Rapport final projet Mini-Shell

RHAYOUTE ABDELMALEK

Introduction :

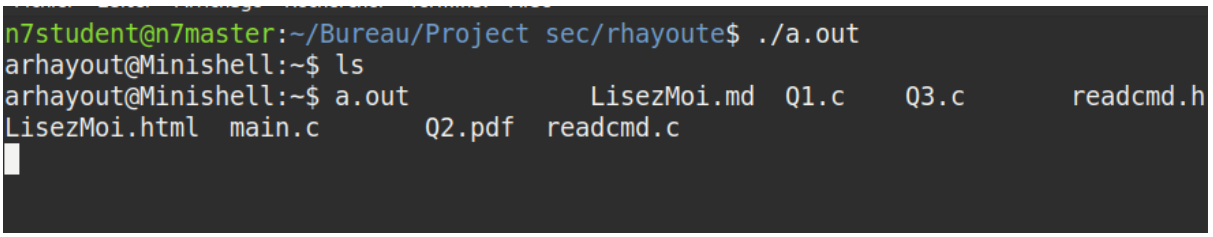
Ce projet a pour but la réalisation d'un Mini-Shell en mettant en pratique les notions de base vues en TD et TP, comme la gestion des processus, des signaux et même des entrées/sorties. Cet interpréteur de commandes développé à la fin offre plusieurs fonctionnalités de bases des shells comme celui de Bash, PowerShell, etc.

Questions :

Dans ce rapport, je vais répondre aux questions posées dans le sujet.
J'expliquerai les choix de conception et j'ajouterai des captures de test au fur et à mesure.
Partie 1 :

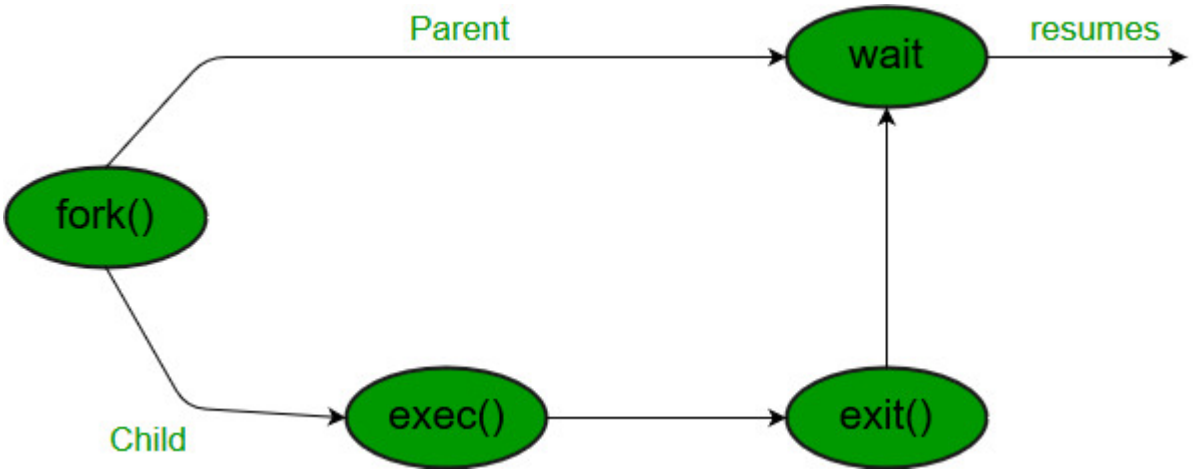
1-D'abord, une boucle infinie (while(1)) à l'intérieur de laquelle la création des processus était exécutée. Ensuite, on teste le retour de système et on ajoute certaines commandes.

2- Effectivement, l'affichage de l'invite se mêle à l'exécution du processus fils (L'invite de commande s'affiche avant la terminaison de la commande, donc le processus père (le minishell) n'attend pas la terminaison du fils (la commande exécutée)



3-Le processus parent n'attend pas son fils, donc j'ai utilisé wait(NULL) dans la partie exécutée par le père.

