



Ire année Sciences du Numérique
SYSTEMES D'EXPLOITATION CENTRALISES

2020-2021

Rapport de Projet (mini)Projet (mini)Shell

I- Architecture de l'application

Dans les dernières versions du code j'ai choisi d'essayer de rendre aussi clair et aussi modulaire que possible mon application. Ainsi j'ai essayé de respecter un objectif de moins de 200 lignes de code par fichier écrit. Vous pourrez constater que ceci a été relativement bien respecté sauf dans les fichiers `command.c` (ainsi que `command_avant_tube.c` et `command_finale.c`) et `list_process.c` qui comporte trop d'opérations. J'aimerais attirer votre attention sur le fait que `command.c` couplé aux fichiers `QXX.c` (avec `XX` un numéro) forment le cœur de mon travail.

Pour ce qui est de l'organisation du dépôt voici un extrait du `README.txt` qui est à la racine du projet :

- livrables : fichiers demandés (`Q1.c`, `Q3.c`,... ainsi que `Q2.pdf` et `rapport.pdf`)
- `src` : & `modules_persos` regroupe le coeur du projet et se subdivise lui même en plusieurs dossiers :
 - **Execution_commandes** : c'est le **COEUR du projet** ou figure notamment les commandes internes dans des fichiers séparés, le module permettant de faire des redirections mais aussi et surtout les fichiers `command*` qui comporte les méthodes qui lancent les processus fils ou exécutent les commandes internes
 - `Gestion_erreurs` : fichiers qui permettent de réaliser un affichage

systématique des erreurs de minishell

- sous-section Table_process : Gestion de la table avec les processus lancés (suppression, insertion,...)

□ fournitures : parser fourni sur Moodle

J'ai aussi pris la liberté de vous synthétiser ce readme sous la forme d'une image avec en rouge les fichiers qui sont les plus importants à mon sens :

