Rapport de projet : minishell

Assala ASSELLALOU
2A - Parcours B - ENSEEIHT

Introduction

Ce rapport présente la validation progressive des fonctionnalités du projet **minishell**, en suivant les étapes du sujet. Chaque étape est illustrée par une capture d'écran du terminal pour montrer le comportement observé.

1 Méthodologie de tests

Chaque fonctionnalité a été testée séparément à l'aide de commandes simples et de cas limites (commandes inexistantes, redirections invalides, signaux clavier). Les tests ont été réalisés en ligne de commande, avec des captures d'écran pour les étapes clés.

2 Étape 2 : Exécution d'une commande simple

Cette étape consiste à exécuter une commande saisie par l'utilisateur (par exemple `ls`). J'ai implémenté la création d'un processus fils avec **fork()**. Le processus fils exécute la commande saisie par l'utilisateur grâce à **execvp()** .

Figure 1: Exécution d'une commande simple

Cependant comme vous pouvez le remarquer dans la figure 1, on a pas encore la main après l'exécution de la commande. C'est ce qu'on va traiter dans la prochaine étape.

3 Étape 3 : Attente de la terminaison du fils

Après avoir lancé la commande, le shell attend la fin du processus fils avant de rendre la main à l'utilisateur grâce à **waitpid()**, on peut aussi utiliser **wait()**.

Figure 2: Attente de la terminaison du fils

4 Étape 4 : Exécution en tâche de fond

Si la commande se termine par **&**, elle s'exécute en tâche de fond et le shell ne l'attend pas. Pour réaliser cela on met le **waitpid()** à l'interieur d'une **clause if** sous la condition suivante : `(commande->backgrounded == NULL)` pour le cas des commandes en avant-plan.

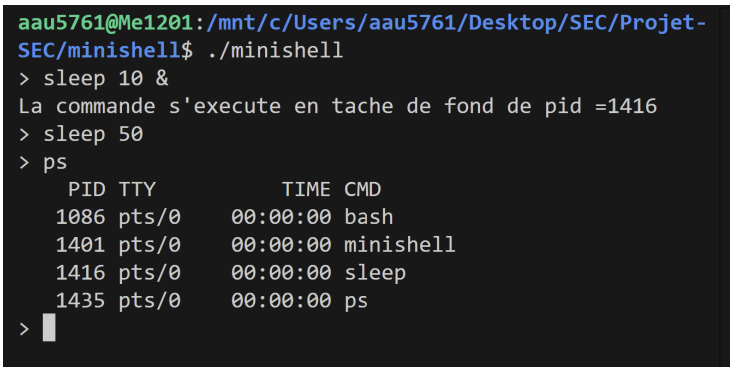

Figure 3: Commande en tâche de fond

5 Étape 5 : Traitement du signal SIGCHLD

On gère le signal SIGCHLD pour détecter la terminaison des processus fils. Pour cela on ecrit la procédure **traitement_sigchld**.

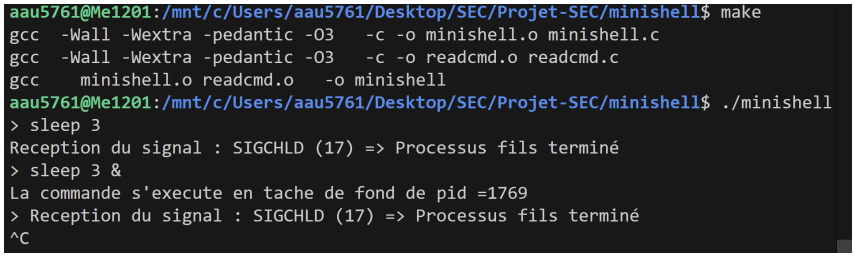

Figure 4: Traitement du signal SIGCHLD

6 Étape 6 : Gestion avancée de SIGCHLD

On affiche un message à chaque terminaison de fils, en utilisant `waitpid(-1, &status, WNOHANG | WUNTRACED | WCONTINUED)` dans la nouvelle procédure **traitement_sigchld**.

