Processus et mécanismes de redirection

1 La notion de processus Unix

1.1 Exercice Rappelez le nom de la commande permettant de donner la liste des processus actuellement actifs sur votre machine.

1.2 Exercice Sur quel processeur (i.e., de quelle machine) s'exécutent les commandes que vous invoquez dans votre `xterm`[5].

Définition : Un **programme** est un fichier exécutable, et un **processus** est une occurrence d'un programme en exécution.

Dans la suite du TD, n'hésitez pas à utiliser les aides `man` ou `info`

2 Première vision des processus

Dans cette partie, vous allez commencer à manipuler les processus à travers divers exemples. Les commandes à utiliser pour répondre aux questions suivantes sont : **ps, kill, jobs, fg, bg**.

2.1 Exercice Lancez le programme `xload` en tapant la commande 'xload'. Avez-vous toujours le contrôle dans la fenêtre `xterm` depuis laquelle vous avez lancé `xload` ?

2.2 Exercice Testez la commande `Ctrl-z` dans votre fenêtre xterm. Que se passe-t-il ? Pouvez notamment interagir avec `xload` ?

2.3 Exercice Testez la commande `jobs`. Qu'indique-t-elle ?

2.4 Exercice Testez la commande `fg`. Interrompez à nouveau le processus.

2.5 Exercice Testez la commande `bg`. Que se passe-t-il ? Avez-vous toujours le contrôle dans la fenêtre xterm.

2.6 Exercice Lancez à nouveau la commande `xload`. Interrompez-la.

2.7 Exercice Qu'indique maintenant la commande `jobs` ?

2.8 Exercice Que provoque la commande `kill -9 %1` ?

2.9 Exercice Testez la commande `ps`. Trouvez les **pid** de vos processus.

2.10 Exercice Testez la commande `ps -f`. Quelles sont les différences par rapport à la commande précédente ?

[5] Si vous ne le sazez pas, ou bien si vous voulez en être sûr, faites `man hostname`.

2.11 Exercice Quel est le numéro du processus `xload` restant ?

2.12 Exercice Testez la commande `ps -ef`.

2.13 Exercice Essayez de tuer (par la commande `kill -9 pid`) un processus d'un autre utilisateur. Que se passe-t-il ?

2.14 Exercice Tuez votre processus `xload` restant en utilisant cette fois-ci son numéro de processus (**pid**).

2.15 Exercice Lancez la commande `xload &`. Avez-vous toujours le contrôle dans la fenêtre xterm ?

2.16 Exercice Faites passer le processus en **avant-plan** par la commande `fg`.

2.17 Exercice Faites à nouveau passer le processus en **arrière-plan** avec la commande `bg`.

3 Les mécanismes de redirection

3.1 Redirection de la sortie standard

L'opérateur de redirection de la sortie standard dans une ligne de commande est « `>` ».

3.1 Exercice Lancez la commande `ls` depuis la racine de votre répertoire d'accueil.

3.2 Exercice Relancez la commande en redirigeant cette fois-ci la sortie standard dans un fichier dont le nom est `resultats`. Pour cela, utilisez la commande `ls > resultats`.

3.3 Exercice Éditez le fichier `resultats`. Que contient-il ?

3.2 Redirection de l'entrée standard

L'opérateur de redirection de l'entrée standard dans une ligne de commande est « `<` ».

3.4 Exercice Affichez le fichier `resultats`, en redirigeant l'entrée standard vers le fichier `resultats`. Pour cela, utilisez la commande `cat < resultats`. Que se passe-t-il ?

3.5 Exercice Quel est la différence entre la commande précédente et `cat resultats` ?

3.3 Redirection simultanée de l'entrée et de la sortie standard

Il est possible de combiner les redirections : l'exécution d'un programme **mon_programme** par la commande `mon_programme < donnee > resultats`, ira chercher les données en entrées dans le fichier `donnees` et écrira les résultats dans le fichier `resultats`, « court-circuitant » ainsi complètement clavier et écran.

Remarques :

1) Si on introduit l'ordre `mon_programme < donnees` et si le fichier `donnees` n'existe pas, l'interprète de commandes produira un message d'erreur à l'écran et arrêtera là l'exécution du programme.
2) Une autre façon d'exécuter le programme `mon_programme` permet encore d'ajouter, à la fin du fichier[6] `resultats` existant, de nouvelles données : `mon_programme < donnees >> resultats`.

