Processus (2)

1 Les mécanismes de redirection

Un programme, lorsqu'il s'exécute, a généralement besoin de données en entrée et doit produire des résultats (sinon il n'est pas très utile). Lors de l'élaboration d'un programme [1], il est donc nécessaire de préciser **d'où proviennent les données** et **où iront les résultats**. Sous Unix, ce sont les fichiers qui sont utilisés comme support pour contenir ces données et ces résultats.

Chaque fichier est identifié par un nom. Parmi ces fichiers, il en est deux qui ont un statut particulier ; on les appelle l'**entrée standard** [2] et la **sortie standard** [3]. Bon nombre de programmes écrits par des utilisateurs et bon nombre de commandes utilisent l'entrée et la sortie standard.

Remarque : Par défaut, Unix associe l'entrée standard au **clavier** et la sortie standard à la **fenêtre dans laquelle l'ordre d'exécution a été donné** (généralement une fenêtre xterm). Lire le fichier **entrée standard** équivaut donc à lire ce qui est introduit au clavier et écrire sur le fichier **sortie standard** équivaut généralement à afficher les résultats dans votre **xterm**.

Exemple : La commande `ls` écrit son résultat (*i.e.* le contenu du répertoire courant) dans le fichier de la sortie standard, qui est par défaut la fenêtre dans laquelle on a tapé la commande `ls`.

Néanmoins, il est possible, grâce au système, d'indiquer lors de l'exécution d'un programme qu'il faut « aller chercher » l'entrée standard ailleurs qu'au clavier et « déposer » les résultats ailleurs qu'à l'écran. On parle dans ce cas de **redirection de l'entrée et de la sortie standard**.

1.1 Redirection de la sortie standard

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

1.1 Exercice Lancez la commande `ls` depuis votre **home directory**.

1.2 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`.

1.3 Exercice Ouvrez le fichier `resultats`. Que contient-il ?

1.2 Redirection de l'entrée standard

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

1.4 Exercice Envoyez vous un courrier, en redirigeant l'entrée standard vers le fichier `resultats`. Pour cela, utilisez la commande `Mail votre_email < resultats` avec une adresse email que vous consultez habituellement. Que se passe-t-il ?

1.5 Exercice Lisez le courrier que vous venez de vous envoyer. Que contient-il ?

1. Dans le code du programme, ou dans les paramètres de lancement d'une commande.
2. L'entrée standard est appelée **stdin**.
3. La sortie standard est appelée **stdout**.

1.3 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 < donnees > resultats`, ira chercher les données en entrée dans le fichier `donnees` et écrira les résultats dans le fichier `resultats`, « court-circuitant » ainsi complètement clavier et écran.

Remarque :

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 d'ajouter les nouveaux résultats à la fin du fichier[4] `resultats` existant : `mon_programme < donnees >> resultats`.

2 Enchaînement des processus

2.1 Enchaînement séquentiel indépendant

Il est possible d'enchaîner plusieurs commandes sur une même ligne en les séparant par un « ; ».

2.1 Exercice Effacez votre fichier `resultats`.

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

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

Remarque : Attention : l'enchaînement des commandes par un « ; » se fait de façon séquentielle. En effet, avec une ligne de commande du type :

```
$ commande_1 ; commande_2 ; ...; commande_n-1 ; commande_n
```

le processus associé à la commande 2 ne débute que lorsque le processus associé à la commande 1 est terminé, et ainsi de suite. Le processus associé à la commande n ne débutera que lorsque le processus associé à la commande $n-1$ sera terminé[5]. Les processus sont exécutés séquentiellement et indépendamment les uns des autres[6] en suivant l'ordre dans lequel ils ont été tapés sur la ligne de commande.

2.2 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 (re)voir qu'il est possible de créer des communications entre processus. L'exécution des processus peut alors dépendre des messages qu'ils reçoivent. Pour cela analysons l'exemple suivant :

Exemple :

```
$ ls -l /usr/bin > resultats ; more < resultats ; rm resultats
```

Si on regarde attentivement cette ligne de commande, on se rend compte que

1) `ls -l /usr/bin > resultats` écrit dans le fichier `resultats` par redirection de la sortie standard la liste des fichiers du répertoire `/usr/bin` ;
2) `more` lit dans le fichier `resultats` par redirection de l'entrée standard ;
3) `resultats` sert de fichier auxiliaire ou temporaire et peut être supprimé à l'issue du `more`.

