

TP noté: programmation d'un shell

1 Introduction

Dans ce TP nous allons implanter un “mini-shell”. Vous avez 4 séances pour travailler sur votre implantation. Lors de la dernière séance vous devrez répondre à des questions posées binôme par binôme. Vous devez déposer votre code sur CELENE avant le 17/01/2016 à 23h59. Pour rappel, la note de ce TP correspond à 50% de la note finale du cours.

2 Fonctionnalités du shell

Fonctionnalités souhaitées :

- Exécution de commandes avec des pipes
- Exécution de commandes avec des redirection d'entrée / sortie
- Un chien de garde (à chaque exécution d'une commande, le shell met fin aux processus générés au bout de 5 secondes si celle-ci est sans réponse).

Fonctionnalités bonus :

- Auto completion
- Historique des commandes

Typiquement votre shell doit être capable d'exécuter correctement une commande de type :

```
cat < /var/log/messages | grep ACPI | wc -l >> truc.txt
```

3 Décomposition d'une commande

Une commande est séparée en un certain nombre de membres, délimités par des pipes (et le début et la fin de la commande).

Chaque membre comporte une ou plusieurs redirections d'entrées sorties :

- `cmd < f` est équivalent à un `cat f | cmd`. Note : un membre comportant un `<` ne peut pas être précédé d'un autre membre.
- `cmd > f` redirige la sortie standard de `cmd` vers un fichier `f`, qui est écrasé. Note : un membre comportant un `>` ne peut pas être suivi d'un autre membre.
- `cmd >> f` est identique au symbole précédent, mais l'écriture dans le fichier sur fait à la fin de celui-ci. Il n'est pas écrasé.
- `cmd 2>f` et `cmd 2>>f` sont identiques aux deux précédents, à cela près qu'ils portent sur la sortie d'erreur.

L'exécution d'une commande se décompose ainsi :

1. Séparer les différents membres
2. Regarder si les membres comportent des redirections d'entrée sortie
3. Effectuer le bon nombre de *fork*
4. Rediriger correctement STDIN, STDOUT et STDERR
5. Effectuer les *exec*
6. Attendre la fin des fils

Il est nécessaire et important de définir une structure adaptée pour stocker une commande. La structure décrite ci-dessous devra donc être utilisée.

Listing 1 – command struct

```
typedef struct command {
    //the command originally inputed by the user
    char *init_cmd;

    //number of members
    unsigned int nb_cmd_members;

    //each position holds a command member
    char **cmd_members;

    //cmd_members_args[i][j] holds the jth argument of the ith member
    char ***cmd_members_args;

    //number of arguments per member
    unsigned int *nb_members_args;

    //the path to the redirection file
    char ***redirection;

    //the redirecction type (append vs. override)
    int **redirection_type;
} cmd;
```

en gardant l'exemple donné précédemment, cette structure doit être initialisée ainsi :

Listing 2 – exemple

```
init_cmd="cat<_/var/log/messages_|_grep_ACPI_|_wc_-l_>>_truc.txt"
nb_cmd_members=3
cmd_members[0]="cat<_/var/log/messages"
cmd_members[1]="grep_ACPI"
cmd_members[2]="wc_-l_>>_truc.txt"
cmd_members_args[0][0]="cat"
cmd_members_args[0][1]=NULL
cmd_members_args[1][0]="grep"
cmd_members_args[1][1]="ACPI"
cmd_members_args[1][2]=NULL
cmd_members_args[2][0]="wc"
cmd_members_args[2][1]="-l"
```

```
cmd_members_args[2][2]=NULL
nb_members_args[0]=1
nb_members_args[1]=2
nb_members_args[2]=2
redirection[0][STDIN]="/var/log/messages"
redirection[0][STDOUT]=NULL
redirection[0][STDERR]=NULL
redirection[1][STDIN]=NULL
redirection[1][STDOUT]=NULL
redirection[1][STDERR]=NULL
redirection[2][STDIN]=NULL
redirection[2][STDOUT]="truc.txt"
redirection[2][STDERR]=NULL
redirection_type[2][STDOUT]=APPEND
```

Vous mettrez la définition de cette structure ainsi que les prototypes des fonctions utiles à sa manipulation dans un fichier `cmd.h`. Ces fonctions peuvent être :

Listing 3 – Exemple de fonctions en `cmd.h`

```
//Prints the contents of members_args to the console
void print_members_args(cmd *c);

//Frees the memory allocated to store member arguments
void free_members_args(cmd *c);

//Prints the contents of members to the console
void print_members(cmd *c);

//Frees the memory allocated to store member information
void free_members(cmd *c);

//Prints the redirection information for member i
void print_redirection(cmd *c, int i);

//Frees the memory allocated to store redirection info
void free_redirection(cmd *c);

//Remplit les champs cmd_args et nb_args_membres
void parse_members_args(cmd *c);

//Remplit les champs initial_cmd, membres_cmd et nb_membres
void parse_members(char *s, cmd *c);

//Remplit les champs redir et type_redir
void parse_redirection(unsigned int i, cmd *c);
```

Une première version de `cmd.h` est disponible sur CELENE.

Créez une fonction `void exec_command(cmd c)`, dans un fichier `shell_fct.c`, qui prend une commande dûment initialisée et qui effectue la création des pipes, les fork et les execs correspondants. Étant donné le formatage des arguments, il est judicieux d'utiliser l'appel `execvp`.

Un *makefile* vous ait également proposé pour compiler votre projet.

Pour la gestion des entrées de l'utilisateur, je vous conseille de regarder du côté de la librairie GNU/Readline (<https://cnswww.cns.cwru.edu/php/chet/readline/rltop.html>), bien que cela ne soit pas une obligation.