

Telecom Nancy - deuxième année

Module RS : Réseaux et Systèmes

Rapport du Projet de Systèmes

TESH

ASSELIN-BOULLÉ Charlotte et Valentin MOREAU

À rendre le Vendredi 22 Décembre 2017

Table des matières

1 Introduction

Dans le cadre du Module de Réseau et Système du premier semestre de deuxième année il nous a été demandé de développer en langage C un *shell* sur l'exemple du *bash* que nous utilisons quotidiennement. Ce projet sera appelé ***tesh*** dans toute la suite de ce rapport. Il devait s'agir d'un exécutable avec lequel un utilisateur pourrait communiquer dans le but de lui faire effectuer certaines commandes. Le but était donc d'approfondir les connaissances vues lors du cours de Système (et de C-SHELL de la première année) tout en les mettant en pratique à l'échelle d'un projet.
Ce bref rapport contient nos choix de conception pour ce projet, nos difficultés et leur résolution (éventuelle) que nous avons mis en place ainsi que la répartition de travail au sein de notre groupe et un bilan personnel.

2 Remerciements

Pour leur aide précieuse dans les moments de difficultés nous tenons a remercier Monsieur Charles GUILLAUME et Monsieur Jean CALVET, tous les deux élèves eux aussi en deuxième année à TELECOM Nancy, qui ont su nous aiguiller sur les erreurs que nous aurions pu commettre et nous guider ainsi dans des pistes de résolutions.

3 Notre projet

Cette section contient la présentation des différentes parties avec les difficultés rencontrées et manières de les surmonter.
Finalement notre ***tesh*** permet d'exécuter les commandes simples, les redirections et les enchaînements uniques avec l'affichage d'un prompt en mode itératif.

3.1 Général

D'une manière globale nous avons souvent rencontré des erreurs simples mais très pénalisantes de syntaxe qui nous ont clairement aidé à nous familiariser toujours plus avec le langage C. Il s'agissait parfois de ne pas oublier de mettre à jour le *Makefile* ou encore d'inclure les fichiers nécessaires (avec le chemin d'accès), d'erreurs d'étourderie qui nous valait de nous arracher les cheveux au moment de compiler ou bien encore des problèmes de pointeurs parfois mal appelés ou libérés...

3.2 Fonctionnement de base

Dans son fonctionnement basique le ***tesh*** doit attendre une commande, l'exécuter puis rester en attente jusqu'à la prochaine commande. Nous avons passé énormément de temps sur cette partie, surtout Charlotte, car bien que basique elle demandait beaucoup d'anticipation et de compréhension du problème. Par ailleurs nous avons eu a exécuter des fonctions dont le fonctionnements étaient assez obscurs. Finalement nous avons choisis d'utiliser un *fgets* qui lit la ligne saisie. Ensuite il découpe en mots la ligne grâce à la fonction *strtok* avec comme séparateur l'espace, les mots étaient une liste de mot, eux-mêmes liste de caractères (ce qui implique les *malloc* allant de pair !). Ensuite il vérifie qu'il n'y a pas une commande particulière et si ce n'est pas le cas il exécute le cas général à savoir exécuter la commande dans le fils du processus (après un *fork*), le père étant le ***tesh*** global qui ne se ferme qu'avec le signal de fin "*crtl+c*" ce qui nous a d'ailleurs amené à faire un boucle "*tant que (1)*" dès le départ.

3.3 Enchaînements conditionnels de commandes et redirections

Dans les cas particuliers à traiter nous avons distingué deux catégories : les redirections : $>$, $>>$, $<$ et | et les enchaînements conditionnels : **;** , **&&** et || Dans les deux cas le fonctionnements de reconnaissance est similaire : On copie tous les mots dans un élément qui constituera la commande 1 (*cmd1*) et si l'on croise un des éléments de redirections on copie les mots suivant dans la commande 2 (*cmd2*) et on lance la fonction de redirection (du fichier **red.c**) de même avec les éléments d'enchaînements (fonction du fichier **ench.c**). Nous avons un *fork* et suivant les cas le fils exécute une des commandes (généralement la première) et le père qui en prenant en argument la sortie de l'exécution grâce un un *pipe* dans le cas des redirections, ou en fonction du résultat de la sortie dans le cas d'un enchainement, exécute une seconde commande, le tout en respectant la syntaxe du *shell* basique pour ces commandes.
Notre ***tesh*** ne traite cependant pas la possibilité de succession de ces éléments il aurait fallut pour ce faire vérifier qu'il n'y a pas de nouveaux éléments à vérifier dans la suite des mots (c'est à dire dans le "*cmd2*").
Cette partie fut répartie entre nous deux étant donné le nombre de cas différents, ceci nous permettait de prendre en main tous les deux les éléments essentiels à la compréhension des processus, en particulier les *fork()* et la gestion des entrées et sorties.

