Système d'Exploitation
Projet: Mini-Shell

Etienne PENAULT

Semestre 3

Sommaire

Introduction

Le langage Shell est le premier langage de programmation créé sous UNIX par Steve BOURNE. Il est présent actuellement sur toutes les distributions Linux. Le Shell est utilisé par l'utilisateur pour intéragir avec le système via un terminal, ou exécuter des actions écrites dans un script (.sh). Dans ce projet, nous nous focaliserons sur cette première utilisation, celle dans un terminal et allons tenter d'arriver à un résultat similaire.

Mode D'Emploi

"./mini-shell" pour lancer le programme.
"make" pour compiler.
"make clean" pour nettoyer les traces de compilations.

Les fonctionnalités sont:

- Affichage du répertoire courant devant notre prompt
- Affichage du nom de l'utilisateur et de la machine devant le prompt
- ";" pour marquer la fin d'une commande
- "cd" pour changer de répertoire
- "&&" pour lancer une commande si la précédente a réussi
- "&" pour lancer une commande en arrière plan
- Les redirections (">", ">>", "&>", "2>", "2>>", "2>&1")
- "|" qui gère seulement ce qui le précéde
- "author" ou "creator" pour l'identité du réalisateur du projet
- "man mini" pour le manuel
- "exit" pour sortir de notre Mini-Shell

Fonctionnement

Le fonctionnement de ce programme repose sur plusieurs boucles. Tout d'abords, nous avons celle du main qui permet d'actualiser le dossier courant, ainsi que les découpages (par exemples celui de ';'). Ensuite, dans cette boucle, nous avons une sous-boucle où nous traitons chaque commandes précedemment découpées. Ici, nous appelons la fonction "lanceurdefonction()" qui va traiter, cas par cas, les commandes que nous pouvons retrouvé, dans ce qui découle du découpage avec ';', c'est ici qu'est le coeur de notre programme.

Une fois dans cette fonction, nous allons analyser si la commande passée en argument (par le biais du tableau "mot"), comporte les symboles d'une commande en particulier. Si c'est le cas, on la traite.

Pour chaque type de commande, nous retournons une valeur spécifique. Cela nous permet, dans la boucle du main, de vérifier si il nous reste une action à faire selon le type de commande, que nous ne pourrions pas faire au sein de la fonction "lanceurdefonction()" (Par exemples, si nous avons à faire à la commande "cd", nous allons retourner 3, et nous allons donc appliquer l'actualisation du chemin courant uniquement si la fonction a retourné 3.).

Nous pouvons aussi noter de la récursivité dans la fonction "lanceurdefonction()", en cas de création de processus enfants (cf: "&&" par exemple).

Exemples

Voici un exmple d'exécution de notre Mini-Shell:

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~ MINI-SHELL ~~~ ETIENNE PENAULT ~~~ SYSTEME D'EXPLOITATION ~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

[etienne@Knuckles Projet Mini Shell]% cd ..
```

```
[etienne@Knuckles OS]% cd ..
[etienne@Knuckles S3]% ls
'Algorithme et structure de données'    'Programation Impérative 2'
"Fondement de l'intelligence artificielle"  'Programation orienté objet'
 OS
[etienne@Knuckles S3]% cd OS
[etienne@Knuckles OS]% echo a; echo b
a
b
[etienne@Knuckles OS]% ls TP9 && echo ca existe !
ls: cannot access 'TP9': No such file or directory
[etienne@Knuckles OS]% ls tp9 && echo ca existe !
a.out  main.c  sys.h
ca existe !
[etienne@Knuckles OS]% echo test du et && sleep 4 & ;echo On n'attend
test du et                                                          pas !
On n'attend pas !
[etienne@Knuckles OS]% echo test > toto
[etienne@Knuckles OS]% cat toto
test
[etienne@Knuckles OS]% echo ca rajoute >> toto
[etienne@Knuckles OS]% cat toto
test
ca rajoute
[etienne@Knuckles OS]% ls fichier_non_existant &> toto
[etienne@Knuckles OS]% cat toto
ls: cannot access 'fichier_non_existant': No such file or directory
[etienne@Knuckles OS]% ls &> toto
[etienne@Knuckles OS]% cat toto
a.out
fork.c
Projet Mini Shell
Projet Mini Shell.tar.gz
support_JM.pdf
TD3
TD4
toto
TP
```

```
TP5
TP7
TP8
tp9
[etienne@Knuckles OS]% ls fichierQuiNExistePas 2> toto
[etienne@Knuckles OS]% cat toto
ls: cannot access 'fichierQuiNExistePas': No such file or directory
[etienne@Knuckles OS]% cat fichierInconnu 2>> toto
[etienne@Knuckles OS]% cat toto
ls: cannot access 'fichierQuiNExistePas': No such file or directory
cat: fichierInconnu: No such file or directory
[etienne@Knuckles OS]% ls && ls existePas 2>&1
 a.out  'Projet Mini Shell'  support_JM.pdf   TD4    TP    TP7   tp9
 fork.c  'Projet Mini Shell.tar.gz'   TD3        toto   TP5   TP8
[etienne@Knuckles OS]% ls | sort
 a.out  'Projet Mini Shell'  support_JM.pdf   TD4    TP    TP7   tp9
 fork.c  'Projet Mini Shell.tar.gz'   TD3        toto   TP5   TP8
[etienne@Knuckles OS]% echo a && cat toto > newtoto && sleep 4 & ; cat
                                                    newtoto 2> /dev/null
a
ls: cannot access 'fichierQuiNExistePas': No such file or directory
cat: fichierInconnu: No such file or directory
ls: cannot access 'existePas': No such file or directory
[etienne@Knuckles OS]% exit
A la revoyure!

#On revient a notre shell de notre distribution
[etienne@Knuckles Projet Mini Shell]$
```

Nous pouvons voir ici que notre Mini-Shell est capable de gérer plusieurs commandes d'affilée sur une même ligne, que cela soit par des "&&" ou encore des ';'.
Toutefois, nous pouvons également constater qu'il ne gère pas correctement les pipes, la partie droite du pipe ne prend pas en entrée la sortie de la partie gauche du pipe.
De plus, notre Mini-Shell supporte très mal les retours chariots quand la ligne est vide. Cela provoque une erreur de segmentation, je ne sais pas pourquoi.

Conclusion

Ce projet m'a permis de me familiariser avec les commandes Shell ainsi que de mieux comprendre leur fonctionnement.
Il est clair que ce Mini-Shell est bien loin du Shell que nous pouvons retrouver de base sur nos distribution linux, mais toutefois, il a le mérite de fonctionner, malgrès ses options limités par rapport à l'original.