POLYTECH

RICM 2ème année

TP SYSTÈME N°1 : INTERPRÈTE D'UN MINI-LANGAGE DE COMMANDE

1. INTRODUCTION

Un système d'exploitation fournit à ses utilisateurs une interface de programmation, ensemble de services dont on peut déclencher l'exécution par des « appels système». Ces appels système sont lancés par l'intermédiaire de sous-programmes de bibliothèque qui peuvent être appelés depuis un programme quelconque, par exemple un programme en C. Un autre mode de déclenchement consiste à utiliser un interprète d'un langage de commande. C'est alors l'interprète qui transforme une commande tapée sous forme textuelle en un ou plusieurs appels système. Dans le système Unix, l'interprète du langage de commande est appelé un shell.

Exemple. Pour créer un nouveau processus en Unix, on dispose d'un appel système fork() qui clone le processus qui l'exécute et un appel système execvp(…) qui permet de modifier l'espace mémoire d'un processus pour lancer l'exécution d'un nouveau programme. Lorsqu'un utilisateur frappe une commande, le shell interprète le premier mot de la commande comme le nom d'un fichier contenant un programme à exécuter et lance l'exécution de ce programme dans un nouveau processus en appelant successivement les appels systèmes fork et execvp.

On se propose dans ce TP de réaliser un interprète pour un langage de commande simplifié. L'objectif est d'une part de comprendre la structure d'un shell 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.

2. 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
```

3. Travail demandé

3.1 Analyse des lignes frappées au clavier

La procédure de lecture d'une ligne et son analyse vous sont fournies (fichiers readcmd.h et readcmd.c). Un programme test.c qui utilise readcmd est également joint pour vous aider à comprendre ce qui est renvoyé par readcmd().

La fonction readcmd() renvoie un pointeur vers une structure struct cmdline dont les champs sont les suivants :

char *err message d'erreur à afficher, null sinon

char * in nom du fichier pour rediriger l'entrée, null si pas de redirection

char *out nom du fichier pour rediriger la sortie, null sinon

`char *** seq` une commande est un tableau de mots (`char **`) dont le dernier terme est un pointeur `null` ; une séquence est un tableau de commandes (`char ***`) dont le dernier terme est un pointeur `null`.

La complexité de la structure n'est qu'apparente ; vous vous apercevrez lors de la réalisation du programme qu'elle est très bien adaptée, tout particulièrement pour les appels à `execvp`.

3.2 Réalisation

Il vous est demandé de programmer un interprète du langage de commande défini en 2, en utilisant `readcmd()` pour la lecture des lignes. Nous vous suggérons d'organiser la réalisation comme suit :

3.2.1 Compréhension de la structure `cmdline` et des résultats de `readcmd`.

3.2.2 commande simple

3.2.3 commande avec redirection entrée ou sortie

3.2.4 pipe entre deux commandes

3.2.5 suite de pipes avec redirection

Une difficulté de la programmation d'un shell est la détection de toutes les erreurs commises par l'utilisateur : un shell ne doit pas « se planter ».

4. Pour aller plus loin (facultatif)

4.1 Exécution de commandes en arrière-plan

Lorsqu'une commande est terminée par le caractère `&`, elle s'exécute en tâche de fond, c'est à dire que le shell crée le processus destiné à exécuter la commande, mais n'attend pas sa terminaison.

Remarque. La détection du caractère `&` dans une ligne de commande implique une modification de la fonction `readcmd()` et de la structure `cmdline`.