

# Rapport de projet : minishell

Assala ASSELLALOU  
2A - Parcours B - ENSEEIHT

## Introduction

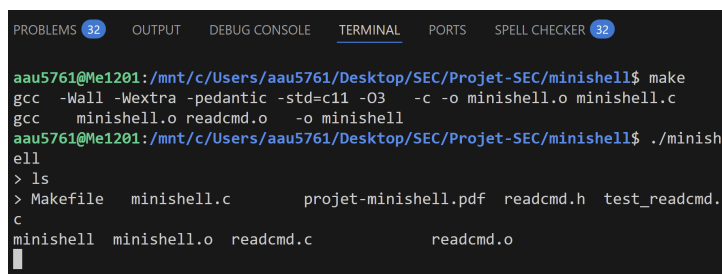
Ce rapport présente la validation progressive des fonctionnalités du projet **minishell**, en suivant les étapes du sujet. Chaque étape est illustrée par une capture d'écran du terminal pour montrer le comportement observé.

## 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()`.



```
PROBLEMS 32 OUTPUT DEBUG CONSOLE TERMINAL PORTS SPELL CHECKER 32
aau5761@Me1201:/mnt/c/Users/aau5761/Desktop/SEC/Projet-SEC/minishell$ make
gcc -Wall -Wextra -pedantic -std=c11 -O3 -c -o minishell.o minishell.c
gcc minishell.o readcmd.o -o minishell
aau5761@Me1201:/mnt/c/Users/aau5761/Desktop/SEC/Projet-SEC/minishell$ ./minishell
>
> ls
> Makefile  minishell.c      projet-minishell.pdf  readcmd.h  test_readcmd.c
minishell  minishell.o  readcmd.c            readcmd.o
```

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()`.

```

aau5761@Me1201:/mnt/c/Users/aau5761/Desktop/SEC/Projet-SEC/minishell$ make
gcc -Wall -Wextra -pedantic -std=c11 -O3 -c -o minishell.o minishell.c
gcc minishell.o readcmd.o -o minishell
aau5761@Me1201:/mnt/c/Users/aau5761/Desktop/SEC/Projet-SEC/minishell$ ./minishell
> ls
Makefile minishell.c projet-minishell.pdf readcmd.h terminal-screenshots
minishell minishell.o readcmd.c readcmd.o test_readcmd.c
> █

```

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'intérieur d'une **clause if** sous la condition suivante : (`commande->backgrounded == NULL`) pour le cas des commandes en avant-plan.

```

aau5761@Me1201:/mnt/c/Users/aau5761/Desktop/SEC/Projet-SEC/minishell$ ./minishell
> sleep 10 &
La commande s'exécute en tâche de fond de pid =1416
> sleep 50
> ps
  PID TTY          TIME CMD
 1086 pts/0    00:00:00 bash
  1401 pts/0    00:00:00 minishell
  1416 pts/0    00:00:00 sleep
  1435 pts/0    00:00:00 ps
> █

```

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 écrit la procédure `traitement_sigchld`.

```

aau5761@Me1201:/mnt/c/Users/aau5761/Desktop/SEC/Projet-SEC/minishell$ make
gcc -Wall -Wextra -pedantic -O3 -c -o minishell.o minishell.c
gcc -Wall -Wextra -pedantic -O3 -c -o readcmd.o readcmd.c
gcc minishell.o readcmd.o -o minishell
aau5761@Me1201:/mnt/c/Users/aau5761/Desktop/SEC/Projet-SEC/minishell$ ./minishell
> sleep 3
Reception du signal : SIGCHLD (17) => Processus fils terminé
> sleep 3 &
La commande s'exécute en tâche de fond de pid =1769
> Reception du signal : SIGCHLD (17) => Processus fils terminé
^C

```

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`.

```

aau5761@Me1201:/mnt/c/Users/aau5761/Desktop/SEC/Projet-SEC/
minishell$ ./minishell
> sleep 10
> sleep 10 &
La commande s'exécute en tâche de fond de pid =2119
> Reception du signal SIGCHLD (17): processus fils 2119 ter
miné
^C
aau5761@Me1201:/mnt/c/Users/aau5761/Desktop/SEC/Projet-SEC/
minishell$ ./minishell
> sleep 2
> sleep 7 &
La commande s'exécute en tâche de fond de pid =2144
> sleep 2
> slReception du signal SIGCHLD (17): processus fils 2144 t
erminé

