Rapport sur la logique implémentée dans MiniShell

# Introduction

MiniShell est un interpréteur de commandes minimaliste implémenté en C. Il offre la possibilité d'exécuter des commandes système ainsi que des commandes internes, tout en gérant des fonctionnalités avancées telles que les redirections, les pipes, et l'exécution de commandes en arrière-plan. Ce rapport présente la logique sous-jacente à son implémentation.

# Fonctionnalités principales

## 1. Exécution des commandes simples

Les commandes simples (écrites sans redirection ni opérateurs) sont gérées grâce à la fonction run\_single\_command().

* La commande est divisée en arguments en utilisant strtok.
* Un processus enfant est créé avec fork() pour exécuter la commande.
* L'enfant utilise execvp() pour lancer la commande spécifiée, tandis que le processus parent attend sa terminaison via waitpid().

Si la commande se termine par &, elle est exécutée en arrière-plan. Dans ce cas, le parent ne bloque pas sur waitpid() et retourne directement au prompt.

**Exemple :**

my\_shell> ls -l

## 2. Gestion des commandes multiples

La fonction execute\_commands() permet d'exécuter plusieurs commandes sur une seule ligne. Les commandes peuvent être chaînées par des opérateurs comme ;, &&, ou || :

* **;** : Les commandes sont exécutées séquentiellement sans condition.
* **&&** : La commande suivante s'exécute uniquement si la précédente réussit (code de retour 0).
* **||** : La commande suivante s'exécute uniquement si la précédente échoue (code de retour différent de 0).

Chaque opérateur est traité dans une boucle via des appels à strsep.

**Exemple :**

my\_shell> mkdir test && cd test || echo "Erreur"

## 3. Gestion des redirections (<, >, >>, <<)

**Redirections standard**

La fonction handle\_redirection() gère les redirections suivantes :

* **>** : Redirige la sortie standard vers un fichier (crée ou écrase le fichier existant).
* **>>** : Redirige la sortie standard vers un fichier (ajoute à la fin du fichier existant).
* **<** : Redirige l'entrée standard depuis un fichier.

Lorsqu'une redirection est rencontrée dans les arguments d'une commande, elle est traitée comme suit :

* Le fichier est ouvert avec open().
* Le descripteur de fichier est dupliqué vers STDOUT\_FILENO (pour > et >>) ou STDIN\_FILENO (pour <) via dup2().

**Redirection << (heredoc)**

La redirection << permet de fournir une entrée multi-lignes jusqu'à un "terminateur" défini par l'utilisateur. La logique :

* Un pipe est créé pour transmettre les données saisies.
* Un processus enfant lit les lignes de l'utilisateur jusqu'à ce que le "terminateur" soit rencontré, et écrit ces lignes dans le pipe.
* Le parent redirige ensuite l'entrée standard (via dup2()) pour utiliser les données du pipe.

**Exemple :**

my\_shell> cat << EOF

> Ligne 1

> Ligne 2

> EOF

## 4. Gestion des pipes (|)

**Logique actuelle**

La fonction handle\_pipes() permet d'exécuter des commandes chaînées par des pipes. Chaque commande dans la chaîne transmet sa sortie à l'entrée de la suivante via un pipe.

**Processus :**

1. La commande complète est divisée en sous-commandes avec strtok sur le caractère |.
2. Pour chaque sous-commande :
   * Un pipe est créé avec pipe().
   * Un processus enfant est créé avec fork().
   * L'enfant redirige sa sortie standard vers le pipe.
   * Le parent redirige son entrée standard vers le pipe pour la prochaine sous-commande.
3. Le processus parent attend la fin de tous les enfants via waitpid().

**Exemple :**

my\_shell> ls | grep "main" | sort

## 5. Commandes internes

Les commandes internes sont gérées dans handle\_builtin\_commands(). Cette fonction traite directement certaines commandes sans passer par execvp() :

* **cd** : Change le répertoire courant avec chdir().
* **pwd** : Affiche le répertoire courant avec getcwd().
* **exit** : Termine le programme avec exit().
* **history** : Affiche l'historique des commandes sauvegardées dans le fichier history.txt.

**Exemple :**

my\_shell> cd /home

my\_shell> pwd

/home

## 6. Gestion de l'exécution en arrière-plan (&)

Si une commande se termine par &, elle est exécutée en arrière-plan. Dans ce cas :

* Le parent ne bloque pas sur waitpid() et retourne directement au prompt.
* L'identifiant du processus enfant (PID) est affiché.

**Exemple :**

my\_shell> sleep 10 &

[Processus en arrière-plan lancé : PID 1234]

# Conclusion

MiniShell implémente une logique robuste pour gérer les commandes système et internes avec des fonctionnalités avancées comme les redirections, les pipes, et l'exécution en arrière-plan. La gestion des opérateurs conditionnels (&&, ||), des redirections complexes (<<), et des pipes multiples permet une expérience proche des shells modernes comme Bash. Cependant, certaines optimisations peuvent encore être apportées pour améliorer les performances et gérer les cas limites.