

# Introduction aux systèmes d'exploitation (IS1)

## TP n° 7 : les processus UNIX

### 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), en général 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...

### Préliminaires

#### Exercice 1 – bloquer et débloquer son terminal








1. Dans votre terminal, lancer la commande « `~is1p/cbonsai` ». 🐞 Pouvez-vous écrire dans le terminal ? Pourquoi ?
2. Taper maintenant, la combinaison de touches `ctrl-Z`. 🐞 Que constatez-vous concernant l'exécution de « `cbonsai` » ? Pouvez-vous écrire dans le terminal ? Pourquoi ?
3. Tapez maintenant la commande « `fg` ». 🐞 Que constatez-vous concernant l'exécution de « `cbonsai` » ? Conclure sur ce qu'a fait la combinaison de touches `ctrl-Z` de la question précédente. Qu'a fait la commande « `fg` » ?
4. En utilisant la combinaison de touches `ctrl-C`, reprendre la main sur votre terminal. 🐞 Que répond alors la commande « `fg` » ? En déduire ce qu'a fait la combinaison de touches `ctrl-C`.

## Obtenir des informations sur les processus

« ps » permet d'afficher des informations concernant un ensemble de processus ; la nature des informations et la liste des processus concernés varient selon les options, et le comportement sans option varie selon les installations ; sous linux, « ps » sans option liste les processus de l'utilisateur courant rattachés au terminal courant.

### Exercice 2 – lister les processus : la commande « ps »

Lancer quelques programmes (navigateur web, « subl ») 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 ouvrir un second terminal (toujours sans fermer les programmes « xcalc » et « xclock ») et tester la commande « ps » dans celui-ci. Quels processus voyez-vous dans chaque cas ?
2.  Sans fermer les programmes que vous aviez lancé dans le premier terminal, lancer sur le second terminal la commande « firefox & ». 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 quelconque ? Pour chaque processus, quelle est la signification des informations affichées par « ps » ?
3. 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, TT(Y), S(TAT).
4.  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 ?
5.  Quelle option utiliser pour afficher uniquement le processus de PID n fixé ? (remplacer n par un PID existant).
6.  Pour chacun des processus que vous avez lancés dans le premier 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 au terminal.

## 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 (ou des) processus ; malgré son nom, elle ne sert pas *seulement* à « tuer » un processus.  
Syntaxe d'envoi d'un signal à un processus : « kill -signal PID » où *signal* est un numéro ou un nom de signal (optionnel, 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 ou 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 ? Quels sont les numéros correspondant à SIGTERM, SIGKILL, SIGINT, SIGSTOP et SIGCONT ?
2. Tester les cinq signaux ci-dessus sur des processus « xcalc & ». Dans les cas où l'envoi d'un signal tue le processus « xcalc », il faudra en relancer un nouveau.  
➤ Que se passe-t-il ? Dans les cas où la fenêtre de la calculatrice paraît survivre, pouvez-vous vous en servir ?
3. ➤ Essayer maintenant d'envoyer ces mêmes signaux au processus « bash » depuis lequel « xcalc & » a été lancé. Noter et comparer les effets obtenus, notamment en ce qui concerne le sort du fils (« xcalc ») et la survie de la fenêtre de terminal.  
(Vous ouvrirez une nouvelle fenêtre, d'où vous lancerez un nouvel « xcalc & » à chaque fois que nécessaire).
4. 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 ?

« killall » permet d'envoyer des signaux en ciblant tous les processus exécutant une certaine commande, et non en fonction de leur identifiant.

5. ➤ Tester « killall » sur les processus correspondant à la commande xclock après avoir lancé plusieurs processus exécutant cette commande.
6. Exécuter la commande « ~is1p/hydré-de-lerne », et vérifier (par exemple avec « pstree ») que cela a créé 9 processus. Ces processus sont très bien protégés contre la plupart des signaux... Essayer de les tuer avec « killall hydré-de-lerne » ou avec *ctrl-C*, et observer le résultat (toujours avec « pstree »). ➤ Comment tuer tout de même tous ces processus à l'aide de « killall » ?

**Exercice 4 – le signal *SIGHUP***

« sleep » se contente d'attendre le temps (en secondes) passé en paramètre.

1. 🚧 Lancer un terminal et y exécuter « sleep 1000 & ». Relever le PID du processus, puis fermer le terminal. Dans un autre terminal, afficher les informations concernant le processus « sleep ». Que constatez-vous ?

« nohup » permet de protéger un processus contre le signal SIGHUP (envoyé par le système aux processus dépendant d'un terminal lorsque celui-ci est fermé, ce qui par défaut provoque leur terminaison).

2. 🚧 Relancer un processus « sleep » 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 « sleep » avant et après la fermeture du terminal ?  
(on dit que le processus orphelin a été adopté)
4. Tester la commande « killall » sur les processus exécutant bash. 🚧 Que constatez-vous ? Pourquoi ? Lire la section *SIGNALS* du manuel de bash et 🚧 donner une ligne de commande utilisant killall pour tuer tous les processus bash d'un coup (sans le faire!).

**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 (jobs)* 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. Lancer ensuite un processus (par exemple « subl ») **sans le & final**. Dans quel état se trouve ce processus ? Presser dans le terminal la combinaison de touches *ctrl-Z*, 🚧 Dans quel état se trouve ce processus ?

3. La combinaison de touches *ctrl-Z* correspond à l'envoi d'un signal. 🐛 Déterminer lequel à l'aide du manuel de « *kill* ».
4. 🐛 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* » affiche la liste des jobs du shell courant, triée par numéro, ainsi que leur état actuel. Un signe "+" marque le job courant, un "-" le précédent.

#### Exercice 6 – la commande « *jobs* »

1. 🐛 Dans un nouveau terminal, tester cette commande et comparer sa sortie à celle de « *ps* » sans argument. Ensuite exécuter « *xclock &* » et répéter la comparaison des sorties de « *jobs* » et de « *ps* ».
2. 🐛 Dans le même terminal, exécuter « *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 job à l'avant-plan (« *fg* ») ou de faire reprendre son exécution à l'arrière-plan à un job interrompu (« *bg* »). Sans argument, ces commandes s'appliquent au job 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* », puis *ctrl-Z*, puis « *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 ?

## Pour aller plus loin

**Sur les systèmes Linux** (mais pas macOS), un *pseudo*-système de fichiers enraciné en « /proc » permet d'accéder aux informations du système concernant les processus en cours d'exécution. Par exemple, « /proc/PID/stat » contient les informations de base du processus d'identifiant PID donné (pid, commande exécutée, état, ppid...). Voir « man procfs » pour plus de précisions.

### Exercice 8 – le pseudo-système de fichiers des processus

1. Afficher les informations de base associées au processus 1.
2. En utilisant une seule commande, lister les informations de base de tous les processus sur votre machine.
3. En utilisant une seule commande, lister les informations de base des processus dont le pid contient le chiffre 5.