Rapport projet Mini-Shell

Préambule :

J'explique dans ce court rapport les fonctionnalités que j'ai réussi à implémenter, les difficultés rencontrées ainsi que les solutions et les choix de mon implémentation.

Explication de l'implémentions des commandes (choix et difficultés rencontrées**)**

1) Commande simple

L'implémentation d'une commande simple a été réalisée sans difficultés. Cependant, il a fallu réfléchir à une solution pour éviter d'allouer le maximum de mémoire pour stocker les arguments. Pour bien gérer la mémoire lors du stockage des arguments, j'ai utilisé un compteur d'argument et en fonction de ce compteur et de la taille de l'argument, j'ai alloué l'espace nécessaire. Cela permet d'éviter d'allouer pour chaque argument la taille maximale, on gagne ainsi en mémoire. Le reste de cette partie a été facile à implémenter.

2) Traitement de l'attente

Cette commande a été réalisée grâce à la fonction « wait ». Lors du lancement d'une commande en arrière-plan, le processus père n'attend pas immédiatement le processus fils et l'utilisateur peut continuer à saisir d'autres commandes. La principale difficulté rencontrée dans cet exercice était dans la manière d'arrêter le processus lancé en arrière-plan. En effet, si ce processus n'est pas arrêté, on remarque que celui-ci devient un processus zombie.
Pour arrêter ce processus zombie, il est nécessaire d'arrêter au préalable la commande en arrière-plan dès qu'elle se termine, avant de lancer la prochaine commande. On réalise ceci grâce à une boucle « wait (NULL) » qui va récupérer le « pid » de tous les processus fils finis : si le « pid » récupéré est celui d'une commande en arrière-plan, alors ce processus fils sera arrêté et la boucle continue tant que le « pid » n'est pas celui du processus au premier plan.

3) Traitement des redirections

Cette partie n'a pas particulièrement posé de difficultés. Cependant, une légère contrainte a été rencontrée lors d'une redirection avec troncature, au début j'ai utilisé la fonction « write » permettant d'écraser directement ce qui est déjà écrit, mais cette technique n'est pas toujours efficace car si le fichier possède par exemple 100 mots et qu'on vient l'écraser avec une commande qui écris 50 mots, on aura alors 50 mots en plus qui « n'ont pas de sens » à la lecture du fichier. J'ai donc décidé de supprimer le fichier puis de le recréer (ainsi il est vide), de cette façon la lecture d'une commande redirigée avec troncature devient plus lisible.

Par ailleurs, une implémentation des messages d'erreur en cas de non-ouverture d'un fichier ou autre a été réalisée pour améliorer le code, ce qui m'a permis de comprendre d'où provient une erreur lors de l'implémentation, de détecter rapidement mes erreurs et de gagner du temps. Notons que la fermeture de la sortie ou l'entrée lors d'une redirection est importante pour éviter d'éventuelles erreurs.

4) Traitement du tube

L'une des étapes de cette partie a été assez difficile à réaliser. Au début, j'ai suivi les instructions d'implémentation que j'ai réussi à appliquer sans difficulté, mais la partie difficile concerne le traitement des extrémités non utilisées du tube. En effet, lors d'une commande nécessitant un tube, j'ai constaté que mon programme se « bloque » dans le minishell. J'ai résolu mon problème en comprenant que pour lire dans un tube il faut que ce dernier reçoive « une fin de fichier » et celle-ci est envoyée lorsque le nombre d'écrivain est égal à 0, il faut donc impérativement inclure « close(pipe[1]) » dans le processus père .

5) Traitement du point-virgule

Cette partie n'a pas posé de problème. Certes un point-virgule se traite comme un retour chariot, mais il faut faire attention à ne pas afficher le prompt. En effet, à la fin de l'exécution de la partie gauche du point-virgule il ne faut pas afficher le prompt mais continuer directement vers l'exécution et l'affichage de la partie de droite.

NB : La compréhension de exercices de TP et les explications du professeur BENJAMIN NEGREVERGNE sur Mycourse m'ont beaucoup aidé à réaliser le projet.