Rapport Minishell - SADURNI Thomas

INTRODUCTION :

Mon projet comprend deux fichiers minishell : un fichier où la taille des processus dynamique, dans le fichier minishell_dynamique.c et l'autre où la taille n'est pas dynamique dans le fichier minishell.c. J'ai décidé de rendre ces deux fichiers car je n'ai pas eu le temps de vérifier dans la totalité les fonctionnalités et les erreurs possibles(Seg Fault) dans minishell_dynamique.c. En revanche le code reste globalement le même entre les deux, j'ai seulement ajouté une nouvelle structure nommée "liste_job_struct" qui comprend la taille et la liste des processus.

STRUCTURE DES FICHIERS :

Vous trouverez reaccmd.c/h, les fichiers donnés en début de projet qui permette de récupérer les commandes de l'utilisateur, en gérant les pipes et les commandes en arrière plan en utilisant "&".

J'ai finalement décidé de laisser tout dans le même fichier, car j'ai rendu deux minishell et je ne voulais pas que vous confondiez les fichiers h et c pour chaque main.
Ainsi, vous trouverez en début de code les différentes couleurs ROUGE VERT JAUNE etc. pour pouvoir mettre de la couleur dans le shell.
Ensuite vient les fonctions pour les processus. Les constantes UNDEF, FG, BG, ST respectivement pour undefined, foreground, background et stopped représentent les états des processus. J'ai décidé de créer une nouvelle structure nommée job_struct qui définit le processus avec son pid (jobpid), son id (jobid), son état (jobetat) et sa commande (cmd).
C'est à partir d'ici que commence la différence entre les deux fichiers. Dans le premier ("minishell.c") j'ai créé un tableau de "job_struct" : listedesjobs[JOBMAX] avec JOBMAX limité à 15. Dans le deuxième, j'ai implanté une nouvelle structure comme dit en introduction, pour pouvoir gérer la taille de façon dynamique. Elle s'appelle aussi listedesjobs.
Les différentes fonctions et le main() viennent après.

LES FONCTIONS :

- **display_prompt :** affiche le prompt avec le nom de l'utilisateur, le dossier courant. Cette fonction "free" le dossier à chaque fois et affiche le prompt au couleur du drapeau français.
- **clearjob :** remise à zéro du processus
- **clearliste** (minishell_dynamique.c) : remise à zéro de la liste des processus, en utilisant clearjob()
- **initjob** : initalisation des jobs, utilise respectivement clearjob et clearliste pour minishell.c et minishell_dynamique.c
- **maxjobid** : retourne l'ID maximale de l'ensemble des processus.
- **addjob** : ajoute le job à la liste des processus

- **suppjob :** supprime le jonb de la liste des processus
- **changeEtat :** changer l'etat des processus
- **getJobFromPid** : retourne le processus selon un PID
- **getJobIdFromPid :** retourne l'id du processus selon un PID
- **getJobPidFromId :** retourne le processus selon un ID
- **freecommandejobs** : libère les commandes d'un processus
- **listjobs :** liste les processus avec leur ID, PID, Etat, et commande, retourne rien si pas de jobs.
- **handler_fils** : traitant du signal SIGCHLD
- **handler_tstp :** traitant du signal SIGTSTP.
- **handler_int :** traitant du signal SIGINT
- **stop :** commande stop, qui permet de suspendre un processus.
- **bg et fg :** ne sont pas utile ici.
- **redirection :** gestion des redirection
- **freeall** : libère la mémoire des commandes.

J'ai décidé de traiter séparement les signaux SIGINT et SIGTSTP pour la gestion des CtrlC CtrlZ, pour que ce soit plus clair.
Les commandes **bg fg** et **stop** fonctionnent normalement. Des exemples ou instructions sont affichés en tapant seulement fg, stop ou bg en ligne de commande.
La commande **jobs** ou **list** permet de lister les processus en cours d'exécution.
La commande **exit** permet de quitter le minishell.

PRÉCISION POUR LE MAIN :

La première boucle est la boucle principale, qui ne s'arrête que lorsque la commande exit est tapée avec le clavier. La premiere boucle for est faite pour apercevoir la commande que l'utilisateur tape, elle est optionnel, cependant je la laisse en guise d'aide pour l'utilisateur.
En parcourant le code du main(), plusieurs if/elseif sont implantés, ils correspondent aux commandes internes (cd et exit) et intégrées (fg, bg, stop, list, jobs...).
Pour la gestion des fils, je bloque les signaux SIGTSTP et SIGINT pour ne pas que les CtrlZ ou CtrlC affectent ces processus, je ne les démasque pas.
A chaque commande tapé est affiché une phrase indique que le processus commence, puis avertit lorsqu'elle est stoppée ou arrêtée. C'est là aussi en option, mais utile pour l'utilisateur. En revanche cela est très encombrant lors de l'utilisation des pipes, mais je laisse comme ça. De plus des messages peuvent être décommentés dans les fonctions addjob et suppjob pour avertir l'utilisateur de l'ajout ou la suppression d'un processus dans la liste.

MAKE :

La commande make du makefile génère les exécutables shell et shelldynamic.

Difficultés rencontrées : Difficile de faire un shell entier en partant de rien, donc tout a été très compliqué à implanter, surtout la gestion des signaux SIGINT, SIGTSTP et des pipes. J'ai donc passé énormément de temps sur ce projet.

TEST :

Commande	Résultat	OK/PAS OK
ls -l > toto	ecirre la sortie de ls -l dans le fichier toto	OK
rm toto	supprime le fichier toto	OK
cat minishell.c | wc -l	compte le nombre de ligne dans le fihcier minishell.c	OK
cat minishell.c | grep listedesjobs | wc -l	compte le nombre de ligne dans minishell .c avec listedesjobs	OK

Si vous souhaitez que je garde qu'un seul fichier minishell et que je fasse des .h et .c pour les processus et les fonctions, je peux le faire. J'ai jusqu'au 30 mai, date limite de rendu.