[6] Concaténation des données.

4 Enchaînement des processus

4.1 Enchaînement séquentiel indépendant

Si vous désirez lancer plusieurs commandes les unes à la suite des autres, vous pouvez soit les taper une par une au clavier, soit les taper sur une seule ligne de commande en les séparant par un « ; ».

4.1 Exercice Effacez votre fichier `resultats`.

4.2 Exercice Lancez la commande `ls > resultats ; ls >> resultats ; ls >> resultats`.

4.3 Exercice Éditez le fichier `resultats`. Que contient-il ?

4.2 Enchaînement parallèle avec communications

Nous avons dit au début de ce TD que les processus s'exécutaient indépendamment les uns des autres. Néanmoins, vous allez maintenant voir qu'il est possible de créer des communications entre processus. L'exécution des processus peut alors dépendre des messages qu'ils reçoivent.

Définition : Un **pipe** est un fichier de caractères[7] géré en file (FIFO[8]) et de taille limitée[9].

- Si le **pipe** est plein, le processus qui cherche à écrire dedans est suspendu (*i.e.* il attend que le **pipe** se vide).
- Si le **pipe** est vide, le processus qui cherche à lire dedans est suspendu (*i.e.* il attend que le **pipe** se remplisse).
- Le **pipe** est dédruit automatiquement par le système uniquement lorsque les deux processus (*i.e.* celui qui écrit et celui qui lit dedans) se sont terminés.

L'opérateur de pipe est « `|` ».

4.4 Exercice Essayez la commande `ls -l /usr/bin | more` et expliquez son fonctionnement.

4.3 Applications

Les questions suivantes vont vous permettre d'utiliser les mécanismes de redirection et d'enchaînement des processus ('<', '>', '|', '>>', ...) avec quelques commandes Unix simples (`cat`, `wc tail head`). Bien sûr, dans cette partie, vous n'avez pas le droit d'utiliser emacs . Toutes les réponses aux questions suivantes se font dans votre fenêtre xterm.

[7] C'est en fait un espace de mémoire partagée.
[8] First In First Out.
[9] Généralement de 4 Ko.

4.5 Exercice Essayer quelques exemples de redirections (< > |).

Exemples

```
$ cat > tmp
 ...
^D

$ cat < tmp

$ ls -L > LS-L

$ ls | wc
```

```
$ ps

$ sort

$ ps | sort

$ ps -af  | grep bash | sort
```

Remarquer les commandes `head` et `tail`, qui donnent (respectivement) le début et la fin d'un flot de lignes ; et la commande `sort`, qui trie les lignes d'un flot.

4.6 Exercice Comparer et tester l'enchainement de commandes ainsi que les résultats des commandes suivantes :

```
$ ps -fax | less
$ du > output.txt ; sort -n output.txt
$ du | sort -n
$ du | sort -rn | less
```

4.7 Exercice Créer un fichier d'exemple au moyen de

```
$ yes ligne | head -10 | cat -n > Lignes
```

et puis trouver des combinaisons de `head` et `tail` qui donnent :
- les premières trois, les dernières trois lignes du fichier
- tous sauf les premières trois lignes
- la première et la dernière ligne
- les trois lignes à partir de la deuxième ligne

5 La sortie d'erreur standard

Par défaut, les erreurs envoyées sur la sortie d'erreur standard s'affichent au même endroit que les résultats envoyeés sur sortie standard, c'est-à-dire votre feneçtre xterm. Néanmoins, il vous est aussi possible de rediriger cette sortie d'erreur standard.
- Pour rediriger la sortie d'erreur sur un nouveau fichier : '`2>`' ;
- Pour rediriger la sortie d'erreur sur un fichier existant (par concaténation) : '`2>>`'.

Pour mieux comprendre, nous allons exécuter une commande sous Unix produisant une erreur.

5.1 Exercice Placez-vous dans votre **home directory**. Essayez de vous déplacer dans le répertoire `pouf` en utilisant la commande `cd pouf`.

5.2 Exercice Est-il apparu un message d'erreur à l'exécution de la commande précédente ? Si oui, où est-il apparu ?

5.3 Exercice Essayez de vous déplacer dans le répertoire `pouf` en utilisant la commande `cd pouf 2> erreur`.

5.4 EXERCICE Est-il apparu un message d'erreur dans votre **xterm** à l'exécution de la commande précédente ? A-t-il été créé un fichier `erreur` ? Si oui, visualisez-en le contenu. Qu'en concluez-vous ?

