UNIVERSITÉ DE LORRAINE
TELECOM nancy
Ingénieurs du numérique • Inspiring your digital future

Promotion 2016
2A G5 – 2A G2

Rapport de réseaux et systèmes

Création d'un Leaning SHell

par

Valentin Giannini
Eric Perlinski

15/12/2014

Table des matières

1. Déroulement du projet

Pour ce projet nous avons décidé d'exécuter les taches dans un ordre différent de celui du sujet. Nous avons commencé par faire la partie “exécuter une commande”. Pour cette partie nous avons implémenté notre propre commande “exec” nommée execSimple, qui prend en charge la création du processus fils chargé d'exécuter la commande, ainsi que la gestion des tubes permettant la communication entre les différents processus. Nous avons par conséquent mis en place un fichier nommé “test.c” dans le but de pouvoir tester les commandes mis en place, avant de les implémenter dans la partie main.

Nous avons ensuite mis en place les structures nécessaires au stockage des commandes tapées par l’utilisateur. Afin de représenter une commande, nous avons créé la pseudo-classe “exec” dont la fonction exec_init prend en argument la chaîne de caractère issue de l’entrée de commandes par l’utilisateur, découpe cette chaîne de caractère suivant les caractères (&&, ||, etc …) et qui initialise une série de commandes “cmd” en fonction de ce découpage. Cette structure nous permet de prendre en compte les redirections (<, >, >>) de ce fait notre structure était adaptée aux parties “bonus” du sujet. A partir de ce moment nous pouvions exécuter une suite de commandes chaînées avec les redirections dans des fichiers.

Nous avons ensuite mis en place la lecture du fichier “meta”, via une structure de donnée appropriée, qui stocke l’intégralité des commandes utilisables. Cette pseudo classe permet également de vérifier si les commandes tapées par l’utilisateur sont correctes en regardant s’il s’agit d’une commande autorisée ou non.

Nous avons ensuite repris le fils conducteur du sujet, implémenté cd, pwd, exit et également mis en place la boucle de jeu. Nous avons de même mis en place les handler pour les signaux afin de terminer les commandes de l’utilisateur.

La plus grosse partie du projet étant faite nous avons implémenté quelques bonus. Tout d’abord nous avons ajouté la librairie readline afin de gérer l’historique et l'auto-complétion des commandes. Puis nous avons ajouté la gestion des espaces avec les doubles « quotes ». Ensuite nous avons ajouté le globbing puis la possibilité d’utiliser des backquotes.

Le cryptage du fichier meta a également été mis en place, il s’agit d’un cryptage de césar ce qui est plutôt basic mais nous avions des taches plus importantes. Pour crypter un fichier meta, il suffit de lancer le leash avec comme argument “-C <fichier_meta>”. Cette commande produit en sortie un fichier meta.crypt qui devra être placé dans l’archive. Le fichier devra conserver l’extension .crypt car lors de la décompression, le décryptage ce fait s’il y a l’extension.

Nous avons également placé un petit easter egg pour le fun, mais il va falloir fouiller dans le code source.

2. *Petites explications techniques*

La structure “cmd” contient des informations décrivant une commande, comme son pid, son nom, ses arguments, ainsi que le résultat. Il y a également les pipes qui remplaceront l’entrée et la sortie standard afin de pouvoir lier les commandes avec des fichiers ou d’autres commandes.

Ensuite il y a l’attribut “backquoted” qui est un booléen et qui permet de savoir s’il y a une commande passé en paramètre avec des backquotes, dans ce cas elle sera exécutée puis le résultat sera placé dans le tableau des arguments.

Lors de l'exécution d’une commande, un processus fils est créé avec un fork. Ce fils va brancher les tubes sur son entrée et sortie standard puis va s'exécuter. Pendant ce temps, le père attend la fin de l’exécution de son fils et écrit si besoin le résultat (stdout du fils) sur l'écran.

Une fois le fils terminé, il vérifie le code de retour puis renvoie éventuellement une erreur si ce même code est une valeur différente de 0.

La structure “exec” contient un tableau de toutes les commandes relatives à une seule ligne entrée par l’utilisateur, ainsi que le lien existant entre toutes ces commandes. Ces liens sont définis grâce à des constantes permettant de brancher les tubes et d’afficher le résultat lors de l'exécution.

