Rapport Du Projet : Systèmes d'Exploitation Centralisée.

ALOUANE Issam

Département Sciences du Numérique - Première année
2021-2022

Contents

1 Question 1

Pour la première question j'ai choisi pour implanter la boucle de base de l'interpréteur une boucle répéter avec un boolean car la sortie du minishell (en tapant "exit") doit être sans erreur, alors on va se baser sur le boolean qui va nous indiquer le moment où on va sortir du boucle et arrêter le minishell.

2 Question 2

Pour mettre en évidence ce comportement, on peut utiliser un programme qui crée un processus fils et fait un appel à sleep pour un bon moment (20 secondes par exemple) et en même temps le père est forcé (par wait) à attendre la fin de son fils. A titre d'exemple, on peut utiliser le programme $pre\_fils\_wait.c$ qui a été fourni avec le tuto de tp processus, là, le processus principal attend la fin de ses fils pour terminer l'exécution, et on peut voir bien qu'on peut taper par exemple « ls » sur la ligne de commande et le résultat va être affiché au milieu de l'exécution du premier programme.

3 Question 3, 4, 5

Pour l'attente de la fin de la dernière commande lancée, j'ai choisi d'utiliser une boucle while dans laquelle y'en a une pause bloquante qui attend un signal, SIGCHLD en particulier pour dire que le fils est suspendu ou arrêté ou terminé, pour pouvoir ensuite reprendre le minishell et demander la commande suivante. Ainsi, pour les deux commandes internes, j'ai ajouté une conditionnelle if, afin de tester la commande entrée, et s'elle fait partie des commandes intérieures on la traite indépendamment. Ensuite, pour le lancement en arrière-plan, j'ai utilisé un boolean qui indique est-ce que la tâche est en arrière-plan ou en avant-plan et c'est ce boolean qui est la condition de boucle de pause cité précédemment.

4 Question 6

Pour la première commande lj, j'ai créé un module liste, qui va m'aider à enregistrer les commandes et les pid de leur processus et leur identifiant propre au minishell dans une liste qui sera une variable globale car on va la manipuler presque partout. Ainsi, le processus est enregistré à l'état actif quand il vient d'être écrit ou quand on lui envoie un SIGCONT, il est suspendu quand on lui envoie un SIGSTOP, et quand il est terminé ou tué par un signal on le supprime de la liste, et j'ai utilisé un waitpid pour m'aider à distinguer ces cas dans le handler du signal SIGCHLD, et aussi parce qu'elle donne la possibilité d'attendre un processus avec un pid donné. Pour la commande sj, on envoie au processus un signal SIGSTOP pour le suspendre, et pour bg, on envoie au processus un SIGCONT et on indique que le boolean d'attente (pause) est en false, car la tâche est en arrière-plan, et pour fg on envoie un SIGCONT et on indique que le boolean est en true, car la tâche deviendra en

avant-plan. Enfin, j'ai ajouté ces commandes internes à la conditionnelles dans la boucle principale.

5 Question 7, 8

Pour les frappes du Ctrl+Z et Ctrl+C, j'ai écrit 2 handlers qui envoient au fils qui fait la tâche en avant-plan les 2 signaux convenables : SIGKILL et SIGSTOP. Je me suis encore servit du boolean qui indique est-ce que la tâche est en avant-plan ou pas, ainsi, j'évite d'envoyer ses signaux à tous les processus et je les envoie juste au processus qui est en avant-plan. Ensuite, j'ai ajouté la commande interne susp qui envoie un signal SIGSTOP au minishell.

6 Question 9

Pour cette question, je me suis servi des composantes in et out de la commandes pour rediriger l'entrée standard vers le nom du fichier indiqué en in (s'il existe), et rediriger la sortie standard vers le nom du fichier indiqué en out.

7 Question 10, 11

Pour ces 2 questions, je me suis servi du sous-programme récursif *exec_cmd_pipe*, qui réalise des commandes liées par des tubes, et sa nature récursive nous donnes l'avantage qu'il va réaliser toutes les commandes contenant des tubes, quelque soient leur nombre.