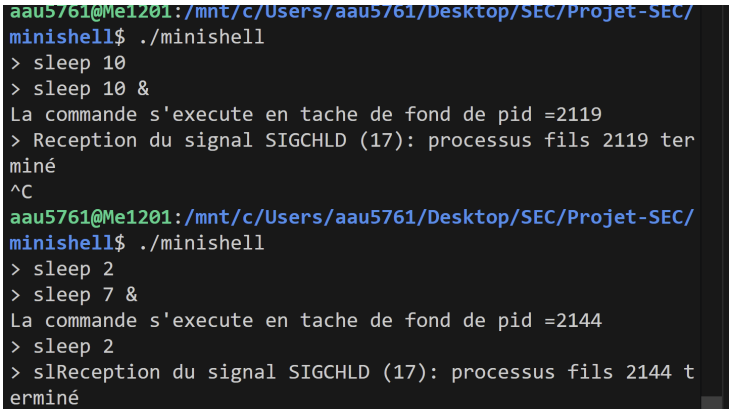

Figure 5: Gestion avancée de SIGCHLD

7 Étape 7 : Attente d'un signal

On utilise `pause()` pour attendre la terminaison des fils et **waitpid()** est désormais mit en commentaire. Et puis on test le fonctionnement.

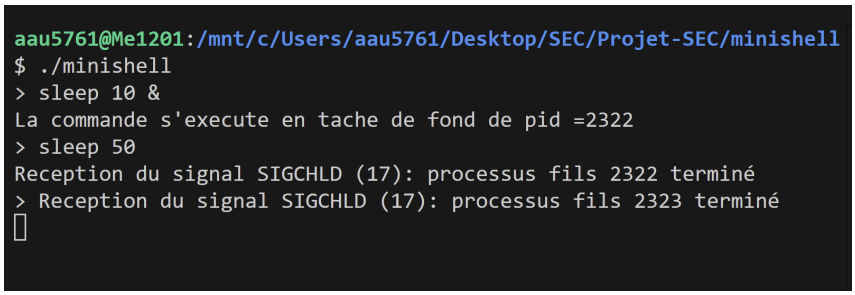

Figure 6: Utilisation de pause()

8 Étape 8 : Gestion des signaux Ctrl-C et Ctrl-Z

Il est demandé de tester l'envoi des signaux SIGSTOP et SIGCONT vers un processus en arrière-plan.
On lance : **sleep 120 &** dans un premier terminal et on recupere son **pid = 2646**.
Dans un deuxieme terminal on envoi le signal SIGSTOP avec : **kill -STOP 2646** et le signal SIGCONT avec : **kill -CONT 2646** . On verifie les processus par la commande ps.

