MLVChell

Projet Système L3

23 avril 2013

1 Conditions de rendu

Le projet est à réaliser en binôme. Il est à rendre le **28 mai 2013** (ou avant). La date de soutenance vous sera communiquée ultérieurement.

Le projet est à faire par niveau de difficulté, inutile de passer au niveau n+1 si le niveau n ne marche pas au risque de voir son travail non évalué. Ce n'est pas la peine non plus de faire la partie bonus si tous les niveaux ne fonctionnent pas correctement. Le projet devra être rendu sous la forme d'une archive compressée au format zip et devra contenir le répertoire `projet_L3_Systeme_login1_login2`. Ce répertoire devra contenir :

- `doc/doc.pdf` : un rapport décrivant et justifiants brièvement vos choix et vos solutions (5 pages maxi)
- `src/` : le répertoire contenant les sources
- `bin/` : le répertoire contenant les exécutables
- `Makefile` : le fichier qui permet de compiler le projet avec la commande `make`
- `Readme` : le fichier qui décrit comment compiler et exécuter le programme

2 Description

Un shell est un programme interactif qui interprète des commandes utilisateur sur un système d'exploitation (i.e. Linux). Les commandes sont entrées sur une interface de ligne de commande et chaque commande peut avoir des arguments et des options. Linux offre différents shells (e.g. bourne, csh, tcsh, bash, korn,... etc) qui proposent à peu près les même commandes avec certaines différences sémantiques. De plus, l'utilisateur peut aussi implémenter son propre shell pour satisfaire des besoins particuliers.

Les appels système représentent des appels à des fonctions qui sont implémentées dans le noyau (kernel) de Linux. Les développeurs doivent utiliser ces appels système pour demander des services au noyau parce qu'ils n'ont pas d'accès direct aux ressources du système pour des raisons de sécurité.

3 Niveau 1

Vous devez écrire un programme en C qui peut être utilisé comme un shell pour la gestion des fichiers et des exécutions de programmes. Vous devez utiliser les appels système vus en TP et les commandes du système pour gérer les fichiers et les processus pour implémenter votre shell.

Votre programme doit s'appeler `"mlvchell"` et doit proposer les commandes shell décrites ci-dessous. Il doit afficher un invite de commande (i.e. `mlvchell>`) au démarrage et ensuite l'utilisateur doit pouvoir entrer une commande ou une combinaison de commandes shell (avec les options et les arguments adéquats) dans l'invite de commande suivie d'une validation par la touche "Entrée". Le shell doit réafficher à nouveau l'invite de commande après la terminaison de la commande. Si la commande est invalide, votre shell doit afficher un message d'erreur.

Pour le premier niveau, les commandes suivantes doivent s'exécuter correctement sur votre programme (supposant que `"cmd"` est une commande donnée par son chemin absolue ou relatif) :

Exécution de commandes
- `cmd` : le shell doit créer un nouveau processus dans lequel la commande `"cmd"` va s'exécuter. Les entrées/sorties de la commande doivent êtres soigneusement redirigées vers ceux de votre programme.
- `cmd &` : doit exécuter la commande en tache de fond.

Redirection des entrées/sorties
- `cmd > file` : exécute la commande en redirigeant sa sortie standard vers le fichier spécifié.
- `cmd 2> file` : idem pour la sortie d'erreur.
- `cmd < file` : idem pour l'entrée standard.
- `cmd1 | cmd2` : crée deux processus tel que la sortie de `"cmd1"` est redirigée vers l'entrée de `"cmd2"`.

Commandes internes
- `copy file dest` : copie le fichier `"file"` dans le répertoire `"dest"`.
- `move file dest` : déplace le fichier `"file"` dans le répertoire `"dest"`.
- `rename file name` : renomme le fichier `"file"` en `"name"`.
- `kill pid name` : tue le ou les processus lancés par votre shell et qui portent l'identifiant `"pid"` ou le nom `"name"`.

4 Niveau 2

Une fois votre niveau 1 fonctionne correctement, vous pouvez implémenter les fonctionnalités suivantes :
- `help` : une commande interne qui affiche de l'aide à l'utilisateur.
- `cd dir` : commande interne qui change le répertoire courant vers le répertoire "dir".
- `exit` : commande interne qui termine l'exécution du shell.
- `var=value` : ajoute une variable à l'ensemble des variables internes du shell et lui donne la valeur `"value"`. Ainsi, les commandes qui suivent, `"$var"` est remplacé par `"value"`.
- `export var` : ajoute la variable interne `"var"` à la liste des variables d'environnement.

5 Niveau 3

Vous pouvez compléter votre programme avec combinaisons suivantes :
- implémentation de `Ctrl+C` pour arrêter la commande en cours
- implémentation de `Ctrl+Z` pour suspendre une commande et `"bg"` pour la reprendre en tache de fond.
- exécution d'une commande sur une machine distante qui héberge une instance de `"mlvchell"`.

6 Bonus

Pour les passionnés, vous avez carte blanche pour rajouter d'autres fonctionnalités, en voici quelques suggestions :
- la gestion de la variable `PATH` pour pouvoir exécuter une commande sans spécifier tout le chemin.
- un interpréteur de script.
- les commandes conditionnelles.
- gestion des processus lancés par votre shell (e.g. lister, pid, ppid, temps d'exécution ... etc).
- implémenter l'historique de commande et la possibilité de réexécuter une ancienne ligne de commande.

Astuce : Vous pouvez profiter de votre cours de compilation en utilisant *"flex"* pour l'analyse des commandes et des scripts.