Programmation Système — Projet

1 Objectifs

Il s'agit d'écrire un *shell* permettant d'interpréter des expressions et lançant des processus pour les exécuter. La grammaire suivante décrit la forme minimale des expressions qui doivent être traitées. Il est possible d'ajouter d'autres constructions.

expression ⟶ `commande`
| *expression* `;` *expression*
| *expression* `||` *expression*
| *expression* `&&` *expression*
| `(`*expression*`)`
| *expression* `&`
| *expression* `|` *expression*
| *expression* `>` `fichier`
| *expression* `<` `fichier`
| *expression* `>>` `fichier`

La signification des différents symboles est la même qu'avec les shells habituels, en particulier une expression parenthésée indique que celle-ci doit être interprétée dans un sous-shell. Une commande simple pourra être soit interne, soit externe, et peut évidemment comporter des arguments.

1.1 Commandes internes

Les commandes internes suivantes devront en particulier être interprétées, avec leur signification habituelle : `echo`, `date`, `cd`, `pwd`, `history` [1], `hostname`, `kill`, `exit`.

On rappelle que les *commandes internes* sont implantés directement dans le shell comme des fonctions et ne font pas appel à un processus fils, par opposition aux *commandes externes*.

1.2 Remote shell

On souhaite ajouter une commande interne `remote` faciltant l'interaction distante avec un ensemble de machines, voici un exemple ou un *shell maitre* crée deux *shells distants* pour leur faire exécuter la commande `date`.

```
mini_shell(0):remote add infini1 infini2
mini_shell(0):remote all date
--- infini1 ----
mardi 20 octobre 2015, 15:45:19 (UTC+0200)
--- infini2 ----
```

1. Voir `man 3 history`.

```
mardi 20 octobre 2015, 15:45:19 (UTC+0200)
mini_shell(0):
```

Voici les commandes à implémenter :

`remote add liste-de-machines` crée un mini-shell contrôlé à distance sur chaque machine apparaissant dans la liste ;

`remote remove` termine tous les mini-shells distants ;

`remote list` liste l'ensemble des machines connectées ;

`remote nom-machine commande-simple` exécution d'une commande simple par le mini-shell (déjà créé) sur la machine donnée en paramètre puis affichage de la sortie produite par cette exécution ;

`remote all commande-simple` exécution de la commande simple par tous les mini-shell distants (déjà créés) et affichage de la sortie de chaque exécution.

Afin d'exécuter des commandes dans un shell distant, l'idée est d'exécuter ce shell à travers la commande SSH (ex. *ssh localhost bash*). Il faut ensuite rediriger l'entrée et la sortie standard du shell distant (plus précisément, de la commande *ssh*) vers votre mini-shell. Par ailleurs, on se limitera à l'exécution de commandes simples se terminant toujours, comme par exemple *date, hostname, ls -l, ...*

Le travail peut être décomposé en plusieurs étapes - ces étapes vous sont simplement suggérées.

1. La première étape se limite au traitement d'un *unique* shell *faussement distant*, c'est-à-dire sans utiliser *ssh*. De plus, pour commencer, on peut utiliser comme shell *bash* plutôt que votre mini-shell.
2. La deuxième étape ajoute la possibilité d'exécuter plusieurs shells réellement distants avec *ssh*. Toutes les réponses aux commandes distantes s'affiche dans votre mini-shell. Dans cette solution, il faut dans votre mini-shell créer n fils pour exécuter les shells distants (*ssh*), et encore n fils pour attendre et afficher les réponses de chaque shell distant directement dans votre mini-shell. Bien sûr, il faudra mettre en place tous les pipes nécessaires pour rediriger les entrées & sorties standards des shells distants. Attention, il est recommandé d'utiliser des pipes avec une *bufferisation en ligne*[2]. Néanmoins, les réponses aux commandes risquent de s'entrelacer.
3. Afin d'améliorer la présentation des résultats, la troisième étape met en œuvre un système de fenêtrage rudimentaire. Il s'agit ici d'afficher dans des fenêtres séparées les réponses aux commandes de chacun des shells distants. Pour ce faire, on vous fournit le script *xcat.sh*, qui est un programme simple qui affiche dans une fenêtre X les caractères envoyés sur son entrée standard. D'autres solutions sont envisageables.

