FALLOT
Ysabel
Groupe E

Rendu projet minishell

Les résultats des deux premières questions ont été générés avec la version du minishell rendu après le TP 1, et pour laquelle le code dans le père a été commenté.

Question 1 :

Pour cette question, les commandes de tests ont été « ls », « ls -l », « ls -a », « sleep 5 » et « exit ».

```
ysa@ysa-HP-Laptop-15-dw0xxx:~/Téléchargements/minishell-yft5609/minishell$ ./minishell
> ls
commande : ls
Makefile  minishell  minishell.c  minishell.o  readcmd.c  readcmd.h  readcmd.o  test_readcmd.c
> ls -a
commande : ls -a
.  ..  Makefile  minishell  minishell.c  minishell.o  readcmd.c  readcmd.h  readcmd.o  test_readcmd.c
> ls -l
commande : ls -l
total 56
-rw-r--r-- 1 ysa ysa  1076 mars  27 17:54 Makefile
-rwxrwxr-x 1 ysa ysa 16872 mai   22 22:30 minishell
-rw-r--r-- 1 ysa ysa  3311 mai   22 22:30 minishell.c
-rw-rw-r-- 1 ysa ysa  3904 mai   22 22:30 minishell.o
-rw-r--r-- 1 ysa ysa  5360 mars  23 23:50 readcmd.c
-rw-r--r-- 1 ysa ysa  2156 mars  23 16:25 readcmd.h
-rw-rw-r-- 1 ysa ysa  6480 mai   22 22:17 readcmd.o
-rw-r--r-- 1 ysa ysa  2012 mars  23 16:26 test_readcmd.c
> sleep 5
commande : sleep 5
> exit
Au revoir ...
```

Figure 1.1 – Commandes exécutées pour la question 1

On peut voir en plus de l'affichage console que les commandes s'exécutent bien car le processus exécutant sleep 5 apparaît bien en statut S+ (indiquant que le processus est en cours) après avoir tapé la commande. De plus exit fait bien quitter le minishell.

```
ysa@ysa-HP-Laptop-15-dw0xxx:~$ ps fj
   PPID     PID    PGID     SID TTY        TPGID STAT   UID   TIME COMMAND
  11722   12504   12504   12504 pts/1      12523 Ss    1000   0:00 bash
  12504   12523   12523   12504 pts/1      12523 R+    1000   0:00  \_ ps fj
  11722   11745   11745   11745 pts/0      12492 Ss    1000   0:00 bash
  11745   12492   12492   11745 pts/0      12492 S+    1000   0:00  \_ ./minishe
  12492   12522   12492   11745 pts/0      12492 S+    1000   0:00      \_ sleep
```

Figure 1.2 – Processus en cours après avoir tapé « sleep 5 » dans le minishell

Question 2

On voit qu'ici le chevron « > » s'affiche puis d'autres caractères lié à l'exécution d'une commande précédente s'affichent sur la même ligne. Cela signifie bien que le processus père n'attend pas la fin de l'exécution de son fils pour attendre une nouvelle lecture de commande.

