TP3 Shell et couteau suisse

nicolas.ollinger@univ-orleans.fr

27 octobre 2017

> ***Contrôle continu :*** *Sont à rendre selon les modalités détaillées sur la page celene, une archive contenant le couteau suisse développé dans la deuxième partie de cette fiche sous la forme d'une archive* `toolbox.tar.gz`*.*

1 Shell élémentaire

Le langage C est un langage de programmation très permissif, qui permet d'écrire facilement des programmes illisibles et incompréhensibles tant au niveau syntaxique qu'au niveau sémantique. Cependant, il est moins aisé d'écrire des programmes qui soient très illisibles, imbriquant plusieurs niveaux de difficulté. Ainsi, le programme C suivant met en oeuvre un mini-shell incluant la plupart des fonctionnalités que nous allons développer dans cette fiche de TP (à l'exception de la gestion des variables d'environnement et des scripts) :

```
int exit(int);
#define D ,close(
char*c,q[512],m[256],*v[99],**u,*i[3];int f[2],p;main(){for(m[m[60]=m[62]=
32]=m[*m=124[m]=9]=6;e(-8),gets(1+(c=q))||exit(0);r(0,0))for(;*++c;);}
r(t,o){*i=i[2]=0;for(u=v+98;m[*--c]^9;m[*c]&32?i[*c&2]=
*u,u-v^98&&++u:3)if(!m[*c]){for(*++c=0;!m[*--c];);*--u=
++c;}u-v^98?strcmp(*u,"cd")?*c?pipe(f),o=f[1]:1,(p=fork())?e(p),o? r(o,0)D
o)D*f):4,wait(0):(o?dup2(*f,0)D*f)D o):*i?1 D 0),e(open(*i,0)):5,t?
dup2(t,1)D t):i[2]?9 D 1),e(creat(i[2],438)):2,e(execvp(*u,u))):
e(chdir(u[1])*2):6;} e(x){x<0?write(2,"?\n$ "-x/4,2),x+1||exit(1):5;}
```

— B. Rakitzis et S. Dorward, IOCCC 1990

Voici quelques conseils, en vrac, afin de faciliter votre travail :

- lisez attentivement les consignes et respectez les ;
- utilisez un éditeur de texte raisonnable (par exemple, **emacs**) ;
- indentez votre code source ;
- commentez vos en-têtes de fonctions ;
- la documentation est dans le manuel en ligne : *man(1)* (c'est-à-dire **man 1 man**) ;
- utilisez les options de détection d'erreurs du compilateur C ;
- utilisez un Makefile pour compiler avec **make**, en voici un minimal :
  ```
  CC=gcc
  CFLAGS=-W -Wall -ansi -g
  ```
- lisez les messages d'erreur et les avertissements du compilateur ;
- utilisez les outils de débuggage comme **gdb** ;
- testez les valeurs de retour des fonctions.

Exercice 1 (Analyse syntaxique)**.** Écrire une fonction `int parse_line(char *s);` qui, à partir de la chaîne de caractères `s`, terminée par `'\0'`, analyse cette chaîne et appelle la fonction `int simple_cmd(char *argv[]);` chargée de l'interpréter. La fonction `simple_cmd` se contente d'afficher ce qu'on lui passe en argument et retourne `0`. Dans le cas où le nom de la commande passée en paramètre est `exit`, le programme est arrêté. Une ligne de commandes est une suite de mots-clefs `commande argument1 ... argumentn` qu'il faut passer à `simple_cmd`. Elle peut contenir des commentaires, le caractère `#` indiquant que tout ce qui suit est un commentaire et doit donc être ignoré. La fonction `main` affiche une invite de commande `$ ` et attend qu'une ligne soit tapée avant de l'envoyer à `parse_line`.

Références : *fgets(3), strpbrk(3), strcmp(3).*

Exercice 2 (Exécution de commandes simples)**.** Modifier la fonction `simple_cmd` afin d'exécuter la commande passée en paramètre, d'attendre sa terminaison, puis de retourner dans la boucle principale du shell. Dans le cas où la commande invoquée est `cd`, changer de répertoire sans faire d'appel à `fork()`. Pourquoi faut-il exécuter cette commande à l'intérieur du shell ?

Références : *fork(2), execvp(3), wait(2), chdir(2).*

Exercice 3 (Variables d'environnement)**.** Modifier la fonction `parse_line` afin d'ajouter un nouveau type de ligne de commandes : les lignes de la forme `chaine=valeur` qui servent à mettre la valeur `valeur` dans la variable d'environnement `chaine`. Faites en sorte que les occurrences de mots de la forme `$chaine` soient remplacées par la valeur de la variable d'environnement `chaine`.