3.4 Commandes internes, gestion des répertoires courants et prompts

La commande *cd* est particulière car, exécutée dans un processus fils, son effet est limité à ce processus, et n'a donc pas d'effet sur la localisation du processus globale. Nous l'avons donc traitée à part (dans le fichier **cd.c**) avec une fonction permettant d'influer directement sur le placement courant (*chdir*).
Par la même occasion nous avions a gérer une invitation de commande sous forme d'un prompt donnant l'utilisateur courant et la localisation du répertoire courant, ce que nous avons fait dans le fichier p**rompt.c** que nous appelons dès qu'on a fini d'exécuter une commande.
Cette partie fut essentiellement assurée par Valentin.

3.5 Mode non-itératif

Nous avons inclus un mode non-itératif à notre projet car les tests étant effectués automatiquement, ils étaient sous la forme *" echo 'echo foo' | ./tesh "* ainsi nous n'avions pas besoin de prompt puisque l'entrée standard n'était pas le terminal. Pour ce faire nous avons eu beaucoup de soucis pour au final l'inclure directement dans le ***tesh*** général en lui demander de tester avec *isatty(0)* pour savoir s'il avait à utiliser le prompt (*isatty(0)* renvoyant 0 si nous ne sommes pas dans le terminal).

4 Ce que nous n'avons pas fait

Ayant rencontré beaucoup de difficultés et travaillant en mode itératif nous n'avons pas réussit à programmer tout ce qui nous a été demandé mais nous avons prit le temps d'y réfléchir.

4.1 Gestion des erreurs

Le sujet nous proposait de rajouter l'option *-e* qui arrêtait le terminal si le code de retour n'était pas 0 (et donc qu'il y avait un code d'erreur). Etant donné que nous avon écrit un *exit(1)* à la fin de chaque exécution dans les fils puisque par définition on ne peut pas revenir d'une exécution, il nous aurait suffit de vérifier, lors du passage de cet argument ce code retour en plus du code isatty. Nous avons essayer de le faire néanmoins notre ***tesh*** construit ainsi supportait mal la gestion d'un argument de ce type.

4.2 Gestion de différents processus

La gestion des processus en arrière plan nous demandait de renvoyer, en plus de l'exécution, le *PID* du processus l'ayant exécuter et le code de retour allant avec, nous n'avons pas réussi a gérer le concept de processus d'arrière plan dans la vision du sujet et repartir en arrière est beaucoup trop coûteux au stade d'avancement où nous étions.

4.3 Readline

Nous avons longuement essayer d'utiliser la librairie *Readline* lors du passage de l'option *-r* pour remplacer *fgets*, néanmoins nous n'avons pas réussi à gérer l'historique allant avec avec et avons préféré nous concentrer sur la partie globale plutôt que refondre le code et risquer de perdre le travail déjà

effectué. Néanmoins nous avons mis un point d'honneur à comprendre la gestion de cette fonction pour l'avenir.

4.4 Idées d'amélioration à apporter

Le projet se voulait déjà très ambitieux et nous n'avons hélas pas réussi à tout effectuer mais nous avons eu le loisir de songer à des améliorations possibles :

- la gestion de l'auto-compression qui nous est précieuse dans le *shell* usuel
- gérer proprement les signaux
- lors d'une erreur, indiquer précisément d'où elle provient
- etc...

5 Gestion de projet

5.1 Répartition du travail et temps passé sur les parties

Nous n'avons pas forcément passé beaucoup de temps à programmer car le programme n'est pas long en soit, en revanche sa complexité a fait que nous avons passé énormément de temps sur la manière de fixer des problèmes et sur la réflexion d'implémentation. Nous avons souvent été au cas par cas et essayons toujours d'avoir une version la plus fonctionnelle sur le git.
De plus à chaque implémentation d'une nouvelle fonctionnalité nous ré-effectuons des tests en plus des tests blancs lancés régulièrement.

