Note sur le rendu final

Questions complètement traitées: Questions 1 à 11

Choix de conception:

Liste des processus courants:

Pour conserver en mémoire la liste des jobs non terminés, j'ai décidé d'utiliser une liste chaînée. Chaque cellule contient un processus et ses informations (identifiant, commande tapée, pid, s'il est actif ou suspendu et s'il est en premier plan ou en background), ainsi que le processus suivant.
Pour initialiser la liste, afin d'éviter les problèmes de pointeur liés à une liste de vide, je crée une cellule avec un job de pid -2 (car ce n'est pas un pid qu'on peut trouver naturellement).
La liste est une variable globale car j'avais besoin de l'utiliser dans les traitants de signaux (handler_z et handler_c), or ils ne peuvent prendre que l'entier correspondant au signal en paramètre.

Les commandes ctrl+z et ctrl+c:

Pour implémenter ces commandes, j'ai utilisé sigaction et deux handler (un pour ctrl+z et un pour ctrl+c). Dans le handler, signal(sig, SIG_IGN) permet d'ignorer les signaux SIGINT et SIGSTP. A l'intérieur des handler, il faut redéfinir sigaction (sinon les signaux ne seront pas traités les fois d'après).
On kill le processus avec SIGSTOP pour ctrl+z et SIGKILL pour ctrl+c. Grâce à cette méthode, on suspend ou tue bien le processus en premier plan sans influer sur le minishell.

Redirection:

J'utilise dup2 pour rediriger l'entrée standard et la sortie standard sur un fichier donné si l'utilisateur l'a précisé (command->in ou command->out non null).

Les tubes:

Pour réaliser l'enchaînement de commande grâce aux tubes, je commande par compter le nombre de tubes nécessaires: s'il n'y en a pas, c'est une commande simple, on l'exécute comme d'habitude.
S'il y a besoin de tubes, je traite la première commande et la dernière commande à part: en effet, la première commande ne prendra pas son entrée dans un tube, et la dernière commande ne renvoie pas sa sortie dans un tube.
Au milieu, les commandes ont des tubes "des deux côtés". Je crée une boucle qui réalise autant de tubes que nécessaire, grâce à un array de tubes (dont la taille a été calculée au début en comptant le nombre de tubes nécessaires.

Test:

Les commandes sj, fg, bg, lj:

```
lweisbec@vador:~/Annee_1/SEC/projet/fournitures$ ./minishell
lweisbec@machine $ sleep 30&
lweisbec@machine $ lj
lweisbec@machine $ Processus d'identifiant 1 | PID : 1177034 | Etat : actif | commande:  sleep 30

lweisbec@machine $ sj 1
lweisbec@machine $ lj
lweisbec@machine $ Processus d'identifiant 1 | PID : 1177034 | Etat : suspendu | commande:  sleep 30

lweisbec@machine $ bg 1
lweisbec@machine $ lj
lweisbec@machine $ Processus d'identifiant 1 | PID : 1177034 | Etat : actif | commande:  sleep 30

lweisbec@machine $ fg 1
```

Ici, on peut voir l'exécution d'une commande an arrière plan qui est suspendu avec sj, puis on reprend son exécution avec bg, et on la remet au premier plan.

Les commandes ctrl+c et ctrl+z:

```
lweisbec@vador:~/Annee_1/SEC/projet/fournitures$ ./minishell
lweisbec@machine $ sleep 30
^Zlweisbec@machine $ lj
lweisbec@machine $ Processus d'identifiant 1 | PID : 1179325 | Etat : suspendu | commande:  sleep 30
bg 1
lweisbec@machine $ fg 1
^Clweisbec@machine $
lweisbec@machine $
lweisbec@machine $ lj
lweisbec@machine $
```

Ici on commence par exécuter une commande en premier plan. On la suspend avec ctrl+z et on la remet en premier plan, puis on on tue le process avec ctrl+c. Le job a bien disparu de la liste.

Les redirections:

```
lweisbec@machine $ ls
LisezMoi.html                   minishell     minishell8    out.txt           rendu_etapes_1_5
LisezMoi.md                     minishell2    minishell84   ps.txt            rendu_etapes_1_5.tar
lj.txt                          minishell5    minishell8.c  readcmd.c         test.txt
ls.txt                          minishell5.c  minishell.c   readcmd.h         toto.c
lweisbec_rendu_intermediaire    minishell6    out3.txt      rendu_etape_6
lweisbec_rendu_intermediaire.tar minishell6.c out4.txt      rendu_etape_6.tar
lweisbec@machine $ ls > out.txt
lweisbec@machine $
```

out.txt

```
1 LisezMoi.html
2 LisezMoi.md
3 lj.txt
4 ls.txt
5 lweisbec_rendu_intermediaire
6 lweisbec_rendu_intermediaire.tar
7 minishell
8 minishell2
9 minishell5
10 minishell5.c
11 minishell6
12 minishell6.c
13 minishell8
14 minishell84
15 minishell8.c
16 minishell.c
17 out3.txt
18 out4.txt
19 out.txt
20 ps.txt
21 readcmd.c
22 readcmd.h
23 rendu_etape_6
24 rendu_etape_6.tar
25 rendu_etapes_1_5
26 rendu_etapes_1_5.tar
27 test.txt
28 toto.c
```

Ici on exécute la commande ls d'abord dans la sortie standard, puis on le redirige vers un fichier "out.txt". On vérifie que ce fichier contient bien le résultat de la commande.

Les tubes:

```
lweisbec@machine $ cat toto.c | grep int | wc -l
3
lweisbec@machine $ ls -a | wc -l
32
lweisbec@machine $
```

Ici, on peut voir des exemples d'utilisation de tube pour réaliser des commandes chaînées.