Projet : `minishell`

Systèmes d'exploitation centralisés

Comparetto Matthieu
1ère année Sciences du Numérique

Mai 2025

Introduction

Ce projet consiste à implémenter un mini-interpréteur de commandes Unix, permettant de lancer des processus, de gérer les redirections, les pipes, et certains signaux. Il a été développé en langage C.

1 Architecture générale

Le projet repose sur une boucle principale lisant les commandes de l'utilisateur via la bibliothèque `readcmd`. Les commandes sont analysées, puis exécutées via des processus fils créés dynamiquement.

2 Architecture du code

Tout le code est regroupé dans un unique fichier `minishell.c`, ce qui rend le projet autonome et facile à manipuler. Ce fichier contient la boucle principale ainsi que les fonctions auxiliaires pour la gestion des commandes internes, des redirections, des tubes et des signaux. Le parsing est assuré par la bibliothèque `readcmd` fournie.

3 Fonctionnalités principales

- **Lancement de commandes** (avec `execvp`).
- **Support des pipes** entre plusieurs commandes.
- **Redirections** d'entrée et de sortie (`<`, `>`).
- **Commandes internes** : `exit`, `cd`, `dir`.
- **Gestion de signaux** : interruption, suspension, exécution en arrière-plan.
- **Affichage d'un prompt interactif**.

4 Choix de conception

- Les commandes enchaînées sont exécutées dans une boucle, chaque processus étant forké, puis relié via des tubes si nécessaire.
- En mode interactif (hors arrière-plan), seul le **dernier processus** est attendu, ce qui permet une exécution parallèle des processus connectés par des pipes.
- Les signaux sont gérés via `sigaction` pour garantir un traitement fiable des processus fils.

5 Méthodologie de test

Des tests manuels ont été réalisés sur des cas simples et complexes pour valider le bon fonctionnement. (Voir captures d'écran plus bas.) Voici quelques cas testés :

- Vérification que le shell affiche la commande saisie : `echo`.
- Création d'un processus fils et exécution : utilisation de `fork()` et `exec()` dans le code.
- Exécution en arrière-plan : `commande->backgrounded == NULL` dans le code, `&` dans le terminal.
- Gestion des signaux : test du handler avec `Ctrl+C` et `Ctrl+Z`, sans fermeture du shell.
- Redirections E/S : test des opérateurs `<` et `>` dans l'interpréteur.
- Vérification des pipes : test de `|` dans des commandes chaînées.

6 Spécificité de conception

Pour que les pipes fonctionnent correctement, il a fallu enregistrer le PID du dernier processus pour effectuer un `wait` hors de la boucle `for`. Cela garantit que tous les processus d'un pipe peuvent s'exécuter ensemble, sans bloquer sur un tampon saturé en cas de traitements lourds.