Pour ce qui est de l’extraction, un dossier .leaSh est créé dans le répertoire home de l’utilisateur et il contiendra les différents niveaux. Nous avons choisi de placer ce dossier dans le home de l’utilisateur afin d’éviter tout type de problème relatif aux droits d’utilisations. Les niveaux sont simplement des archives que l’utilisateur donne en paramètre lors du lancement du programme LeaSh. Chaque archive est extraite dans un dossier du même nom et de ce fait l’utilisateur gardera une trace de ce qu’il a fait.

Nous avons également ajouté la commande about affichant de manière dynamique le nom du programme ainsi que nos noms et prénoms. Cet affichage est fait grâce à Ncurses qui est plutôt simple d’utilisation. Cela nous a permis d’apprendre plus en détails le fonctionnement de cette librairie.

3. Problèmes et solutions

Pour le cas du test avec le fichier loop, nous avons rencontré un problème avec les tubes, en effet lorsque nous lancions la commande ./loop rien ne s'affichait car le père ne lisait pas la sortie de la dernière commande et de ce fait nous ne voyions pas ce qu'affichait le programme. Ce problème se généralise pour tous les programmes s'exécutant longtemps car il faut attendre la fin de l'exécution de ce programme pour afficher à l'écran les différentes informations.

De ce fait, quand nous lancions un signal tel que le SIGINT sur le processus fils, le processus père reprenait la main suite au waitpid, et affichait sur la sortie standard l'intégralité du niveau ./loop c'est à dire "Entering an infinite loop …, >Interrupted!".

Pour résoudre ce problème nous avons mis un place une sorte de 'Y' sur les tubes. Si le programme doit afficher sa sortie sur l'écran (il n'y a pas de redirection de fichier ou de pipes après la commande) alors on lit la sortie du programme, on l'affiche puis on réécrit ce qui a été lu dans un autre tube. Ce tube remplacera le tube de sortie de la commande (cmd->fd_out). Ce tube sera examiné à la fin de l'exécution des commandes de l'utilisateur pour vérifier si c'est le résultat attendu ou non.

L'implémentation et le traitement des commandes dont les attributs sont "backquotés" a été plutôt difficile à mettre en œuvre. En effet, il faut repérer la commande qui est à l'intérieur des "backquotes", puis l'exécuter et enfin stocker le résultat dans un tableau. Ce tableau sera par la suite inséré à l'endroit où se trouvait la commande dite "backquoté". On sépare les résultats par l'obtention du caractère de retour à la ligne, tous les résultats sont stockés dans une liste qui est ensuite transformée en tableau. Ce tableau sera placé dans le tableau des arguments de la commande principal par copie.

Pour l'ajout de chroot, nous avons fait une version normale et une version avec chroot (que l'on peut choisir lors de la compilation) mais le problème est qu'après l'application du chroot, lorsque l'on veut faire un execvp le fichier binaire n'est plus accessible. Effectivement notre chroot ce fait sur /home/User/.leaSh/level or les exécutable sont par exemple dans /usr/bin et nous ne pouvons plus y accéder. De ce fait nous avons abandonné la version chroot par manque de temps (la branche chroot est toujours visible sur le dépôt).

4. Répartition du travail et organisation du projet.

Le déroulement du projet s'est effectué principalement autour de 4 phases complémentaires : La conception, le développement, les tests et la résolution des bugs, et enfin la rédaction du rapport. La phase de conception est une phase à ne pas négliger car c'est elle qui permet d'orienter le projet et de rendre la phase de développement beaucoup plus simple. Une phase de conception plus élaborée permet de diminuer drastiquement la durée de la phase de développement. Nous avons donc passé beaucoup de temps sur cette phase.

Concernant le déroulement du projet, nous avons alterné les phases de conception, de développement et de tests / résolution de bugs, afin de nous adapter au mieux aux différentes contraintes qui peuvent survenir lors de la phase de développement. La durée totale des heures passées sur l'ensemble du projet est de 80 heures pour l'ensemble des membres du groupe (45 heures pour Valentin Giannini, et 35 heures pour Eric Perlinski), réparties équitablement et uniformément à 30% sur la conception, 40% sur le développement, 30% sur les tests et la résolution des bugs et enfin à 10% sur la rédaction du rapport.

5. remerciements

Man linux.
Globbing : http://man7.org/linux/man-pages/man3/glob.3.html
ReadLine : http://web.mit.edu/gnu/doc/html/rlman_2.html
Ncurses : http://www.tldp.org/HOWTO/NCURSES-Programming-HOWTO/windows.html

6. licence WTFPL