Nous avons découpé le projet en 4 grandes parties à se répartir :

- Le squelette, effectué en majorité par Charlotte : il s'agit du programme central du ***tesh*** qui lit la commande entrée, la découpe en différentes commandes et teste quel fichier elle doit appeler pour exécuter la ligne de commandes.
- Les redirections ($>$ et $>>$ développées par Valentin, $<$ et | par Charlotte), et enchaînements (développés par Valentin) : ces deux parties sont dans des fichiers à part à inclure. Le focntionnement que nous avons choisi est précisé dans la section 3.
- La gestion du répertoire implémentée par Valentin : il s'agit du développement de la commande interne correspondant au "*cd*" et de l'établissement du chemin où on se trouve, qui est d'ailleurs retourné dans le prompt.
- Nous avons passé beaucoup de temps à chercher des solutions pour les fonctions non implémentées et considérons qu'il s'agit d'un élément à part entière dans notre travail.

Enfin il est a noté que nous nous conseillions et corrigions régulièrement l'un l'autre puisque nous pouvions avoir plus de recul, notamment Charlotte avait une vision plus générale ayant fait le fonction globale, et Valentin justement une vision de ce que nécessitaient les programmes annexes à inclure.
Le tableau ci-dessous représente la responsabilité et surtout le temps de travail de notre groupe sur chaque partie.
Lorsque nous travaillions sur les éléments dans leur intégration ou sur plusieurs éléments nous avons établit qu'on considérait que les heures de travail comptabilisées étaient mises dans le fonctionnement global.

Tâches	Charlotte	Valentin
Gestion de Projet (Rapport, gestion du git, études de l'art)	5h	2h
Fonctionnement de Base et fonctionnalités globales (lecture des commandes, découpage, exécutions)	29h	7h
Redirections, enchaînements	11h	15h
Gestion des dossiers / fichiers (cd, ls, pwd, path...)	1h30	13h
Tentatives pour les autres fonctionnalités	17h	12h

Matrice représentant notre répartition du travail avec le temps passé

5.2 Bilans personnels du projet

5.2.1 Charlotte

Ce projet fut très intéressant de mon point de vue et surtout très enrichissant. N'étant au préalable pas très à l'aise en langage C il m'a permis de m'épanouir dans cette syntaxe particulière, et de l'apprécier encore davantage dans la multitude d'options qu'elle peut nous offrir. Il m'a aussi permis de mettre à profit les connaissances sur la gestion des processus que nous avons vues en cours, et les mettre en pratique à bien plus grande échelle que ce que nous permettaient les Travaux Pratiques.
En outre il fut aussi pour moi l'occasion de beaucoup de recherches et de prendre le réflexe de me documenter sur les fonctions possibles et un gros travail par itérations puisqu'il fallait toujours avoir une version plus ou moins fonctionnelle ce qui m'a permis de me rendre compte que cette gestion par jalons pouvait se révéler parfois complexe car on a du mal à avoir une vision de l'ensemble, mais très bénéfique quand ce que nous essayons d'implémenter ne marche pas : on peut toujours repartir du jalon fonctionnel précédent.
Par ailleurs au vue de la difficulté du projet au premier abord, et bien que notre implémentation ne couvre pas tout le sujet, je suis plutôt fière, tout du moins contente, de ce que j'ai réussi à faire, et en tout cas très satisfaite de ce que ce projet a pu m'apporter.

5.2.2 Valentin

Malgré le fait que nous n'ayons pas réussi à implémenter toutes les fonctionnalités que nous aurions voulu, ce projet m'a apporté des connaissances plus approfondies concernant ce qui a été apporté en cours de Système mais m'a aussi permis de revoir certaines choses étudiées lors du cours de C de première année.
Néanmoins, n'ayant pas réussi à comprendre tout ce qui a été abordé lors des Cours Magistraux, Travaux Dirigés et Travaux Pratiques de RS, j'ai trouvé ce projet un peu compliqué pour moi.
Je ressors, malgré cela, content et enrichi de ce projet car il m'a permis d'appliquer plus concrètement les choses que dans le cadre des cours.

6 Conclusion

Ce projet ambitieux non par sa longueur mais par la complexité des éléments et structures que nous avions a manipuler nous a été bénéfique et bien que un peu déçus de n'avoir pas réussi à implémenter toutes les fonctions qu'il nous était proposé de faire dans le sujet, nous sommes parvenus à une version fonctionnelle basique de ce qui nous était demandé ce qui nous donne satisfaction au vu du temps que nous avons chacun passé à essayer de surpasser nos difficultés.

7 Annexe : Structure de notre programme

Voici une vue schématique présentant la structure de notre makefile avec les différents fichiers et leur liens pour la compilation.