



Projet Minishell

Oussakel Rachida

Département Sciences du numérique – Première année
2021-2022

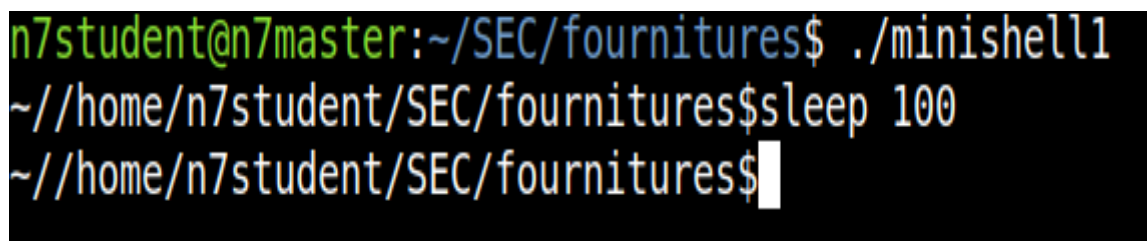
Introduction :

L'objectif du travail présenté dans ce rapport est de réaliser un 'Minishell' offrant les fonctionnalités de base d'une Shell UNIX, tout en se basant sur les notions déjà vues en TP englobant le traitement des signaux, la gestion des processus et des tubes.

Boucle principale de mon programme :

En premier temps, j'étais emmenée à réaliser une boucle basique comme vu au 2ème TP qui permet d'interpréter ce qui est entré en ligne de commande tout en se basant sur le module 'Readcmd' qui offre des fonctionnalités pour la lecture et traitement d'une commande tapée par l'utilisateur.

Ce processus Shell lance un fils, puis se met immédiatement en attente de lecture de la prochaine ligne de commande, ce comportement est bien mis en évidence sur la capture d'écran ci-dessous :



```
n7student@n7master:~/SEC/fournitures$ ./minishell1
~//home/n7student/SEC/fournitures$sleep 100
~//home/n7student/SEC/fournitures$
```

Quand on lance un nouveau processus, 'sleep x' par exemple, on voit que le Minishell donne la main à l'utilisateur immédiatement après le lancement pour pouvoir entrer d'autres commandes, ce qui sera traité par la suite dans la 5 - ème question .

En deuxième temps, j'ai ajouté les commandes internes telles que la commande **cd** (change directory) qui permet de changer le répertoire courant, ce répertoire peut être exprimé selon plusieurs formes :

- * le chemin relatif : ce chemin représente un répertoire à partir du répertoire courant

- * **~** : représente le répertoire de connexion de l'utilisateur.

- * **cd** tout court permettra aussi de réaliser la fonctionnalité de '**cd ~**'

Ainsi que la commande **exit** qui termine la session courante de l'utilisateur.

Traitement des commandes en BACKGROUND :

En effet, si un utilisateur lance par exemple un processus à l'aide de la commande '**sleep 100**', celui-ci bloque l'invité de commande et par conséquent, le système attend la fin de la tâche lancée. Alors qu'en lançant le processus en tâche de fond à l'aide du caractère **&** à la fin de la commande, l'interpréteur de commandes réussira à exécuter le programme tout en redonnant la main, ce qui permettra d'exécuter une nouvelle ligne de commande :

```
n7student@n7master:~/SEC/fournitures$ ./minishell
```

```
~/home/n7student/SEC/fournitures$ sleep 100
```

```
~/home/n7student/SEC/fournitures$ sleep 5&
```

```
~/home/n7student/SEC/fournitures$ ls
```

```
[1]      pid: 716245      Actif(BG)      sleep
[2]      pid: 716373      Actif(BG)      sleep
[3]      pid: 716393      Actif(BG)      sleep
```

```
~/home/n7student/SEC/fournitures$
```

Gestion des processus lancés depuis le Shell :

Avant toute implémentation des fonctions qui permettent la gestion des processus, il fallait réfléchir à une structure pour englober l'ensemble des processus du Shell courant (**Jobs**), pour se faire je me suis basée sur une liste simple même si celle-ci est couteuse en mémoire par rapport à une liste chaînée, mais elle reste plus facile à gérer et moins couteuse en temps en termes d'accès aux différents Jobs.

