Réalisation d'un mini Shell

On veut réaliser un « mini shell » (interpréteur de commandes) permettant les fonctionnalités suivantes :

Il affiche un prompt contenant le nom de la machine (HOSTNAME) suivi du nom du répertoire courant et suivi d'un dollar.
Exemple : Station14:/home/etudiant$

Il permet de saisir une chaîne de caractères (composée d'au plus 20 arguments de 200 caractères maximum). Une commande peut être soit interne (primitive) soit externe (appel à exec). Une commande simple consiste en un nom de commande suivi d'une suite optionnelle d'arguments séparés par des espaces ou des caractères de tabulation, chacun d'entre eux étant un seul mot ou une chaîne entre guillemets (") .

Il permet d'exécuter toutes les commandes externes (accessibles grâce au PATH) et les commandes internes : **set** et **cd**

Pour les commandes externes on considère qu'elles peuvent recevoir des options (caractères) indiquées derrière le signe '-' suivi d'arguments (les séparateurs sont des espaces et/ou des tabulations) la fin de la ligne de commande est indiquée par le caractère retour chariot '\n'. Les caractères : '<', '>' , '>>' et '|' indiquent respectivement la redirection de l'entrée standard, la redirection de la sortie standard, l'ajout dans un fichier et le pipe. On suppose que dans une ligne de commande il n'existe qu'au plus un seul de ces caractères.

Pour simplifier ce mini-shell ne traitera pas les méta-caractères tels que : '*', '?', etc.
et ne disposera pas de structures de contrôles telles que if, while etc.

Primitives utiles :

La primitive **chdir** change le répertoire courant en un répertoire donné en paramètre

int chdir(char *path)

cette fonction retourne –1 en cas d'erreur et 0 en cas de succès.

La primitive **dup** duplique un descripteur de fichier existant et retourne un nouveau descripteur de fichier qui est ouvert sur le même fichier ou conduit. La primitive garanti que la valeur du descripteur retournée est la plus petite possible.

int dup(int fd)

cette fonction retourne –1 en cas d'erreur et la valeur du nouveau descripteur en cas de succès.

Un petit programme (**parser.c**) vous est fourni pour analyser les lignes d'entrée. Cette analyse consiste à regrouper les caractères en unités syntaxiques, les lexèmes. Des exemples de lexèmes sont les mots, les chaînes entre guillemets et les symboles spéciaux comme < et |. Chaque lexème est représenté par une constante symbolique :

MOT : un argument ou un nom de fichier, s'il est entre guillemets, ces derniers sont supprimés une fois le lexème reconnu.
TUB : le symbole |.
INF : le symbole <.
SUP : le symbole >.
SPP : le symbole >>.
NL : le caractère nouvelle ligne.
FIN : lexème spécial signifiant que la fin du fichier a été atteinte. Si l'entrée standard est un terminal, l'utilisateur a tapé un EOT (Control D).

A chaque appel l'analyseur lexical retournera un lexème. Si le lexème est MOT, il retournera également une chaîne contenant les caractères qui le composent. Il est également capable de reconnaître les blancs et les tabulations sans rien retourner ou de retourner les lexèmes correspondant à une redirection ou un tube.

Notre parser est un automate fini : à la lecture les caractères sont soit reconnus immédiatement comme lexème, soit ils sont accumulés (par exemple les caractères d'un mot). A la lecture de chaque caractère, l'analyseur peut passer dans une autre état. Nous avons retenu quatre états : **Neutre**, **Spp**, **Equote**, **Emot**
Neutre : Etat de départ. Les espaces et les tabulations sont ignorés. Les caractères |, < et nouvelle ligne sont directement reconnus comme lexème (**TUB**, **INF** et **NL**).Le caractère > provoque le passage à l'état **Spp** dans lequel on teste s'il est suivi d'un autre > car > et >> sont deux lexèmes différents. Un guillemet provoque le passage dans l'état **Equote**, où la chaîne entre guillemets est assemblée. Tout autre caractère est considéré comme le début d'un mot sans guillemets. Le caractère est sauvegardé dans un tampon (mot) et l'état courant est changé en **Emot**.
Spp :Cet état signifie que le caractère > vient d'être lu. Si le caractère suivant est également >, le lexème **SPP** est retourné sinon **SUP** est retourné. Comme dans ce cas nous avons lu un caractère de trop, il est restitué à l'entrée (fonction ungetc()).
Equote : Cet état indique qu'un guillemet de début de chaîne a été lu. Les caractères sont accumulés dans un tampon jusqu'à ce qu'un guillemet fermant soit lu. Le lexème **MOT** ainsi que la chaîne seront retournés.
Emot : Cet état signifie que le premier caractère d'un mot a été lu et rangé dans un tampon. Les caractères suivants seront accumulés jusqu'à la rencontre d'un caractère spécial (comme |, espace, tab etc). Ce dernier est restitué à l'entrée puis le lexème **MOT** est retourné.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

