# Mise en Œuvre 1/2

2 décembre 2013

L'objectif du TP d'aujourd'hui est de poser les bases de l'implémentation de votre shell.

## 1 Fonctionnalités du shell

**Fonctionnalités souhaitées :**

- Exécution de commandes avec des pipes
- Exécution de commandes avec des redirection d'entrée / sortie
- Un chien de garde (à chaque exécution d'une commande, le shell met fin aux processus générés au bout de 10 seconde si celle-ci est sans réponse).

**Fonctionnalités bonus :**

- Auto completion
- Historique des commandes

Typiquement votre shell doit être capable, d'exécuter correctement une commande de type

```
cat < /var/log/messages | grep ACPI| wc -l > truc.txt
```

## 2 Décomposition d'une commande

Une commande est séparée en un certain nombre de membres, délimités par des pipes (et le début et la fin de la commande).

Chaque membre comporte une ou plusieurs redirections d'entrées sorties :

- `cmd < f` est équivalent à un `cat f | cmd`. Note : un membre comportant un < ne peut pas être précédé d'un autre membre.
- `cmd > f` redirige la sortie standard de `cmd` vers un fichier f, qui est écrasé. Note : un membre comportant un > ne peut pas être suivi d'un autre membre.
- `cmd >> f` est identique au symbole précédent, mais l'écriture dans le fichier sur fait à la fin de celui-ci. Il n'est pas écrasé.
- `cmd 2>f` et `cmd 2>>f` sont identiques aux deux précédents, à cela près qu'ils portent sur la sortie d'erreur.

Pour exécuter une commande se décompose ainsi :

1. Séparer les différents membres
2. Regarder si les membres comportent des redirections d'entrée sortie
3. Effectuer le bon nombre de fork
4. Rediriger correctement STDIN, STDOUT et STDERR
5. Effectuer les exec.

6. Attendre la fin des fils

Il va très vite être nécessaire de définir une structure adaptée pour stocker une commande, je vous propose d'utiliser la structure suivante :

```
typedef struct commande {
 char cmd_initiale;      /* La chaine initiale tapee par l'utilisateur */
 char **cmd_membres;     /* a chaque case i chaine du membre i */
 unsigned int nb_cmd_membres; /* Nombre de membres */
 char ***cmd_args; /* cmd_args[i][j] contient le jieme mot du ieme membre *
 unsigned int *nb_args; /* Nombre d'arguments par membres */
 char ***redirection; /* Pour stocker le chemin vers le fichier de redirect
 int **type_redirection;/* Pour stocker le type de redirection */
} cmd;
```

en gardant l'exemple donné précédemment, cette structure doit être initialisée ainsi :

```
exemple.cmd_initiale=
        ''cat < /var/log/messages | grep ACPI| wc -l >> truc.txt''
exemple.cmd_membres[0]=''cat < /var/log/messages''
exemple.cmd_membres[1]=''grep ACPI''
exemple.cmd_membres[2]=''wc -l > truc.txt''
exemple.nb_cmd_membres=3
cmd_args[0][0]=''cat''
cmd_args[0][1]=NULL
cmd_args[1][0]=''grep''
cmd_args[1][1]=''ACPI''
cmd_args[1][2]=NULL
cmd_args[2][0]=''wc''
cmd_args[2][1]=''-l''
cmd_args[2][2]=NULL
nb_args[0]=1
nb_args[1]=2
nb_args[2]=2
redirection[0][STDIN]=''/var/log/messages''
redirection[0][STDOUT]=NULL
redirection[0][STDERR]=NULL
redirection[1][STDIN]=NULL
redirection[1][STDOUT]=NULL
redirection[1][STDERR]=NULL
redirection[2][STDIN]=NULL
redirection[2][STDOUT]=''truc.txt''
redirection[2][STDERR]=NULL
type_redirection[2][STDOUT]=APPEND
```

Vous mettrez la définition de cette structure ainsi que les prototypes des fonctions utiles à sa manipulation dans un fichier `cmd.h`. Ces fonctions incluent :

- `void aff_args(cmd *c);`
- `void free_args(cmd *c);`
- `void parse_args(cmd *c);` Remplit les champs cmd_args et nb_args_membres
- `void parse_membres(char *chaine,cmd *ma_cmd);` Remplit les champs initial_cmd, membres_cmd et nb_membres.
- `void aff_membres(cmd *ma_cmd);`

- `void free_membres(cmd *ma_cmd);`
- `int parse_redirection(unsigned int i,cmd *c);` Remplit les champs redir et type_redir du membre i
- `void free_redirection(cmd *c);`
- `void aff_redirection(cmd c, int i);`

Les fonctions aff et free effectuent l'affichage des informations (pour le debuggage par exemple) et la libération de l'espace mémoire alloué lors du remplissage des champs.

Créez une fonction `void exec_commande(cmd c)`, dans un fichier `shell_fct.c`, qui prend une commande duement initialisée et qui effectue la création des pipes, les fork et les execs correspondants. Étant donné le formattage des arguments, il est judicieux d'utiliser l'appel `execvp`.

**Pour la gestion des entrées de l'utilisateur, je vous conseille de regarder du côté de la librairie GNU/Readline (man 3 readline), bien que cela ne soit pas une obligation.**