Il est possible de simplifier l'écriture de l'exemple précédent. Pour ce faire, Unix met à votre disposition un mécanisme particulier appelé **pipe**[7]. C'est un opérateur permettant de connecter directement la

4. Concaténation des données.
5. Quel que soit le mode d'achèvement, c'est-à-dire même si le processus s'est terminé avec une erreur.
6. Il n'y a pas d'échange direct d'informations entre les processus.
7. **Tube** ou **tuyau** en français, mais on préférera généralement le terme anglais.

sortie standard d'un premier programme à l'**entrée standard** d'un second sans devoir spécifier de fichier auxiliaire. L'opérateur de pipe est « `|` ».

Exemple : Reprenons l'exemple précédent, et réécrivons-le en utilisant cette fois-ci un **pipe** :

```
$ ls -l /usr/bin | more
```

Le processus `ls -l /usr/bin` envoie son résultat sur l'entrée standard du processus `more`, ainsi, il n'est plus nécessaire d'utiliser un fichier temporaire comme dans l'exemple précédent (fichier `resultats`).

Le **pipe** est un mécanisme de communication qui permet d'échanger des informations entre processus.

Définition : Un **pipe** est un fichier de caractères [8] géré en file (FIFO [9]) et de taille limitée [10].

- 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 détruit automatiquement par le système uniquement lorsque les deux processus (*i.e.* celui qui écrit et celui qui lit dedans) se sont terminés.

Remarque : Il faut bien comprendre que, contrairement à l'enchaînement séquentiel des processus (par utilisation du `;`), une commande du type :

```
$ commande_1 | commande_2 | ...| commande_n-1 | commande_n
```

provoque dans le système la création « simultanée » d'un processus distinct associé à chacune des commandes. Ainsi, dans cet exemple, cohabiteront simultanément un processus 1, 2, ..., $n-1$ et n.

2.3 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.