Pour être spécifique, on utilise waitpid()




```
n7student@n7master:~/Bureau/Project sec/rhayoute$ gcc -Wall -o executable readcmd.c
Q3.c
n7student@n7master:~/Bureau/Project sec/rhayoute$ ./executable
arhayout@Minishell:~$ ls
a.out       LisezMoi.html  main.c       Q1.c  readcmd.c
executable  LisezMoi.md    minishell.c  Q3.c  readcmd.h
Parent pid = 100444
Child pid = 100470
arhayout@Minishell:~$
```

4-Il était nécessaire de contrôler l'entrée de ces commandes même avant la création du processus fils avec une simple structure if... else if....

```
.:executable : commande introuvable
n7student@n7master:~/Bureau/Project sec/rhayoute$ ./executable
arhayout@Minishell:~$ ls
a.out       LisezMoi.html  main.c       Q1.c  Q4.c       readcmd.h
executable  LisezMoi.md    minishell.c  Q3.c  readcmd.c
Parent pid = 107645
Child pid = 107671
arhayout@Minishell:~$ cd
arhayout@Minishell:~$ cd
arhayout@Minishell:~$ exit
n7student@n7master:~/Bureau/Project sec/rhayoute$ ls
a.out       LisezMoi.html  main.c       Q1.c  Q4.c       readcmd.h
executable  LisezMoi.md    minishell.c  Q3.c  readcmd.c
n7student@n7master:~/Bureau/Project sec/rhayoute$ cd
n7student@n7master:~$
```

5- Si l'attribut Background de la structure cmdl n'est pas NULL, le processus père n'attend pas le processus fils exécutant la commande en tâche de fond.

```
n7student@n7master:~/Bureau/Project sec/rhayoute$ gcc -Wall -o executable readcmd.c
Q5.c
n7student@n7master:~/Bureau/Project sec/rhayoute$ ./executable
arhayout@Minishell:~$ ls &
arhayout@Minishell:~$ a.out         LisezMoi.html  main.c       Q1.c  Q4.c  readcmd.
c
executable  LisezMoi.md    minishell.c  Q3.c  Q5.c  readcmd.h
^C
n7student@n7master:~/Bureau/Project sec/rhayoute$ ./executable
arhayout@Minishell:~$ ls
a.out       LisezMoi.html  main.c       Q1.c  Q4.c  readcmd.c
executable  LisezMoi.md    minishell.c  Q3.c  Q5.c  readcmd.h
Parent pid = 122859
Child pid = 122889
arhayout@Minishell:~$ cd &
```

Partie 2 :

Pour mieux visualiser les tests, j'ai ajouté des couleurs à l'affichage

6-

Pour pouvoir stocker les informations sur chaque processus lancé, j'ai créé le type job_t et les fonctions de manipulation dans un fichier séparé "job.c"

```
n7student@n7master:~/Bureau/Project sec/rhayoute$ ./executable
arhayout@Minishell:~$ ls
 a.out                       job.c           main.c        Q3.c   Q6.c
'Document 1 non enregistré'  LisezMoi.html   minishell.c   Q4.c   readcmd.c
 executable                  LisezMoi.md     Q1.c          Q5.c   readcmd.h
arhayout@Minishell:~$ list
Id         PID            État           ligne de commande lancée
[1]      160145           Actif(fg)           ls
arhayout@Minishell:~$ fg 160145
fg : Processus non trouvé.
arhayout@Minishell:~$ ls 1
ls: impossible d'accéder à '1': Aucun fichier ou dossier de ce type
arhayout@Minishell:~$ bg 1
arhayout@Minishell:~$ list
Id         PID            État           ligne de commande lancée
[1]      160145           Actif(bg)           ls
[2]      160524           Actif(fg)           ls
arhayout@Minishell:~$ fg 2
ls
arhayout@Minishell:~$ ls
 a.out                       job.c           main.c        Q3.c   Q6.c
'Document 1 non enregistré'  LisezMoi.html   minishell.c   Q4.c   readcmd.c
 executable                  LisezMoi.md     Q1.c          Q5.c   readcmd.h
arhayout@Minishell:~$ list
Id         PID            État           ligne de commande lancée
[1]      160145           Actif(bg)           ls
[2]      161023           Actif(fg)           ls
arhayout@Minishell:~$
```

