Projet Minishell

Adam Bouam

25 mai 2022

Question 3 : Une solution au problème de la question 2

A ce niveau j'ai utilisé la primitive *wait* qui permet au père d'attendre le processus fils executant la commande lancé.

Question 4 : Les commandes internes *cd* et *exit*

les commandes internes sont exécutées directement par le processus père sans avoir à utiliser la primitive *fork*.

- Pour la commande *cd* j'ai utiliser la primitive *chdir* qui permet de changer le répertoire courant.
- Pour *exit* on utilise le *exit*(0) normal.

Question 5 : Les commandes lancées en arrière-plan

A ce niveau j'ai utilisé la primitive *waitpid* d'une manière normale si la commande est lancée en foreground, d'une manière bloquante avec le macro WNOHANG entré en options si la commande est lancée en background.

Question 6 : Les commandes *lj*, *sj*, *fg* et *bg*

A ce niveau du code j'ai fait des changements sur le comportement du processus père et j'ai ajouter des variables et types, voici une liste non-exhaustive de ce que j'ai réalisé :

- J'ai défini un type enregistrement qui regroupe les informations sur chaque processus lancé en background, ces processus vont être regroupés dans une liste.
- J'ai défini des variables qui vont servir pour caractériser le processus lancé en avant plan et ce processus seulement.
- J'ai défini quelques fonctions supplémentaires qui vont faciliter la gestion des processus et leurs états.
- *lj* va permettre d'afficher tout les processus qui sont background qui sont soit actifs ou suspendu.
- Pour *sj* J'ai utilisé la primitive *kill* avec le signal SIGSTOP.
- Pour *bg* J'ai utilisé le signal SIGCONT qui remet automatiquement un processus arrêté en arrière plan.

Dans cette partie du projet j'ai rencontré un nombre de difficultés dont je vais citer une partie ci-dessous avec les approches que j'ai suivi :

Les solutions déjà proposées pour la question Q2 ne marchent plus, j'ai dÃ» changer d'approche. Car pour un processus lancé le père va devoir l'attendre et suivre son état jusqu'à sa terminaison. Cela ne sera pas possible avec plusieurs processus en arrière plan qui sont tous en cours d'exécution. J'ai trouvé du mal à gérer leurs états à travers d'un *waitpid* du père. Pour cela j'ai utilisé un simple *sleep* à chaque fois un processus en avant plan est lancé. Pour les processus en bg, On ne fait rien et on les enregistre dans notre liste. Lorsque un processus fils change d'état j'ai pris l'avantage du signal SIGCHLD qui va être envoyé au père et j'ai changer le handler à ce signal pour en mettre un *waitpid* qui va rien faire mais prendre connaissance de ce changement d'état à l'aide des différents primitives adéquates.

La commande fg devient à ce niveau simple à implanter avec toujours l'appel à *sleep* tant que ce processus n'a pas changé d'état (fini).

Questions 7 et 8 : Gestions des signaux

J'ai traité cette partie partiellement en avance dans la question 6 mais je n'ai pas trouvé de difficultés particulières. J'ai ajoutés un handler vide au signal SIGINT et j'ai modifié le handler de SIGSTOP en ajoutant le fait que si c'est le shell qui est en avant plan ce signal ne va rien faire.

J'ai ajouté des masques pour ces signaux pour les processus fils.

Pour la commande *susp* j'ai utilisé directement la primitive *kill* avec *getpid* et le signal SIGSTOP.