RAPPORT DE PROJET SYSTEM D'EXPLOITATTION

Interpréteur de commandes

Minish

ABDULSALAM Mohammad Kabir

SADIK Sofiane

Licence 3 Informatique

Système d'Exploitation

Table of Contents

1. Présentation du Sujet

Notre programme est un mini interpréteur de commandes similaire à celle utilise dans le linux, **bash**, capable de reproduire les fonctionnalités de bases des différents interpréteurs de commandes d'UNIX.

2. L'objectif du Projet

L'objectif est de créer un mini interpréteur de commandes pour un system d'UNIX capable d'exécuter correctement un ensemble de commandes dont la structure est semblable à celles interprétées dans une console UNIX. Pour ce faire, notre interpréteur de commande devra être capable de gérer les tubes et les redirections de flux etc.

2.1 Fonctionnalités disponibles

- Gestion des commandes en tâches de fonds
- Redirection d'entrées sorties
- Gestion des tubes
- Prise en compte des séparateurs (guillemets et apostrophe)
- Gestions des variables simples et des variables d'environnements
- Gestion du répertoire courant et changement de répertoire
- Gestion des historiques pour rappeler une commande déjà exécutée

3. Choix de Structure de Données

3.1 Organisation des Modules

Notre projet est découpe en 4 modules (4 fichiers « header »). Chaque module à ces propres fonctionnalités. Les modules sont les suivants :

- parser.h : Le parseur des commandes tapé par l'utilisateur. Il est chargé de d'analyser et découpée ces commandes selon certain critères.
- var_manager.h : chargé de la gestion de variables du shell.
- utility.h : contient des fonctions utiles. Ces fonctions sont utilisées dans d'autre module.
- minishell.h : chargé d'exécuté les commandes tape par l'utilisateur

3.2 Descriptions des Modules

3.2.1 parser.h

Ce module est chargé d'analyser et découpé les command taper par l'utilisateur et de la décomposer par ces fonctions en une liste de commandes. Sa structure est le suivant :

Structure de données

```
#define MAX_ARGUMENTS 2048
typedef struct noeud
{
   char*  arguments[MAX_ARGUMENTS];
   int argCount;
   char const* inputRedirect;
   char const* outputRedirect;
   int background;
   struct noeud* next;
}cmdLine;
```

Où arguments[] est un tableau contiendra les arguments déjà décomposé par les fonctions, argCount est le nombre d'arguments, inputRedirect contiendra les arguments avant un symbole de redirection (<, >, >>), outputRedirect contiendra les arguments avant un symbole de redirection (<, >, >>), background est un booléen(0 ou 1) qui détermine si un command a été tapée avec et commercial (&) et next la liste suivant.

Les fonctions de parser.h

- char* parseWord(char* str)
  Découpe *str* en prenant compte les symboles de redirection. Si une de ces symboles est rencontrer, on commence à découpé de cette endroit.

- void extractRedirections (char* strLine, cmdLine *command)
  Extraire les arguments avant et après un symbole de redirection (<, >, >>) dans *strLine* et l'affecte ces commandes a *command*.

- char* strClone (const char* source)
  Créer simplement une copie de la *source* et renvoie un pointeur sur cette copie.

- int isEmpty(const char* str)
  Vérifie si *str* est vide.

- cmdLine* parseSingleCmdLine(const char* strLine)
  Découpe *strLine* avec espace comme délimiteur

- cmdLine* parseCmdLines(char* line)
  Découpe *line* avec pipe (|) comme délimiteur

- cmdLine* parseCmd(const char* strLine)
  Découpe *strLine* en prenant compte les pipes(|), l'espace et les redirections.

- void freeCmdLines(cmdLine* command)
  Libère les tous les informations dans la liste chaînée *command*.

3.2.2. var_Manager.h

Ce module est chargé à vérifier si un command taper est correspond à une affection, de créer une variable et de supprimer une variable dans la liste. Il est aussi charger de remplacer la valeur d'un variable existant dans la liste. Sa structure est le suivant :

Structure de données

```
#define MAX_NAME 512
#define MAX_VALUE 2048

typedef struct node
{
  char name[MAX_NAME];
  char value[MAX_VALUE];
  struct node *next;
} variable;
```

Où name est le nom du variable et value est la valeur du variable.

Les fonctions de var_Manager.h

- variable* getVariable(const char* name, variable* var)
  S'il existe dans la liste, renvoie le variable *var* avec le nom *name*

- int findVariable(const char* name, variable* var)
  Si le variable *var avec* le nom name *existe* dans la liste, il renvoie 1 sinon 0

- variable* removeVariable(const char* name, variable* var)
  Supprime le variable *var avec* le nom name s'il *existe* dans la liste

- variable* addVariable(const char* name, const char* value, variable* var)
  Ajoute le variable *var avec* le nom name *existe* dans la liste.

- void freeEnvironment(variable* var)
  Supprime tous les variables dans la liste.

- int check_syntax(char* userInput)
  Vérifie si le *userInput* est une syntaxe de création d'un variable.

- void expandVariables(cmdLine* command)
  Entendre les variables dans *command*. Ca teint en compte des commandes commençant par un signe du dollar ou un guillemet ou une apostrophe.

- void expandArguments(cmdLine *command)
  Entendre les arguments. Prise en compte du tilde (~).

- void executeVariables(char* userInput, variable* var)
  Créer les variable d'environnement, s'il ces variables existes, il fait la mise à jour de leur valeurs.

3.2.3 utility.h

Ce module contient 2 fonctions utiles qui seront utilisés par les autres modules.

Les fonctions de utility.h

- char *replaceCharacter(char *str, char *original, char *replace)
  Cherche le string *str* dans le string *original* et s'il le trouve il le remplace par *replace.*

- char* trimspace(char *str)
  Enlève les espaces avant et après *str*.

- int countPipes(char *userInput)
  Compte le nombre de pipe (|) présent dans *userInput.*

3.2.4 minishell.h

Ce module contient des fonctions qui exécutent les commandes entré par utilisateur. Il est le cœur du minishell. Il effectue toutes les opérations nécessaires pour la communication entre processus.

Les fonctions de minishell.h

- void runPipedCommands(cmdLine* command, char* userInput)
  Cette fonction exécute les commandes qui contiennent des pipes (|). Il est chargé de relier les entrée et sorties des commandes entre les pipes (|). On commence par récupérer le nombre de pipes dans *userInput,* déclare et initialise un tableau des pipes. Puis tant que *command* n'est pas null, pour chaque pipe rencontré, on fork un processus. On exécute dans les processus fils les commandes.

- void handle_signal(int signum)
  Fonction chargé d'ignorer les cas où l'utilisateur tape Ctrl+C sur le clavier. Le handler d'un *struct sigaction* sera initialise par cette fonction.

- void changeDirectory(char* argv[])
  Fonction chargé de changer le répertoire. Il utilise la fonction *getenv* et *chdir*. Il vérifie d'abord si l'utilisateur à donner le nom de répertoire comme argument. S'il n'a pas donné ce nom, il change le répertoire au répertoire *HOME,* sinon il change le répertoire à celle que l'utilisateur à donner en arguments, s'il existe.

- int executeCommands(cmdLine* command, char* userInput, variable* var)
  Fonction chargé d'exécute les commandes internes, externes et affectation des variables simples dans la list *command*. Il vérifie aussi si le command est à exécute en tache de fond. Si oui, il n'attend pas que le processus finisse, sinon il attend.