Systèmes d'Exploitation Centralisés

Rapport de projet - BONTINCK Laérian 1AF

À noter : ce projet à été rendu en retard d'une journée, par inattention. Je vous prie de m'excuser.

I - Introduction

L'objectif du projet est de réaliser un shell rudimentaire qui permettra à l'utilisateur les choses suivantes :

- Exécuter des commandes
  - dans la boucle principale
  - en arrière plan
- Gérer les processus
- Gérer les signaux
- Effectuer des redirections
- Utiliser des tubes (*pipes* en anglais)

Chacune de ces actions sera étudiée dans une partie, chacune correspondant à une séance de TP.

II - Boucle principale (*main loop*)

La boucle principale, au départ, a simplement comme vocation d'exécuter les commandes qu'on lui donne dans l'entrée standard (le terminal) dans un fils :

```
> ls -l
Command : ls -l
total 80
-rw-r--r-- 1 lae lae   975 May 25 15:08 copier.c
-rw-rw-r-- 1 lae lae  2584 May 25 15:08 copier.o
-rw-r--r-- 1 lae lae  1057 May 25 15:08 Makefile
-rwxrwxr-x 1 lae lae 21760 May 25 16:01 minishell
-rw-r--r-- 1 lae lae  4937 May 25 15:16 minishell.c
-rw-rw-r-- 1 lae lae  7064 May 25 16:01 minishell.o
drwxrwxr-x 2 lae lae  4096 May 25 15:19 rapport
-rw-r--r-- 1 lae lae  5360 May 25 15:08 readcmd.c
-rw-r--r-- 1 lae lae  2156 May 25 15:08 readcmd.h
-rw-rw-r-- 1 lae lae  6568 May 25 15:08 readcmd.o
-rw-r--r-- 1 lae lae  2012 May 25 15:08 test_readcmd.c
(Process n°20956 terminated)
> fortune
Command : fortune
I despise humanity because it is not my humanity. I hate tyrants and I detest slaves.
        -- Renzo Novatore

(Process n°20969 terminated)
```

> On voit bien que le shell est capable d'exécuter des commandes shell de base, comme `ls`, mais aussi des commandes installées sur la machine, comme `fortune`.

On peut aussi quitter le shell avec la commande `exit` :

```
> exit
Shutting down...
```

> Le programme se termine bien quand on invoque la commande `exit`. On peut également lancer des commandes en fond (exemple en utilisant `sleep`):

```
> sleep 5 &
Command : sleep 5
[1] 20667
> pwd
Command : pwd
/home/lae/Documents/Work/N7/1A/SECs/Projet/minishell

(Process n°20669 terminated)
>
(Process n°20667 terminated)
```

> On voit dans cet exemple que le processus 20667 se lance, et ne finit qu'après avoir attendu 5s. En attendant, le premier plan est capable de lancer la commande `pwd` sans problème.

III - Gestion des processus et signaux

Dans cette partie, nous allons ajouter la capacité de mettre en pause ou d'arrêter un processus enfant, à l'aide des combinaisons de touches **Ctrl+C** et **Ctrl+Z** respectivement.
On ajoute aussi ici des messages qui indiquent l'action sur le processus (*terminated*, *manually paused*, *manually terminated*)

```
> sleep 5
Command : sleep 5

(Process n°21394 terminated)
> sleep 5
Command : sleep 5
^Z
(Process n°21403 manually paused)
> sleep 5
Command : sleep 5
^C
(Process n°21406 manually terminated)
```

> On voit effectivement que les *shortcuts* **Ctrl+C** et **Ctrl+Z** mettent en pause et arrêtent les processus.

IV - Masques signaux

Dans cette partie, nous allons masquer tous les signaux pour qu'ils ne s'appliquent que sur des processus **actifs** et **au premier plan**.

```
> sleep 1
Command : sleep 1
^C
(Process n°23056 manually terminated)
> sleep 1 &
Command : sleep 1
[2] 23057
> ^C
(No process to kill...).

(Process n°23057 terminated)
```

> On voit dans cet exemple qu'on peut bien arrêter un processus en premier plan, mais que quand on essaie d'arrêter un processus en arrière plan, le shell n'en est pas capable.

V - Redirections

Dans cette partie, on implémente les redirections, c'est-à-dire la capacité à utiliser un fichier en entrée d'une commande, ou d'écrire l'output d'une commande dans le fichier, en utilisant les tokens `<` et `>`.

```
> ls -l > out.txt
Command : ls -l

(Process n°24734 terminated)
```

```
(contenu du fichier out.txt)
total 80
-rw-r--r-- 1 lae lae   975 May 25 15:08 copier.c
-rw-rw-r-- 1 lae lae  2584 May 25 15:08 copier.o
-rw-r--r-- 1 lae lae  1057 May 25 15:08 Makefile
-rwxrwxr-x 1 lae lae 21760 May 25 16:15 minishell
-rw-r--r-- 1 lae lae  4933 May 25 16:15 minishell.c
-rw-rw-r-- 1 lae lae  7136 May 25 16:15 minishell.o
-rw-rw-r-- 1 lae lae     0 May 25 17:02 out.txt
drwxrwxr-x 2 lae lae  4096 May 25 15:19 rapport
-rw-r--r-- 1 lae lae  5360 May 25 15:08 readcmd.c
-rw-r--r-- 1 lae lae  2156 May 25 15:08 readcmd.h
-rw-rw-r-- 1 lae lae  6568 May 25 15:08 readcmd.o
-rw-r--r-- 1 lae lae  2012 May 25 15:08 test_readcmd.c
```

> On voit bien dans cet exemple que l'output de la fonction `ls -l` a été écrit dans le fichier out.txt, qui est même présent dans la liste des fichiers car il est nécessairement créé avant le début de l'exécution de la commande.

VI - Tubes

Dans cette partie, nous allons implémenter des tubes, c'est-à-dire une redirection directe entre les commandes. Exécuter `ls -l | wc -l` revient à exécuter `ls -l > tmp` puis `wc -l < tmp`, mais nous faisons ici tout ça sans fichier externe, dans la boucle principale.

```
> ls -l | wc -l
Command : ls -l

(Process n°25239 terminated)
Command : wc -l
13

(Process n°25240 terminated)
```

> Dans cet exemple de tube simple, on voit que l'output de la commande `ls -l` (qui liste les fichiers du cwd) est utilisée comme entrée de la commande `wc -l`, qui compte le nombre de lignes de son input.

Il est aussi possible d'utiliser des tubes en séquences (*pipelines* en anglais), c'est-à-dire d'enchaîner plusieurs tubes sur une ligne.

```
> cat minishell.c readcmd.c | grep int | wc -l
Command : cat minishell.c readcmd.c

(Process n°25484 terminated)
Command : grep int

(Process n°25485 terminated)
Command : wc -l
31

(Process n°25486 terminated)
```

> Dans cet exemple, la pipeline fonctionne ainsi :
>
> - `cat minishell.c readcmd.c` renvoie le contenu des fichiers `minishell.c` et `readcmd.c`
> - `grep int` récupère les lignes qui contiennent le token 'int' (explication simplifiée de la commande Global Regular Expression Print)
> - `wc -l` compte les lignes
>
> Ainsi, nous avons une commande qui, en une ligne, nous renvoie le nombre de lignes contenant 'int' dans les fichiers `minishell.c` et `readcmd.c`.

Conclusion

Ce projet nous a guidé dans l'élaboration d'un shell basique mais incorporant toutes les fonctionnalités requises d'un shell. Cela nous a permis de mieux comprendre comment les shells, composants de base de nos OS, fonctionnent.