typedef enum {MOT, TUB, INF, SUP, SPP, NL, FIN} LEX;

static LEX getlex(char *mot){
        enum {Neutre,  Spp, Equote, Emot } etat=Neutre;
```

```
int c;
char *w;
w=mot;
while ((c=getchar()) != EOF){

        switch(etat){
        case Neutre:
                switch(c){
                case '<':
                        return (INF);
                case '>':
                        etat=Spp;
                        continue;
                case '|':
                        return (TUB);
                case '"':
                        etat=Equote;
                        continue;
                case ' ':
                case '\t':
                        continue;
                case '\n':
                        return(NL);
                default:
                        etat=Emot;
                        *w++=c;
                        continue;
                }
        case Spp:
                if(c=='>')
                        return(SPP);
                ungetc(c,stdin);
                return(SUP);
        case Equote:
                switch(c){
                case '\\':
                        *w++=c;
                        continue;
                case '"':
                        *w='\0';
                        return(MOT);
                default:
                        *w++=c;
                        continue;
                }
        case Emot:
                switch(c){
                case '|':
                case '<':
                case '>':
```

```
                        case ' ':
                        case '\t':
                        case '\n':
                                ungetc(c,stdin);
                                *w='\0';
                                return(MOT);
                        default:
                                *w++=c;
                                continue;
                        }
                }
        }
        return(FIN);
}

main(int argc, char *argv[]){

        char mot[200];

        while(1)
                switch(getlex(mot)){
                case MOT:
                        printf("MOT: %s\n",mot);
                        break;
                case TUB:
                        printf("TUBE\n");
                        break;
                case INF:
                        printf("REDIRECTION ENTREE\n");
                        break;
                case SUP:
                        printf("REDIRECTION SORTIE\n");
                        break;
                case SPP:
                        printf("REDIRECTION AJOUT\n");
                        break;
                case NL:
                        printf("NOUVELLE LIGNE \n");
                        break;
                case FIN:
                        printf("FIN \n");
                        exit(0);
                }
}
```

L'étape suivante sera donc l'étude d'une fonction **commande** qui traite une commande terminée par nouvelle ligne. Nous envisagerons cinq cas de figure :

- la commande est simple sans redirection ni pipe. Les arguments sont simplement rangés dans un tableau (char *argv[]) pour utilisation ultérieure dans un appel **execvp**.
- la commande comporte un lexème INF,
- la commande comporte un lexème SUP,
- la commande comporte un lexème SPP,
- la commande comporte un lexème TUB.

Il y a trois possiblilités pour l'entrée standard :

- l'entrée par défaut (fd=0),
- un fichier, en présence d'une redirection d'entrée (lexème INF),
- l'extrémité de lecture d'un tube (lexème TUB).

Pour la sortie standard, il y a quatre possibilités :

- la sortie par défaut (fd=1),
- un fichier, en présence d'une redirection de sortie (lexème SUP),
- un fichier (mode ajout), en présence de >> (lexème SPP),
- l'extrémité d'écriture d'un tube (lexème TUB).

Ecrire donc la fonction **commande** permettant de sélectionner chaque cas (fonction **getlex**) puis une fonction propre à chaque lexème. Enfin, un programme principal qui affiche le prompt, permet la saisie de la commande et son exécution.