```

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.

```

aau5761@Me1201:/mnt/c/Users/aau5761/Desktop/SEC/Projet-SEC/minishell
$ ./minishell
> sleep 10 &
La commande s'exécute en tâche de fond de pid =2322
> sleep 50
Reception du signal SIGCHLD (17): processus fils 2322 terminé
> Reception du signal SIGCHLD (17): processus fils 2323 terminé

```

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`.

```

aau5761@Me1201:/mnt/c/Users/aau5761/Desktop/SEC/Projet-SEC/minishe
$ ./minishell
> sleep 120 &
La commande s'exécute en tâche de fond de pid =2646
> Reception du signal SIGCHLD (17): processus fils 2646 terminé
Reception du signal SIGCHLD (17): processus fils 2646 terminé

```

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

```

> kill -STOP 2646
Reception du signal SIGCHLD (17): processus fils 2679 terminé
> ps
  PID TTY          TIME CMD
  2619 pts/2        00:00:00 bash
  2678 pts/2        00:00:00 minishell
  2680 pts/2        00:00:00 ps
Reception du signal SIGCHLD (17): processus fils 2680 terminé
> kill -CONT 2646
Reception du signal SIGCHLD (17): processus fils 2681 terminé
> ps
  PID TTY          TIME CMD
  2619 pts/2        00:00:00 bash
  2678 pts/2        00:00:00 minishell
  2682 pts/2        00:00:00 ps
Reception du signal SIGCHLD (17): processus fils 2682 terminé

```

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**.

```

5761/Desktop/SEC/Projet-SEC/minishell$ make
make: Nothing to be done for 'all'.
aau5761@Me1201:/mnt/c/Users/aau5761/Desktop/SEC/Projet-SEC/minishell$ ./minishell
> sleep 120 &
La commande s'exécute en tâche de fond de pid = 3215
> Processus fils 3215 suspendu (19)
Processus fils 3215 repris
Processus fils 3215 terminé par un signal (15)
[]

aau5761@Me1201:/mnt/c/Users/aau5761/Desktop/SEC/Projet-SEC/minishell$ ./minishell
> kill -STOP 3215
Processus fils 3224 terminé normalement (0)
> kill -CONT 3215
Processus fils 3229 terminé normalement (0)
> kill 3215
Processus fils 3230 terminé normalement (0)
>

```

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.

```

aau5761@Me1201:/mnt/c/Users/aau5761/Desktop/SEC/Projet-SEC/minishell$ ./minishell
> sleep 50
^C
aau5761@Me1201:/mnt/c/Users/aau5761/Desktop/SEC/Projet-SEC/minishell$ ./minishell
> sleep 10
Processus fils 701 terminé normalement (0)
> sleep 10 &
La commande s'exécute en tâche de fond de pid = 702
> ^Z
[1]+  Stopped                  ./minishell
aau5761@Me1201:/mnt/c/Users/aau5761/Desktop/SEC/Projet-SEC/minishell$

```

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**.

```

aau5761@Me1201:/mnt/c/Users/aau5761/Desktop/SEC/Projet-SEC/minishell$ ./minishell
> sleep 10
^C
Ctrl-C ignoré
> > Processus fils 431 terminé par un signal (2)
sleep 10 &
La commande s'exécute en tâche de fond de pid =432
> ^Z
Ctrl-Z ignoré
> Processus fils 432 suspendu (20)
exit 0
Au revoir ...
aau5761@Me1201:/mnt/c/Users/aau5761/Desktop/SEC/Projet-SEC/minishell$