Références : *getenv(3), setenv(3), putenv(3).*

Exercice 4 (Fichiers scripts)**.** Modifier la fonction `main` afin d'ajouter l'exécution de scripts de commandes. Lorsque le programme est invoqué avec un nom de fichier en argument, ce fichier est ouvert et les commandes qu'il contient sont exécutées, le programme s'arrête à la fin du fichier ou lorsqu'il rencontre la commande `exit`. Dans le cas d'un script, le shell n'affiche pas d'invite de commande.

Références : *fopen(3), fclose(3).*

Exercice 5 (Redirections d'entrées-sorties)**.** Modifier la fonction `parse_line` afin d'ajouter la gestion des redirections vers des fichiers en entrée `<` et en sortie `>`. Pour exécuter la commande, on utilisera la fonction `int redir_cmd(char *argv[], char *in, char *out);` (à la place de `simple_cmd`) qui reçoit en paramètre les noms des fichiers à ouvrir en entrée et en sortie (`NULL` si on utilise le fichier standard).

Références : *open(2), dup2(2), close(2).*

Exercice 6 (Gestion des filtres)**.** Modifier la fonction `parse_line` afin d'ajouter la gestion des filtres. Une ligne de commande est alors une suite de commandes simples séparées par des pipes `|`. La première commande peut comporter une redirection en entrée et la dernière une redirection en sortie. Pour exécuter l'ensemble des commandes on utilisera trois fonctions (et éventuellement des variables globales) :

- `start_cmd(char *argv[], char *in);` pour la première commande ;
- `next_cmd(char *argv[]);` pour les commandes intermédiaires ;
- `last_cmd(char *argv[], char *out);` pour la dernière commande.

Références : *dup2(2), pipe(2).*

2 Couteau suisse

Avec un seul utilitaire et en utilisant des liens, il est possible d'obtenir un grand nombre de fonctionnalités différentes. L'objectif de cette partie est d'étendre le squelette qui est présent dans l'archive `/home/ollinger/toolbox.tar.gz`. Pour le moment, cette archive ne possède que la fonctionnalité `echo`.

Dans un premier temps, récupérer puis décompresser l'archive. Le contenu se présente de la façon suivante :

- un dossier `man` contenant les page de manuel (à utiliser avec `man -Mman`) ;
- un dossier `bin` contenant les exécutables ;
- un programme `txt2tags` convertissant le format `t2t` en page de man ;
- un exemple de fichier `t2t` (`manpage.t2t`) ;
- un fichier `toolbox.h` contenant **toutes** les inclusions de bibliothèques ;
- un fichier principal `toolbox.c` ;
- un fichier `Makefile`.

Pour chaque fonctionnalité `toto` à ajouter à cette toolbox, il faut créer le fichier `toto.c` contenant une fonction `int main_toto(int argc, char *argv[])`. Ce fichier ne doit inclure que le fichier `toolbox.h` (ajoutez les `#include` nécessaires dans `toolbox.h`). Une fois ce ficher créé, il reste encore à ajouter cette fonction dans la liste des fonctionnalités de `toolbox.c` et à modifier le `Makefile`. Pour finir, il est demandé de réaliser une courte page de manuel sous forme d'un fichier `.t2t` et de le rajouter à la variable `MANS` du `Makefile` pour qu'il génère la page de manuel correspondante. Pour toute ces opérations, on pourra s'inspirer de ce qui est fait avec la fonctionnalité `echo` déjà fournie.

Exercice 7 (Intégration du shell)**.** Ajouter à la toolbox la fonctionnalité `sh` en utilisant le shell réalisé dans la partie précédente.

Exercice 8. Ajouter les fonctionnalités de la liste suivantes (on pourra, au besoin, s'appuyer sur les TPs précédents) :

- `mkdir` qui crée le répertoire donné en argument ;
- `rmdir` qui supprime le répertoire donné un argument si celui-ci est vide ;
- `link` qui créer un lien physique vers le fichier donné en premier argument sous le nom donné en deuxième argument ;
- `ln` qui créer un lien physique ou un lien symbolique (`-s`) vers le fichier donné en premier argument sous le nom donné en deuxième argument ;
- `rm` qui supprime le liens physique donné en argument ;
- `ls` qui liste le contenu du répertoire donné en argument ou du répertoire courant si aucun argument n'est présent ;
- `time` qui chronomètre le temps d'exécution du programme passé en paramètre ainsi que son temps d'activité utilisateur et noyau ;
- `cp` qui copie le contenu du fichier donné en premier argument dans le fichier donné un deuxième argument ;
- `mv` qui déplace le fichier donné un premier argument vers celui donné un deuxième argument ;
- `cat` qui affiche la concaténation des fichiers passés en argument ;
- `true` qui renvoie la valeur 0 ;
- `false` qui renvoie toujours une valeur non nulle ;
- `echo` qui affiche les arguments séparés par des espaces.