Introduction aux systèmes d'exploitation (IS1)

TP 4 – Entrées/Sorties

Chaque processus peut être amené à manipuler des fichiers au cours de son exécution. À cette fin, il maintient une liste numérotée de 0 à `n` de descripteurs de fichiers, qui recense les fichiers ouverts en lecture ou en écriture par le processus.

Parmi ceux-ci, trois fichiers dits « fichiers standards » occupent une fonction particulière. Ce sont des fichiers seulement au sens logique, c'est à dire qu'ils se comportent comme des fichiers mais peuvent correspondre en réalité à d'autres mécanismes :

- l'entrée standard, qui correspond au descripteur de fichier n°0, où un processus peut conventionnellement recueillir des informations à traiter ;
- la sortie standard, descripteur n°1, où un processus écrit les données qu'il produit ;
- la sortie standard d'erreur, descripteur n°2, où un processus transmet les messages d'erreurs éventuels rencontrés au cours de son exécution.

Par défaut, ces trois fichiers logiques sont associés au terminal de l'utilisateur (symbolisé par le fichier `/dev/tty`), ce qui permet de lire les données fournies au clavier par l'utilisateur et d'afficher les résultats ou les messages d'erreur sur le terminal.

1 Notion de commande « filtre »

Même si tous y ont accès, certains processus n'utilisent pas les fichiers standards. Par exemple, la commande `ls` n'attend pas d'entrée de la part de l'utilisateur, mais traite des données présentes dans le système de fichier. La commande `cd`, quant à elle, travaille uniquement sur ses arguments et ne fournit en général aucune sortie.

Les processus qui traitent des données provenant de l'entrée standard et produisent un résultat sur la sortie standard sont communément appelés « filtres ». Un exemple typique de filtre est la commande `c t` lancée sans argument, qui recopie les données lues sur l'entrée standard vers la sortie standard.

Exercice 1 – Le filtre le plus simple.

1. Lancez `c t` sans argument. L'invite change de forme pour vous permettre de fournir des données.
2. Tapez quelques caractères puis frappez la touche entrée. Observez le résultat.
3. Tapez encore quelques lignes, puis pour indiquer au processus la fin des données à traiter, pressez la combinaison de touches `Ctrl-D` (aussi appelé EOF, ou end of file).
4. Que se passe-t-il si à la fin d'une ligne contenant des caractères, vous pressez `Ctrl-C` au lieu de `Ctrl-D` ?

Ceci est le comportement par défaut d'un filtre. **Attention :** presser la combinaison `Ctrl-D` depuis l'invite du shell provoque sa fermeture !

2 Redirection simple

L'un des grands avantages d'un système de type Unix est de permettre à un utilisateur averti (comme vous) de combiner entre elles à volonté un certain nombre de briques de base, à l'aide de mécanismes puissants et flexibles. L'un de ces mécanismes permet de reconfigurer les canaux d'entrée et de sortie des processus.

Si `m chin` est une commande quelconque, il est possible de rediriger son entrée standard sur le fichier fic grâce à la syntaxe `m chin <` fic (raccourci pour `0< fic`), ou de rediriger sa sortie standard vers fic grâce à la syntaxe `m chin >` fic (raccourci pour `1> fic`)[1].

Exercice 2 – Rediriger l'entrée ou la sortie d'un processus

1. Faites la liste détaillée (option `-l`) du contenu du répertoire /usr/lib. Quel est le premier fichier de ce répertoire dans l'ordre alphabétique ?
2. Redirigez la sortie de la commande précédente vers un fichier `liste`, puis affichez le contenu de ce fichier avec la commande `less` par exemple. Quel est l'avantage de cette méthode ?
3. Redirigez l'entrée standard de la commande `c t` vers le fichier `liste` et observez ce qui se produit. Comme pour un certain nombre de commandes, le même résultat peut être obtenu en passant le fichier `liste` directement en argument de la commande `c t` (sans symbole de redirection). Essayez.
4. Qu'y a-t-il dans le fichier `liste` si l'on exécute maintenant la commande : `echo` une phrase > liste (utilisée au TP2) ?
5. La syntaxe >> fic a un comportement différent. Testez-la comme dans la question précédente et concluez.
6. Sans utiliser ni `em cs` ni `kwrite`, **ajoutez** au fichier `liste` la ligne "C'était vraiment très intéressant !".
7. Qu'y a-t-il dans le fichier `liste` après l'exécution de la commande suivante :
   `(echo "Bonjour" > liste ; echo "Au revoir" > liste ;`
   `ls - l >> liste) || echo "Bonjour" > liste` (sur une seule ligne) ?
   Et si l'on remplace `||` par `&&` ?
8. On souhaite copier liste dans liste-bis sans utiliser la commande `cp`. Comment peut-on procéder ?
9. Que se passe-t-il lorsqu'on exécute la commande `c t <` fichier_ultra_important > fichier_ultra_important ? (Choisissez pour faire ce test un fichier non vide, mais pas trop important tout de même). Comment peut-on expliquer ce résultat ?

