TP de Système n° 11

I) Démon d'impression

Exercice 1 :

Écrire une commande de nom `lpd` chargée de simuler le démon d'impression d'un système. Cette commande lira dans tube nommé (de nom `/tmp/printer0`) la référence d'un fichier (sous la forme d'une simple chaîne de caractère) à imprimer, recopiera alors le contenu du fichier correspondant dans un périphérique (de nom `/dev/lp0`[1]) puis recommencera. On fera ce qu'il faut pour qu'en cas d'existence d'un exemplaire de ce démon, toute tentative de lancement de l'exécution d'un autre provoque la terminaison immédiate du second.

Exercice 2 :

Quels droits/propriétaires doit-on positionner sur les fichiers `/tmp/printer0`, et `/dev/lp0`, de sorte que personne ne puisse écrire directement sur l'imprimante, personne d'autre que la commande `lpd` ne puisse extraire les requêtes et que ce démon puisse lire effectivement les fichiers correspondant aux requêtes ?

Exercice 3 :

Écrire une commande de nom `lpr` chargée de simuler la commande de requête d'impression d'un système. Cette commande prendra en paramètre la référence d'un fichier et écrira la référence absolue correspondante dans le tube correspondant du démon lpd.

Exercice 4 :

On souhaite désormais modifier le démon de sorte qu'il puisse gérer des impressions sur deux imprimantes, c'est-à-dire imprimer les fichiers correspondant aux requêtes de `/dev/printer0` sur `/dev/lp0`, et ceux de `/dev/printer1` sur `dev/lp1`. Quels problèmes cela pose t-il ? Quelle(s) solution(s) entrevoyez-vous ?

Exercice 5 :

Écrivez le code correspondant à vos réponses à l'exercice 4.

II) Mini-shell

Le but de ce TP est de programmer un petit sous-ensemble des fonctionnalités d'un *shell* comme `sh` ou `bash`. On l'appellera `dsh` (Diderot Shell). C'est un projet en plusieurs fichiers source ; vous écrirez tout au long du TP un Makefile qui compile l'ensemble des fichiers source en un executable `dsh`.

1. Sur les machines du TP, il se peut que vous n'ayez pas les permissions nécessaires. De plus, même si vous les aviez, vous ne souhaitez pas gaspiller de papier. Ainsi, il est suggéré de créer un tube nommé que vous utiliserez à la place de `/dev/lp0`. Vous pourrez utiliser la commande `tail -f` dans un terminal pour surveiller ce qui "s'imprime" sur ce faux périphérique.

Exercice 6 : main.c

1. Implémenter la boucle principale de `dsh`, qui lit des commandes sur une ligne (`fgets(3)`) et les execute. Une commande est soit `exit` pour quitter le shell, soit un nom d'executable dans le `PATH`, suivi de ses arguments séparés par des espaces[2]. Pour cela on écrira une fonction `int execute(char *commande)` qui découpe `commande` en un nom de commande et ses arguments, puis l'execute.

Exercice 7 : builtins.c

Étendre l'ensemble des commandes intégrées au *shell* avec `cd`, `pwd`. Les fonctions (même triviales) implémentant ces commandes seront dans un fichier à part. Faire afficher à l'invite de commande le nom du répertoire courant et le code de retour de la dernière commande.

Rappel : ces fonctions ont déjà été programmées dans les TD/TP précédents...

Exercice 8 : combinators.c

Implémenter les fonctions de combinaison de commandes :

```
int and (char *c1, char **argv1, char *c2, char **argv2)
int or  (char *c1, char **argv1, char *c2, char **argv2)
int pipe(char *c1, char **argv1, char *c2, char **argv2)
```

Étendre la boucle principale du *shell* pour accepter des commandes simples de la forme :

```
gcc exo1.c -o exo1 & ./exo1
grep calais fichier.txt % echo pas_de_calais
cat fichier.txt  sort | less
```

utilisant respectivement `and`, `or` et `pipe`[3]. Pour simplifier, une ligne de commande ne pourra pas mélanger les combinateurs. Par contre elle pourra contenir autant de combinateurs du même genre que voulu.

Exercice 9 : builtins.c

S'il vous reste du temps, implémenter les commandes intégrées `pushd`, `popd` et `dirs`. Ces commandes maintiennent une liste de répertoires dans une pile (pour éviter d'avoir des très longues lignes de `cd` à écrire) ; `pushd` empile le répertoire courant dans cette pile, `popd` en dépile un et va dedans (`man builtins`). Vous pourrez vous appuyer sur votre implémentation des listes chaînées du TD2.

2. Une astuce bien pratique : Dans tout ce TP, `strtok(3)` vous facilitera la vie pour analyser la suite d'arguments, découper une suite de commandes etc.

3. Remarquez que l'on n'utilise pas la syntaxe standard des shells ici (`&&`, `||` et `|`). C'est pour simplifier l'analyse grammaticale.