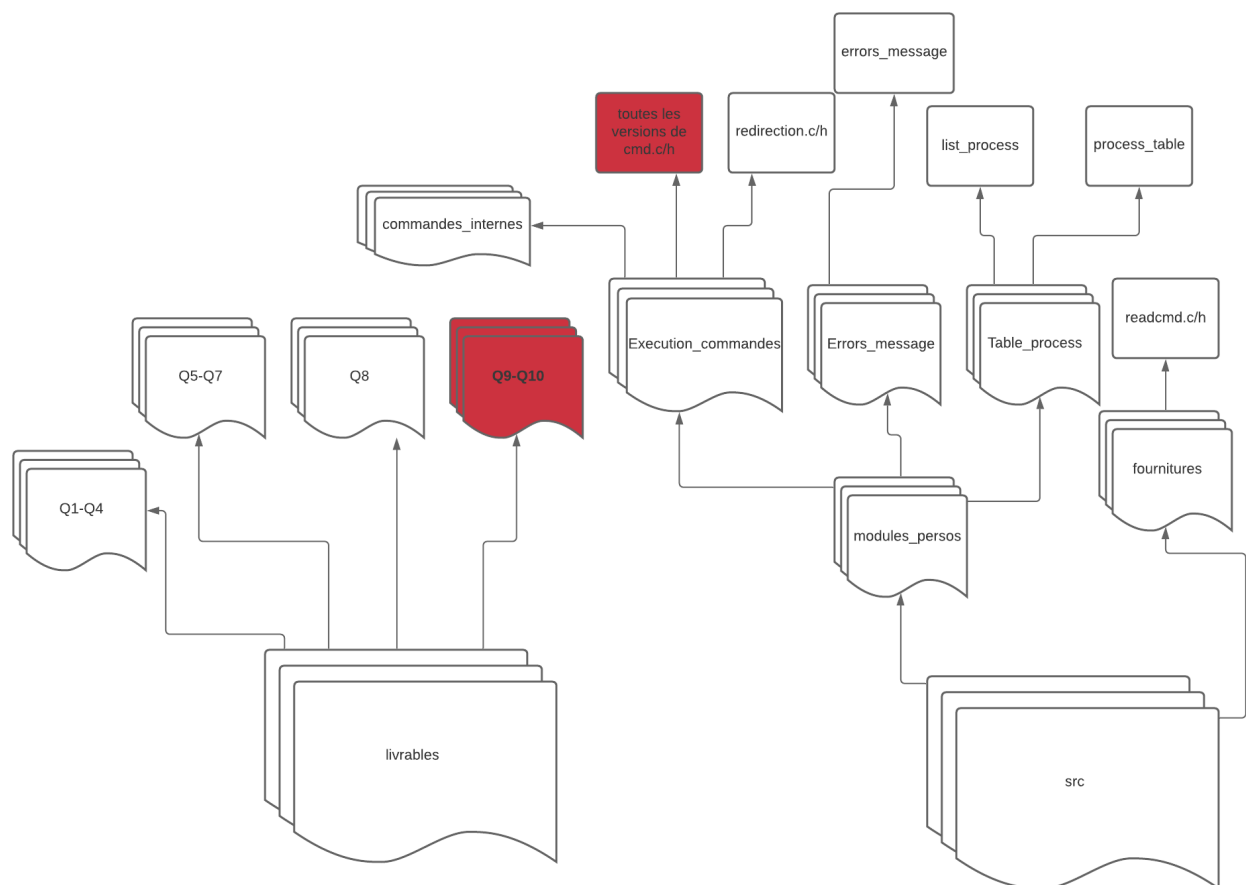


Figure 1 -Schéma expliquant l'organisation du rendu

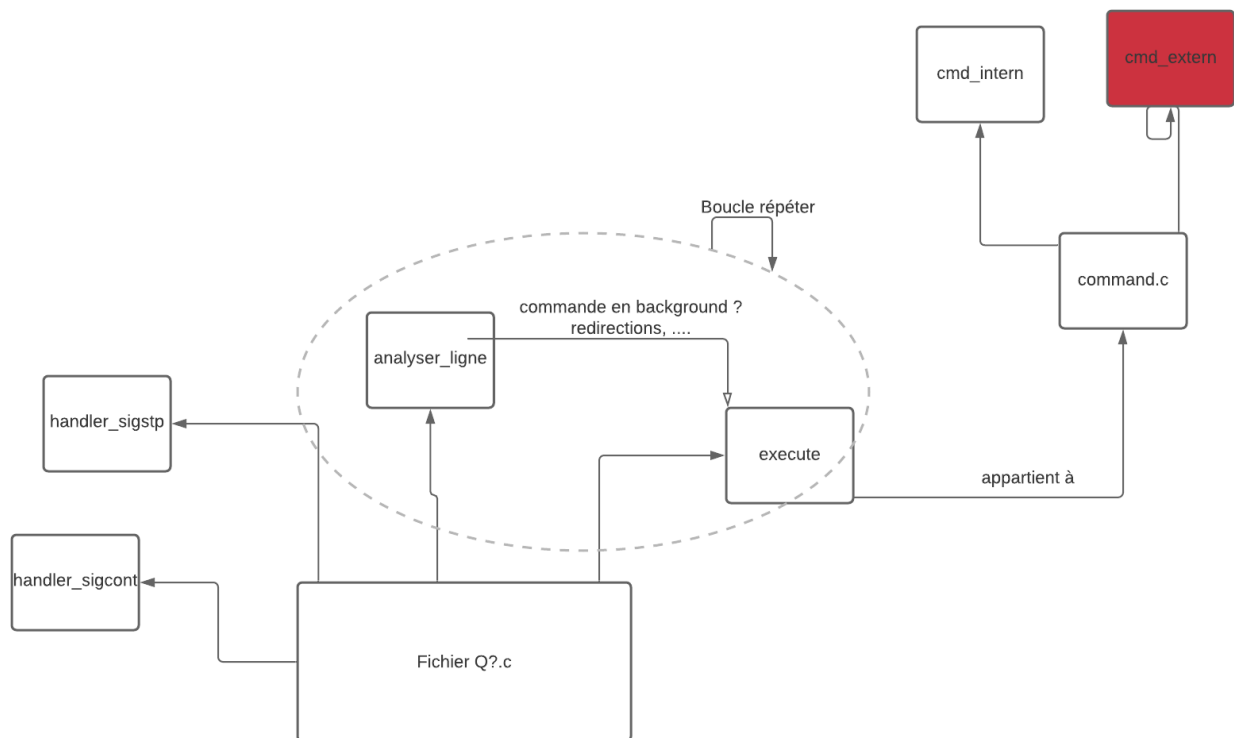


Figure 2 -Schéma simplifiant la compréhension de minishell

Sur le schéma ci-dessus vous aurez un aperçu de la manière dont est construite mon application. En effet le fichier en Q*.c se contente de positionner les handler et de faire l'analyse de la ligne de commande avant de renvoyer vers la procédure execute du fichier command.c (command_finale.c pour les version 9 et 10) qui elle-même fait appel à cmd_intern puis cmd_extern. Le cœur de minishell se situe dans cmd_extern même si c'est une procédure assez longue et difficile à comprendre. C'est cette procédure qui se charge de créer les fils avec un fork, de lancer les commandes, d'attribuer les tubes et elle fait également les redirections si nécessaire. Notez enfin que les commandes internes sont dans des fichiers séparés dans le dossier Execution_commandes/cmd_internes.

