

Rapport de projet Systèmes d'exploitation centralisés Minishell

Arthur Bongiovanni 1SN TP-B

Résumé

1. Question du sujet	1
A. questions traités	1
B. question non traité	1
C. question partiellement traités	1
2. Architecture de l'application	2
3. Tests	2

1/ Questions du Sujet

- A. Questions traités
 - question 1
 - question 2
 - question 3
 - question 4
 - question 7
 - question 8
 - question 9
 - question 10
- B. Questions non traités
 - question 5
- C. Questions traités partiellement
 - question 6 : la question 5 n'étant pas traité, une partie de la question 6 n'as pas été traitée : le fonctionnement de la fonction « sj ». De plus, la commande interne "susp" n'a pas été implémentée.

2/ Architecture de l'application

L'application est composée de plusieurs fichiers :

- Un makefile,

- minishell.c qui correspond au programme principal et contient le main.

Pour l'exécuter il faut rentrer les commandes suivantes dans un terminal :

- make minishell

- ./minishell

- readcmd.c qui correspond au programme d'analyse de la commande entrée par l'utilisateur.

J'ai choisi dans le fichier minishell.c de ne pas faire de sous-programme (sauf pour le traitement des signaux) car il n'y a quasiment pas de redondance donc la création d'un sous-programme ne ferait que « déplacer » des bouts de code et entraînerait un appel à un sous-programme (coûtant de la mémoire)

3/ Tests

Question 1 : testé avec les commandes ls, ps et pwd

résultat :

```
> ls
Makefile  minishell  minishell.c  minishell.o  minishell.pdf  'Rapport projet SEC.pdf'  readcmd.c  readcmd.h  readcmd.o  test readcmd.c
processus [175154] terminé avec le code 0
>
> █
```

Question 2 :

```
gcc -o minishell.o readcmd.o -o minishell
argio@argio-ThinkPad-T460:~/Bureau/N7_FISE/1A_SN/minishell$ ./minishell
> ls
commande : ls
> Makefile  minishell  minishell.c  minishell.o  minishell.pdf  readcmd.c  readcmd.h  readcmd.o  test readcmd.c
█
```

On voit bien que l'invite de commande apparaît avant la terminaison du processus en cours. Ceci n'empêche en rien l'exécution d'une nouvelle commande et n'est en soit qu'un problème d'affichage qui se règle en introduisant un wait dans le code du père empêchant l'apparition de l'invite de commande avant la fin de l'exécution du fils.

Page 10 of 10

```
> ps
  PID TTY          TIME CMD
175035 pts/0    00:00:00 bash
175099 pts/0    00:00:00 minishell
176547 pts/0    00:00:00 ps
processus [176547] terminé avec le code 0

>
> pwd
/home/argio/Bureau/N7_FISE/1A_SN/S2/SECs/minishell
processus [176573] terminé avec le code 0

>
> █
```

Question 4 : testé avec *sleep 5* et *sleep 5 &*

résultat :

```
> sleep 5 &
> ps
  PID TTY          TIME CMD
175035 pts/0    00:00:00 bash
175099 pts/0    00:00:00 minishell
177309 pts/0    00:00:00 sleep
177329 pts/0    00:00:00 ps
processus [177329] terminé avec le code 0
>
> processus [177309] terminé avec le code 0
> █
```

Question 6 : j'ai testé de faire **Ctrl+Z** sur un processus en cours au premier plan, et **kill -s STOP pid_processus** sur un processus en cours en arrière plan (pour m'assurer du bon fonctionnement de la commande). Et Ctrl+Z directement dans le minishell. les processus sont ensuite relancé avec la commande **kill -s CONT pid_processus**.

résultat

```
arglo@arglo-ThinkPad-T460:~/Bureau/N7_FISE/1A_SN/S2/SEC:
> sleep 20
^Zprocessus [216805] est stoppé
> kill -s CONT 216805
processus [216805] est continué
processus [216998] terminé avec le code 0
processus [216805] terminé avec le code 0
^Zpwd
/home/arglo/Bureau/N7_FISE/1A_SN/S2/SECs/minishell
processus [217165] terminé avec le code 0
> sleep 50 &
processus [217569] est stoppé
processus [217569] est continué
processus [217569] terminé avec le code 0
>
arglo@arglo-ThinkPad-T460:~$ ps -auxf | grep sleep
arglo  217569  0.0  0.0  5728  2176 pts/0    S   21:28  0:00 |           |           \ sleep 50
arglo  217586  0.0  0.0  10920  1920 ?        S   21:28  0:00 |           |           \ sleep 1
arglo  217604  0.0  0.0  11648  2560 pts/1    S+  21:28  0:00 |           |           \ grep --color=auto sleep
arglo@arglo-ThinkPad-T460:~$ kill -s STOP 217569
arglo@arglo-ThinkPad-T460:~$ kill -s CONT 217569
arglo@arglo-ThinkPad-T460:~$
```

(la commande `pwd` est utilisée ici pour montrer que le minishell n'est pas interrompu par un **Ctrl+Z**. J'ai utilisé un second terminal pour le processus en arrière plan par soucis de simplicité)

Question 7 : j'ai testé **Ctrl+C** sur un processus en cours (afin de vérifier que la commande fonctionne bien) et Ctrl+C directement dans le minishell.

résultat :

