TP N° 5
Les Processus et leurs Communications

Objectifs

- Effectuer des commandes de gestion des processus et des tâches,
- Exécution des tâches en avant-plan et en arrière-plan (tâches de fond),
- Comprendre les canaux de communication standards utilisés par les processus des commandes shell,
- Utiliser les commandes avec les différents types de redirections,
- Comprendre la notion de tube (pipe) et son avantage par rapport à la redirection.

1) Les Processus

Un processus (ou tâche) est un programme (application) ou une commande **en cours d'exécution.** (chargé en mémoire et exécuté).

ps : (*processes snapshot*) affiche les informations sur les processus actifs dans le shell.

- **Tester les options –a, -l, -x**

- **Identification : PID** (Processus IDentification). Un processus est connu par le système d'exploitation par un numéro unique.

  pidof <nomprocessus> : retourne le PID d'un processus.

  - **Quel est le PID du processus du terminal et le PID du processus de la commande ps ?**
  - **Quel est le nom du processus du terminal ?**
  - **Utilisez pidof pour déterminer le PID du processus du terminal ?**

- **Filiation** : un processus est lancé par un autre processus. Notion de parenté père-fils. Il existe un processus unique, à l'origine de tous les autres.

  ps -f

  - **Vérifiez à travers le PPID que le père du processus de la commande ps –f est le shell ?**

  pstree : permet d'afficher les processus sous forme d'arborescence et donc de voir leurs indépendance. Permet aussi de connaitre le nom d'un processus.

- **Appartenance** : un processus appartient à un utilisateur et à un groupe.

  ps –u : tous les processus de l'utilisateur courant

  ps –aux : tous les processus en cours ⇔ **ps -ef**

- **Ressources système** : chaque processus pouvant accéder à des ressources du type fichier, une table de fichiers ouverts associée à chaque processus et mise à jour. (table des fichiers ouverts).

  top : permet d'afficher les processus en cours d'exécution (en mise à jour constamment). Il permet aussi de suivre les ressources consommées par les processus ?

  - **A quoi peut bien être utile la commande top** ?

  - **Quel est le PID et le nom du processus père de tous les autres processus ?**

- **Graphiquement**, vous pouvez aussi gérer les processus par le programme « ***Moniteur Système*** » d'Ubuntu.

- **Etat de fonctionnement** : à une date donnée, un processus ne peut être que dans l'un des états de fonctionnement suivants : actif (running), en attente de ressources (waiting) , en sommeil (sleeping).
  kill <signal> <pid> : permet de tuer un processus en cours d'exécution.
  Exemple :
  kill -9 7804 : Envoie le signal **9**, ou **KILL**, au processus ayant le PID 7804 ce qui a pour effet de **tuer** le processus.

  Si la fermeture d'un programme nécessite une permission, alors précéder le **kill** par un **sudo**.
  Killall permet de tuer le processus **par son nom** et non par son PID.
  - **Lancer firefox puis fermer le par un kill puis par un killall ?**

  Pour tuer une commande en cours d'exécution on peut utiliser aussi un **CTRL + C**.

Exercice A : processus

1. Taper la commande **gnome-terminal** pour ouvrir un nouveau shell à partir de votre shell.
2. Vérifier le PID de chacun des deux shell. Vérifier qu'ils ont le même père.
3. Tuer le processus du premier terminal par un kill.

- **Travail en tâche de fond** : un programme qui s'exécute en **tâche de fond** (**arrière-plan** ou **background**) est un **job**. Si on ne perd pas la main sur le shell, alors c'est un job en arrière-plan.
  Pour **suspendre** (**arrêter**, **stopper**) un job en tâche de fond on utilise un **Ctrl+Z.**

  jobs : permet de connaitre le statut d'un job dans une session courante.
  - **Taper jobs → rien ne s'affiche**
  - **Lancer gedit à partir du shell → le shell est en arrière-plan**
  - **Faites un Ctrl+Z → gedit est suspendu en arrière-plan et le shell est en avant-plan**
  - **Taper jobs → déterminer le numéro job et l'état du job du programme gedit**

  bg <numéro job> : envoyer un processus suspendu en arrière-plan (**background**)
  fg <numéro job> : envoyer un processus d'arrière-plan en avant-plan (**foreground**)