Pour les éléments de ma liste c'étaient des jobs caractérisaient par leurs **id**, **pid**, **état** (Actif en BG, Actif en FG, Suspendu) et la ligne de commande lancée.

En se basant sur la structure précédente, j'ai pu implémenter les quatre fonctionnalités demandées.

Pour tester ces dernières, j'ai lancé un processus en arrière-plan 'sleep 10&', et avec la commande **fg(foreground)** j'ai pu continuer son exécution en avant plan, ainsi que de le suspendre avec la commande **Sj (Stop job)**, tout en affichant chaque fois l'ensemble des processus avec la commande **lj (list jobs)** pour vérifier leurs états.

```
~//home/n7student/SEC/fournitures$lj
[1]      pid: 716245      Actif(BG)      sleep
[2]      pid: 716373      Actif(BG)      sleep
[3]      pid: 716393      Actif(BG)      sleep
[4]      pid: 716548      Actif(BG)      sleep

~//home/n7student/SEC/fournitures$fg 1
sleep

~//home/n7student/SEC/fournitures$lj
[2]      pid: 716373      Actif(BG)      sleep
[3]      pid: 716393      Actif(BG)      sleep
[4]      pid: 716548      Actif(BG)      sleep

~//home/n7student/SEC/fournitures$$sj 2

~//home/n7student/SEC/fournitures$lj
[2]      pid: 716373      Suspendu      sleep
[3]      pid: 716393      Actif(BG)      sleep
[4]      pid: 716548      Actif(BG)      sleep
```

Contrôle des processus en avant plan :

Pour traiter la frappe ctrl-c qui envoie un signal SIGINT, j'ai défini un traitant handlerSIGINT (int signal) qui envoie le signal SIGKILL au dernier processus lancé en foreground. Ce qui interrompe l'exécution de ce dernier et le supprime de la liste des processus par la suite, ce comportement est ainsi montré sur la figure ci-dessous :

```
n7student@n7master:~/SEC/fournitures$ ./minishell
~//home/n7student/SEC/fournitures$ sleep 10
^C
~//home/n7student/SEC/fournitures$ lj
~//home/n7student/SEC/fournitures$
```

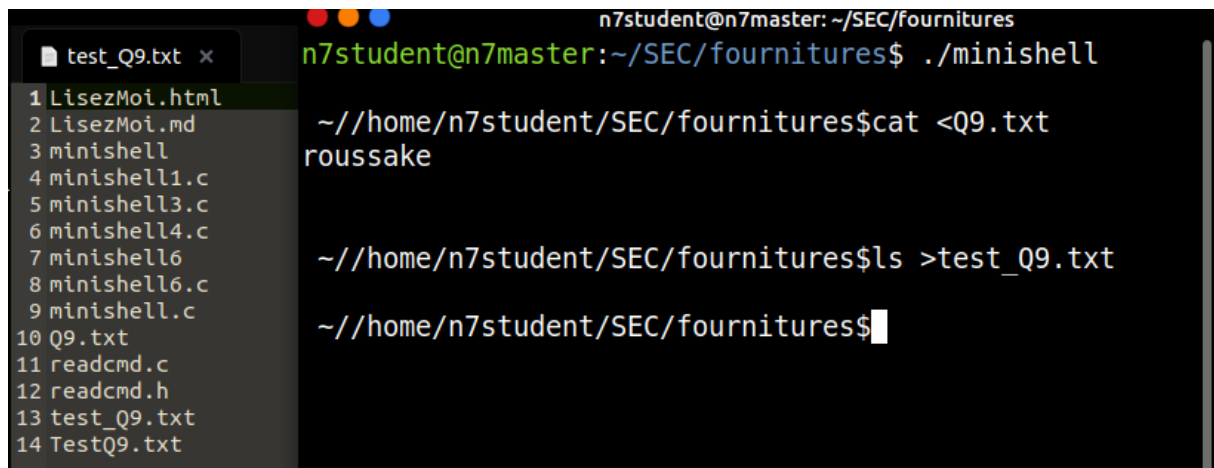
Tandis que la frappe ctrl-z était partiellement traitée, j'ai fait ignorer alors le signal SIGSTP aux fils avec la commande signal (SIGSTP, SIG_IGN).

