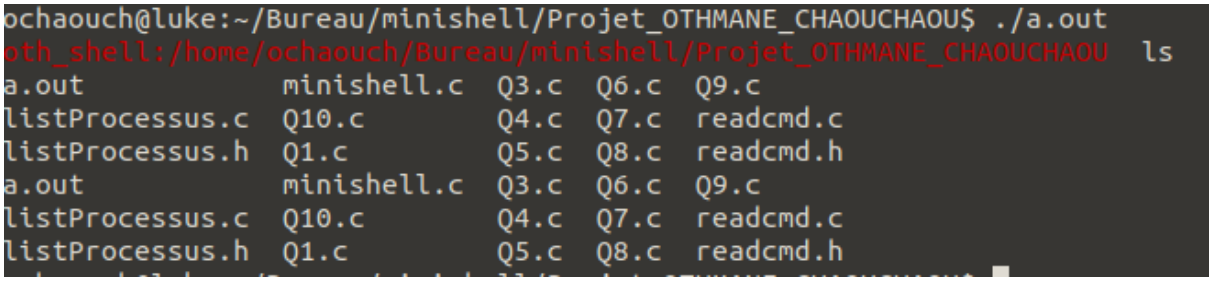
Systèmes d’exploitation centralisée

Rapport minishell

Othmane CHAOUCHAOU

Question 1 : Pour chaque commande saisie, un processus fils est créé.J'ai rencontré un problème que je n'ai pas pu résoudre. **Le loop infini (while (true)) ne fonctionne pas.**

Question 2 : Test initial du minishell



Question 3 : le processus père n'attend pas son fils .Il suffisait d'ajouter wait(NULL) dans la partie exécutée par le père.

Question 4: j'ai utilisé les méthodes « chdir » et « getenv » ainsi que la variable HOME qui contient le chemin du répertoire d'accueil de l'utilisateur.

Question 5: Il suffit de lancer la commande avec « execvp ».

Question 6:

Pour pouvoir enregistrer les informations concernant chaque processus fils, j'ai défini le type job_t qui contient l'identifiant du processus (géré par le minishell), son pid , son état (foregrounded , backgrounded, ou Suspendu) et la commande qui l'avait lancé.
Puis, j'ai créé une liste liste_jobs dont les éléments sont de type job_t. Pour manipuler cette liste, j'ai défini les méthodes suivantes: Int get_pid(int id) retourne le pid du processus dont le id géré par le mini shell est en entrée.
chaque signal SIGCHDL reçu par le processus père implique la modification de liste_jobs
grâce au handler : suivi_fils (int sig).
a) La commande 'list' :
Se traduit par l'appel à la méthode void lister_jobs ( ) qui affiche tous les processus enregistrés dans liste_jobs.
b) La commande 'stop' :
Se traduit par l'appel à void stop_processus (char** cmd) qui envoie le signal SIGSTOP au
processus.

voici un exemple d'utilisation de list et stop :
c) La commande 'bg' :

Se traduit par l'appel à void background (char** cmd) qui envoie le signal SIGCONT au processus.

Voici un exemple :
d) La commande 'fg' :

Se traduit par l'appel à void foreground (char** cmd) qui affiche la commande qui avait lancé le processus, envoie le signal SIGCONT au processus , fait le père attendre la terminaison de ce processus puis le supprimer de liste jobs.

e) Traitement de ctrl + z :

Pour le signal SIGTSTP (ctrl z), j'ai choisi qu'il soit ignoré car le traitement sig_ign est conservé chez les processus fils.
De plus, le masquage des signaux bloquent les signaux et les signaux bloqués ne sont pas ignorés, mais simplement mis en attente, (en général pour être délivrés ultérieurement) ce qui n'est pas notre but.

Question 7: Terminaison du processus en avant-plan
Pour traiter la frappe ctrl-c, j'ai défini le handler sigint_handler(int sig) qui envoie le signal
SIGINT au dernier processus lancé actif en foreground dont le pid est enregistré dans le
variable global 'pid_fg' .

Question 8: Pour cette question, il fallait utiliser les attributs in et out de la structure du module readcmd.
L'implantation, qui a été faite dans la partie du fils (après le fork), était assez simple:
if in != NULL && err == NULL
rediriger stdin vers le descripteur du fichier in
if out != NULL && err == NULL
rediriger stdout vers le descripteur du fichier out
Il faut bien sûr aussi tester les retour des dup2 et des open si jamais ces commandes engendrent une erreur.
L'exécution de la commande est assurée par l'ancien execvp.

Question 9-10:

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 comme le montre le schéma suivant :

Commande 0 | Commande 1 | Commande 2 | Commande 3 ….

Pipes[0] Pipes[1] Pipes[2] Pipes[3] Pipes[4] Pipes[5]

```
oth_shell:/home/ochaouch/Bureau/minishell/Projet_OTHMANE_CHAOUCHAOU$ ls | grep Q
a.out            minishell.c  Q3.c  Q6.c  Q9.c
listProcessus.c  Q10.c        Q4.c  Q7.c  readcmd.c
listProcessus.h  Q1.c         Q5.c  Q8.c  readcmd.h
Q10.c
Q1.c
Q3.c
Q4.c
Q5.c
Q6.c
Q7.c
Q8.c
Q9.c
```