7 Exemple d'utilisation

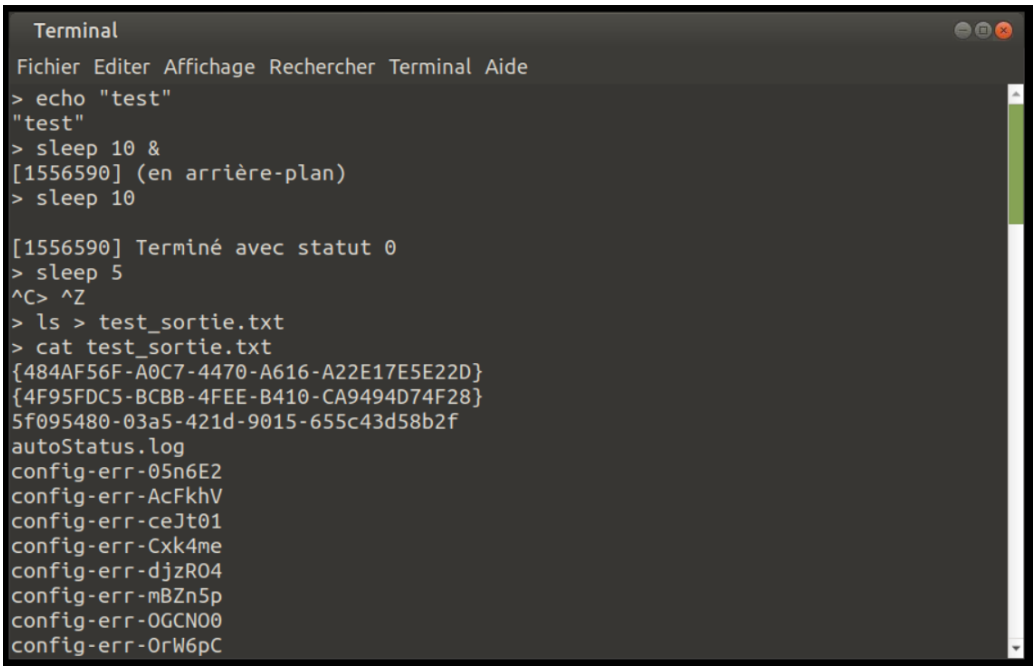

FIGURE 1 – Echo, exécution en arrière-plan, redirection de la sortie : `ls > test_sortie.txt`

On observe que `echo` affiche bien la commande, l'ajout de `&` permet l'exécution en arrière-plan, et la redirection `>` envoie correctement la sortie dans un fichier texte, consulté ensuite avec `cat`.

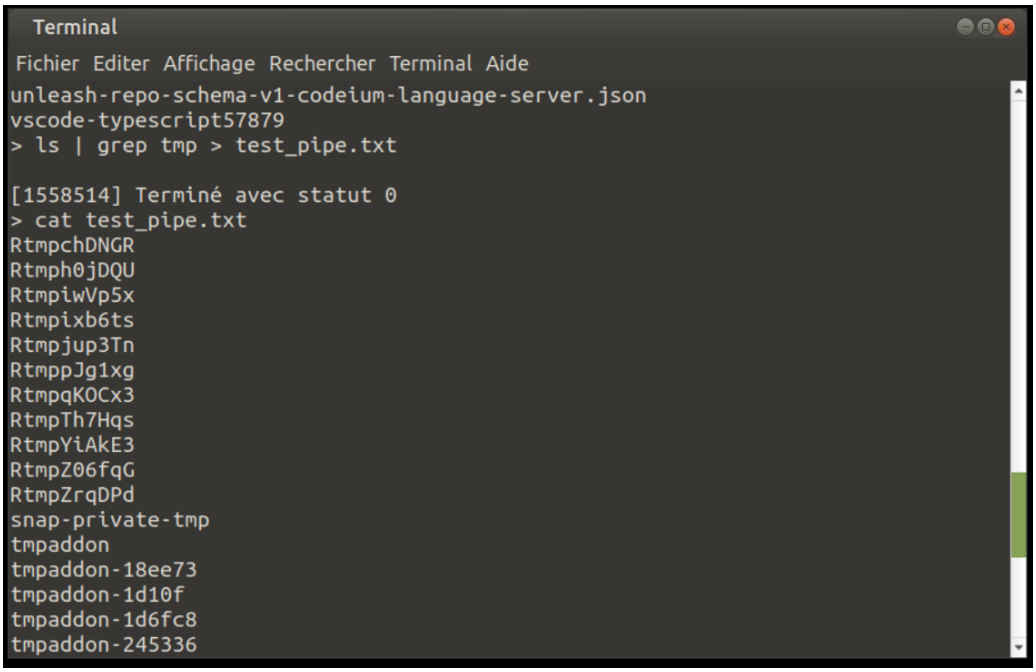

FIGURE 2 – Utilisation d'un pipe avec redirection : `ls | grep tmp > test_pipe.txt`

On voit que la commande `ls` passe sa sortie à `grep`, qui filtre les fichiers contenant "tmp", et que le résultat est redirigé correctement dans un fichier texte.

Conclusion

Ce projet a permis de mettre en œuvre les concepts fondamentaux des systèmes UNIX : gestion des processus, communication inter-processus, redirections et signaux. Le minishell fonctionne comme prévu et constitue une base simple mais robuste pour un interpréteur personnalisé.