5.1 Application - redirection simultanée de la sortie et de la sortie d'erreur

Vous allez maintenant combiner ces mécanismes de redirection. Pour ce faire, nous allons utiliser une commande pour laquelle il est très pratique de rediriger la sortie et la sortie d'erreur dans deux fichiers différents.

Supposons que vous cherchiez à localiser dans votre **home directory** le répertoire `indentation`, mais vous ne vous rappellez plus où se trouve exactement ce répertoire..

5.5 EXERCICE Placez-vous dans votre **home directory**.

5.6 EXERCICE Exécutez la commande : `find . -name indentation -print`. Avez-vous localisé le répertoire `indentation` ?

Cette commande `find` accepte de nombreux paramètres et options de recherche[10]. Examinons ensemble la syntaxe de la commande de l'exercice précédent. Celle-ci se décompose comme suit :

`find` <u>`pathname`</u> `-name` <u>`file`</u> `-print`

et signifie : *rechercher*[11] *tous les fichiers ou répertoires portant le nom* `file`[12] *à partir du* *répertoire* `pathname`[13] *et* *afficher le résultat de la recherche sur* *la sortie standard*[14].

5.7 EXERCICE Recherchez à partir du répertoire `/` tous les fichiers ou répertoires portant le nom `libjpeg` Avez-vous eu également des messages d'erreur ?

En fait, les messages d'erreur qui apparaissent à l'exécution de la commande viennent perturber la visualisation des résultats de la recherche. Vous allez donc modifier la commande pour simplifier la visualisation des résultats.

5.8 EXERCICE idem à l'exercice précédent, mais en redirigeant la sortie standard dans un fichier `resultats_recherche`. Qu'apparaît-il cette fois-ci dans votre **xterm** ? Listez le contenu du fichier `resultats_recherche`.

5.9 EXERCICE Comment faire pour exécuter ces programmes en tâche de fond -arrière plan- ?

Pour éviter de polluer votre **xterm** avec les messages d'erreur de la commande, il est préférable de rediriger la sortie d'erreur standard dans un fichier particulier.

5.10 EXERCICE Effacez le fichier `resultats_recherche`.

5.11 EXERCICE Recherchez à partir du répertoire `~` tous les fichiers ou répertoires portant le nom `.*rc`[15], mais vous redirigerez la sortie standard dans un fichier `resultats_recherche` et la sortie d'erreur standard dans un fichier `erreur_recherche` et vous lancerez la commande en *tâche de fond.*

5.12 EXERCICE Vérifiez à l'aide de la commande `ps`, que le processus associé à la commande `find` de l'exercice précédent, est achevé.

5.13 EXERCICE Des messages dus à la commande `find` sont-ils apparus dans votre fenêtre xterm ? Listez les contenus des fichiers `resultats_recherche` et `erreur_recherche`.

---

[10] Pour une description exhaustive de la commande, lancez `info find`.
[11] `find`
[12] `-name` <u>`file`</u>
[13] <u>`pathname`</u>
[14] `-print`
[15] Attention, penser à déspécialiser le caractère `*` par `\*`.

5.2 Un périphérique bien pratique

Vous avez peut-être déjà vu que le répertoire `/dev` contenait une liste de fichiers particuliers correspondant à des périphériques (*devices*). L'un d'eux est le périphérique `null`. Celui-ci peut être très utile, car il joue le rôle de *poubelle* ou plutôt de *trou noir*. En effet, vous avez la possibilité de rediriger la sortie standard ou sortie d'erreur standard sur ce périphérique, et toute information écrite dans ce périphérique sera inexorablement perdue à jamais.

Dans l'exercice utilisant la commande `find`, les messages d'erreur ne sont dans ce cas précis que d'une piètre utilité. D'où l'intérêt de rediriger les messages d'erreur vers le périphérique `/dev/null`.

5.14 Exercice Effacez les fichiers `resultats_recherche` et `erreur_recherche`. Recherchez à partir du répertoire `/` tous les fichiers ou répertoires portant le nom `libjeg`, en prenant soin de rediriger les messages d'erreur vers le périphérique `/dev/null`. Visualisez le contenu du fichier `/dev/null`. Que contient-il ?