Projet : minishell

BAURIAUD Laura

June 2, 2023

1 Introduction

L'objectif de ce projet est de mettre en pratique les notions que nous avons vu en TD et en TP, autour de la gestion des processus, des signaux et des E/S, dans un contexte réaliste. Il s'agit plus précisément de développer un interpréteur de commandes simplifié, offrant des fonctionnalirés de base des shells Unix, comme le bash.

2 Étapes du projet

2.1 Étapes complètement traitées

Les 9 premières questions ont été entièrement traitées.

1. Lancement de la commande : mon minishell peut répondre à des commandes simples.
2. Exemple : une partie du rapport y est dédié
3. Enchainement séquentiel des commandes : je peux lancer plusieurs commandes à la suite
4. Commandes internet : nous pouvons faire appel aux commandes cd et exit qui ne seront pas excecutés dans un fils mais bien dans le père.
5. Lancement de commandes en tâche de fond : le minishell est capable de lancer des tâches de fonds.
6. Gérer les processus lancés depuis le shell : la commande lj permet bien de lister les processus, sj permet de suspendre un processus, les commandes bg et fg fonctionnent également.
7. SIGSTP : un processus peut être suspendu à partir des différentes commandes demandées
8. SIGTINT : idem avec le ctrl-C
9. Redirections : mon minishell est capable de rediriger l'entrée ainsi que la sortie standards d'une commande vers un fichier.

2.2 Étapes partiellement traitées

Les questions partiellement traitées sont les question sur les tubes (pipes), c'est-à-dire les question 10 et 11. En effet je pense avoir un problème soit dans l'ouverture et la fermeture des pipes, soit dans la redirection, ce qui est étonnant car ma redirection standard semble bien fonctionner.
J'ai pensé à fermer mes pipes dans le père mais ça ne règle pas mon problème. Je n'ai aucun problème à la compilation mais mes pipes ne fcontionnent quand même pas. En utilisant des *print*, que j'ai enlevé par la suite, je n'arrive pas à trouver le ou les problèmes.
De plus, il y a parfois quelques problèmes d'affichage, les »> ne sont pas toujours affichés au bon endroit. J'ai essayé au maximum de faire une interface friendly pour l'utilisateur mais je doute d'avoir toujours réussi.

2.3 Étapes non traitées

J'ai essayé de traiter l'intégralité du sujet même si ce n'est pas toujours concluant.

3 Architecture de l'application

Dans un premier temps, nous allons considérer les types de variables :

- gestion : une énumération qui rerésente un processus avec son état, son pid, sa commande ainsi que son identifiant dans le minishell
- globale : un tableau de l'ensemble des processus en cours, qu'ils soient au premier plan ou en arrière plan. Il s'agit d'un table de "gestion"

Nous allons maintenant voir, les différentes fonctions présentes dans minishell.c. Je vais les présenter dans l'ordre dans lequel elles apparaissent dans mon code.

- Redirections : les redirections sont gérées par *redirection_entree* et *redirection_sortie*. La fonction globale *redirection_globale* permet de gérer en une seule fois les deux redirections possibles. Cette fonction globale est utilisée lorsqu'il n'y a pas de pipe.
- Définition des commandes : il nous ai demandé de créer différents handler pour des commandes particulières entrées en ligne de commande ainsi que lors de la réception de signaux. J'ai donc codé :

  1. *handler_jobs* qui correspond à la commande lj, lister l'ensemble des processus processus en cours.
  2. *handler_sj* pour sj qui va arrêter le processus indiqué.
  3. *handler_bg* pour placer un processus en arrière plan.
  4. *handler_fg* pour déplacer le processus en premier plan.
  5. *handler_cd* qui est utilisé après la lecture de cd au clavier.
  6. *handler_int* à la reception du signal SIGINT, c'est ce traitant qui va être appelé
  7. *handler_tstp* qui est utilisé à la reception du signal SIGTSTP.
  8. *handler_child* est le traitant que je lance dans le fils.
  9. *handler_gen* est le traitant général qui va gérer la création des fils et la gestion du père.
  10. $handle_cmd$ traitement des commandes spéciales
  11. *recup_sigaction* initialiser le sigaction qui va récupérer les signaux.
  12. *main* programme principal lancée à partir du terminal.

  Pour chaque fonction, j'ai essayé de traiter tous les codes d'erreur, d'être exhaustive au maximum pour faciliter l'utilisation du minishell par l'utilisateur.

4 Choix et spécificités de conception

J'ai choisi d'utiliser un tableau pour gérer l'ensemble de mes processus. L'identifiant propre à chaque processus correspond à son indice dans le tableau. Pour supprimer un processus, il suffit de remettre l'identifiant à 0. Le seul problème que je vois à l'utilisation du tableau est la taille fixe de celui-ci. En effet, j'ai choisi de pouvoir gérer au maximum 500 processus mais ce nombre a été choisi arbitrairement. Il serait pertinent d'étudier le nombre moyen de processus lancé pour être certain que mon tableau peut contenir tous les processus.

5 Méthodologie des tests - méthodologie de tests signicatifs

5.1 Créer le fichier à compiler

Afin de tester mon minishelle :

- décompresser le fichier Minishell_Laura.tar
- ouvrir le terminal dans le dossier
- faire le fichier à compiler : gcc -Wall -o minishell minishell.c readcmd.c
- ./minishell

5.2 Tests

Le minishell est lancé, vous allez maintenant pouvoir lancer les commandes et donc faire les tests suivants :

- Commandes simples : écrire ls ; ls -l; pwd
- Crtl-C et ctrl-Z : effectuer un sleep 40 puis ctrl-C, le minishell agit correctement
- Commandes en arrière plan : commencer avec un premier lj -> aucun processus n'a été lancé. Écrire successivement sleep 40 , sleep 50 , lj -> les processus sont maintenant affichés en arrière plan. Nous pouvon maintenant utiliser l'identifiant de ces processus et les remttre au premier plan. Nous pouvons égalelement tester les ctrl-C et ctrl-Z, ils n'ont aucun effet car les processus sont en arrière plan.
- Redirection : écrire ls -l > f1, le fichier f1 vient d'être créé nous pouvons donc l'ouvrir et constater qu'il contient bien le résultat du ls -l
- tubes : nous pouvons écrire ls | wc -l, le résultat n'est pas celui attendu, nous obtenons deux fois le résultat du ls

Cette liste de tests n'est bien sûr pas exhaustive mais elle permet de voir les principales fonctionnalités et les défauts de mon minishell.

6 Conclusion

Ce projet m'a permis de mettre en application les éléments de SEC vus en cours, en TD et en TP. De plus, ce projet a mis en lumière qu'il existait une multitude de façon de coder un shell. Il existe toujours plusieurs possibilités pour écrire une fonction et la mettre en application. Les choix de conception sont riches et variés ce qui fait la beauté de cette matière.
Je regrette de ne pas voir réussi à faire fonctionner mes pipes mais je n'arrive pas à débeugger mon code pour que cela fonctionne.