Université Joseph Fourier, Grenoble
Licence d'Informatique L3

Introduction aux Systèmes et Réseaux

TP n°4 : Réalisation d'un mini-*shell*

Ce TP (qui est prolongé par des Apnées) consiste à réaliser un mini-*shell* pour Unix.

1 Introduction

Un systeme d'exploitation fournit a ses utilisateurs une interface de programmation comprenant des fonctions de création de processus, de manipulation des E/S, de travail avec les fichiers, etc. Ces *appels système* peuvent être faits dans un programme C quelconque (en utilisant les fonctions de bibliothèque standard, p.ex. `fork`, `open`, `dup`, ...) ou alors en ligne de commande. Dans le deuxième cas, un interprête de langage de commande qui transforme une commande tapee sous forme textuelle en un ou plusieurs appels systeme. Dans le systeme Unix, l'interprete du langage de commande est appele un *shell*. Des exemples de shell sont `tcsh`,`bash`, `ksh`, etc.

2 Le langage de commande (rappels et compléments)

Une commande est une suite de mots separes par un ou plusieurs espaces. Le premier mot d'une commande est le nom de la commande a executer et les mots suivants sont les arguments de la commande. *Chaque commande doit s'executer dans un processus autonome, fils du processus shell. Le shell doit attendre la fin de l'execution d'une commande.* Les appels systemes `wait` et `waitpid` permettent de realiser cette attente et de recuperer la valeur de retour de la commande (`status`).

Une sequence est une suite de commandes separees par le delimiteur "—" ; la sortie standard d'une commande doit alors etre connectee a l'entree standard de la commande suivante. Une telle connexion s'appelle en Unix un tube. La valeur de retour d'une sequence est la valeur de retour de la derniere commande de la sequence.

L'entree ou la sortie d'une commande (ou l'entree de la premiere commande d'une sequence ou la sortie de la derniere commande d'une sequence) peuvent etre redirigees vers des fichiers. On utilise pour cela les notations usuelles d'Unix :

— `< toto` : redirige l'entree standard vers le fichier `toto`
— `> lulu` : redirige la sortie standard vers le fichier `lulu`

Voici quelques exemple de séquences de commandes, avec ou sans redirection :

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

3 Travail à réaliser

Le but du mini-projet est de realiser un interprete pour un langage de commande simplifie. L'objectif est d'une part de comprendre la structure d'un shell et d'autre part d'apprendre à utiliser quelques appels systemes importants, typiquement ceux qui concernent la gestion des processus, les tubes et la redefinition des fichiers standards d'entree et de sortie.

3.1 Analyse des lignes frappees au clavier

La procedure de lecture d'une ligne et son analyse vous sont fournies (fichiers `readcmd.h` et `readcmd.c`). Un programme `tst.c` est egalement joint pour vous aider a comprendre ce qui est renvoye par `readcmd()`.

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

— `char *err` : message d'erreur a afficher, null sinon
— `char * in` : nom du fichier pour rediriger l'entree, 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 sequence est un tableau de commandes (`char ***`) dont le dernier terme est un pointeur null.

La complexite de la structure n'est qu'apparente ; vous vous apercevrez lors de la realisation du programme qu'elle est tres bien adaptee, tout particulierement pour les appels a `execvp`.

3.2 Organisation du travail

Il vous est demandé de programmé un shell qui peut interpéter les commandes écrites avec le langage défini dans 2. Vous devez passer par/réaliser les étapes suivantes :

1. *Comprehension de la structure cmdline et des resultats de readcmd.*
   Lisez attentivement les fichiers fournis et appropriez vous le code. N'hesitez pas à poser des questions sur la logique ou sur le langage C.
2. *Interprétation de commande simple*
   Pour cette étape, vous devez utiliser vos connaissances sur les fonctions de création de processus, notamment la famille de la commande `exec`, que nous avons vu en TP1. Se référer aux points techniques à la fin du sujet et aux compléments fournis.
3. *Commande avec redirection d'entree ou de sortie*
   Pour cette étape, vous devez utiliser vos connaissances sur les fonctions de redirection d'entrée/sortie, notamment `dup` et `dup2`.
4. *Séquence de commandes composée de deux commandes reliées par un tube*
   Pour cette étape, vous devez utiliser vos connaissances sur la gestion de tubes,, notamment `pipe`.
5. *Séquence de commandes composée de plusieurs commandes et plusieurs tubes*
   Pour cette étape vous réutiliserez les points traités précédemment.

4 Pour aller plus loin

4.1 Execution de commandes en arriere-plan

Lorsqu'une commande est terminee par le caractere &, elle s'execute en tache de fond, c'est a dire que le shell cree le processus destine a executer la commande, mais n'attend pas sa terminaison.

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

4.2 Changer l'état du processus en premier plan

Implémenter la gestion de `control-c` et `control-z` pour qu'ils envoient respectivement un signal SIGINT et un signal SIGTSTP au(x) processus de premier plan.

4.3 Gestion des zombis

Une tâche lancée en arrière plan ne doit pas être attendue par le shell. Nénamoins, le shell doit ramasser les processus terminés pour éviyer la prolifération de zombis. Pour cela implémenter un traitant de `SIGCHLD`. Utiliser les options suivantes de la primitive waitpid : `waitpid(-1, &status, WNOHANG|WUNTRACED)`

4.4 Commande integrée `jobs`

Les commandes exécutées en arrière-plan s'appellent des jobs. Un job peut être désigné par le PID du processus qui l'exécute (exemple 14567) ou par son numéro de job précédé de % (exemple %3). Les numéros de jobs sont des entiers positifs, attribués à partir de 1.

Implémenter la commande `jobs` qui donne la liste des commandes lancées.

4.5 Agir sur les commandes en arrière plan

Implémenter les commandes `fg`, `bg` et `stop` qui agissent sur les jobs d'un shell et respectivement mettent un job en premier plan, en arrière plan ou l'arrêtent.

5 Présentation des résultats

Chaque binôme rendra le code source *commenté* des programmes réalisés et un bref compte-rendu présentant : (a) les principales réalisations et (b) une description des tests effectués.

Une démonstration des réalisations sera organisée en fin de semestre (détails indiqués en temps utile).