes l'effet est imperceptible.

Lancer en processus en tâche de fond Pour démarrer un processus, il suffit d'utiliser une commande quelconque dans un terminal. Cette commande est lancée en tant que processus fils du shell. Tant que la commande n'est pas terminée, vous n'avez plus la main au niveau du shell (vous ne disposez plus du prompt). En réalité le processus du shell est en attente de la terminaison du processus fils créé.

Vous avez cependant la possibilité de modifier ce comportement par défaut afin d'indiquer au processus shell de ne pas attendre la fin du processus fils pour redevenir interactif. Pour cela, il vous suffit de rajouter un `&` à la fin de commande. Celle-ci se lancera en *tâche de fond*, et vous reviendrez directement au prompt du shell. À la suite de la saisie de la commande suivie d'un `&`, le shell vous donne immédiatement la main, et affiche le numéro du PID du processus lancé.

Vous avez également la possibilité de demander au shell de vous rendre la main avant la fin du processus fils en envoyant un signal particulier (`CTRL+Z` : *suspend*) au processus fils, puis d'exécuter la commande `bg` pour basculer le programme utilisateur en tâche de fond.

QUESTION 1. Expliquer ce qui se produit lorsque vous effectuez les lancements d'applications suivantes :

`commande`	`commande &`	`commande1; commande 2`
`commande1 & commande 2`	`(commande 1; commande 2) &`	`{commande 1; commande 2} &`
`commande 1 && commande 2`	`commande 1 || commande2`	`exec commande`

Gérer les signaux Un signal associé à un événement peut arriver vers un processus, sans que l'événement se soit réellement produit. La commande `kill(1)` et l'appel système `kill(2)` permettent d'envoyer n'importe quel signal vers n'importe quel processus (du moins vers ceux qui appartiennent au même propriétaire).

Il est également possible sous Linux de rediriger les signaux dans un terminal via la commande `trap`.

QUESTION 1. Expliquez comment supprimer un processus en arrière plan.

QUESTION 2. Expliquez la séquence interactive suivante :

Listing 1 – Commande `kill`

```
(while [ 1 -eq 1 ]; do :; done) & #boucle infinie
ps -l $!      # relever le pid ($!) du processus
kill -STOP pid
ps -l
kill -CONT pid
ps -l
```

Question 3. Comparer `kill -KILL pid` et `kill pid`

Question 4. Expliquer la commande `killall`

Question 5. Expliquer les 3 commandes suivantes

`trap 'rm /tmp/temporaire; exit' 2 3`	`trap ' ' 2`	`trap 2`

Question 6. Exécuter, puis expliquez le fichier suivant :

Listing 2 – Commande `trap`

```
trap 'echo touche \<ctrl c\>' 2
trap 'echo touche \<ctrl \>' 3
while true; do
i=1
done
```