Les signaux :(C'est la partie ma plus délicat)

J'ai pas arrivé a relever tous les défis et il m'en reste pas de temps

```
"je suis plein"
arhayout@Minishell:~$ lj
Id         PID            État           ligne de commande lancée
[1]      619006           Actif(fg)           ▯▯▯ U
[2]      619233           Actif(fg)
arhayout@Minishell:~$ ^Z
Erreur de segmentation (core dumped)
n7student@n7master:~/Bureau/rhayoute$ ./executable
arhayout@Minishell:~$ sleep 100&
[1]       619720
arhayout@Minishell:~$ ^Zaucune processus en fg.

Erreur de segmentation (core dumped)
n7student@n7master:~/Bureau/rhayoute$ ls
a.out  executable  f1  f2  job.c  LisezMoi.html  LisezMoi.md  lulu.c  main.c  Q10.c  Q11.c  Q11_v2.c  Q1.c  Q3
n7student@n7master:~/Bureau/rhayoute$ ./executable
arhayout@Minishell:~$ sleep 100
^Zaucune processus en fg.
^C
^Zaucune processus en fg.
```

7-Ctrl-Z

Pour traiter la frappe ctrl-z, j'ai défini le handler **handler_SIGTSTP.** Mais ça ne marche pas

8-Ctrl-C

Pour traiter la frappe ctrl-c, j'ai défini le handler **handler_SIGINT.** Mais ça ne marche pas trop

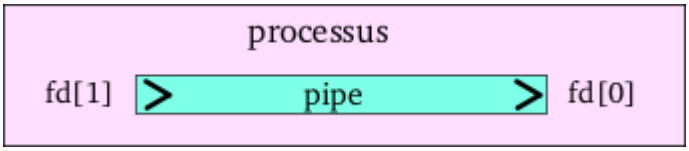
9-Redirections

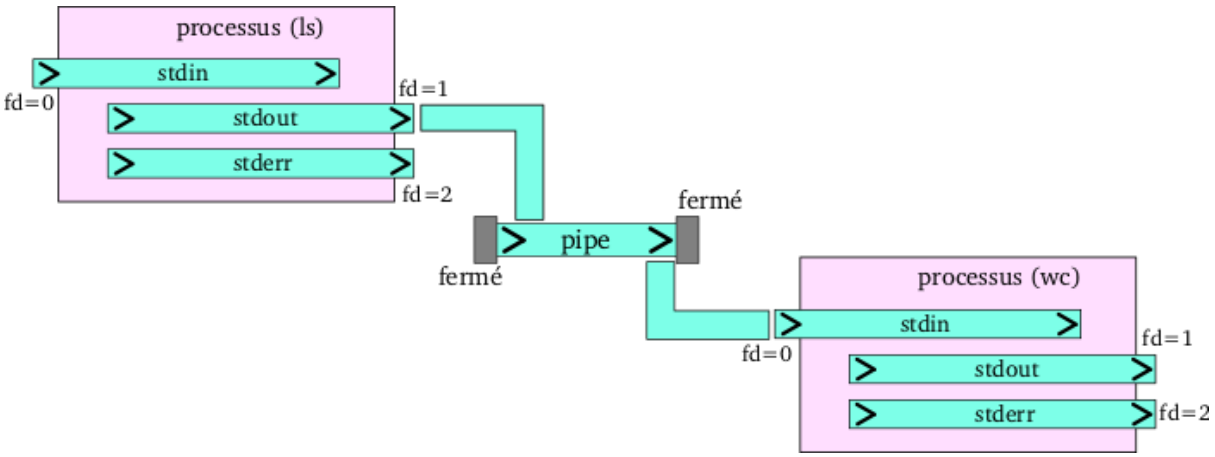
La redirection est faite en testant la commande (cmd->in == null) et (cmd->out == null) en changeant alors l'entrée et la sortie standard.

```
arhayout@Minishell:~$ echo "je suis plein" > f1
arhayout@Minishell:~$ cat f1
"je suis plein"
arhayout@Minishell:~$ cat < f1 > f2
arhayout@Minishell:~$ cat f2
"je suis plein"
arhayout@Minishell:~$
```

10-Tubes simples

Pour traiter deux commandes simultanément, j'ai utilisé une seule pipe, où le père exécute la 2e commande et le fils exécute la première commande.