NB : dans le dossier Execution_commandes vous trouverez plusieurs versions de command.c, ceci est dû au fait que certaines versions de minishell ont nécessité de grandes transformations de ce fichier et comme il nous est demandé de rendre toutes ces versions il fallait conserver ces fichiers.

II- Choix de conception

Un des premiers que j'ai dû opérer a été d'interrompre minishell en cas d'erreur « grave » (cad échec d'un fork, d'un wait, etc..).

En effet, selon moi, une telle erreur implique un problème important dû au système d'exploitation (plus de mémoire RAM disponible dû à un effondrement par exemple). Il est donc inutile de poursuivre l'utilisation de minishell, d'où son arrêt.

Un autre choix évident que j'ai dû réaliser a été celui de créer, à part, un module pour gérer une table de processus en temps réel. Il s'agit simplement d'une liste chaînée mais compte tenu du fait que celle-ci change constamment elle me semblait plus adaptée qu'un tableau qui prendrait beaucoup trop de mémoire.

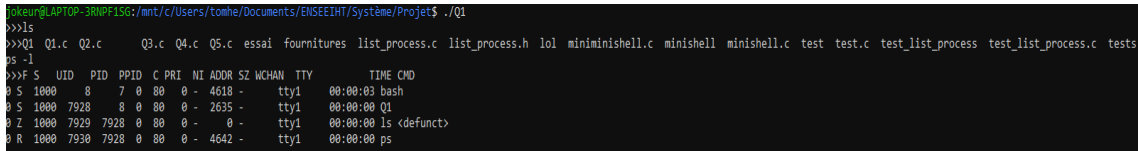
Pour la question 6 et 7 j'ai aussi dû faire un choix important qui m'a d'ailleurs valu un temps de recherche très important. En effet, pour ces deux questions il fallait faire en sorte que minishell ne soit pas interrompu par la frappe de contrôle C ou contrôle Z au clavier mais seulement le processus en avant-plan. J'ai d'abord tenté d'utiliser un système de masque de processus mais je me suis rapidement retrouvé dans une impasse (impossible de seulement démasquer le processus en avant-plan de minishell par exemple). J'ai alors opté pour la première solution proposée sur moodle qui est d'associer le signal SIGSTP à SIGSTP et le signal SIGCONT à SIGKILL bien que cela empêche leur interception par les fils. Ayant passé un temps considérable sur cette partie je n'ai pas pu approfondir plus mais il me semble tout à fait réalisable d'utiliser d'autres signaux que SIGKILL ou SIGSTOP pour réaliser ce traitement.

Pour la question 8 j'ai dû à nouveau opérer un choix (du au manque de temps). En effet j'avais commencé à implanter les redirections pour les commandes internes comme pour celles externes mais face à certaines difficultés (notamment vis à vis de la commande cd) j'ai dû abandonner cette possibilité pour les commandes internes. Vous noterez que la redirection fonctionne très bien pour les commandes externes cependant.

Enfin j'ai fait un choix qui n'a pas été bon vis à vis des tubes. En effet, au lieu de créer N-1 tubes pour exécuter N commandes je voulais créer seulement 2 tubes et les partager en les faisant tourner au fur et à mesure que les fork créent les nouveaux processus. C'était une mauvaise solution qui n'a pas pu marcher car il était alors impossible de fermer ces tubes dans le père et cela causait beaucoup

d'erreurs. Je suis finalement revenu sur un raisonnement plus classique en employant N-1 tubes.

