Minishell - Rapport

BRENNER Quentin, NAEGELY Jeannot

Contexte

Ce projet, réalisé dans le cadre de notre module de systèmes avancés de deuxième année d'école d'ingénieur, est une version plus légère d'un shell, appelé minishell.

Logique implémentée & choix de conception

Premièrement, concernant la structure du projet que nous avons choisie, celle-ci est plutôt simple, mais permet de bien identifier ses différentes parties. En effet, il est découpé en quatre dossiers : **src**, **include**, **doc** et **output**, avec respectivement les fichiers .c, .h, la documentation et l'exécutable du projet.

Ensuite, pour ce qui est du fonctionnement du code, nous avons tenté au maximum de créer le minishell en suivant la logique **REPL** (Read-Eval-Print Loop). Pour ce faire, nous avons donc mis en place une boucle principale dans le main, qui attends perpétuellement un input de l'utilisateur. Une fois celui-ci reçu, il va être **parsé** et modifié par les fonctions du fichier **parser.c**, qui vont l'analyser et en sortir un tableau de Commands, une structure réalisée par nos soins qui permet le stockage d'informations sur plusieurs commandes et opérateurs (définition commentée disponible dans le fichier **typedef.h** et le pdf généré par doxygen trouvable sur le repo).
Une fois l'entrée de l'utilisateur analysée, on va vérifier si les commandes utilisées correspondent ou non à des alias existants et les remplacer par leur contenu si c'est le cas.
Enfin, la structure Commands va être envoyée aux méthodes correspondantes du fichier executor.c afin de vérifier la présence de commandes built-in. Si c'est le cas, on va l'exécuter telle qu'elle a été reconstruite, en respectant les opérateurs, dans un des fichiers suivants : **internal_commands.c**, **aliases.c**, et **history.c**. Dans le cas contraire, on utilise **exec(vp)** pour exécuter la commande classique correspondante.

Pour plus de détails, voici ci-dessous les différentes features disponibles de ce minishell :

- L'exécution de commandes simples (exemples : ls -a, ps)
- L'exécution de sous-ensemble de commandes
    - Opérateurs &&, ||, |, <, >, <<, >>, & et ;
    - exemples :
        - ls -alF > file.txt
        - ps aux | grep minishell > output.txt
- L'exécution de commandes built-in
    - cd
    - pwd
    - exit
    - echo
    - history
    - alias
    - unalias
- Le maintien d'un historique dans le fichier ~/minishell_logs/command_history

- Un mode de lancement "batch"
    - ./[path_to_minishell_executable] [OPTION] <command>
    - Les différentes options possibles sont :
        - -c, --command <command>: Execute the specified command and exit.
        - --help: Display help message and exit.
- La prise en charge d'alias temporaires
    - Définition d'alias
    - Affichage de la commande correspondant à un alias
    - Affichage de toutes les définitions d'alias
    - Suppression d'alias
    - Suppression de tous les alias

Difficultés rencontrées

Comme il nous a été conseillé sur le sujet, nous avons procédé par étapes pour simplifier le développement de ce projet. C'est effectivement une méthode qui fonctionne, mais dans notre cas, nous avons eu du mal à passer d'une structure qui permet de gérer une seule commande, à celle qui nous permet aujourd'hui de gérer les commandes par sous-ensembles. Nous avons réussi à surmonter cette difficulté en discutant entre nous et en sélectionnant les meilleures idées qui en sortaient.

L'ajout de certaines fonctionnalités, telles que la gestion des différents opérateurs (input/output redirection) nécessitait une bonne compréhension de leur fonctionnement et des recherches préalables à leur implémentation dans notre projet.

Qui plus est, bien que les tâches aient été correctement réparties entre les membres du groupe, certains ajouts étaient considérés comme bloquants pour l'implémentation de fonctionnalités, obligeant l'autre membre à attendre que celle-ci soit complétée.

Il est toutefois notable que la bonne préparation et répartition des tâches nous a permis d'avancer efficacement sur le projet, que nous avons particulièrement apprécié.

Heures passées sur les étapes du projet par personne

	Conception	Code	Tests	Documentation / rapport
Quentin	2h	15h	2h	2h
Jeannot	1h	15h	2h	5h