Test

```
n7student@n7master:~/Bureau/rhayoute$ gcc -g  -Wall -o executable readcmd.c Q10.c
n7student@n7master:~/Bureau/rhayoute$ valgrind --leak-check=full --show-leak-kinds=all ./executable
==158976== Memcheck, a memory error detector
==158976== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==158976== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==158976== Command: ./executable
==158976==
arhayout@Minishell:~$ ls
Ce programe est un test de la question 10.
arhayout@Minishell:~$ ls | wc -l
19
arhayout@Minishell:~$ ls | wc-l
wc-l: No such file or directory
```

/ 11-Pipelines

Ma première stratégie :
Pour la gestion d'une commande contenant un nombre quelconque de tubes, je calcule d'abord le nombre de tubes, je crée tous les tubes nécessaires.

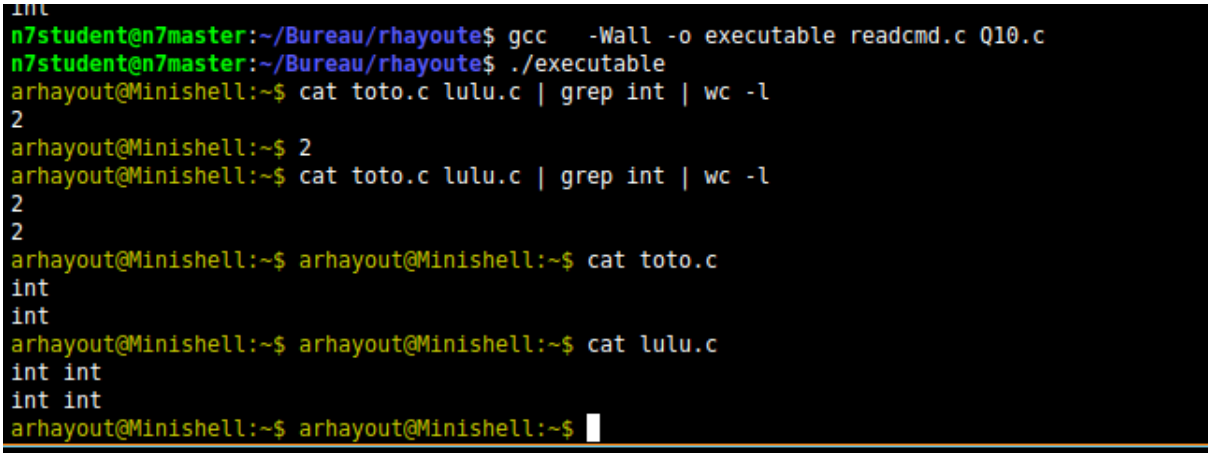
Puis chaque commande élémentaire est exécutée durant une itération de la boucle, sauf la première et la dernière commande exécutée en dehors de la boucle While.
C'est un peu difficile et source de beaucoup d'erreurs

(J'ai abandonné cette version qui se base sur la question 10)

Deuxième stratégie

Assez facile et se base sur la source GitHub suivante :

https://gist.github.com/aspatic/93e197083b65678a132b9ecee53cfe86



Remarque :

J’ai rencontré beaucoup d’erreur de mémoire avec Valgrind et je n’ai pas réussi à réparer ces erreurs vu qu’il m’indique qui ‘ils viennent de readcmd.c

Conclusion :

Ce projet est un bon exercice pour tester les connaissances requises dans les sessions TD et TP, il peut donc être amélioré en ajoutant d'autres fonctionnalités utiles telles que l'historique, et on peut aussi penser à ce que font les flèches haut/bas et droite/gauche comme dans un shell standard.