

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. 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 années)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 (éventuelles) 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, tout 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 partie est 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 rencontrer 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 appelé 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 ligne 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 exécute la commande dans le fils du processus (après un fork), le père étant le tesh global qui ne se ferme que 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 traité nous avons commencé deux catégories : les redirections : ¿ ¿¿ ¡ et — et les enchaînements conditionnels : ; —— . Dans les deux cas le fonctionnements de reconnaissance est similaire : On copie tous les mots dans un 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 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, suivant les cas nous avons le fils qui exécute une des commandes (généralement la première) et le père qui en prenant en argument (grâce à un pipe) la sortie de l'exécution, ou en fonction du résultat de la sortie, 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 (ie dans le "cmd2").
Cette partie fut répartie entre nous deux étant donné le nombre de cas différents, ceci nous permettait de prendre en main tout 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 serait limité à ce processus, et n'a donc pas d'effet sur la localisation du processus globale. Nous l'avons donc traité à part (dans le fichier cd.c) avec une fonction permettant d'influer directement sur le placement courant. Par la même occassion 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 pompt.c que nous appellons dès qu'on a fini d'exécuter une commande.
Cette partie fut essentiellement assurée par Valentin.

3.5 Mode non itératif

Nous avons inclue un mode non-itératif à notre projet car les tsts étant efeectués 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 pour savoir s'il avait à utiliser le prompt.

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 execution dans les fils puisque par définition on ne peut pas revenir d'une execution, il nous aurait suffit de vérifier, lors du passage de cet argument ce code retour en plus du code issaty. Nous avons essayer de le faire néanmoins notre tesh construit ainsi supportait mal la gestion d'un argument de ce type.

4.2 Readline

Nous avons 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 concentré sur la partie globale plutôt que refondre le code et risquer de perdre le travail déjà effectué.

4.3 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
—
— etc...

5 Gestion de projet

5.1 Répartition du travail et temps passé sur les parties

Le tableau ci-dessous représente la responsabilité et surtout le temps de travail de notre groupe sur chaque partie (considéré comme des jâlons.

Tâches	Charlotte	Valentin
Gestion de Projet (Rapport, gestion du git, études de l'art)	3h	.h
Fonctionnement de Base et fonctionnalités globales (lecture des commandes, découpage, exécutions)	21h	.h
Re-directions, processus en arrière plan	12h	.h
Gestion des dossiers / fichiers (cd, ls, pwd, path...)	3h30	.h

Matrice représentant notre répartition du travail avec le temps passé

5.2 Bilan personnel du projet

6 Conclusion

7 Annexes

7.1 Structure de notre programme

Voici la Carte Conceptuelle présentant la structure de notre makefile avec les différents fichiers et leur liens pour la compilation.