Projet Systèmes d'Exploitations Centralisée

Ilyasse Alioui

21 Avril 2023

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.

Question 2:

Pour mettre en évidence ce comportement, il est possible d'utiliser un programme qui crée un processus fils et utilise la fonction "sleep" pour mettre en pause l'exécution pendant un certain temps (par exemple, 20 secondes), pendant que le processus parent est contraint (via la fonction "wait") d'attendre la fin de l'exécution du processus fils.

Par exemple, le programme "pre_fils_wait.c", fourni dans le tutoriel du TP Processus, permet au processus principal d'attendre la fin de l'exécution des processus fils avant de terminer l'exécution. On peut alors exécuter d'autres commandes en parallèle, comme par exemple "ls", et le résultat sera affiché pendant l'exécution du premier programme.

Question 3,4,5:

J'ai décidé d'utiliser une boucle while pour attendre la fin de la dernière commande lancée en utilisant une pause bloquante qui attend un signal SIGCHLD spécifique indiquant que le fils est suspendu, arrêté ou terminé. Cela permet de reprendre le minishell et de demander la commande suivante. Pour traiter les deux commandes internes, j'ai ajouté une condition if pour tester la commande entrée et la traiter indépendamment si elle fait partie des commandes internes. Ensuite, pour le lancement en arrière-plan, j'ai utilisé un booléen pour indiquer si la tâche est en arrière-plan ou en avant-plan et c'est ce booléen qui est utilisé comme condition de la boucle de pause mentionnée précédemment.

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.

Quelques pistes de reflexion pour les question 7 et 8

J'ai presque fini cette partie de code mais je prefere avoir plus de temps pour revoir ce que j'ai fait (Tant que c'est permis).j'ai écrit 2 handlers qui envoient au fils qui fait la tâche en avant-plan les 2 signaux convenables : SIGKILL et SIGSTOP.Ensuite, j'ai ajouté la commande interne susp qui envoie un signal SIGSTOP au minishell.