Rapport du projet de systèmes d'exploitation centralisés

Minishell

Myriam ROBBANA

Groupe F

MAI 2024

Table des matières

1 Introduction

Ce projet vise à développer un interpréteur de commandes simplifié, offrant les fonctionnalités de base des shells Unix, comme le bash...

2 Choix et spécificités de conception

Le code est constitué de deux boucles `WHILE` imbriquées. Dans celle d'extérieure, on s'assure que l'utilisateur sort du shell uniquement si ce dernier saisit `exit`. Cependant, la deuxième boucle `WHILE`, s'assure que tant qu'il y a encore des élements à lire dans ligne de commande, on les lit et on les exécute.

Ainsi, pour la conception de ce minishell, j'ai pris en compte la gestion des signaux qui permet de réagir aux interruptions de l'utilisateur. Ainsi, j'ai mis en place des gestionnaires pour les trois signaux : `SIGCHLD` (terminaison du fils), `SIGINT` (pour le Contrôle C) et `SIGTSTP` (pour le Contrôle Z).

Pour la gestion des commandes en avant et arrère plan, j'ai fais en sorte que les commandes en avant-plan bloquent le shell jusqu'à leur achèvement. De même, j'ai fais en sorte que les commandes en arrière-plan ne bloquent pas le shell lui permettant ainsi de recevoir des commandes directement.

J'ai également fait le choix d'utiliser les masques à la place de `sigaction` pour la gestion des signaux `SIGTSTP` et `SIGINT`, car cela me permettait d'alléger le code et par conséquent de le rendre plus lisible, pour un même résultat.

Enfin, pour la communication entre les processus avec des commandes en pipeline, j'ai utilisé les tubes. Cela permet de rediriger les entrées et sorties des commandes de manière plus fluide. Les commandes étant excécutées dans le processus fils, c'est donc à cet endroit que j'ai réalisé la gestion des tubes. Les tubes ont cependant été créés avant la création du fils par le père (avant le `fork()`).

3 Retours sur quelques questions.

Voici quelques questions du projet :

Question 2 : Le processus shell lance un fils, puis se met immédiatement en attente de lecture de la prochaine ligne de commande ce qui n'est pas toujours désirable. On peut mettre en évidence ce comportement avec un `sleep(10)`. Après cette commande on devrait attendre 10 secondes avant d'avoir la main à nouveau, cependant, ici le shell nous la redonne directement.

Question 7 : La saisie de Controle C et Controle Z au clavier se traduit par les envois respectifs au shell des signaux `SIGINT` et `SIGTSTP`. La réception de ces signaux ne doit pas provoquer la terminaison du shell, ni celle de ses processus en arrière-plan, mais devra amener la terminaison du processus en avant-plan (éventuel) du shell. Pour ce faire, j'ai utilisé `sigprocmask` afin de bloquer les

signaux `SIGINT` et `SIGTSTP` afin d'empêcher les interruptions indésirables. Ainsi, l'ajout d'un masque de signaux permet de bloquer ces signaux et de les débloquer au moment voulu. Ainsi, à l'aide de

(`commande -> backgrounded != NULL`) on s'assure que l'on est en avant plan, et c'est ici que l'on souhaite bloquer les signaux `SIGINT` et `SIGTSTP`. Pour cela, dans cette partie du code, j'ai ajouté la commande `setpgrp()` qui place le fils dans un groupe ayant les mêmes caractéristiques que le père, et donc qui permet de reprendre le blocage réalisé par le père.

Question 8 : J'ai ajouté dans le code une partie qui gère la redirection des entrées ('`<`') et les sorties ('`>`') pour permettre l'utilisation de fichiers comme sources d'entrée ou destinations de sortie.

Ainsi, dans le fils j'ai rajouté un code qui fait en sorte que lorsque l'utilisateur spécifie un fichier pour l'entrée (`commande -> in`), celui-ci est ouvert en lecture (`O_RDONLY`) puis lié à l'entrée standard de la commande grace à `dup2()`. De même, si un fichier est spécifié comme étant la sortie (`commande -> out`), celui-ci est ouvert en écriture et lié à la sortie standard de la commande également.

4 Méthodologie de tests suivie.

4.1 Tests significatifs

Voici quelques tests réalisés :

Tests sur la gestions des arrières/avant-plans

Voici les tests réalisés :
`sleep 5 &`
`sleep 5 &`
`sleep 5`

Code affiché :

Voici l'affichage obtenu :
```
Le processus 2884265 s'est terminé avec le code de retour 0 suite au signal 17.
Le processus 2884304 s'est terminé avec le code de retour 0 suite au signal 17.
Le processus 2884343 s'est terminé avec le code de retour 0 suite au signal 17.
```

Tests sur Contrôle C et Contrôle Z

Test 1 :
`sleep 50`
```
Le processus 2885686 s'est terminé à cause d'un signal avec le numéro 2 suite au signal
17.
```

Test 2 :
`sleep 50 &`
Il n'y a rien qui se passe.

Test 3 :
`sleep 50`
`Le processus 2886335 a été suspendu par un signal SIGSTOP suite au signal 17.`

Test 4 :
`sleep 50 &`
Il n'y a rien qui se passe.

Test sur cat
J'ai écrit dans le fichier f1.txt : dqsdfqsdg
`cat < f1.txt > f2.txt`
Cette commande a crée un fichier f2.txt qui contient bien : dqsdfqsdg

Tests sur les tubes
`ls | grep mini | wc -l`
Voici l'affichage obtenu : `3`