III– Réponse à la question 2



```

okun@LAPTOP-3NPF156:/mnt/c/Users/tomhe/Documents/ENSEEIH/Système/Projet$ ./Q1
>>>ls
>>>Q1 Q1.c Q2.c Q3.c Q4.c Q5.c essai fournitures list_process.c list_process.h lol miniminshell.c minishell minishell.c test test.c test_list_process test_list_process.c tests
>>>ps -l
>>>F S UID PID PPID C PRI NI ADDR SZ WCHAN TTY TIME CMD
  S 1000 8 7 0 00 0 - 4618 - tty1 00:00:03 bash
  S 1000 7928 8 0 00 0 - 2635 - tty1 00:00:00 Q1
  Z 1000 7929 7928 0 00 0 - 0 - tty1 00:00:00 ls <defunct>
  R 1000 7930 7928 0 00 0 - 4642 - tty1 00:00:00 ps

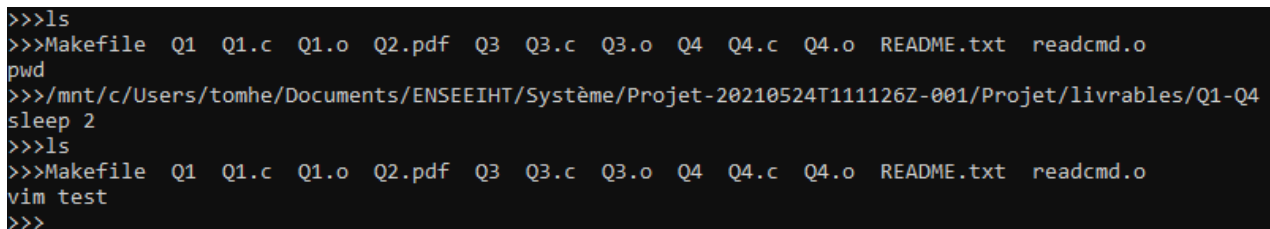
```

Figure 3 -Capture d'écran mettant en évidence la réponse à la question 2

Sur la capture d'écran ci-dessus (en zoomant) vous pourrez vous rendre compte du comportement évoqué dans la question 2 du projet. En effet, la version basique et naïve de minishell ne comporte aucun appel à waitpid dans le processus père. Ainsi minishell n'attend pas du tout la terminaison de ses fils pour proposer à l'utilisateur de rentrer une nouvelle commande. Si on le compare avec le bash on s'aperçoit que la première version de minishell lance des processus en arrière-plan. C'est d'ailleurs une possibilité développée dans la version 5.

IV – Tests opérés et résultats obtenus

Dans les versions 1 à 5 je n'ai observé aucun bug et toutes les fonctionnalités sont opérationnelles. Voici quelques captures d'écran :

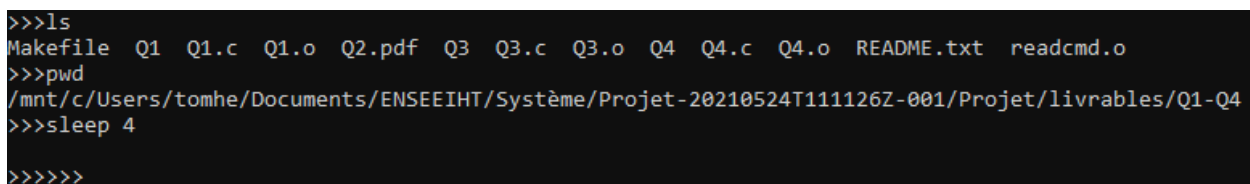


```

>>>ls
>>>Makefile Q1 Q1.c Q1.o Q2.pdf Q3 Q3.c Q3.o Q4 Q4.c Q4.o README.txt readcmd.o
>>>pwd
>>>/mnt/c/Users/tomhe/Documents/ENSEEIH/Système/Projet-20210524T111126Z-001/Projet/livrables/Q1-Q4
>>>sleep 2
>>>ls
>>>Makefile Q1 Q1.c Q1.o Q2.pdf Q3 Q3.c Q3.o Q4 Q4.c Q4.o README.txt readcmd.o
>>>vim test
>>>

```

Figure 4 -Capture d'écran mettant en évidence le fonctionnement de la version 1



```

>>>ls
>>>Makefile Q1 Q1.c Q1.o Q2.pdf Q3 Q3.c Q3.o Q4 Q4.c Q4.o README.txt readcmd.o
>>>pwd
>>>/mnt/c/Users/tomhe/Documents/ENSEEIH/Système/Projet-20210524T111126Z-001/Projet/livrables/Q1-Q4
>>>sleep 4
>>>>>

```

Figure 5-Capture d'écran mettant en évidence le fonctionnement de la version 3 (attente des fils)

```
>>>pwd
/mnt/c/Users/tomhe/Documents/ENSEEIH/Projet-20210524T111126Z-001/Projet/livrables/Q1-Q4
>>>cd ..
>>>ls
Q1-Q4 Q5-Q7 Q8 Q9-Q10 mini_rapport.pdf
>>>cd Q5-Q7
>>>pwd
/mnt/c/Users/tomhe/Documents/ENSEEIH/Projet-20210524T111126Z-001/Projet/livrables/Q5-Q7
>>>exit
```

