Programmation des systèmes d'information

Thème : Réalisation d'un minishell

Travail réalisé par : - SAD SAOUD Farid
- ZOUAOUI Mohamed Cherif

Exécution :
Pour lancer le minishell, on commence par compiler le fichier **Makefile**. Ce fichier spécifie comment compiler et lier les différents composants ou fichiers de notre programme source pour créer l'exécutable final. Puis, on utilise la commande "**make**" pour compiler le Makefile et générer le fichier exécutable "**minishell**", qui sera ensuite exécuté (**./minishell**).

Fonctionnement de notre Minishell :
Dans le répertoire "**src**", chaque fichier source joue un rôle essentiel dans l'exécution du **minishell** :

parser.h, parser.c : Ces deux fichiers source sont dédiés à la manipulation des chaînes de caractères, notamment pour traiter les commandes fournies en ligne de commande. La distinction entre "**parser.h**" et "**parser.c**"réside dans le fait que ce dernier contient le code source, y compris les définitions des fonctions, tandis que l'autre est un fichier qui rassemble les déclarations des fonctions et les constantes. Ce fichier sera par la suite inclus dans le fichier "**main.c**".
Voici les différentes fonction de notre programme :

`-int trim( char* str )` : Cette fonction prend en paramètre une chaîne de caractères et supprime les espaces au début et à la fin de cette chaîne.

`-int clean( char* str )` : Cette fonction prend en paramètre une chaîne de caractères et supprime les espaces doubles dans celle-ci.

`-void ajouter_espace_adroite(char* str , int position , int len )` : Cette fonction prend en paramètre une chaînes de caractères et 2 entier “position” et “len : longueur” qui va ajouter un espace avant la position donnée.

`-void ajouter_espace_agauche(char* str , int position , int len )` : cette fonction prend en paramètre une chaînes de caractères et 2 entier “position” et “len : longueur” qui va ajouter un espace après la position donnée.
Les deux dernières fonctions seront utilisées dans la fonction separate_s().

`-int separate_s ( char* str , size_t max )` : Cette fonction prend en paramètre une chaîne de caractères et une longueur max et ajoute un espace avant et après un caractère spéciale (“>>”,”<<”,”;”,”||”,”&&”,”&”,”2>>”,”2>&1”,”2>”,”>“,“<“,“|”,”>&2”).

`-int substenv ( char* str , size_t max )` : Cette fonction prend en paramètre une chaine de caractère et une longueur max et remplace les mots de type $mot ou ${mot} par les variables d’environnement de mot correspondant.

`-int strcut ( char* str , char sep , char ** tokens , size_t max)`: prend en paramètre une chaîne de caractère, un tableau de chaînes, un séparateur sep et une longueur max et qui découpe la chaîne sur un caractère choisi(espace dans notre cas) en ses différents mots dans un tableau de chaînes.

Fichier cmd.h et cmd.c : Le fichier cmd.h contient la structure d'une commande ainsi les déclaration des fonctions qui sont utilisés pour l'initialisation, l'analyse et l'exécution des commandes. Le fichiers cmd.c contient l'implémentation de ces fonctions.

- `struct cmd_t` : C'est la structure de la commande, elle contient toutes les informations nécissaire pour l'exécution d'une commande.

- `int init_cmd(cmd_t * p )` : Cette fonction permet d'initialiser la structure d'une commande passée en paramètre avec des valeurs par défaut avant l'exécuter.

- `int exec_cmd(cmd_t * p )`: Cette fonction prend en paramètre une commande en fonction de son nom. Si elle est **"built-in"** elle l'exécute directement dans le processus du shell, sans nécessiter le lancement d'un nouveau processus distinct. Sinon elle l'exécute dans un autre processus en utilisant la fonction **execvp**.

- `int parse_cmd(char * tokens[], cmd_t * cmds , size_t max )`: Cette fonction prend en paramètre un tableau des tokens (expliqué dans le fichier main) et un tableau de commandes qui va être remplit en fonction de contenu de ces tokens.

- `void add_fdclose(int * fdclose , int fd )`: une fonction qui permet d'ajouter un descripteur à la fin de tableau **fd_close**.

- `void merge_fdclose ( int * tab1 , int * tab2 )`: une fonction qui nous permet de fusionner deux tableaux de descripteurs de fichiers.

Fichier builtin.h et builtin.c : L'ensemble des fonctions présentes dans ces deux fichiers est destiné à la gestion des commandes intégrées.

- `int is_builtin( const char * cmd )`: Cette commande prend en paramètre une chaîne de caractère (une commande) et vérifie si une commande est built-in. Elle retourne 0 si c'est vrai et 1 sinon.

- `int builtin(cmd_t * cmd )`: Cette fonction prend en paramètre une commande et elle l'exécute.

- `int cd(const char * path , int fderr )`: Cette fonction sert à changer le répertoire de travail actuel de minishell.

- `int export(const char * var , const char * value , int fderr )`: La fonction "export" prend une variable d'environnement('var') et un descripteur de fichier d'erreur ('fderr'). Elle vise à exporter cette variable d'environnement avec sa valeur actuelle, en utilisant la fonction 'setenv. Si la variable est vide, elle affiche un message d'erreur et retourne un code d'erreur.

-*int* exit_shell(const *char* * *var* , const *char* * *value* , *int* *fderr* *:* Cette fonction nous permet de quitter le minishell.

Fichier main.c : C'est le programme principal où toutes les fonctions nécessaires à l'exécution de **minishell** sont appelées. On importe les deux fichier **cmd.h** et **parser.h** pour utiliser l'ensemble des fonctions.
Dans ce programme, trois tableaux sont déclarés : "**cmdline**" pour stocker la commande entrée dans la ligne de commande, "**cmdtoks**" qui contient les différents mots composant cette commande, et "**cmds**" est un tableau de commandes qui sera rempli en fonction du tableau de tokens.

Dans ce programme, une boucle infinie est utilisé afin que le programme reste en cours d'exécution jusqu'à ce que l'utilisateur choisisse de le quitter. À l'intérieur de la boucle on a l'ensemble des opérations suivantes :

- Effacer le contenu des tableaux **"cmds"**, **"cmdline"**, **"cmdtoks"**.
- Initialiser les valeurs par défaut dans cmds en utilisant la fonction **init_cmd()**.
- Lire une ligne dans **cmdline**.
- Supprimer les espaces en début et en fin de ligne en utilisant la focntion **trim()**.
- Ajouter d'éventuels espaces autour de ; ! || && & ... en utilisant la fonction la fonction **separate_s()**.
- Supprimer les doublons d'espaces en utilisant la fonction **clean()**.
- Traiter les variables d'environnement en utilisant la fonction **substenv()**.
- Découper la ligne dans **cmdtoks** en utilisant la fonction **strcut()**.
- Traduire la ligne en structures **cmd_t** dans cmds en utilisant la fonction **parse_cmd()**.
- Exécuter les commandes l'une après l'autre, en utilisant la fonction **exec_cmd()** puis avancer dans la boucle en vérifiant s'il y a des prochaines commandes à exécuter.