Info4 – TP-1 Système
Interprète d'un mini-langage de commande

On se propose dans ce TP de réaliser un interprète pour un langage de commande simplifié, appelé communément « shell ». L'objectif est d'une part de comprendre la structure d'un tel interprète et d'autre part d'apprendre à utiliser quelques appels systèmes importants, typiquement ceux qui concernent la gestion des processus, les pipes et la redéfinition des fichiers standards d'entrée et de sortie.

Le Langage de Commande

Une commande est une suite de mots séparés par un ou plusieurs espaces. Le premier mot d'une commande est le nom de la commande à exécuter et les mots suivants sont les arguments de la commande. Chaque commande doit s'exécuter dans un processus autonome, fils du processus shell. Le shell doit attendre la fin de l'exécution d'une commande. Les appels systèmes wait et waitpid permettent de réaliser cette attente et de récupérer la valeur de retour de la commande (status).

Une séquence est une suite de commandes séparées par le délimiteur «|» ; la sortie standard d'une commande doit alors être connectée à l'entrée standard de la commande suivante. Une telle connexion s'appelle en Unix un pipe. La valeur de retour d'une séquence est la valeur de retour de la dernière commande de la séquence.

L'entrée ou la sortie d'une commande (ou l'entrée de la première commande d'une séquence ou la sortie de la dernière commande d'une séquence) peuvent être redirigées vers des fichiers. On utilise pour cela les notations usuelles d'Unix :

< toto associe le fichier toto à l'entrée standard
>lulu associe le fichier lulu à la sortie standard.

Exemples de commandes et de séquences, avec ou sans redirections.

```
ls -a
ls -a >toto
ls jacques | grep en
ls < fichier1 | grep en > fichier2
```

Ce qui vous est fournit

Vous avez votre cours, ce document, et un petit programme qui vous montre comment lire l'entrée standard, ligne à ligne, et comment découper une ligne en mots, en suivant la syntaxe de notre langage de commande. Voir le fichier « readline.c »

Travail Demandé

Ecrire un shell qui fonctionne. Une difficulté de la programmation d'un shell est la détection de toutes les erreurs commises par l'utilisateur ou par les commandes externes lancées: un shell ne doit **jamais** « planter ». Cela veut dire :

- Résister à n'importe quelle ligne de commande erronée.
- Résister aux crash des processus fils du shell.
- Ne pas laisser trainer des zombies.
- Ne pas laisser trainer des fichiers ou des pipes ouvert.

Lors de votre implémentation, vous allez utiliser les fonctions suivantes :

fork, wait, execve, pipe, dup2
open,read,write,close

1) Commandes internes simples :

Les commandes internes sont implémentées par le code de l'interprèteur. Nous en ferons que quelques unes :

cd	pour change directory
pwd	pour print working directory

Tout shell maintient des variables d'environnement sous forme : nom=valeur, où la valeur est une string. Il est possible d'assigner une variable « PATH », comme ceci :

set **PATH ''/bin:/sbin/''**

Attention, la valeur ne peut contenir les caractères spéciaux : ' ', '\t', '<', '>', '|', '', '='
Il est aussi possible d'afficher cette variable, comme ceci :

print PATH

Enfin, il est possible d'afficher toutes les variables, comme ceci :

print

Votre shell hérite de l'environnement de son shell père, via l'argument envp de la fonction « main » :

void **main**(int argc, char** argv, char**envp) ;

et il passe son environement aux commandes externes qu'il lance.

2) Commandes externes

Il ne s'agit pas d'implémenter ces commandes, mais de les lancer via **fork** et **execve**.
Vous pouvez tester avec les commandes « ls », « cat », « more », et « grep »

Attention, il ne faut pas laisser trainer des zombies partout. Il faut donc utiliser « wait ».

3) Commandes en background

Une commande peut être lancer pour s'exécuter en arrière plan :

ls &

Ce qui est facile à tester, mais n'a pas beaucoup d'intérêt.
La prise en édition d'un fichier est un bon example :

gedit toto &

Une longue copie récursive peu aussi être un bon example :

cp -r toto titi &

4) Les pipes
Il s'agit là de savoir rediriger les entrées et sorties standards (stdin et stdout).
Comme par example avec les commandes ci-dessous :

```
cat /etc/passwd | grep root
ls -a ~ | grep .bash
```

5) Les redirections :
Il s'agit là de savoir rediriger les entrées et sorties standards (stdin et stdout).
Comme par example avec les commandes ci-dessous :

```
ls -al > toto
cat toto >> titi
```