Figure 6 -Capture d'écran mettant en évidence le fonctionnement de la version 4 (cd et exit)

```
>>>sleep 456789&
>>>ls
Makefile Q5.c Q6 Q6.o Q7.c README.txt cmd_bg.o cmd_exit.o cmd_jobs.o command_avant_tube.o list_process.o readcmd.o
Q5 Q5.o Q6.c Q7 Q7.o bug.txt cmd_cd.o cmd_fg.o cmd_stop.o errors_message.o process_table.o redirection.o
>>>ls &
>>>Makefile Q5.c Q6 Q6.o Q7.c README.txt cmd_bg.o cmd_exit.o cmd_jobs.o command_avant_tube.o list_process.o readcmd.o
Q5 Q5.o Q6.c Q7 Q7.o bug.txt cmd_cd.o cmd_fg.o cmd_stop.o errors_message.o process_table.o redirection.o
```

Figure 7 -Capture d'écran mettant en évidence le fonctionnement de la version 5 (commande en arrière-plan)

A partir des versions 6 et 7 j'ai poussé les tests encore plus loin et malheureusement même si la plupart des fonctionnalités sont remplies jusqu'à la question 9 j'ai détecté quelques bugs (cf README.txt du dossier Q5-Q7) dont voici un exemple :

```
>>>sleep 5678&
>>>fg 1
^Z
^Z>>>jobs
ID    PID    STATE    COMMAND
0     3579    R       ./Q6
1     3580    S       sleep 5678
>>>sleep 5679
```

Figure 8 -Capture d'écran mettant en évidence un bug de ctrl Z de la version 6 et + (il faut frapper deux fois ctrl Z), même bug pour ctrl C

Toutes les autres fonctionnalités pour ces deux questions fonctionnent (cad jobs, fg,bg, stop , ctrl Z et ctrl C) comme vous pouvez le constater sur la version 7 :

```

>>>sleep 66
^Z
>>>jobs
ID      PID    STATE    COMMAND
0       3588    R        ./Q7
1       3589    R        sleep 5678
4       3592    S        sleep 66
>>>bg 4
>>>jobs
ID      PID    STATE    COMMAND
0       3588    R        ./Q7
1       3589    R        sleep 5678
4       3592    R        sleep 66
>>>fg 4
^C
^C>>>jobs
ID      PID    STATE    COMMAND
0       3588    R        ./Q7
1       3589    R        sleep 5678
>>>

```

Figure 9 -Fonctionnalités des questions 6 et 7

```

>>>vim toto.txt
>>>cat toto.txt
zefzed
æzfazc
:W
:q
>>>ls
Makefile Q8 Q8.c Q8.o README.txt cmd_bg.o cmd_cd.o cmd_exit.o cmd_fg.o cmd_jobs.o cmd_stop.o command_avant_tube.o errors_message.o
list_process.o process_table.o readcmd.o redirection.o toto.txt
>>>cat < toto.txt > tutu.txt
>>>cat tutu.txt
zefzed
æzfazc
:W
:q
>>>

```

Figure 10 -Redirections fonctionnelles

Enfin vous aurez compris que je n'ai pas réussi (par manque de temps) à obtenir des résultats satisfaisants sur les tubes, voici une illustration :

```

>>>ls
Makefile Q10 Q10.c Q9 Q9.c Q9.o README.txt cmd_bg.o cmd_cd.o cmd_exit.o cmd_fg.o cmd_jobs.o cmd_stop.o command.o errors_message.o list_process.o process_table.o readcmd.o redirection.o
>>>ls | wc -l
0
>>>

```

Figure 11 -Tubes simples non fonctionnels (le résultat est 0 au lieu de 20)

```
>>>vim toto.c
>>>cat toto.c | grep a | wc -l
0
>>>cat toto.c
aa
ab
zczz
fac
a
>>>
```

Figure 12 -Tubes complexes non fonctionnels (le résultat est 0 au lieu de 5)

V – Bilan du projet

Une large partie des fonctionnalités demandées a été implantée au bout de plus d'une soixantaine d'heure de travail au total (estimation basse et sûrement en dessous de la réalité). Je déplore toutefois le fait de ne pas avoir pu régler les bugs sur les signaux des questions 6 et 7 et surtout d'avoir passé tant de temps sur les tubes pour obtenir un résultat qui n'est pas le bon. Il me semble que le problème sur les tubes est que le processus à « gauche » écrit bien dans le tube mais que pour une raison ou une autre celui de l'autre côté y lit un 0.

Dans l'ensemble je considère avoir fourni un travail sérieux en étant seulement pris par le temps et par une situation personnelle délicate. Néanmoins ce projet m'a beaucoup appris.