2 Comment démarrer ?

Un embryon du projet vous est fourni. Vous pouvez vous concentrer sur les fichiers `Shell.h` `Shell.c`. En particulier le programme `Shell.c` est, en l'état, capable d'analyser les lignes de commande qui lui sont soumises. Pour réaliser cette analyse ce programme utilise une fonction d'*analyse syntaxique* qui renvoie l'arbre syntaxique associé à la ligne de commande[3]. C'est un arbre binaire qui décrit comment sont combinées les commandes contenues dans la ligne

2. Indice : Il faut utiliser *fdopen()* pour obtenir un *stream* à partir du *file descriptor* d'un pipe et *setvbuf()* pour changer le mode de bufferisation d'un *stream*.
3. NB. cette fonction retourne `NULL` en cas de ligne syntaxiquement incorrecte.

de commande donnée. Par exemple, une ligne de commande comme `ls -al | grep lol` sera décrite par un arbre dont la racine sera de type `PIPE`, cette racine ayant pour fils gauche une feuille de type `SIMPLE` contenant la commande simple `ls -al` et pour fils droit une autre feuille de type `SIMPLE` décrivant `grep lol`. On trouvera dans le fichier `Shell.c` une description de la structure de données utilisée.

Votre travail consiste à exploiter cet arbre syntaxique afin d'exécuter la ligne de commande entrée par l'utilisateur. Pour ce faire, il vous est demandé de modifier les fichiers fournis pour les adapter à vos développements.

3 Modalités

Équipe de projet. Le projet est à réaliser par équipes de trois étudiants. Les étudiants ne participant pas au contrôle continu peuvent contribuer au projet d'une équipe mais ne seront pas notés. L'équipe utilisera un dépôt svn abrité au CREMI pour le développement du projet. Un accès à ce dépôt devra être communiqué aux enseignants [4] dès sa création. N'hésitez pas à nous [5] contacter en cas de difficulté avant qu'il ne soit trop tard.

Documents à produire. Un document pdf regroupant les sources du projet ainsi qu'un rapport (5 à 15 pages) détaillant les choix effectués et les résultats [6] obtenus seront à transmettre aux enseignants le *lundi 11 janvier* minuit.

Soutenances. Les soutenances auront lieu au cours de la semaine du 11 janvier à une date qui sera précisée ultérieurement. Chaque étudiant présentera en soutenance la partie du travail qu'il a réalisé. Les enseignants pourront observer le dépôt pour mesurer la contribution de chacun. Un doodle sera organisé pour définir le planning des soutenances.

Instructions à suivre pour remettre les documents. Suite au doodle un numéro de session vous sera assigné. Mettre dans une archive tar à la fois le rapport et le code. Le nom de l'archive aura le format suivant :

```
tar czf x-login1-login2-login3-CLEF-SECRETE.tar.gz rapport.pdf code/
```

où x est le numéro de votre session de soutenance, login(1,2,3) sont les logins des membres du trinôme, CLEF-SECRETE une suite de caractères de votre choix. Utiliser ensuite la commande suivante pour vérifier qu'il y a bien tout ce qu'il faut :

```
tar tzf x-login1-login2-login3-CLEF-SECRETE.tar.gz
```

S'assurer de donner le droit en lecture pour tous sans pour autant donner le droit en écriture :

```
chmod a+r x-login1-login2-login3-CLEF-SECRETE.tar.gz
chmod go-w x-login1-login2-login3-CLEF-SECRETE.tar.gz
```

Déplacer enfin l'archive dans le répertoire /net/cremi/pwacreni/SHELL/. Les droits de ce répertoire sont `drwx----wt`, donc normalement : vous pouvez écrire dedans (ouf!), vous ne pouvez pas supprimer les TPs des autres (droit 't'), vous ne pouvez pas lire le contenu du répertoire (donc souvenez-vous bien de votre clé secrète...), `pwacreni` a tous les droits...

4. username : auesnard, aguermou, rnamyst, pwacreni
5. guermouche@labri.fr, namyst@labri.fr, esnard@labri.fr, wacrenier@labri.fr
6. En particulier on étabilra une liste précise des options qui marchent réellement - c'est à dire dans tous les cas - celles qui marchent parfois et celles qui n'ont pas été abordées.

4 Grille d'auto-évaluation

Voici une grille d'auto-évaluation à remettre le jour de la soutenance à l'enseignant. Pour chaque fonctionnalité listée dans le tableau ci-dessous, il est demandé d'évaluer un score (/10) indiquant si la fonctionnalité est correctement implantée. Ainsi, un score de 0/10 indique une fonctionnalité non traitée et un score de 10/10 désigne une fonctionnalité parfaitement implantée. Toute fonctionnalité dont il n'est pas possible de montrer le bon fonctionnement sur au moins un cas vaut également 0/10. Une évaluation de la difficulté de chaque fonctionnalité est donnée à titre indicatif.

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

Nota Bene : Cette grille servira en partie à déterminer votre note. L'enseignant s'appuiera dessus pour vous questionner lors de la soutenance. Soyez honnête, toute tricherie avérée sera lourdement pénalisée !