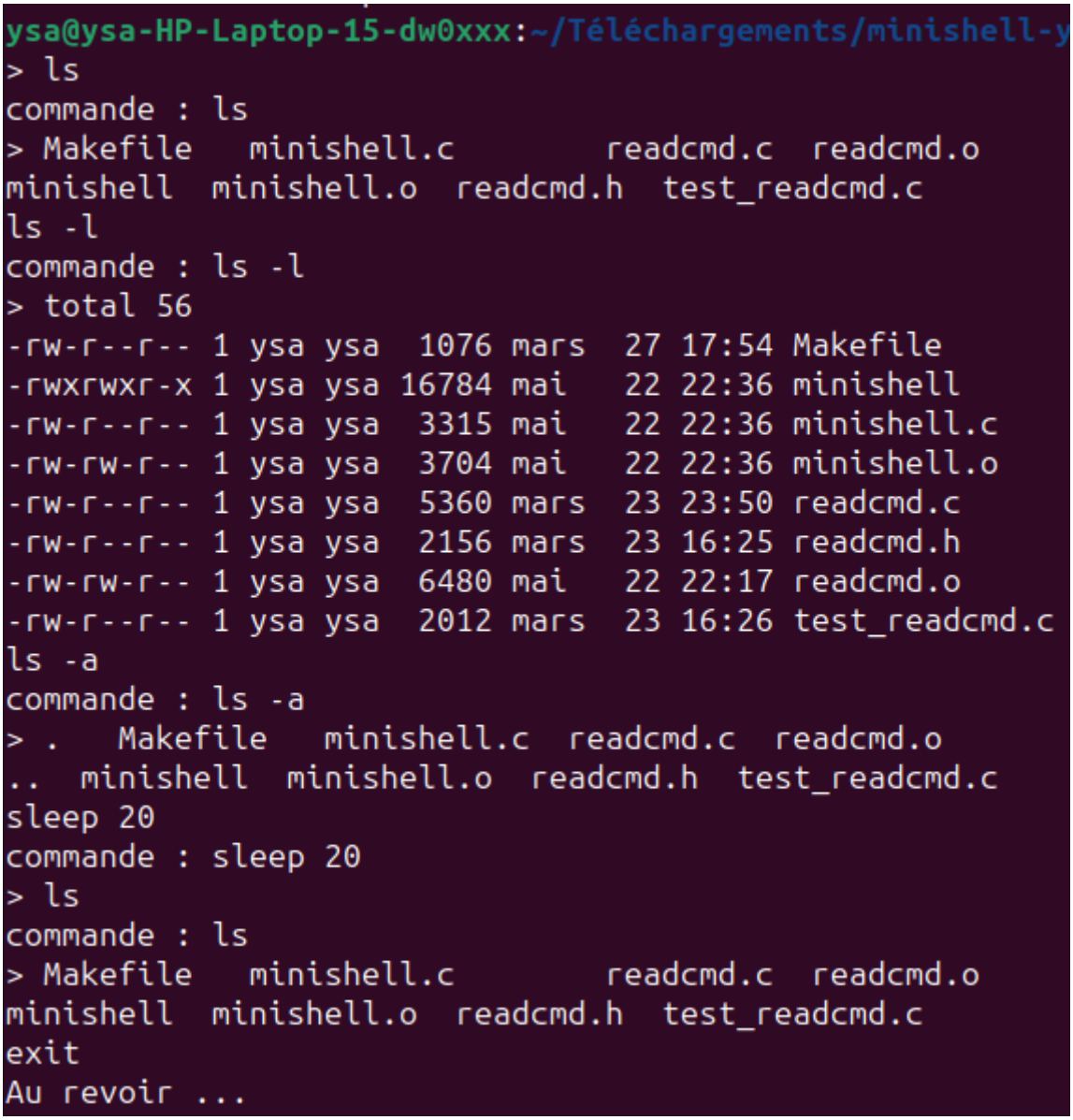

Figure 2 - Commandes exécutées pour la question 2

Pour les questions suivantes, les résultats sont obtenues avec la version finale du minishell.

Question 3 :

