Maxime SOLÈRE

Rapport de projet mini-shell

Fonctionnalités implémentées avec succès :

- Traitement des commandes externes simples
- Redirection des entrées/sorties (<, >, >>)
- Composition de commandes (|)
- Lancement de plusieurs commandes (;)
- Commande cd, avec la fonctionnalité $ cd renvoie vers $HOME
- Capture du signal ^C et mot-clé exit
- Système de variables d'environnement

J'ai également traité le lancement de commandes en arrière plan (&), qui ne fonctionne qu'en partie, c'est-à-dire que $ sleep 10 va afficher le PID du processus lancé, ainsi que le prompt du mini-shell, mais au bout de 10 secondes, il n'est pas affiché que la commande a fini de s'exécuter, et on ne peut pas accéder aux commandes lancées en arrière plan.

Fonctionnalités que je n'ai pas pu implémenter :

- Recherche ^R et recopie des commandes avec les flèches
- Gestions de commandes en arrière plan (fg/bg)
- Complétion automatique des commandes
- Structures de contrôle

Tokens ajoutés à analex.c :

- T_EXIT : signal de la commande $ exit
- T_CD : signal de la commande $ cd
- T_VAR_DECLA : correspond à une variable au moment de sa déclaration
- T_VAR_VAL : correspond à une variable au moment où on appelle sa valeur avec $

Fonctions ajoutées :

- char *copie(char *o) : renvoie la copie d'une chaîne, car la fonction strcpy() posait des problèmes puisque elle ne fait que pointer la première chaîne sur la seconde, et si on l'utilise plusieurs fois sur une même variable (ici word dans la fonction commande), cela modifie tous les éléments qui pointent sur word.
- int look_str(char *chaine, char *tabChaine[ ]) : cherche une chaine dans un tableau de chaine, renvoie sa position ou -1 si elle n'apparait pas. Je l'ai utilisé pour les variables, car j'ai stocké dans deux tableaux de chaînes les identificateurs et les valeurs des variables existantes.

Ajout d'arguments aux fonctions déjà écrites :

- À la fonction execute, j'ai ajouté un argument int pipe, non nul si la commande lancée est dans un pipe, ce qui m'a permis de distinguer le cas où le processus père dans le fork devait attendre le fils, ainsi que fermer la sortie.
- À la fonction commande, j'ai ajoutés trois arguments, pour le traitement des variables d'environnement, ces trois arguments valant NULL par défaut (si la commande ne contient pas de variables) :
    - char *id_var, *val_var : contiennent l'identificateur et la valeur de la variable, pour les stocker dans des tableaux déclarés dans le main, pour qu'ils restent pendant toute l'utilisation du mini-shell.
    - char *tabArgs_var[ ] : c'est le tableaux des arguments de l'entrée, que je dois, dans le cas d'un appel à la valeur d'une variable, utiliser dans le main pour remplacer l'identificateur par sa valeur.

Tests ajoutés :

- Un test pour cd : $ cd tests ; ls 07_cd.ms
- Un test pour les variables : $ x=4 ; x=3 ; z=3 ; echo $y ; echo $z

Problèmes rencontrés :

- Avant de passer par la fonction copie et de mieux comprendre le fonctionnement de analex.c, j'ai mi du temps à réussir à stocker les arguments correctement.
- Je n'avais pas très bien compris le fonctionnement de execvp, donc j'avais écrit une fonction which, qui prenait en argument une commande, et renvoyait sa position dans l'arborescence.
- Pour le traitement des pipes, j'ai eu du mal à gérer les entrées et les sorties, avec plusieurs essais en passant par des fichiers temporaires
- Pour le exit, j'ai eu pendant longtemps un bug où chaque commande exécutées sans succès créaient une nouvelle version fils du mini-shell.
- Avant de tomber sur la fonction getenv(« HOME »), qui renvoie le chemin vers le répertoire home, j'ai essayer de l'implémenter manuellement, pour que $ cd renvoie vers $HOME.

Ce projet m'a permis de mieux comprendre :

- Le fonctionnement général d'un shell et de voir que ce n'est qu'un programme comme un autre
- Le fonctionnement de execvp, notamment couplé au fork
- La redirection de entrées/sorties et manipulations des fichiers avec les appels systèmes