```

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

## 11.2 : Ignorer les signaux

On ignore complètement les signaux avec **SIG\_IGN**.

```

aau5761@Me1201:/mnt/c/Users/aau5761/Desktop/SEC/Projet-SEC/minishell$ ./minishell
> sleep 10
^C
^C
sleep 10 &
Processus fils 762 terminé normalement (0)
> > La commande s'exécute en tâche de fond de pid =763
> ^Z
> ^C
> Processus fils 763 terminé normalement (0)
exit 0
Au revoir ...
aau5761@Me1201:/mnt/c/Users/aau5761/Desktop/SEC/Projet-SEC/minishell$

```

Figure 12: Ignorer SIGINT et SIGTSTP

## 11.3 : Masquer les signaux

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

```

aau5761@Me1201:/mnt/c/Users/aau5761/Desktop/SEC/Projet-SEC/minishell$ ./minishell
1
> sleep 10
^C
Processus fils 815 terminé normalement (0)
> > sleep 10 &
La commande s'exécute en tâche de fond de pid =838
> ^Z
> sleep 10
Processus fils 838 terminé normalement (0)
> ^Z
> Processus fils 841 terminé normalement (0)
^C
> exit 0
Au revoir ...

```

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.

```

aau5761@Me1201:/mnt/c/Users/aau5761/Desktop/SEC/Projet-SEC/minishell$ ./minishell
1
> sleep 20
^C
Processus fils 1482 terminé par un signal (2)
> ^Z
> sleep 20
^Z
Processus fils 1483 suspendu (20)
> sleep 20 &
La commande s'exécute en tâche de fond de pid =1505
> ^Z
> ^C
> exit 0
Au revoir ...
aau5761@Me1201:/mnt/c/Users/aau5761/Desktop/SEC/Projet-SEC/minishell$

```

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.

```

aau5761@Me1201:/mnt/c/Users/aau5761/Desktop/SEC/Projet-SEC/minishell$ ./minishell
> cat < fichier1.txt > fichier2.txt

Processus fils 1031 terminé normalement (0)
> cat fichier2.txt
test 1111111111 2222222222 3333333333

Processus fils 1051 terminé normalement (0)
> ls > liste.txt

Processus fils 1055 terminé normalement (0)
> wc < fichier1.txt
 1  4 38

Processus fils 1059 terminé normalement (0)
> exit 0
Au revoir ...
aau5761@Me1201:/mnt/c/Users/aau5761/Desktop/SEC/Projet-SEC/minishell$

```

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()**.

```

aau5761@Me1201:/mnt/c/Users/aau5761/Desktop/SEC/Projet-SEC/minishell$ ./minishell
> ls
Makefile      minishell.o
TP4-Fichiers.pdf  projet-minishell.pdf
fichier1.txt   readcmd.c
fichier2.txt   readcmd.h
liste.txt      readcmd.o
minishell      terminal-screenshots
minishell.c    test_readcmd.c

Processus fils 589 terminé normalement (0)
> cd terminal-screenshots
> ls
copier.png    etape13.png  etape6.png
etape10.png   etape15.png  etape7.png
etape11-1.png etape2.png   etape8.png
etape11-2.png etape3.png   etape8_ps.png
etape11-3.png etape4.png   etape9.png
etape12.png   etape5.png

Processus fils 593 terminé normalement (0)
> cd blablaba
cd: No such file or directory
> exit 0
Au revoir ...
aau5761@Me1201:/mnt/c/Users/aau5761/Desktop/SEC/Projet-SEC/minishell$

```

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()**.

```
aau5761@Me1201:/mnt/c/Users/aau5761/Desktop/SEC/Projet-SEC/minishell$ ./minishell
> dir
.
..
fichier1.txt
fichier2.txt
liste.txt
Makefile
minishell
minishell.c
minishell.o
projet-minishell.pdf
readcmd.c
readcmd.h
readcmd.o
terminal-screenshots
test_readcmd.c
TP4-Fichiers.pdf
> dir blablabla
dir: No such file or directory
>
```

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.

```
aau5761@Me1201:/mnt/c/Users/aau5761/Desktop/SEC/Projet-SEC/minishell$ ./minishell
> ls | wc -l
15
Processus fils 666 terminé normalement (0)
> exit 0
Au revoir ...
```

Figure 18: Tubes simples