Rapport du projet SEC
Minishell

Ayoub Bouchama

June 2, 2023

Contents

1 Introduction

Le projet consiste à créer un minishell fonctionnel en utilisant l'API Unix, dans le but de mettre en pratique les connaissances acquises lors des cours, des travaux dirigés et des travaux pratiques sur les systèmes d'exploitation centralisés. L'objectif est de développer un shell simplifié capable de gérer différentes commandes courantes du shell Unix en utilisant les fonctions fournies par l'API Unix.

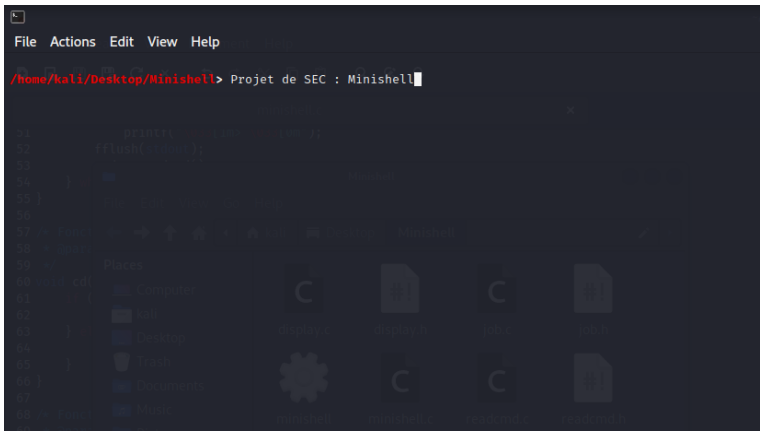

Figure 1: Image du minishell

2 Fonctionnalités du minishell

Le minishell prend en charge une variété de commandes, à la fois internes et externes, similaires à celles du shell Linux.

2.1 Les commandes internes

- **exit :** Quitter le shell.
- **clear :** Effacer l'ecran.
- **cd :** Changer d'emplacement (change directory).
- **susp :** Suspendre le shell.
- **lj :** Afficher les details de chaque processus (ACTIF = 0, SUSPENDU = 1, TERMINE = 2).
- **sj :** Suspendre un processus.
- **bg :** Reprendre un processus dans l'arrière plan.
- **fg :** Reprendre un processus dans l'avant plan.

2.2 Les commandes externes

Les commandes externes sont exécutées à l'aide de la fonction execvp, qui est une fonction de l'API Unix. La fonction execvp permet d'exécuter un programme externe en remplaçant l'image du processus courant par celle du programme spécifié.

2.3 Les redirections

Dans ma réalisation, j'ai pris en charge les redirections de la sortie et de l'entrée standard lorsque la ligne de commande contient les symboles '¡' ou '¿'. Ces symboles indiquent respectivement la redirection de l'entrée standard depuis un fichier ou la redirection de la sortie standard vers un fichier.

Exemple de test :

```
ls -l > TestRedirections.txt
```

2.4 Tubes et pipeline

Dans ma réalisation, j'ai ajouté aussi la fonctionnalité permettant d'enchaîner une séquence de filtres liés par des tubes, ce qui permet d'obtenir un traitement en pipeline. Cette fonctionnalité est couramment utilisée dans les shells Unix pour traiter les données de manière séquentielle à travers plusieurs commandes.

La fonctionnalité a été implémentée de manière récursive, en prenant également en compte les cas de redirection de l'entrée du premier tube et de la sortie du dernier tube.

Exemple de test

```
ls | wc -l | cat > TestPipline.txt
```

3 Question-2

Le problème est que le processus parent continue son exécution sans attendre que le processus fils terminé la commande lancée, ce qui peut entraîner un mélange entre l'affichage de l'invite de commande et l'exécution de la commande elle-même. Par exemple, si on utilise la commande "sleep 10" dans le shell, l'invite de commande apparaît immédiatement après avoir appuyé sur Entrée, sans attendre les 10 secondes de pause.

Pour résoudre cette problématique, j'ai employé la fonction waitpid dans le processus parent. Cette fonction permet d'attendre que le processus fils se termine avant de poursuivre l'exécution normale du programme et d'afficher la ligne de commande suivante. Par la suite, j'ai utilisé les fonctions de test telles que WIFEXITED, WIFSIGNALED et WTERMSIG pour vérifier l'état du processus fils et effectuer les mises à jour nécessaires dans la table des jobs. Ces fonctions permettent de déterminer si le processus fils s'est terminé normalement, s'il a été interrompu par un signal, et quel signal a provoqué cette interruption. Cela me permet de gérer les différents cas et de mettre à jour correctement les informations relatives au processus dans la table des jobs.

Exemple de test :

Remarquer la différence entre l'éxecution des deux commandes :

`sleep 10` et `sleep 10 &`

4 Questions-7/8

La frappe $CTRL + C$ (respectivement $CTRL + Z$) a été gérée à l'aide d'un traitant qui permet de terminer définitivement (respectivement suspendre) le processus fils en avant plan.

Chaque traitant de signal me permet de redéfinir le comportement du signal en le transmettant directement au processus fils en cours d'exécution.

Ensuite, j'ai masqué les signaux dans les processus fils pour éviter les actions inattendus. Si les signaux ne sont pas masqués dans les processus fils, ils peuvent interrompre brutalement l'exécution des commandes et provoquer des comportements inattendus.
Par exemple, si un processus fils est en cours de lecture à partir de l'entrée standard et reçoit un signal SIGINT, cela peut conduire à des résultats indésirables ou à une terminaison prématurée.
En masquant les signaux SIGINT et SIGTSTP dans les processus fils, le shell principal garde le contrôle sur la gestion de ces signaux. Il peut décider de les ignorer, de les traiter spécifiquement ou de les transmettre aux processus fils de manière contrôlée.

Exemple de test :

Lancer la commande `sleep 100` et faites la frappe $CTRL + C$. Vous allez remarquer que le processus se termine définitivement.
Relancer la commande `sleep 100` puis faites $CTRL+Z$, le processus ainsi est suspendu, tapez la commande lj, récupérer le UID de la commande sleep (la dernière) et composez la commande $fg < uid >$ pour reprendre le processus suspendu.

5 Ce qui ne marche pas ?

Malgré le bon fonctionnement des commandes en pipeline, je rencontre des difficultés à ajouter tous les morceaux de la commande saisie en ligne de commande dans le tableau des processus. Seul la première commande s'ajoute, j'ai tout essayé mais sans résultat.

6 Difficulté rencontrée

- J'ai eu des difficultés de mémoire (segmentation fault) dans différent partie de mon implantation mais j'ai réussi finalement à y remédier.
- J'ai eu des diffucltés avec la table des processus (ajout, ajustement d'états des processeurs).
- Debogage des erreurs surtout quand ceci concerne les frappes $CTRL + C$ et $CTRL + Z$ car ca n'affiche pas ou l'erreur existe mais juste une erreur dans un $G_W AIT$.

7 Bilan

Ce projet constitue une excellente mise en pratique des concepts et des fonctionnalités que nous étudions en cours, en travaux dirigés et en travaux pratiques, et m'a permis de consolider ma compréhension des notions clés ainsi que des fonctionnalités offertes par l'API d'UNIX. Grâce à cette expérience, j'ai pu approfondir mes connaissances en sec.

8 Lacer le minishell

Pour lancer le minishell il suffit de créer l'executable **minishell** en executant la commande : `gcc -Wall minishell.c job.c readcmd.c -o minishell`, puis d'éxecuter le programme avec `./minishell`.