Introduction aux systèmes d'exploitation (IS1)
TP n° 6 : les processus UNIX

Attention, sous Ubuntu configuré en français, `bash` interprète `[M-P]` comme `[MnNoOpP]` et non `[MNOP]`. Pour remédier à ce problème, vous devez ajouter la commande suivante dans votre fichier `.bashrc` :

```
export LC_COLLATE=C
```

(sans espace autour du signe =). Faites-le impérativement sur votre compte du script **avant** le contrôle sur machine.

Les processus

On appelle *processus* un objet dynamique correspondant à l'exécution d'un programme ou d'une commande Unix par le système. Un processus est formé du programme qu'il exécute, mais aussi des données nécessaires au programme, et d'un *contexte d'exécution* (informations utilisées par le système pour gérer le processus).

La création d'un nouveau processus s'effectue généralement par clonage d'un processus existant, son *parent* ; on parle d'*arborescence des processus*.

Chaque processus possède des caractéristiques permettant au système de l'identifier :
- son numéro d'identification (PID pour *process identifier*), qui est un entier positif ;
- celui de son processus parent (PPID pour *parent PID*) ;
- son (ses) propriétaire(s), comprenant souvent l'utilisateur qui l'a lancé ;
- éventuellement le terminal dont il dépend (s'il existe) ;
- son répertoire courant (de travail) ;
- sa priorité de travail ;
- son temps d'exécution ;
- etc.

Obtenir des informations sur les processus

« `ps` » appelée sans option, affiche la liste des processus de l'utilisateur courant rattachés au terminal courant.

Exercice 1 – *lister les processus : la commande « `ps` »*

Lancer quelques programmes (navigateur web, emacs) depuis votre interface graphique (sans utiliser le terminal). Ensuite, dans un terminal, lancer les utilitaires « `xcalc` » et « `xclock` » via les commandes « `xcalc &` » et « `xclock &` ». Notez que le caractère `&` permet de reprendre la main immédiatement dans le terminal.

1. ✍ Sans fermer les programmes que vous venez de lancer, tester la commande « `ps` » (sans options) dans ce terminal, puis dans un autre. Quels processus voyez-vous ? Quelle option de « `ps` » permet d'afficher tous vos processus y compris ceux qui ne sont pas rattachés à un terminal ? Quelle option de « `ps` » permet d'afficher tous vos processus rattachés à un terminal ? Pour chaque processus, quelle est la signification des informations fournies ?

2. Certaines options de « `ps` », comme « `-l` » ou « `u` », permettent d'afficher plus de détails sur les processus. ✍ Tester ces deux options et repérer les caractéristiques des processus correspondant à « `xcalc` » et « `xclock` ». ✍ Déterminer (à l'aide du manuel) la signification des colonnes `UID`, `PPID`, `PRI`, `TT(Y)`, `SZ`, `S(TAT)`.

3. ✍ Comment afficher avec « `ps` » la liste de tous les processus du système, y compris ceux dont vous n'êtes pas le propriétaire et ceux qui ne dépendent pas d'un terminal ?

4. ✍ Comment obtenir la liste des processus dont le propriétaire est `machin` ? En déduire comment afficher les processus appartenant au *super utilisateur* (*root*).

5. ✍ Quelle option utiliser pour afficher le processus dont le PID est `n` ? (remplacer `n` par un PID existant).

6. ✍ Pour chacun des processus que vous avez lancés dans le terminal (`xclock`, etc.), quel est le PID de son parent ? Quel est ce processus parent ? Remonter l'arbre généalogique du processus exécutant « `xclock` » et identifier ses ancêtres sur 3 générations.

7. À l'aide de la commande « `pstree` », afficher l'arbre généalogique des processus rattachés à votre terminal.

« `top` » **et** « `htop` » listent dynamiquement (en temps réel) les processus du système via un gestionnaire de tâches interactif.

Exercice 2 – *la commande « `top` »*

1. Lancer la commande « `top` ». Remarquer deux zones principales d'informations. ✍ Que signifient les informations affichées dans les colonnes PID, UID, PRI, SIZE, COMMAND ? Dans quel ordre les processus sont-ils classés ?