[1] Conformément à ce qui a été dit plus haut, taper `machin < /dev/tty > /dev/tty` correspond au comportement par défaut de `machin`.

3 Redirection moins simple

Exercice 3 – Erreur standard et sortie standard

Nous avons vu comment rediriger l'entrée ou la sortie standard d'un processus vers des fichiers quelconques. Pour rediriger la sortie standard d'erreur, il faut utiliser une syntaxe explicite faisant apparaître son numéro de descripteur, autrement dit la syntaxe `2>`.

1. Choisissez un nom de fichier qui n'existe pas (ici, `rEp3`) et testez la commande `ls rEp3`. Que se passe-t-il ?
2. Prédisez le résultat de la commande `ls rEp3 > toto` puis exécutez-la. Que se passe-t-il ? Que contient le fichier `toto` ?
3. Comment modifier la ligne précédente pour que le message d'erreur s'écrive dans le fichier `erreurs` ?
4. Écrivez une ligne de commande qui liste le répertoire courant et un répertoire inexistant (par exemple `rEp3`) et stocke la sortie standard dans `liste` et les erreurs dans `erreurs`. Vérifiez le résultat.
5. On souhaite maintenant qu'une commande écrive son résultat dans le fichier `result t` si elle s'exécute sans erreur et ses messages d'erreur dans le fichier `/dev/null` sinon. Testez avec une commande qui fonctionne et une autre qui produit une erreur.

   Quel est le contenu du fichier `/dev/null` ? Ce fichier spécial du système correspond à un "trou noir" qui absorbe tout ce qu'on y écrit.
6. Pour rediriger la sortie d'erreur et la sortie standard vers le même fichier fic, on utilise la syntaxe abrégée `&>` fic[2]. Redirigez les deux canaux de sortie de la commande précédente vers le fichier `liste`, affichez le contenu de ce fichier et expliquez le résultat.
7. Enfin, on veut que la commande `ls . rEp3` **ajoute** son résultat et ses messages d'erreur dans le fichier `sortie`. Quelle ligne de commande pouvez-vous taper ?

4 Tubes

Exercice 4 – Communication entre processus

La commande `t il` affiche les dix dernières lignes qui lui sont fournies sur son entrée standard. En utilisant des redirections, un fichier intermédiaire et des points virgules, écrivez une ligne de commande composée qui affiche la liste des dix derniers fichiers du répertoire `/usr/lib` mais sans produire de « déchet », c'est à dire en effaçant le fichier intermédiaire.

La commande vue à l'exercice précédent est assez peu commode à utiliser. Dans certains cas, par exemple quand les données intermédiaires sont trop volumineuses, il n'est pas possible de les stocker dans un fichier.

Pour parer à cet inconvénient, la syntaxe cmd1 | cmd2, appelée pipeline, permet de rediriger directement la sortie de cmd1 vers l'entrée de cmd2. Il est possible de l'utiliser en cascade : cmd_1 | cmd_2 . . . | cmd_n.

[2] Qui est une notation raccourcie pour > fic 2> fic, ou encore > fic 2>&1

Exercice 5 – Tubes

1. Documentez-vous sur les commandes `sort` et `grep` (que nous reverrons en détail dans un prochain TP).
2. Que fait la ligne de commande `ls /usr/bin | grep em` ?
3. Triez le contenu de votre répertoire personnel (home directory) par ordre alphabétique.
4. En une seule commande composée, cherchez dans `/bin` tous les noms de fichier contenant la lettre a et triez-les par ordre alphabétique inverse.

5 Digression cryptographique

Exercice 6 – César

Le code de César est une méthode de « chiffrement » rudimentaire : il effectue un simple décalage circulaire de lettres[3]. Ainsi, si le décalage choisi est d'une seule lettre, « abricot » sera codé par « bcsjdpu » et « Z » devient « A ».

1. La commande `tr` permet d'effectuer un codage très simplement. À l'aide de la page de manuel de cette commande, chiffrer un fichier de votre choix en utilisant des redirections.
2. On peut vouloir obtenir la version chiffrée d'un message directement, sans l'écrire préalablement dans un fichier. La redirection effectuée par << sert à cela :

```
tr ... << EOF
> Cette insigne f veur que votre cœur récl me
> Nuit à m  renommée et répugne à mon âme.
> EOF
```

effectue exactement la même opération que la commande `tr ... <MsgS nd` lorsque le fichier `MsgS nd` contient

```
Cette insigne f veur que votre cœur récl me
Nuit à m  renommée et répugne à mon âme.
```

Pour un message d'une seule ligne, on peut encore utiliser directement la syntaxe <<< :

```
tr ... <<< "Cette insigne f veur..."
```

Utilisez cette commande pour envoyer un message secret à votre voisin et déchiffrez la réponse. Inventez éventuellement un codage moins régulier que César pour déjouer les autres voisins...

[3] Le décalage de 13 lettres exactement, appelé `ROT13`, est assez souvent utilisé (car chiffrage et déchiffrage sont identiques).