On refait les mêmes commandes que pour la question précédente, et on remarque ici que le chevron > s'affiche bien lorsque que tous les affichages liées à la commande précédente se sont affichées, c'est à dire que le père a bien attendu la fin d'exécution de son fils avant de lancer une nouvelle commande.

```
ysa@ysa-HP-Laptop-15-dw0xxx:~/Téléchargements/minishell-yft5609/minishell$ ./minishell
> ls
commande : ls
Makefile  minishell  minishell.c  minishell.o  readcmd.c  readcmd.h  readcmd.o  test_readcmd.c
> ls -l
commande : ls -l
total 56
-rw-r--r-- 1 ysa ysa  1076 mars  27 17:54 Makefile
-rwxrwxr-x 1 ysa ysa 16872 mai   22 22:43 minishell
-rw-r--r-- 1 ysa ysa  3311 mai   22 22:43 minishell.c
-rw-rw-r-- 1 ysa ysa  3904 mai   22 22:43 minishell.o
-rw-r--r-- 1 ysa ysa  5360 mars  23 23:50 readcmd.c
-rw-r--r-- 1 ysa ysa  2156 mars  23 16:25 readcmd.h
-rw-rw-r-- 1 ysa ysa  6480 mai   22 22:17 readcmd.o
-rw-r--r-- 1 ysa ysa  2012 mars  23 16:26 test_readcmd.c
> ls -a
commande : ls -a
.  ..  Makefile  minishell  minishell.c  minishell.o  readcmd.c  readcmd.h  readcmd.o  test_readcmd.c
> sleep 10
commande : sleep 10


ls

ls
> > > commande : ls
Makefile  minishell  minishell.c  minishell.o  readcmd.c  readcmd.h  readcmd.o  test_readcmd.c
> > commande : ls
Makefile  minishell  minishell.c  minishell.o  readcmd.c  readcmd.h  readcmd.o  test_readcmd.c
> exit
Au revoir ...
```

Figure 3 - Commandes exécutées pour la question 3

Question 4 :

Pendant que la commande « sleep 20 & » s'exécute, on peut taper « ls » et » sleep 6 », et ces deux commandes viennent à d'exécuter. De plus, en tapant « ps fj » dans une autre console, on voit bien que le processus exécutant « sleep 20 & » tourne toujours (état S+). Ainsi la commande avec un « & » est bien réalisé en arrière plan. Cette commande se termine bien, puisqu'après avoir exécuté « sleep 6 », lorsque que le terminal attend une nouvelle commande (après le message « Je suis le processus 13067 et je me ; suis terminé ») le message « Je suis le processus 13065 et je me suis terminé » apparaît, ce qui signifie que le processus exécutant « sleep 20 & » a bien fonctionné.

```
ysa@ysa-HP-Laptop-15-dw0xxx:~/1A/S6/SEC/minishell$ ./minishell
> sleep 20 &
> ls
Makefile   minishell.c  readcmd.c  readcmd.o
minishell  minishell.o  readcmd.h  test_readcmd.c
Je suis le processus 13066 et je me suis terminé
> sleep 6
Je suis le processus 13067 et je me suis terminé
> Je suis le processus 13065 et je me suis terminé
exit
```