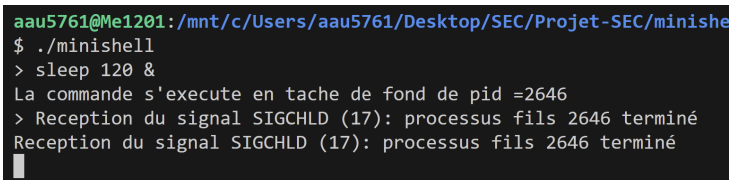

Figure 7: Gestion de Ctrl-C et Ctrl-Z

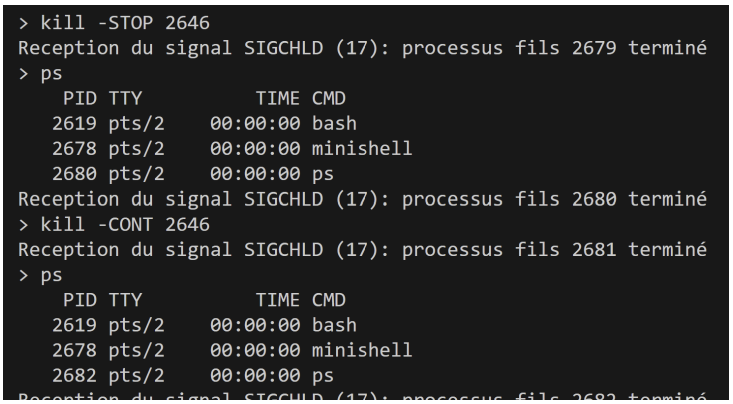

Figure 8: Vérification des processus avec ps

9 Étape 9 : Affichage du signal reçu

On affiche le signal ayant terminé le processus fils : **terminé normalement, repris, ou suspendu.**

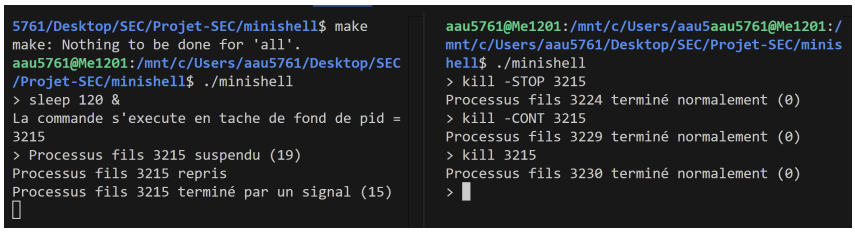

Figure 9: Affichage du signal reçu

10 Étape 10 : Gestion personnalisée des signaux

A cette étape, on personnalise le traitement des signaux pour afficher un message lors de Ctrl-C ou Ctrl-Z.

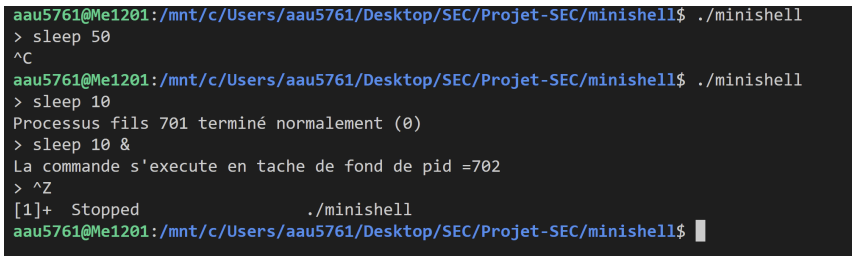

Figure 10: Gestion personnalisée des signaux

11 Étape 11 : Variantes de gestion des signaux

11.1 : traitement personnalisé

Gestionnaire personnalisé qui affiche un message : **Ctrl-C ignoré** ou **Ctrl-Z ignoré**.

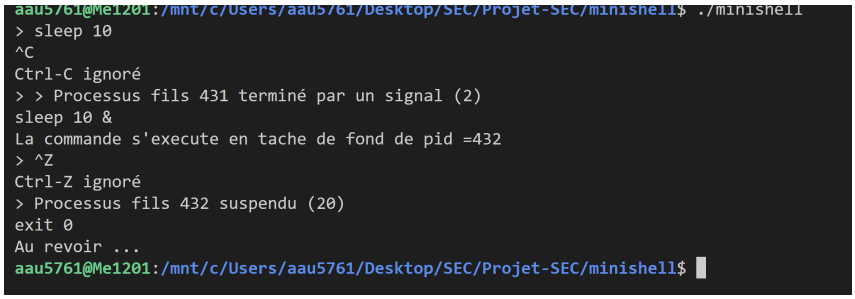

Figure 11: Traitement personnalisé pour Ctrl-C et Ctrl-Z

11.2 : Ignorer les signaux

On ignore complètement les signaux avec **SIG_IGN**.