2. Lancer ensuite « `htop` » et comparer. ✍ Comment trier les processus affichés selon leur durée d'exécution ? afficher l'arborescence des processus ? afficher uniquement les processus vous appartenant ?

Communiquer avec les processus et les contrôler via des signaux

Il est possible d'envoyer des informations aux processus, via des *signaux*, pour les manipuler et changer leur état. Certains signaux provoquent la suspension, l'arrêt ou la mort d'un processus.

« `kill` » permet d'envoyer différents types de signaux à un processus dont on connaît l'identifiant (PID). Malgré son nom, elle ne sert pas *seulement* à "tuer" un processus.
Syntaxe d'envoi d'un signal : `kill -signal PID` où *signal* est un numéro ou un nom de signal (le signal par défaut est `SIGTERM`), et *PID* l'identifiant du processus concerné.

Il existe de nombreux signaux différents dont vous trouverez la signification dans le `man`. Les signaux les plus courants sont `SIGTERM`, `SIGKILL` pour terminer (tuer) un processus, ainsi que `SIGSTOP` et `SIGCONT` qui servent respectivement à arrêter provisoirement / reprendre l'exécution d'un processus là où elle en était restée.

Exercice 3 – *envoyer des signaux : la commande « `kill` »*

1. ✍ Comment obtenir la liste de tous les signaux que l'on peut envoyer aux processus ? Quels sont les numéros correspondant à `SIGTERM`, `SIGKILL`, `SIGINT`, `SIGSTOP` et `SIGCONT` ?

2. Tester les cinq signaux ci-dessus sur des processus « `xcalc &` ». ✍ Que se passe-t-il ?

3. ✍ Essayer maintenant d'envoyer ces mêmes signaux au processus bash d'où « `xcalc &` » a été lancé. Noter et comparer les effets obtenus, notamment en termes du sort destiné au fils « `xcalc` » et de la survie de la fenêtre. (Vous ouvrirez une nouvelle fenêtre, d'ou vous lancerez un nouvel « `xcalc &` » à chaque fois que nécessaire).

4. Créer une fenêtre terminal par la commande « `xterm &` » , dans cette fenêtre lancez un processus « `xcalc &` » . ✍ Remonter l'arbre généalogique du processus exécutant « `xcalc` » et identifier ses ancêtres sur 3 générations.

5. ✍ Tester les signaux `SIGTERM`, `SIGKILL` et `SIGINT` sur le processus « `xterm &` » correspondant au terminal où ce « `xcalc &` » est executé . Noter et comparer les effets obtenus, notamment en termes du sort destiné aux descendants du processus tué (vous vérifierez leur statut par « `ps` » après avoir envoyé le signal) et de la survie de la fenêtre terminal. (Vous ouvrirez une nouvelle fenêtre avec « `xterm &` », d'ou vous lancerez un nouvel « `xcalc &` » à chaque fois que nécessaire).

6. Repérer parmi les processus actifs un processus dont vous n'êtes pas propriétaire et tenter de le stopper par l'envoi du signal `SIGSTOP`. ✍ Que se passe-t-il ? Qu'en déduisez-vous ?

7. ✍ Lancer de nouveaux processus, puis la commande « `top` » ou « `htop` ». Comment envoyer des signaux aux processus à partir de ce gestionnaire ?

8. La commande « `killall` » permet d'envoyer des signaux à tous les processus repérés par leur nom et non par leur identifiant. ✍ Testez-la sur les processus correspondant à la commande `xclock` après avoir lancé plusieurs processus avec cette commande.

Exercice 4 – *le signal `SIGHUP`*

1. ✍ Lancer un terminal et y exécuter « `xclock &` ». Relever le PID du processus, puis fermer le terminal. Dans un autre terminal, afficher les informations concernant le processus « `xclock` ». Que constatez-vous ?

> « `nohup` » permet de protéger un processus contre le signal `SIGHUP` : lorsqu'un terminal est fermé, les processus qui en dépendent reçoivent le signal `SIGHUP`, ce qui par défaut provoque leur terminaison.