Figure 4.1 - Commandes exécutées pour la question 4

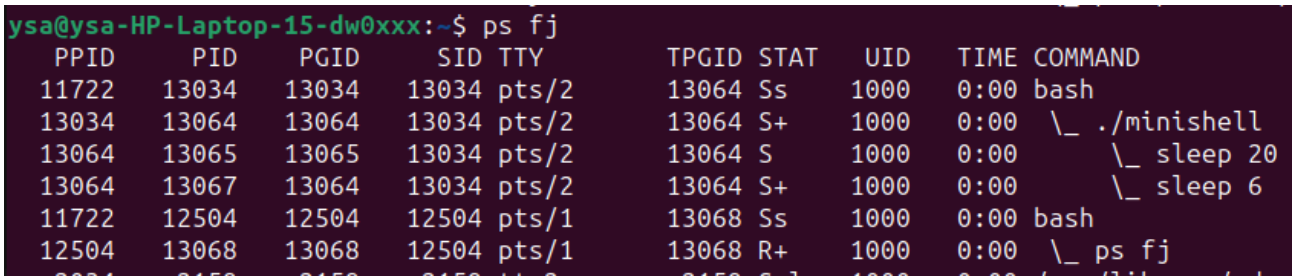

Figure 4.2 – Processus en cours après avoir tapé « sleep 6 » dans le minishell

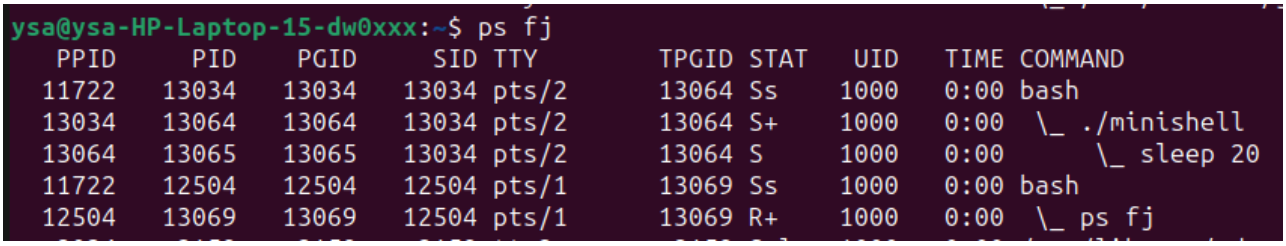

Figure 4.3 – Processus en cours après avoir apparition de « Je suis le processus 13067 et je me ; suis terminé »

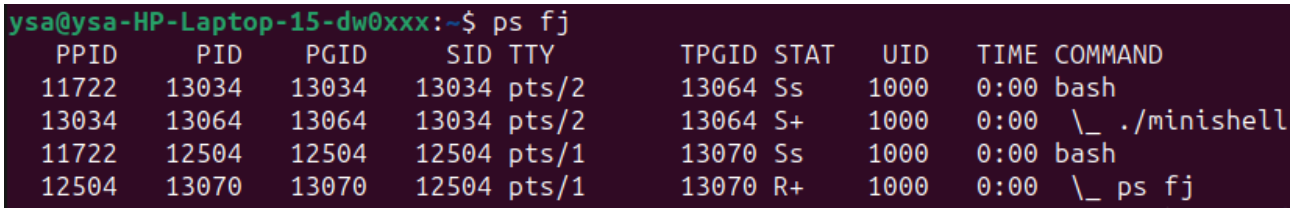

Figure 4.4 – Processus en cours après avoir apparition de « Je suis le processus 13065 et je me ; suis terminé »

Question 5 : Question non traitée.

Pour cette partie traitant du traitement des signaux, j'ai implémenté aussi bien la méthode consistant à ignorer les signaux puis à en tenir compte uniquement dans les fils, que la méthode qui consiste à masquer les signaux.
Je présenterai ici que les résultats de la première méthode (ignorer les signaux) car les résulats de la deuxième méthode sont identiques.
Pour passer d'une méthode à l'autre, il suffit de modifier la valeur de la variable globale entière choix_traitement_signal au début du programme. (2 pour ignorer, 3 pour masquer).

Question 6 :

En ayant lancé une commande en arrière plan et en avant plan, la commande Ctrl Z rend « zombie » (état T+) la commande en avant plan (Figure 6.3). Cette commande fonctionne donc puisque ni la commande en avant plan ni le minishell s'arrête et la tâche d'arrière plan se finit (Figure 6.4) Néanmoins plus aucun processus ne s'exécute dans le minishell, il reste bloqué.

```
ysa@ysa-HP-Laptop-15-dw0xxx:~/1A/S6/SEC/minishell$ ./minishell
> sleep 60 &
> sleep 20
^Z
exit
Je suis le processus 13599 et je me suis terminé
exit
```

Figure 6.1 - Commandes exécutées pour la question 6

```
ysa@ysa-HP-Laptop-15-dw0xxx:~$ ps fj
    PPID     PID    PGID     SID TTY        TPGID STAT   UID   TIME COMMAND
   11722   13568   13568   13568 pts/0    13598 Ss    1000   0:00 bash
   13568   13598   13598   13568 pts/0    13598 S+    1000   0:00  \_ ./minishell
   13598   13599   13599   13568 pts/0    13598 S     1000   0:00      \_ sleep 60
   13598   13600   13598   13568 pts/0    13598 S+    1000   0:00      \_ sleep 20
   11722   12504   12504   12504 pts/1    13601 Ss    1000   0:00 bash
   12504   13601   13601   12504 pts/1    13601 R+    1000   0:00  \_ ps fj
```

