Rapport Projet Testaro

Jean-Baptiste RENAULT, Arthur SCHMIDT

20 Décembre 2020

Table des matières

1 Introduction

Le but de ce projet est d'implémenter un programme "testaro" servant à automatiser des tests sur d'autres programmes.

Il prend en entrée un fichier de description des tests à effectuer et des résultats attendus par ces tests.

Ce projet a pour but de tester les connaissances que nous avons acquises grâce aux cours de RS cette année.

2 Choix de conception

Nous avons suivi les indications présentes dans le sujet afin d'établir une stratégie pour implémenter ce programme.

Nous avons donc commencé par lire un fichier mis en argument dans la ligne de commande, puis nous avons implémenté les fonctions relatives aux lignes spéciales (ligne blanche, cas "cd", commentaires,...)

Nous avons ensuite implémenté les fonctions reconnaissant le type de ligne en entrée (input,output et commande).

Après celà, nous avons implémenté la fonction reconnaissant les backslashs et concaténant les lignes concernées.

Nous avons ensuite créé la fonction permettant d'exécuter la commande en passant au processus fils le stdin.

Nous avons alors créé une fonction permettant de vérifier la valeur de retour de la commande.

Puis nous avons implémenté les fonctions vérifiant si un code d'erreur a été renvoyé ou si un signal a terminé le programme.

Nous avons ensuite implémenté la fonction gérant le cas particulier de la commande "cd".

Puis nous avons géré le timer pour éviter le gel du programme si une commande infinie est lancée.

Depuis le début de ce projet, nous avons implémenté deux modes : le mode "DEBUG" et le mode "normal". En effet, le mode DEBUG permet d'imprimer explicitement pour chaque commande : l'input,l'output attendu et la commande.

Ce mode nous a permis de trouver rapidement la source de certaines erreurs.

Pour passer en mode DEBUG, il suffit de mettre la variable DEBUG dans testaro.h à 1, la mettre à 0 enclenche le mode normal.

3 Extensions

Ayant terminé la partie principale, nous avons décidé d'implémenter certaines des extentions proposées dans le sujet :

- Les lignes 'p' affichent leur argument sans l'exécuter
- Un compteur de lignes a été mis en place pour savoir où se trouve l'erreur (indiquée seulement en mode "DEBUG")
- Les modes "post" et "pre" ont été implémentés
- La commande "! set timeout val" change la valeur du timer (désactivé si val == 0)
- Un "\\ " est considéré comme un simple "\ "

4 Les Tests

Afin de vérifier que notre implémentation est correcte, nous avons mis en place un système de tests lancés via le makefile afin de pouvoir tester rapidement et facilement les nouvelles fonctionalités/extentions implémentées et de vérifier qu'aucune fonctionalité précédente n'a été impactée par les nouvelles.

- Le test de base : un "make testaro" compile le fichier .c et lance le test de base donnée dans le sujet
- Le test de boucle : un "make boucle_infinie" permet de tester le timer mais aussi la réception de certains signaux (quand on kill la boucle).
- Le test poussé : un "make test_marche" permet de tester des fonctionalités avancées de manière plus approfondie que le test de base (le cd,le 'p',...).
- Le test post/pre : un "make test_post" permet de tester l'extension post/pre.
- Le test d'erreur : un "make test_erreur" permet de tester la valeur retournée quand une ligne d'entrée est invalide
- Le test d'erreur de cd : un "make test_erreur_cd" permet de tester le code de retour quand une commande "cd" échoue.

5 Problèmes rencontrés et solutions

Nous avons rencontré au cours de ce projet deux problèmes importants :

Tout d'abord, les tests de base et base++ ne passaient pas alors que notre programme retournait bien le code d'erreur 0 et écrivait bien TOTO TUTU(retour_ligne) dans le fichier "fich". Après discussion avec d'autres élèves, il nous est apparu que ce problème était surement du à la présence de caractères non-imprimables dans le fichier fich : ces caractères ne faisaient pas échouer les tests de notre côté mais pourraient très bien faire échouer d'éventuels tests faits sur le fichier "fich" à la fin du programme "base". Pour palier ce problème, nous avons diminué de 1 la taille du "write", ce qui a permis de ne plus afficher des caractères non-imprimables contenus en mémoire après le (backslash0).

De plus, nous avons passé du temps à essayer de comprendre pourquoi le test "signaux" ne passait pas et changé plusieurs fois la gestion des signaux avant de recevoir un mail du professeur nous disant que celà était dû à une erreur de paramétrage de son environnement (le test "signaux" a été validé quand le professeur a relancé ses batteries de tests).

6 Tableau de répartition du travail

Travail	Arthur SCHMIDT	Jean-Baptiste RENAULT
Reflexions préliminaires	2h	2h
Reconaissance argument	1h	0h
Lecture du fichier	0h	1h
Reconaissance type de ligne	3h	3h
Gestion des backslashs	3h	0h
Implémentation de l'exec	3h	3h
Vérification du retour	0h	3h
Gestion signaux	0h	4h
Gestion erreurs	4h	0h
Implémentation extentions	6h	6h
Gestion erreurs test "signaux"	4h	4h
Gestion erreur tests "base" et "base++"	4h	4h
Total :	30h	30h

7 Conclusion

En conclusion, nous pouvons dire que ce projet nous a permis de renforcer nos connaissances en programmation système et plus particulièrement en gestion des processus, gestion des signaux et gestion des erreurs.

De plus, il nous a permis de comprendre l'intérêt d'un travail de groupe mais également d'acquérir des compétences de communication de groupe.

Enfin, cela nous a permis d'avoir une première approche d'une évaluation par tests automatiques, ce type d'évaluation nous oblige à nous poser des questions sur l'environnement sur lequel les tests sont lancés et nous incite à faire preuve de rigueur pour que les tests précédemment validés ne soient pas invalidés par les nouvelles fonctionnalités implémentées .