- Le symbole **&** lance la tâche en **arrière-plan** et **sans que le shell ne perd la main**. (Ceci n'est intéressant que pour une tâche dont l'exécution est longue)

  - **xeyes & firefox & :** Lancer les programmes xeyes et firefox en arrière-plan.
  - Déterminer leurs **états** par la commande **jobs.** (**[1]** pour xeyes et **[2]** pour firefox)
  - Taper **bg <numjob firefox> (bg 2).** (Firefox est déjà en arrière-plan)
  - Mettre le programme xeyes en **avant-plan** par la commande **fg.** On perd la main sur le shell : Ce dernier devient en arrière-plan.
  - Pour reprendre la main sur le shell, taper **Ctrl+Z** pour suspendre xeyes ou **Ctrl+C** pour le tuer**.** Dans les deux cas, le shell reprend la main (c-à-d devient en avant-plan)

- Un job est identifié par un *numéro unique* renvoyé par la commande **jobs**. Ce numéro est repéré par le système d'exploitation par le signe **%**. **Kill %numjob** : pour tuer le job

Exercice B : états des jobs

1. Lancer xeyes en arrière-plan. Taper jobs et déterminer son état.
2. Mettre xeyes en avant-plan par un fg. On perd la main sur le shell.
3. Suspendre (arrêter) xeyes par un Ctrl+Z. Taper jobs et déterminer son état.
4. Relancer xeyes en arrière plan par un bg. Taper jobs et déterminer son état.
5. Tuer xeyes par un kill par son numéro de job. Taper jobs et déterminer son état.

2) Les canaux de communication standard et les redirections

- Le shell, ou interpréteur de commandes, sert d'interface entre l'utilisateur et la machine (en fait, les processus en exécution).
- Chaque processus utilise des canaux de communication définis par défaut, ou « **canaux de communication standards** » (canaux d'entrées sorties par défaut).
- Il existe trois canaux standards : ces canaux sont aussi appelés « fichiers » et portent un nom.

  Entrée standard : **stdin**
  Sortie standard : **stdout**
  Sortie d'erreur standard : **stderr**
- Par défaut, les canaux standards sont associés à des périphériques et portent un numéro (appelé « **descripteur de fichier** ») :

  Entrée standard : **clavier** **numéro 0**
  Sortie standard : **console** **numéro 1**
  Sortie d'erreur standard : **console** **numéro 2**
- Il est possible de **rediriger** les flots d'informations (en entrée ou en sortie) vers des fichiers (**redirections**).

Redirection en sortie :

- Le résultat de l'exécution d'une tâche peut être envoyé vers un fichier à l'aide de **>**.
- Si l'on souhaite ne pas détruire le fichier lors d'une nouvelle exécution, il faut effectuer une concaténation à l'aide de **>>**.

Exemples : **ls -l > liste.txt**
ls -l >> liste.txt (avec concaténation)

- ❖ **Vérifier le résultat obtenu. Est-il nécessaire que le fichier « liste.txt » existe préalablement ?**

Redirection en entrée :

- Les données sont prise dans un fichier à l'aide de <.

Exemple : **wc < liste.txt** (*wc = word count*)

- ❖ Quel est l'affichage obtenu? Interprétez le résultat de l'affichage.
- Remarque : il n'est pas possible d'effectuer une redirection sur un éditeur de textes.

Redirection en entrée et en sortie :

- Il est possible de lancer une commande en redirigeant simultanément l'entrée et la sortie.
- Remarque : le shell traite la ligne de commande de la gauche vers la droite.

Exemple : **wc < liste.txt > resultat.txt**

Redirection du canal de sortie d'erreur standard :

- Le canal **stderr** porte le numéro 2. On utilise donc la commande : **2>**

Exemple : (le répertoire de travail ne doit pas contenir de répertoire nommé « ce_répertoire »)

- ❖ Lancez successivement les commandes :
  rmdir ce_repertoire
  rmdir ce_repertoire/ 2> erreur.txt
- ❖ Quelles sont les différences d'exécution entre ces deux commandes ?