2.4 Exercice Essayez 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
```

Remarquez les commandes `head` et `tail`, qui donnent (respectivement) le début de la fin d'un flot de lignes ; la commande `wc` qui affiche le nombre de lignes, de mots, et de bytes d'un fichier ; et `sort` qui permet de faire des tris.

2.5 Exercice Trouvez les combinaisons de commandes et de redirection permettant de

- enregistrer le résultat de la commande `wc` lancée sur un fichier de votre choix dans un fichier `res`

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

– trier les 6 premières lignes d'un fichier
– compter le nombre de mots contenus dans les 4 dernières lignes d'un fichier
– enregistrer dans un fichier les 5 premières lignes (par ordre alphabétique) d'un fichier

3 La sortie d'erreur standard

Vous avez vu dans les sections précédentes qu'il était possible d'utiliser et de rediriger l'entrée standard *(stdin)* ainsi que la sortie standard *(stdout)*. Un programme, s'il est correctement écrit doit intégrer un système de gestion d'erreurs. En effet, au cas où une erreur interviendrait durant l'exécution d'un programme, ce dernier se doit d'en informer au mieux l'utilisateur. Pour cela, un moyen efficace est de renvoyer un message d'erreur. Il existe sous Unix un fichier particulier dans lequel un programme peut écrire ses messages d'erreur, c'est la **sortie d'erreur standard** *(stderr)*.

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 fenêtre xterm. Néanmoins, il vous est aussi possible de rediriger cette sortie d'erreur standard.

– Le symbole utilisé pour rediriger la sortie d'erreur sur un nouveau fichier est '`2>`' ;
– Le symbole utilisé pour rediriger la sortie d'erreur sur un fichier existant (concaténation) est '`2>>`'.

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

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

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

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

3.4 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 ?

3.5 Exercice Réexécutez la commande `cd pouf 2> erreur`. Que contient le fichier `erreur` ?

3.6 Exercice Exécutez la commande `cd pouf 2» erreur`. Que contient le fichier `erreur` ?

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

Vous allez maintenant combiner ces mécanismes de redirection. Pour se 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.

Par exemple, si vous cherchez à localiser l'emplacement d'un fichier particulier, sans savoir exactement où il se trouve, alors vous aurez avantage à utiliser la commande `find` qui permet d'effectuer ce type de recherche.

Supposons que vous cherchiez à localiser dans votre **home directory** le répertoire `indentation`[11]. Une première solution consiste à le rechercher *manuellement* en vous déplaçant dans l'ensemble de votre arborescence, et en listant le contenu de chacun des répertoires dans lequel vous entrez. Si votre arborescence est dense et compliquée, cette méthode de recherche deviendra très vite fastidieuse. Or, un bon informaticien est *paresseux*[12] et il préférera utiliser et apprendre une nouvelle commande si celle-ci peut effectuer le travail à sa place. Pour cet exemple, la commande `find` est d'un grand secours. Examinons cette commande de plus près.

3.7 Exercice Placez-vous dans votre **home directory**.

11. Mais vous ne vous rappellez plus où se trouve exactement ce répertoire.
12. Ou plutôt intelligemment *paresseux*.

3.8 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[13]. Examinons ensemble la syntaxe de la commande de l'exercice précédent. Celle-ci se décompose comme suit :

`find pathname -name file -print`

et signifie :
rechercher[14] *tous les fichiers ou répertoires portant le nom* ***file***[15] *à partir du répertoire* ***pathname***[16] *et afficher le résultat de la recherche sur la sortie standard*[17].

Dans l'exercice, `pathname` est égal à '`.`', ce qui signifie *commencer la recherche à partir du répertoire courant*. Comme vous vous êtes préalablement placé dans votre **home directory**, alors la recherche s'effectue à partir de la racine de votre espace de travail personnel.

Remarque : Bien entendu la commande `find` est astreinte aux permissions de lecture des fichiers et de traversée des répertoires. En aucun cas, elle ne peut outrepasser ces droits. Cela signifie que si vous effectuez une recherche en dehors de votre espace personel de travail, et s'il existe des répertoires pour lequels vous n'êtes pas autorisé en lecture, alors la commande `find` vous renverra un message d'erreur vous signifiant qu'elle a échoué en essayant de lire un répertoire interdit pour vous.

3.9 Exercice Recherchez à partir du répertoire `/etc` tous les fichiers ou répertoires portant le nom `emacs`. Combien d'occurrences de `emacs` ont été trouvées ? 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.

3.10 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_recher`

L'exécution de la commande `find` peut prendre beaucoup de temps si vous lui demandez d'effectuer une recherche dans une arborescence relativement importante. Ainsi, pour vous permettre de *garder la main* dans votre **xterm** pendant l'exécution de la commande, il faut lancer cette commande en *tâche de fond*[18] en utilisant le symbole '`&`' à la fin de la ligne de commande. De plus, 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.

3.11 Exercice Effacez le fichier `resultats_recherche`.

3.12 Exercice Recherchez à partir du répertoire `/net/adm` tous les fichiers ou répertoires portant le nom `.*shrc` , 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*.

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

3.14 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`.

13. Pour une description exhaustive de la commande, lancez `man find`.
14. `find`
15. `-name file`
16. `pathname`
17. `-print`
18. En anglais *background*.

3.2 Un périphérique bien pratique

Vous avez 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 perdue définitivement.

Dans l'exercice utilisant la commande `find`, les messages d'erreur ne sont pas utiles, d'où l'intérêt de rediriger les messages d'erreur vers le périphérique `/dev/null`.

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

3.16 Exercice (pour ceux qui sont en avance) Recherchez toutes les occurences d'`emacs` à partir du répertoire `/opt`.

3.17 Exercice (pour ceux qui sont en avance) Affinez la recherche précédente en utilisant des options de la commande `find` pour n'obtenir que :

1) les occurences non-exécutables d'`emacs` ;
2) les occurences exécutables ;
3) les occurences exécutables d'`emacs` qui ne soient pas des répertoires.