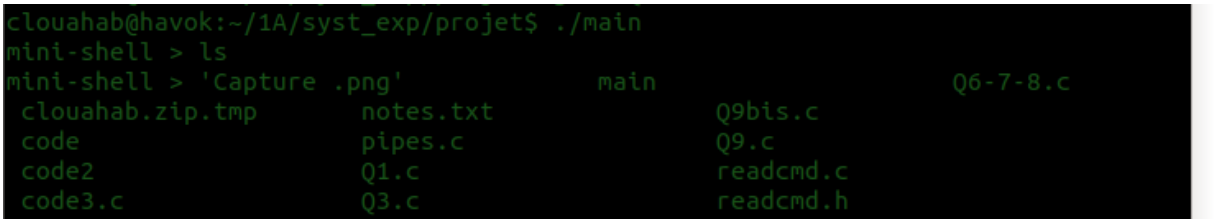
Rapport final du projet Mini-Shell

Réalisé par Chaimaa Louahabi

Dans ce bref rapport, Je répondrai aux questions posées dans le sujet du projet, en expliquant au fur et à mesure les choix de conception et en ajoutant des captures de tests.

Question 1 : Pour chaque commande saisie, un processus fils est créé. Un message d'erreur est affiché si la commande n'existe pas dans la variable « PATH ».

Question 2 : On voit bien que l'affichage de l'invite se mêle à l'exécution du processus fils.



Question 3 : Par défaut, le processus père n'attend pas son fils, pour cela j'ai utilisé la routine "wait".

Question 4 :***La commande interne 'cd'*** :

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

La signature de la méthode implantant la commande 'cd' est :

```
void chage_directory(char ** cmd)
```

Question 5 : ***Lancement de commandes en tâche de fond***

Il suffit de lancer la commande avec « execvp » puisque Le parseur fourni permet d'analyser et reconnaître un & en fin de ligne. Par contre, dans ce cas le processus père n'attend pas le processus fils exécutant la commande en tâche de fond.

Question 6 : ***Gérer les processus lancés depuis le Shell***

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 minishell est en entrée.
- ✓ ***Int*** chercher_max_id() retourne le max des id s des processus gérés par leminishell (on l'utilisera pour mettre à jour le variable global 'max' après la suppression d'un processus.
- ✓ ***Void*** delete_job (***pid_t*** pid) supprime un processus de la liste à partir de son pid.
- ✓ ***Void*** add_job (***int*** pid , ***char**** cmd, etat_t etat) ajoute un processus à ***liste_jobs*** son état est 'etat'.
- ✓ ***Int*** get_ job_indice (**pid_t** pid) retourne l'indice du processus dans la ***liste_jobs*** à partir de son pid.
- ✓ ***void*** print_job (***job_t*** job) affiche sur la sortie standard l'id, le pid, l'état et la commande du processus 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.

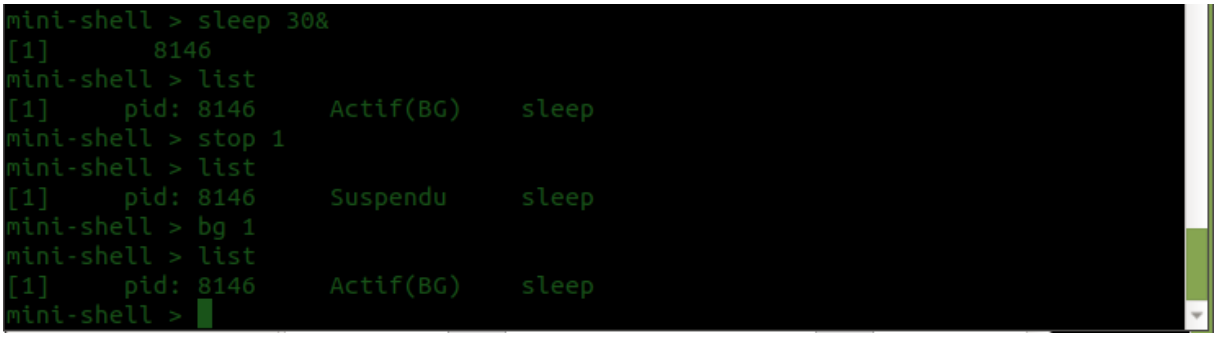
voici un exemple d'utilisation de list et stop :

```
mini-shell > sleep 10&
[1]       5746
mini-shell > sleep 20&
[2]       5755
mini-shell > list
[1]     pid: 5746     Actif(BG)     sleep
[2]     pid: 5755     Actif(BG)     sleep
mini-shell > stop 2
mini-shell > list
[2]     pid: 5755     Suspendu      sleep
mini-shell >
```

c) La commmande '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* : ***Gestion des redirections***

Les méthodes ***void*** rediriger_sortie(***char**** fichier) et ***void*** redirigier_entree(***char**** fichier) permettent respectivement de rediriger la sortie / entrée standard vers le fichier dont le nom est à l'entrée de la méthode.

Question *9- 10* : ***Tubes simples et pipelines***

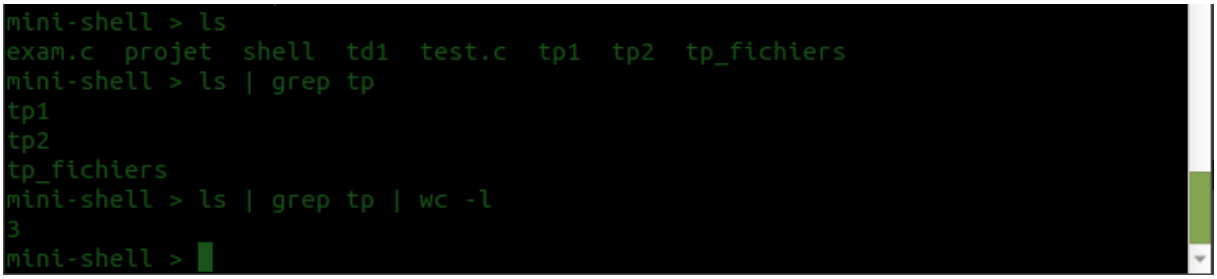
Pour la gestion d'une commande contenant un nombre quelconque de tubes, je calcule d'abord le nombre de tubes, je crée tous les tubes nécessaires comme le montre le schéma suivant :

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

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

Puis chaque commande élémentaire est exécutée durant une itération de la boucle. À la fin de la boucle, le père ferme tous les tubes et attend la terminaison de ses fils.

Voici un test d'une commande de deux tubes :



On peut aussi rediriger l'entrée et/ou sortie :