- Si vous ne voulez pas voir les messages d'erreur s'afficher à l'écran, ni en faire une copie dans un fichier (politique de l'autruche) vous pouvez toujours les envoyer dans un trou noir :

Exemple : **rmdir ce_repertoire 2> /dev/null**

Redirection du canal de sortie standard ET du canal d'erreur standard:

Exemple : **(ls –l ; rmdir ce_repertoire) > fichier 2>&1**

- ❖ Expliquez la syntaxe utilisée pour la redirection ?
- ❖ Pourquoi a-t-on placé des parenthèses ? essayer la même commande sans parenthèses ?

Exercice C : Redirections

1. Enregistrez dans un fichier nommé « **data.txt** » les noms, les terminaux utilisés et les heures de connexion de tous les utilisateurs actuellement connectés sur le système.
   - Avec plusieurs lignes de commandes successives.
   - Avec une seule ligne de commande.
2. A l'aide de la commande **cat**, créez directement à partir du clavier, deux fichiers texte d'une ligne chacun nommés « **abc.txt** » et « **def.txt** ».
3. Expliquez les résultats obtenus par les commandes suivantes :

   cat def.txt >> abc.xyz
   cat def.txt >> def.txt
   cat < def.txt >> abc.txt

3) Les tubes

- On peut exécuter plusieurs commandes successivement de façon totalement indépendante. Cela s'obtient en séparant les commandes les unes des autres par le symbole « **;** » :
  commande ; commande ; commande ; *Exemple* : **ls ; whoami ; echo salut**

Ces trois commandes vont s'exécuter les unes après les autres.

- Cette fois le but est de faire exécuter les processus de ***manière concurrente*** (en parallèle) et communicant entre eux par l'intermédiaire de zones mémoires temporaires c'est-à-dire prises sur la mémoire centrale.
  - ❖ En utilisant la méthode de redirection, afficher la liste des fichiers se trouvant dans le répertoire courant, trié par nom et obtenir une visualisation page par page ?
  - ❖ C'est quoi l'inconvénient de cette méthode ?
- La notion de **pipe** ou **tube** va résoudre les problèmes de création de fichiers intermédiaire et le temps d'exécution. Un tube est symbolisé par le symbole « **|** » :

  ls ~ | sort | more
- Le système Linux assure la synchronisation de l'ensemble :
  - Les 3 processus correspondant aux 3 commandes s'exécutent en parallèle.
  - Chaque processus qui lit dans un tube se bloque lorsque le tube est vide.
  - Chaque processus qui écrit dans un tube s'arrête lorsque le tube est plein.
- Les tubes vont permettre, en enchainant des commandes de bases entre elles, de réaliser des traitements économisant le recours à la programmation.
- On peut sélectionner successivement certaines lignes d'un fichier (selon un motif défini), réaliser un tri sur le fichier, supprimer éventuellement les doublons, ne garder que les 20 premiers.
  - ❖ En utilisant la méthode de redirection, donner la liste de commandes shell permettant de sélectionner successivement certaines lignes d'un fichier (selon un motif défini), réaliser un tri sur le fichier, supprimer éventuellement les doublons, ne garder que les 20 premières lignes.
  - ❖ Exécuter cette commande : **grep http log | sort -n -k2 | uniq | head - 20**

  Cet exemple permet de traiter un fichier de journalisation, de ne sélectionner que les lignes comportant le mot « http », de trier ce fichier selon la deuxième colonne (en numérique), de supprimer les doublons et enfin d'en extraire les 20 premières lignes.

Exercice D :

Réaliser à l'aide de tube les opérations suivantes :

1. Compter le nombre de fichiers se trouvant dans le répertoire courant.
2. Compter le nombre d'utilisateurs connectés.
3. En utilisant le fichier /etc/passwd, donner la liste, classée par ordre alphabétique, des utilisateurs pouvant travailler en Bash.