Diana Heddadji
L2 Informatique
16712076

# Système d'exploitation
# Projet Minishell

Licence : INFORMATIQUE
UFR: MITSIC (Mathématiques, informatique, technologies, sciences de l'information et de la communication)
Année: 2018/2019

## 0.1 Organisation du Projet

### 0.1.1 Les fichiers

- ☐ minishel.h
- ☐ main.c
- ☐ decouper.c
- ☐ moncd.c
- ☐ commandes.c
- ☐ Makefile
- ☐ mon-if.c (+ pas inclut dans le projet)

## 0.2 Description détaillée des fichiers

### 0.2.1 minishel.h

Il s'agit du fichier header du projet, c'est dans celui-ci que nous inclurons toutes les bibliothèques indispensables au projet.

Exemple :

En incluant cette bibliothèque;

**#include <sys/wait.h>**

On pourra par la suite, utiliser l'appel système **wait()** afin d'attendre le changement d'état du fils d'un processus.

On déclare également la constante PROMPT qu'on affichera au cours de l'exécution du programme;

**#define PROMPT "%s@%s$"**

Enfin, on énumérera les différentes variables nécessaires au projet ainsi que les prototypes des fonctions utilisées.

### 0.2.2 main.c

C'est le fichier principale du projet dans lequel on appellera les fonctions dans un ordre précis.

Tout d'abord, on inclut le fichier hearder vu plus haut, puis, on commence par copier le chemin d'accès absolu du répertoire de travail courant dans la chaîne pointée par **dirsbis**.

**getcwd(dirsbis, sizeof(dirsbis));**

On appelle ensuite, la fonction **decouper** qui va se charger de scinder le **PATH** en plusieurs répertoires.

Dans le **for**, on va lire et traiter chaque ligne de commandes en affichant à chaque fois le chemin du répertoire courant avec **dirsbis** et en récupérant la valeur de **USER** avec **getenv** le tout formaté par la constante **PROMPT**.

**for(printf(PROMPT, getenv("USER"), dirsbis); fgets(ligne, sizeof ligne, stdin) != 0; printf(PROMPT, getenv("USER"), dirsbis))**

Puis, on rappelle une nouvelle fois la fonction **decouper**, mais cette fois-ci pour scinder une ligne selon la présence d'un point-virgule afin de traiter plusieurs commandes à la fois. On placera "les mots" trouvés dans **commandes** (c'est-à-dire ceux juste avant chaque ";", il peut en avoir un ou plusieurs car on il peut y avoir des espaces...).

**decouper(ligne, ";", commandes, MaxMot);**

Ensuite, on déclare un deuxième **for**, imbriqué dans le premier, pour parcourir à présent les mots contenus dans **commandes** qu'on a remplit précédemment. Encore une fois, on appelle **decouper** afin de scinder les mots selon la présence d'un espace, d'une tabulatation ou d'un saut de ligne.

**decouper(commandes[c], " ", mot, MaxMot);**

A présent, on a bien uniquement un seul mots par case dans le tableau de chaîne de caractères **commandes**, on va donc pouvoir exécuter ses mots, qui sont enfaîte des commandes.

Afin que la commande qui nous permet de changer de répertoire courant **(cd ou change directory)** soit reconnue, on l'a crée de façon **interne** :

**char * cd = "cd";**

```
if(!strcmp(mot[0],cd)){
moncd(mot, dir, t);
getcwd(dirsbis, sizeof(dirsbis));
continue;
}
```

On déclare une chaîne de caractère initialisée à **"cd"**, qu'on compare ensuite avec le premier mot qu'on entre sur notre **shell**. **strcmp**, déclarée dans **include <string.h>**, va se charger d'établir cette comparaison. Si ces deux chaînes sont équivalentes, on appelle la fonction **moncd** et on récupère le chemin du répertoire courant avec **getcwd** après avoir changer de répertoire.

De même, pour sortir proprement du **shell**, on déclare une chaîne de caractère initialisée à **"exit"**

et si celle-ci est égale à **mot[0]**, on affiche **"Bye"** et on retourne **0**.

A la fin du **main**, on a fait un **fork** dans lequel l'enfant contient la fonction qui traite le cas d'un processus mis en arrière plan ainsi que la gestion de plusieurs commandes.

**monexec(pathname, mot, NULL, NULL);**

Si une commande est introuvable, on affiche un message d'erreur avant de faire appel à **exit**.

../screen/c1.png : minishell
../screen/c2.png : exit

### 0.2.3 decouper.c

Dans ce fichier, on trouvera une seule fonction mais qui a un rôle fondamental dans notre **shell** comme on a pu le constater, il s'agit de la fonction **decouper** qui scinde une ligne, un mot.. en fonction d'un séparateur qu'on indique en paramètre.

Voici son prototype qui est également définie dans le header **minishel.h** :
**void decouper(char * ligne, char * separ, char * mot[], int maxmot);**

Dans le tableau de chaîne **mot**, on y placera les mots trouvés dans **ligne**, qui a été scindé en fonction de **separ**, **maxmot** est la taille de **mot**.

Afin de construire ces mots, la fonction **decouper** utilise **strtok**, définie dans la bibliothèque **include <string.h>**, qui permet justement de scinder une chaîne en une séquence d'éléments lexicaux.

### 0.2.4 moncd.c

Dans ce fichier, on retrouvera la fonction **moncd** qui va se charger de changer le répertoire courant en fonction du chemin qu'indiquera l'utilisateur, son prototype est le suivant :

**void moncd(char ** mot, char * dir, int t);**

Par défaut, si **cd** ne prend aucun paramètre, on se redirige vers le répertoire **home**, si cela n'est pas possible, alors on tente de se placer dans le dossier **tmp**.

Sinon, on se place dans le dossier que l'on a indiqué en paramètre.

Ensuite, on fait un **chdir** dans le répertoire donné en argument ou dans celui indiqué par **HOME** afin de changer de répertoire. A la fin, on teste la valeur de la variable **t** pour savoir si une erreur s'est produite.

../screen/c3png : cd

### 0.2.5 commandes.c

Dans ce fichier, nous allons retrouver la fonction **monexec** qui va nous permettre de faire deux choses, à la fois tester si on souhaite mettre une commande en arrière plan, en comparant les mot présents dans le tableau de chaînes, **char * mot**[], avec **""**, mais aussi exécuter plusieurs commandes à la fois.

Voici donc son prototype :

**int monexec(char pathname[], char * mot[]);**

Pour la mise en arrière plan et la gestion de plusieurs commandes, on fait un **fork** dans lequel l'enfant récupère le **pid** avant d'exécuter les commandes, le parent lui fait un **wait(0)**, le tout dans un **for**.

../screen/c4.png : Gestion de plusieurs commandes
../screen/c5.png : sleep

## 0.3 Améliorations

Ajouter le traitement de &&
Ajouter le traitement des redirections (> » &> 2> 2»)
Ajouter le traitement des symboles (2>&1 1<&2)
Faire un historique des commandes : avec les touches fléchés
Traitement de quelques méta-caractères comme * ou ?
Ajouter le traitement des pipes (|)
Auto-complétion

Sources:

manpagesfr.free.fr/
co-main.c
cn-decouper.c
exo16$_m oncd.c$