Figure 6.2 – Processus en cours après avoir tapé « sleep 20 » dans le minishell

```
ysa@ysa-HP-Laptop-15-dw0xxx:~$ ps fj
    PPID     PID    PGID     SID TTY        TPGID STAT   UID   TIME COMMAND
   11722   13568   13568   13568 pts/0    13598 Ss    1000   0:00 bash
   13568   13598   13598   13568 pts/0    13598 S+    1000   0:00  \_ ./minishell
   13598   13599   13599   13568 pts/0    13598 S     1000   0:00      \_ sleep 60
   13598   13600   13598   13568 pts/0    13598 T+    1000   0:00      \_ sleep 20
   11722   12504   12504   12504 pts/1    13605 Ss    1000   0:00 bash
   12504   13605   13605   12504 pts/1    13605 R+    1000   0:00  \_ ps fj
```

Figure 6.3 – Processus en cours après avoir tapé « Ctrl Z » dans le minishel

```
ysa@ysa-HP-Laptop-15-dw0xxx:~$ ps fj
    PPID     PID    PGID     SID TTY        TPGID STAT   UID   TIME COMMAND
   11722   13568   13568   13568 pts/0    13598 Ss    1000   0:00 bash
   13568   13598   13598   13568 pts/0    13598 S+    1000   0:00  \_ ./minishell
   13598   13600   13598   13568 pts/0    13598 T+    1000   0:00      \_ sleep 20
   11722   12504   12504   12504 pts/1    13627 Ss    1000   0:00 bash
   12504   13627   13627   12504 pts/1    13627 R+    1000   0:00  \_ ps fj
```

Figure 6.4 – Processus en cours après avoir apparition de « Je suis le processus 13599 et je me ; suis terminé »

Question 7 :

En ayant lancé une commande en arrière plan et en avant plan, la commande Ctrl C arrête la commande en avant plan. Cette commande fonctionne donc puisque ni la commande en avant plan ni le minishell s’arrête.

```
ysa@ysa-HP-Laptop-15-dw0xxx:~/1A/S6/SEC/minishell$ ./minishell
> sleep 60 &
> sleep 20
^CJe suis le processus 14000 et j'ai reçu le signal 17
> exit
```

Figure 7.1 - Commandes exécutées pour la question 7

```
ysa@ysa-HP-Laptop-15-dw0xxx:~$ ps fj
   PPID     PID    PGID     SID TTY        TPGID STAT   UID   TIME COMMAND
  11722   13889   13889   13889 pts/2    13998 Ss    1000   0:00 bash
  13889   13998   13998   13889 pts/2    13998 S+    1000   0:00  \_ ./minishell
  13998   13999   13999   13889 pts/2    13998 S     1000   0:00      \_ sleep 60
  13998   14000   13998   13889 pts/2    13998 S+    1000   0:00      \_ sleep 20
  11722   12504   12504   12504 pts/1    14001 Ss    1000   0:00 bash
  12504   14001   14001   12504 pts/1    14001 R+    1000   0:00  \_ ps fj
```

Figure 7.2 – Processus en cours après avoir tapé « sleep 20 » dans le minishell

```
ysa@ysa-HP-Laptop-15-dw0xxx:~$ ps fj
   PPID     PID    PGID     SID TTY        TPGID STAT   UID   TIME COMMAND
  11722   13889   13889   13889 pts/2    13998 Ss    1000   0:00 bash
  13889   13998   13998   13889 pts/2    13998 S+    1000   0:00  \_ ./minishell
  13998   13999   13999   13889 pts/2    13998 S     1000   0:00      \_ sleep 60
  11722   12504   12504   12504 pts/1    14002 Ss    1000   0:00 bash
  12504   14002   14002   12504 pts/1    14002 R+    1000   0:00  \_ ps fj
```