Gestion des redirections :

Cette partie était consacrée à l'association d'une sortie/entrée standards à un fichier, où j'ai traité les trois cas :

- **Commande > fichier** : Exécute la commande et redirige sa sortie standard dans le fichier en écrasant son contenu ou en le créant s'il n'existe pas.
- **Commande < fichier** : Exécute la commande en passant le contenu du fichier dans son entrée standard.
- **Commande <fichier1 > fichier2** : Redirige le résultat du fichier1 vers le fichier2.

Afin de tester que les redirections fonctionnent correctement, j'ai créé en premier temps un fichier Q9.txt contenant le texte (roussake), et par la suite j'ai lancé la commande 'cat < Q9.txt' ce qui a conduit à l'affichage du contenu du fichier au niveau de la console. Ainsi, j'ai lancé la commande 'ls>test_Q9.txt' ce qui a permis d'afficher tous les fichiers se trouvant dans le répertoire courant dans le fichier 'test_Q9.txt' comme envisagé sur la figure ci-dessous :



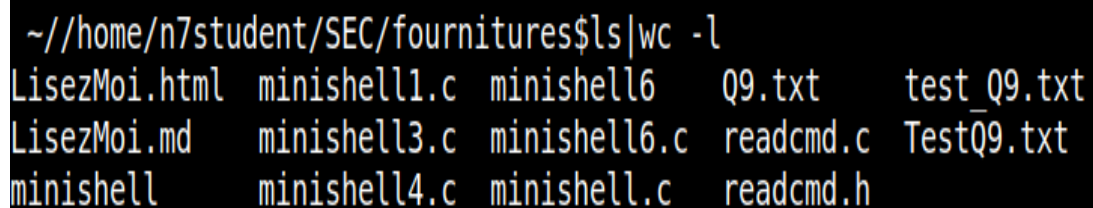
The screenshot shows a terminal window with a file explorer on the left and a terminal on the right. The file explorer lists files: 1 LisezMoi.html, 2 LisezMoi.md, 3 minishell, 4 minishell1.c, 5 minishell3.c, 6 minishell4.c, 7 minishell6, 8 minishell6.c, 9 minishell.c, 10 Q9.txt, 11 readcmd.c, 12 readcmd.h, 13 test_Q9.txt, 14 TestQ9.txt. The terminal shows the following commands and output:

```
n7student@n7master: ~/SEC/fournitures
n7student@n7master:~/SEC/fournitures$ ./minishell
~/home/n7student/SEC/fournitures$ cat <Q9.txt
roussake
~/home/n7student/SEC/fournitures$ ls >test_Q9.txt
~/home/n7student/SEC/fournitures$
```

Gestion des tubes :

- Tubes simples :

Pour pouvoir traiter les commandes à deux séquences de type (ls|wc -l), au début je vérifiais que **cmd->seq [1]** est non **NULL**, et donc je crée une pipe qui lie les deux commandes, où la première était exécutée par le fils, tout en redirigeant son résultat vers la pipe héritée de son père, qui exécute à son tour la deuxième partie de la commande.



The screenshot shows a terminal window with the following command and output:

```
~/home/n7student/SEC/fournitures$ ls|wc -l
LisezMoi.html minishell1.c minishell6 Q9.txt test_Q9.txt
LisezMoi.md minishell3.c minishell6.c readcmd.c TestQ9.txt
minishell minishell4.c minishell.c readcmd.h
```

- **Pipelines :**

Pour traiter cette question, j'ai modifié le code de la question précédente pour généraliser la méthodologie de traitement des pipes. Pour faire, j'ai implémenté une méthode récursive **traitementPipe (int p, int q, int nbPipe)** qui permet de créer un nouveau fils tant qu'on n'est pas arrivé à la dernière commande, tout en incrémentant le compteur des pipes.

Pour tester le bon fonctionnement de cette gestion de pipe, j'ai lancé la commande '**who|grep nom utilisateur| wc -l**' vu au cours de la séance du TP 6 :

```
n7student@n7master:~/SEC/fournitures$ ./minishell
~//home/n7student/SEC/fournitures$who|grep n7student |wc -l
n7student tty7          2022-05-25 22:21 (:0)
```