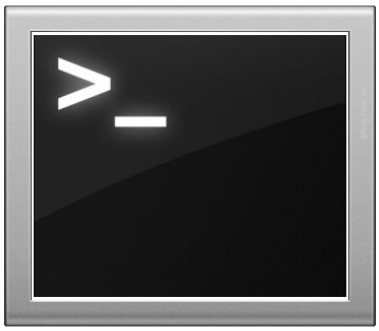
université de BORDEAUX

Projet Remote Shell
Licence 3 Informatique



Corentin DAVIDENKO - Corentin MOREAU - Antoine Rivalier

Dépôt du projet: *https://github.com/corentinmb/shelldmr/*

Le projet

Le projet consiste en la réalisation d'un shell Unix permettant une connexion à distance via SSH. Nous avons utiliser Git et Github pour gérer notre code source. L'embryon de code fourni comporte un analyseur syntaxique programmé à l'aide de l'outil LEX et un analyseur grammatical programmé à l'aide de l'outil YACC.

Les commandes de base du Shell ainsi que la connexion distante étaient donc à implémenter. Pour ce faire, nous sommes passés par plusieurs méthodes que nous allons expliciter dans ce rapport.

Travail réalisé

Fonctionnalités	Difficulté	Score (/10)
Commandes Externes		
commande simple en avant-plan	+	10
commande simple en arrière-plan (&)	+	10
élimination des zombies	++	10
Expressions		
expression ; *expression*	+	10
expression || *expression*	+	10
expression && *expression*	+	10
(*expression*)	++	10
expression | *expression*	++	10
expression > fichier	++	10
expression < fichier	++	10
expression >> fichier	++	10
expressions récursives	+++	0
Commandes Internes		
echo	+	10
date	+	10
cd	+	10
pwd	+	10
history	+	10
hostname	+	10
kill	+	10
exit	+	10
Remote Shell		
gestion d'une liste de machines (add, remove, list)	+	8
exécution sur un unique shell faussement distant (remote localhost ...)	+	10
exécution sur plusieurs shells distants avec *ssh* (remote all ...)	++	10
affichage dans des fenêtres séparées avec *xcat.sh*	+++	9

Les commandes Externes

Par défaut, les commandes sont lancées en avant-plan, si l'on rajoute le symbole & (esperluette) la commande se lance en arrière-plan, c'est-à-dire que la commande est lancée en parallèle du Shell. Dans notre code, nous attribuons la valeur 1 à la variable "bg" pour lancer la commande interne en arrière-plan si l'analyseur syntaxique détecte l'esperluette.
La gestion des zombies est effectuée à chaque tour de boucle du programme:

```
while (1){
  if (my_yyparse () == 0){  /* L'analyse a abouti */
    afficher_expr(ExpressionAnalysee);
    if (evaluer_expr(ExpressionAnalysee) == 0){
      fflush(stdout);
      expression_free(ExpressionAnalysee);
    }
  }
  else {
    /* L'analyse de la ligne de commande a donn? une erreur */
  }
  waitpid(-1, NULL, WNOHANG);
}
```

Expressions

Les commandes séparées par le point-virgule ";" sont exécutées séquentiellement:

```
int sequence(Expression *e)
{
    evaluer_expr(e->gauche);
    return evaluer_expr(e->droite);
}
```

Les commandes séparées par le double pipe “||” sont exécutées séquentiellement uniquement si la première commande n’est pas exécutée avec succès:

```
int sequence_or(Expression *e)
{
    int status = evaluer_expr(e->gauche);
    if ( status == 0)
        return 0;
    else
        return evaluer_expr(e->droite);

}
```

Les commandes séparées par la double esperluette “&&” sont exécutées séquentiellement uniquement si la première commande est exécutée avec succès:

```
int sequence_and(Expression *e)
{
    int status = evaluer_expr(e->gauche);
    if ( status == 0)
        return evaluer_expr(e->droite);
    else
        return status;
}
```

La commande exécutée entre parenthèses est exécutée dans un sous-shell.

Les commandes séparées par un pipe “|” permet d'enchaîner l'exécution des commandes, en branchant la sortie de la première commande sur l'entrée de la seconde.

La commande suivie du chevron ouvrant “<” permet une redirection de l’entrée du programme avec par exemple, un fichier.

La commande suivie du chevron fermant “>” permet une redirection de la sortie du programme avec par exemple, un fichier.

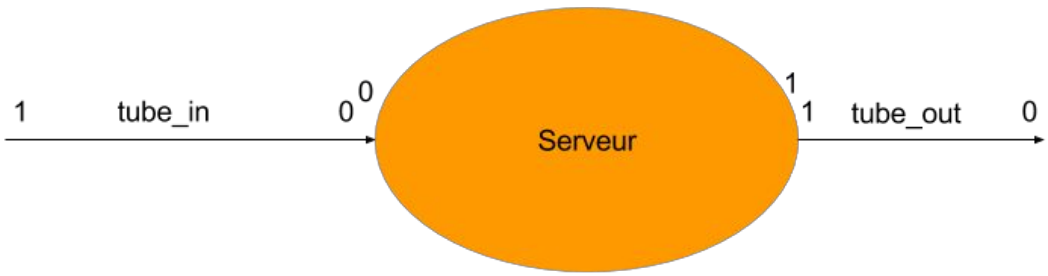
La commande suivie du double chevron fermant “>>” permet une redirection de la sortie du programme avec par exemple, un fichier, de façon à positionner le curseur d’écriture à la fin.

Les commandes récursives n’ont pas été implémentées.

Commandes Internes

`echo`: Elle permet simplement d'afficher une ligne de texte.
`cd`: Elle permet de se déplacer dans un dossier.
`date`: Affiche la date
`pwd`: Affiche le dossier courant
`history`: Affiche l’historique des commandes
`hostname`: Affiche le nom du PC
`kill`: Envoie un signal (généralement SIGKILL)
`exit`: Quitte le Shell

Remote Shell



Le Remote Shell est implémenté grâce à des pipes. La discussion est établie grâce aux branchements des pipes à travers les différentes remote et le Shell principal (bufferisation en ligne).

Nous disposons également de l'affichage distant grâce à l'implémentation du fichier `xcat.sh` qui exécute un terminal pour chaque remote crée. Les informations concernant les Shells distants sont conservées dans un tableau de structures contenant le PID, le hostname ainsi que les tubes (entrée et sortie) de chaque Shell.

Répartition des tâches

Au niveau de la répartition des tâches, nous avons pu diviser le travail suivant les connaissances de chacun.

En l'occurrence, *Corentin Davidenko* avait terminé le TD Machine (TD3) sur le Shell, ainsi il a pu implémenter les expressions.
Corentin Moreau a pu développer la partie concernant les commandes internes (réécriture des commandes, sans la prise en compte des options).
Antoine Rivalier, quant à lui, a pu développer la partie Remote Shell pour interconnecter les Shells à l'aide des pipes, forks... Cette partie était la plus importante à réaliser.