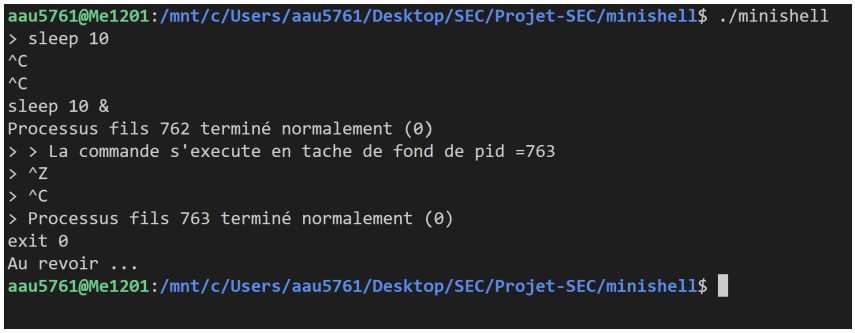

Figure 12: Ignorer SIGINT et SIGTSTP

11.3 : Masquer les signaux

On masque les signaux avec **sigprocmask** pour les bloquer temporairement.

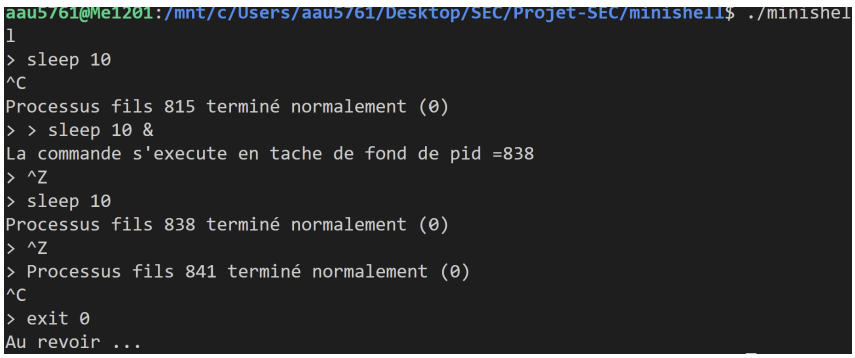

Figure 13: Masquer SIGINT et SIGTSTP

12 Étape 12 : Détacher les processus en arrière-plan

Les processus lancés en arrière-plan sont détachés du terminal avec **setpgrp()** pour éviter qu'ils ne reçoivent les signaux clavier destinés au shell.

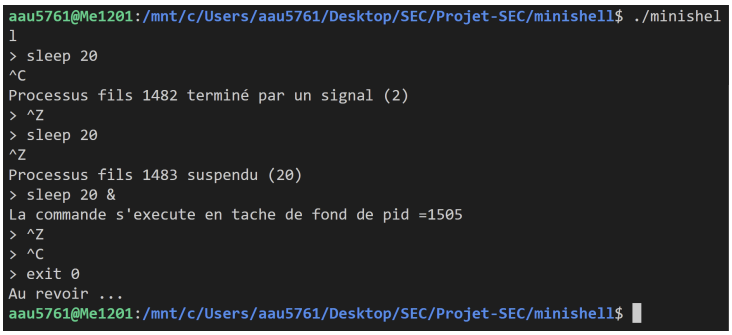

Figure 14: Détachement des processus en arrière-plan

13 Étape 13 : Redirections d'entrée et de sortie

Consiste à permettre d'associer l'entrée standard ou la sortie standard d'une commande à un fichier en utilisant **dup2()** et des descripteurs.

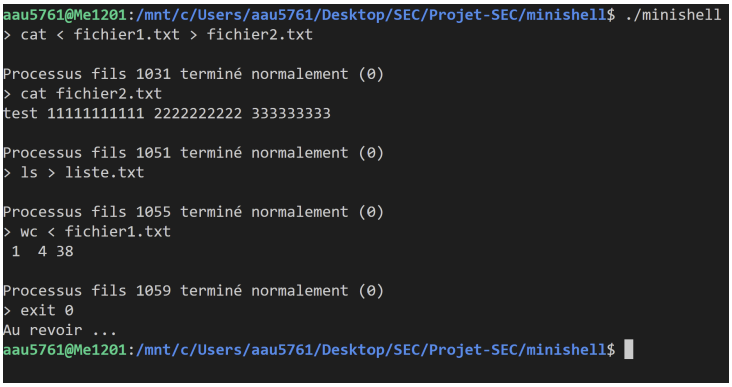

Figure 15: Redirections d'entrée et de sortie

14 Étape 14 : Commande interne cd

On écrit la commande interne `cd` pour changer de répertoire directement depuis le shell.en distinguant deux cas, et en utilisant **chdir()**.

