Rapport projet Programmation Système

LABBE Emeric, NERESTAN Clément, Pellegrini Charles

11 janvier 2016

Table des matières

0.1 Commandes internes 2
0.1.1 echo 2
0.1.2 date 2
0.1.3 cd 2
0.1.4 pwd 2
0.1.5 history 2
0.1.6 kill 3
0.1.7 exit 3
0.2 Remote Shell 4
0.2.1 Gestion d'une liste de machines (add, remove, list) . . 4
0.2.2 Exécution sur un unique shell faussement distant . . . 4
0.2.3 Exécution sur plusieurs shell distants avec ssh 4
0.2.4 Affichage dans des fenetres séparées avec xcat.sh 4
0.3 Expressions 5
0.3.1 instruction simple 5
0.3.2 expression ; expression 5
0.3.3 expression || expression 5
0.3.4 expression & & expression 6
0.3.5 (expression) 6
0.3.6 expression & 6
0.3.7 expression | expression 6
0.3.8 expression > fichier 6
0.3.9 expression < fichier 7
0.3.10 expression » fichier 7
0.4 Bilan 8
0.5 Apport du projet 9

0.1 Commandes internes

0.1.1 echo

Renvoie la chaîne de caractère qui vient d'être entrée.

0.1.2 date

Retourne la date et l'heure au format commun en France (Jour mois Année heure)

0.1.3 cd

Redirige le répertoire courant vers le dossier donnée en paramètre. Ou remonte au précédent en utilisant ".."

Cette commande Interne aura été plus simple que prévu à créer car au début nous pensions qu'il fallait mémoriser le répertoire courant actuel et modifié la chaîne de caractère. Nous pensions aussi qu'il fallait modifier la chaîne de caractère si le paramètre était "..". Nous étions donc partie pour gérer tout cela puis nous avons apprit que l'appel chdir gère tout cela.

0.1.4 pwd

Récupère le répertoire actuel avec la commande get_current_dir_name() et l'affiche.

0.1.5 history

Retourne l'historique des commande entrées. Pour cela nous avons utiliser la librairie readline/history .

0.1.6 kill

Envoie un signal (par defaut Term) mis en option (uniquement via numéro du signal) au(x) PID donné(s) en paramètre. On peux aussi entrer "Kill-l" pour voir la liste de tout les signaux que l'on peux envoyer.

La plus grosse difficulté dans l'implémentation de cette commande interne ne fut pas l'utilisation de l'appel système Kill que nous avions déjà utilisé en cours mais la gestion des chaînes de caractère.

0.1.7 exit

Quitte le shell en transmettant le code de retour (ici atoi(c[0]))

0.2 Remote Shell

0.2.1 Gestion d'une liste de machines (add, remove, list)

0.2.2 Exécution sur un unique shell faussement distant

0.2.3 Exécution sur plusieurs shell distants avec ssh

0.2.4 Affichage dans des fenetres séparées avec xcat.sh

0.3 Expressions

On utilise deux fonctions, evaluer_expr et exec_expr. La fonction evaluer_expr regarde si l'expression est une commande interne : dans ce cas elle appèle directement la fonction correspondante. Autrement elle se fork, et la ligne

```
exit(exec_expr(e));
```

fait que le fils se quittera forcement après l'appel à exec_expr.
Le père, lui, appèle directement waitpid avec le pid du fils. ainsi, il ne peut pas y avoir de processus zombie : le fils est récupéré par le père, et les processus éventuellement crées par le fils seront récupérés par init.
Cette implementation a un autre avantage : on ne redirige les descripteurs de fichiers que des processus fils, il n'y a pas besoin de les sauvegarder avant de les modifier.

0.3.1 instruction simple

e->type == SIMPLE
L'expression est à executer directement. Le processus ayant déjà été forké, il suffit d'appeler execvp.

0.3.2 expression ; expression

e->type == SEQUENCE
L'expression de gauche est passée en paramètre à evaluer_expr, celle de droite à exec_expr. Dans le cas ou celle de gauche est une commande interne, elle peut modifier le comportement de celle de droite (par exemple cd peut changer le répertoire courant de l'expression de droite)

0.3.3 expression || expression

e->type == SEQUENCE_OU
Évaluation de la première expression, si celle-ci retourne 0 (est considérée comme vraie), la suivante ne s'exécute pas. Encore une fois, l'expression de gauche est passée en paramètre à evaluer_expr, celle de droite à exec_expr.

0.3.4 expression & & expression

e->type == SEQUENCE_ET
Comme pour SEQUENCE_OU, mais on évalue la première expression, si elle est fausse on retourne l'évaluation de la deuxième, sinon on ne l'évalue pas mais on retourne vrai.

0.3.5 (expression)

e->type == SOUS_SHELL
Création d'un sous shell dans lequel va s'exécuter l'expression. Ici, on appèle simplement evaluer_expr.

0.3.6 expression &

e->type == BG
Tache en arrière plan. On fork et le fils appèle exec_expr, le père n'attend pas le résultant. Le père se terminant de suite après, le fils est récupéré par init.

0.3.7 expression | expression

e->type == PIPE
Pipe des expressions. Le père et le fils appèlent tous les deux exec_expr et non evaluer_expr, les forker une fois de plus serait inutile.

0.3.8 expression > fichier

e->type == REDIRECTION_O
Redirection de la sortie vers le fichier spécifié.

0.3.9 expression < fichier

e->type == REDIRECTION_I
Redirection du contenu du fichier vers l'entrée standard.

0.3.10 expression » fichier

e->type == REDIRECTION_A
Redirection de la sortie en mode APPEND.

0.4 Bilan

0.5 Apport du projet

Ce projet nous a permis d'utiliser tout ce qu'on nous avons appris cette année en programmation système, et de voir concretement comment cela se passe dans un shell. Nous avons approfondi nos connaissances sur la gestion des processus, notament l'utilisation du fork et la gestion des redirections des entrées et sorties. Nous avons pu aussi revoir les différents signaux ainsi que leur utilisation en fonction du besoin.