Figure 7.3 – Processus en cours après avoir tapé « Ctrl C» dans le minishell

Question 8 :

La commande exécutée est « cat < entree.txt > sortie.txt ». Elle a pour but de copier le contenu du fichier entree.txt dans sortie.txt.
On crée donc le fichier entree.txt qui contient « Je suis le contenu du ficher entree.txt » (Figure 8.2) et le fichier sortie.txt qui contient « Je suis le contenu du ficher sortie.txt » (Figure 8.3).
On obtient bien après exécution de la commande le contenu de entree.txt dans sortie.txt. (Figure 8.4)
Le fichier sortie.txt de l'archive est donc le ficher obtenu après l'exécution de la commande.

```
ysa@ysa-HP-Laptop-15-dw0xxx:~/1A/S6/SEC/minishell$ ./minishell
> cat < entree.txt > sortie.txt
Je suis le processus 11068 et je me suis terminé
>
```

Figure 8.1 - Commandes exécutées pour la question 8

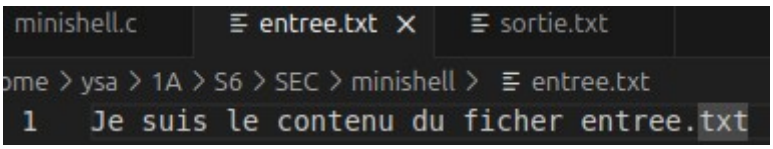

Figure 8.2 – Contenu du fichier entree.txt

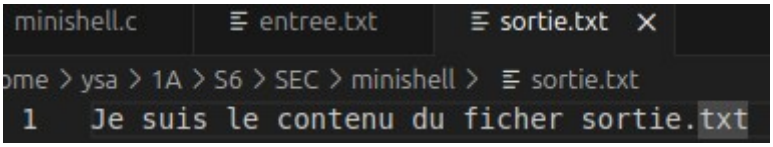

Figure 8.3 – Contenu du fichier sortie.txt avant exécution de la commande

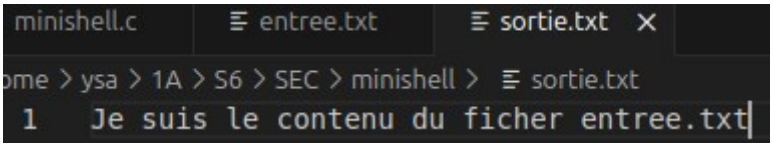

Figure 8.4 – Contenu du fichier sortie.txt après exécution de la commande

Question 9 :

Un tube entre deux commandes fonctionne. Le père n'attend pas la fin de ses fils avant de vouloir lire une nouvelle commande. On retrouve alors le problème de la question 2. J'ai voulu régler ce problème, mais après avoir modifié mon code (partie père de la fonction gestion_pipes) et refait les commandes, le tube ne marchait plus, donc le problème de synchronisation du père n'est pas implémenté avec un tube.