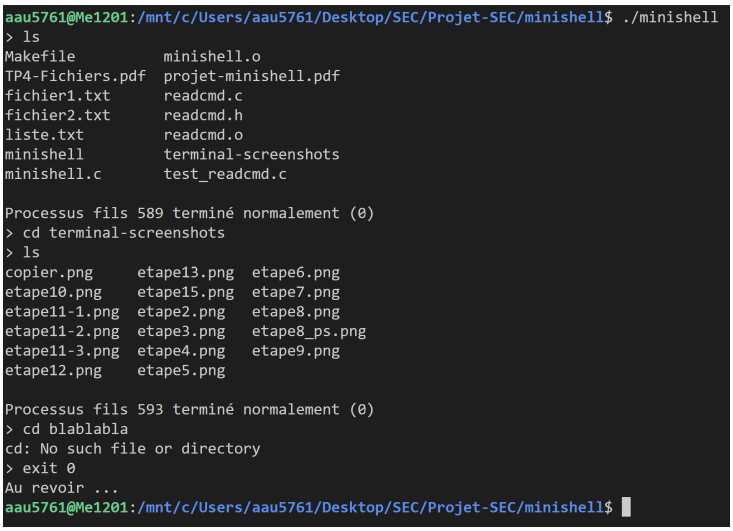

Figure 16: Commande interne cd

15 Étape 15 : Commande interne dir

J'ai ajouté la commande interne **dir** pour afficher le contenu d'un répertoire, en utilisant **opendir()**, **readdir()** et **closedir()**.

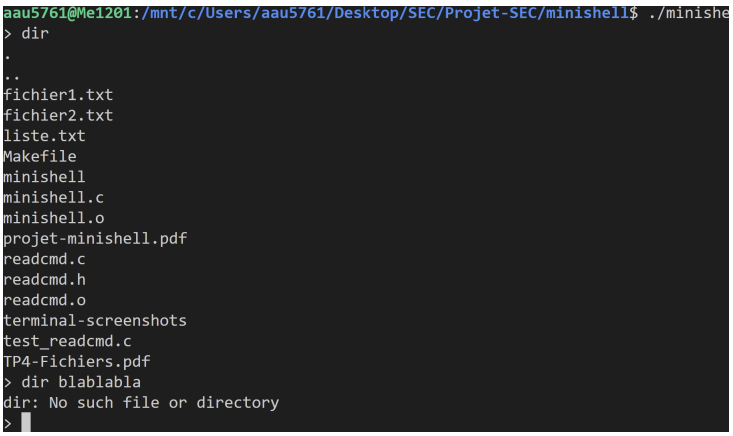

Figure 17: Commande interne dir

16 Étape 16 : Tubes simples

On implémente la gestion des tubes entre deux commandes, permettant de connecter la sortie standard d'une commande à l'entrée standard d'une autre, en utilisant **pipe()** et la création de deux processus fils.

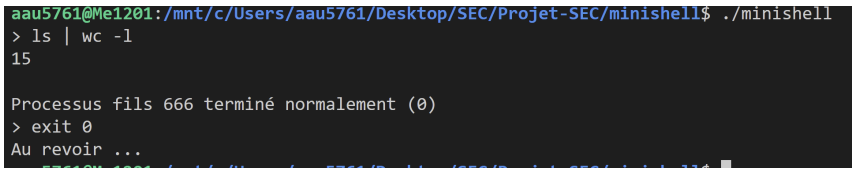

Figure 18: Tubes simples