2. ✍ Relancer un processus « `xclock` » immunisé contre le signal `SIGHUP`, et vérifier qu'il survit à la fermeture du terminal depuis lequel il a été lancé.

3. ✍ Quel est le PPID de « `xclock` » avant et après la fermeture du terminal ? On dit que le processus *orphelin* a été adopté.

Gérer les tâches (ou *jobs*) liées au shell courant

Les shells modernes, en particulier `bash`, fournissent un ensemble de mécanismes pour gérer l'exécution des processus lancés depuis un même terminal. Dans ce contexte, on parle de *tâches* (*job*) de la *session de travail* (définie par le shell courant).

Chaque tâche, correspondant à un *groupe de processus*, est identifiée de manière interne au shell, et peut se trouver dans trois états distincts :
- en avant-plan (*foreground*) : elle peut lire et écrire dans le terminal ; ***au plus une*** tâche peut avoir cet état (par défaut, le shell, qui réagit aux commandes saisies) ;
- en arrière-plan (*background*) : elle s'exécute sans lire ni écrire dans le terminal ;
- *stoppé* ou *suspendu* : la tâche est en sommeil, son exécution est interrompue.

Exercice 5 – *états et changements d'états*

1. Depuis un terminal, lancer un processus « `xcalc` » **avec un `&` à la fin**. ✍ Dans quel état se trouve ce processus ?

2. Maintenant, dans le terminal, lancer un processus (par exemple « `emacs` ») **sans le `&` final**. Dans quel état se trouve ce processus ? Presser dans le terminal la combinaison de touches *ctrl-Z*, aussi appelée commande de suspension (*suspend*). Pouvez-vous encore utiliser « `emacs` » ? ✍ Dans quel état se trouve ce processus ?

3. La combinaison de touches *ctrl-Z* correspond à l'envoi d'un signal. ✍ En utilisant le manuel de « `kill` », déterminer le nom du signal envoyé. Vérifier votre réponse en utilisant « `kill` ».

4. ✍ En utilisant « `kill` », envoyer un signal à un processus suspendu pour lui faire reprendre son exécution.

Nous sommes maintenant capables de lancer des processus à l'avant ou à l'arrière-plan, de suspendre des processus et de reprendre leur exécution.

Pour simplifier la manipulation des processus dépendant d'un même shell, il leur est attribué un numéro de tâche (*job number*) propre au shell qui les contrôle (différent en particulier du PID).

> « `jobs` » permet d'afficher une liste de ces processus triée par numéro de job, ainsi que leur état actuel. Un signe "+" marque le job courant, un "-" le job précédent.

Exercice 6 – *la commande « `jobs` »*

1. ✍ Dans un nouveau terminal, tester cette commande et comparer sa sortie à celle de « `ps` » sans argument.

2. ✍ Exécuter « `xclock &` », « `xcalc &` » puis « `jobs` ». Que constatez-vous sur les numéros de job associés aux programmes ? Comment afficher le PID du processus correspondant à une tâche avec la commande « `jobs` » ?

> Deux commandes supplémentaires permettent de ramener un processus à l'avant-plan (« `fg` ») ou de faire reprendre son exécution à l'arrière-plan à un processus interrompu (« `bg` »). Sans argument, ces commandes s'appliquent au processus courant (cf. remarques précédentes). Elles peuvent aussi être suivies d'un argument de la forme %n, où n est un numéro de job.

Exercice 7 – *les commandes « `fg` » et « `bg` »*

1. Lancer les jobs « `xclock &` » et « `xcalc &` ». ✍ Amener « `xclock` » à l'avant-plan et puis de nouveau en arrière-plan. Combien de processus peuvent être en avant-plan ?

2. ✍ À quelle commande est équivalente la séquence : « `xcalc` » suivi de *ctrl-Z* puis de « `bg` » ?

3. ✍ Terminer chacun de vos jobs en les ramenant à l'avant-plan et en pressant la combinaison de touches *ctrl-C*. À quel signal correspond ce raccourci ?