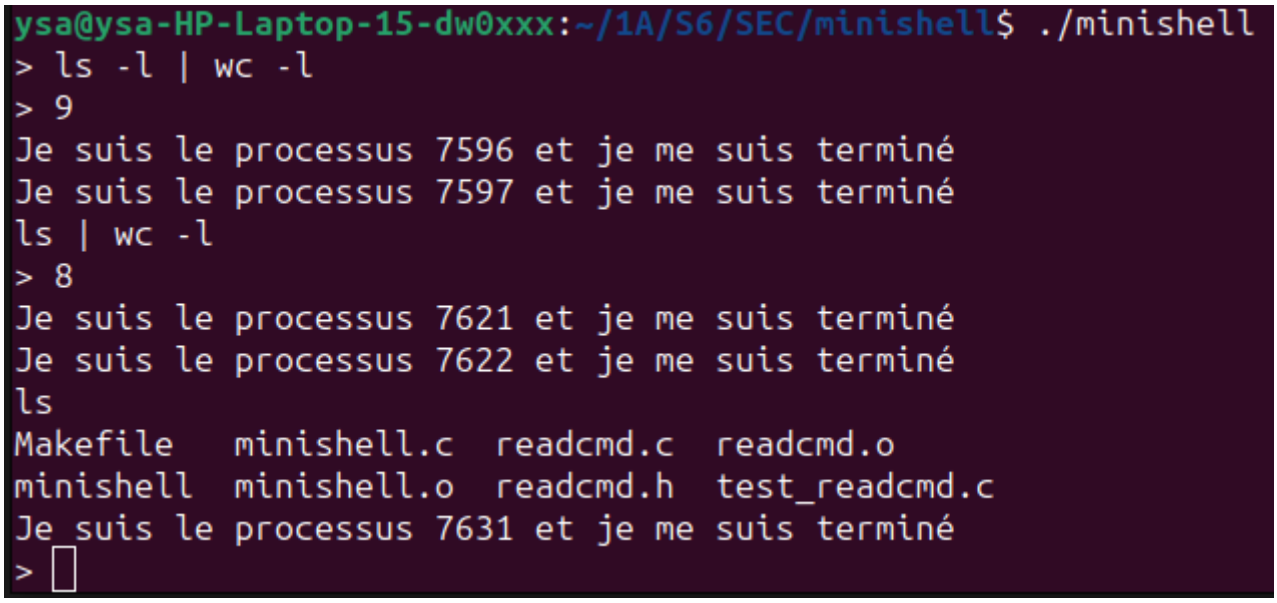

Figure 9 - Commandes exécutées pour la question 9

Question 10 :

De la même manière que la question précédente, les tubes fonctionnent mais le père n'est pas synchronisé avec ses fils ce que provoque un décalage du chevron « > » dans l'affichage.

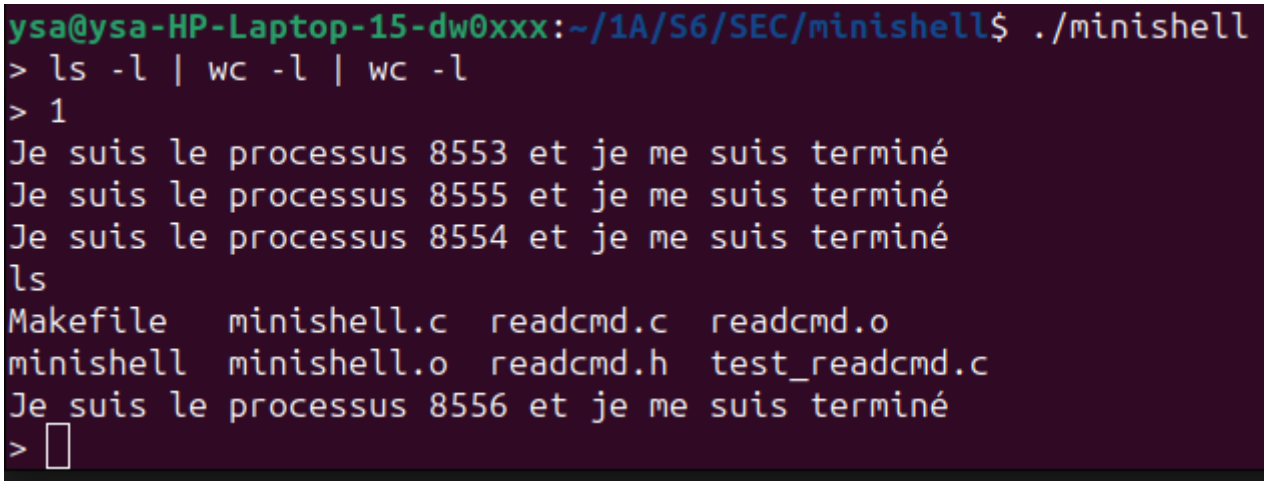

Figure 10 - Commandes exécutées pour la question 10