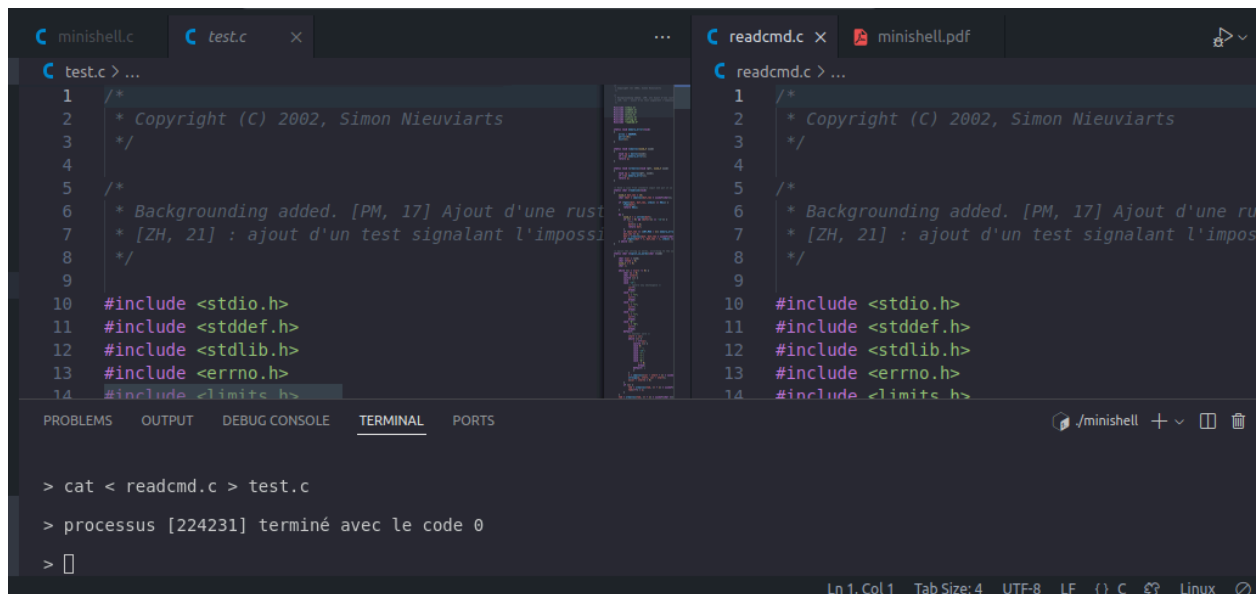
```
> sleep 20
> ^Cprocessus [222327] a envoyé un signal : 2
> sleep 30 &
> ^Cps
>  PID TTY          TIME CMD
210202 pts/0    00:00:00 bash
216600 pts/0    00:00:00 minishell
222482 pts/0    00:00:00 sleep
222508 pts/0    00:00:00 ps
processus [222508] terminé avec le code 0
> processus [222482] terminé avec le code 0
> █
```

(la commande ps permet ici d'illustrer que le **Ctrl+C** n'as pas stoppé le minishell ni le processus en arrière plan)

Pour les question 6 et 7, j'ai choisi de d'abord ignoré les signaux dans le pères avec : `signal(SIGINT, SIG_IGN); signal(SIGTSTP, SIG_IGN);` puis de les "réactiver" à la création du fils en avec les mêmes instructions en changeant `SIG_IGN` par `SIG_DFL`, puis dans le cas où la commande doit être exécuté en arrière plan, je modifie le groupe d'utilisateur du processus avant l'exécution de la fonction `execvp`. J'ai choisi cette stratégie pour sa simplicité d'implémentation (5 instructions seulement qui sont toutes des appels de primitives systèmes).

Question 8 : testé avec `cat < readcmd.c > test.txt`

résultat :



```
1 /*
2  * Copyright (C) 2002, Simon Nieuviarts
3  */
4
5 /*
6  * Backgrounding added. [PM, 17] Ajout d'une rustine
7  * [ZH, 21] : ajout d'un test signalant l'impossibilité
8  */
9
10 #include <stdio.h>
11 #include <stddef.h>
12 #include <stdlib.h>
13 #include <errno.h>
14 #include <limits.h>
```

```
> cat < readcmd.c > test.c
> processus [224231] terminé avec le code 0
> █
```

Question 9 : testé après implémentation de la question 10 avec la commande `ls -l | wc -l`

résultat :

```

> ls -l | wc -l
11
processus [225551] terminé avec le code 0

>
> ps

>
  PID TTY          TIME CMD
 210202 pts/0    00:00:00 bash
 216680 pts/0    00:00:00 minishell
 225552 pts/0    00:00:00 wc <defunct>
 225642 pts/0    00:00:00 ps
processus [225552] terminé avec le code 0

> █

```

(on peut remarquer un bug ici : le minishell ne récupère pas de signal de terminaison du second processus de la pipeline)

Question 10 : testé avec `ls -l | grep mini | wc -l`

résultat :

```

> ls -l | grep mini | wc -l
processus [226990] terminé avec le code 0

> 4
processus [227663] terminé avec le code 0

> processus [228131] terminé avec le code 0

>
> ps

>
  PID TTY          TIME CMD
 210202 pts/0    00:00:00 bash
 216680 pts/0    00:00:00 minishell
 228132 pts/0    00:00:00 grep <defunct>
 228133 pts/0    00:00:00 wc <defunct>
 228169 pts/0    00:00:00 ps
processus [228132] terminé avec le code 0

> █

```

(on remarque que le minishell ne récupère un signal de terminaison que du premier processus de la pipeline. Cependant, il est toujours possible de lancer une nouvelle commande à la suite.)