RAPPORT

DE

PROJET

Prepared By :

Najib Aghenda
Syméon Carle

Course :
Réseaux et Systèmes

Instructor :
Lucas Nussbaum

Date :
22nd December, 2017

Table des matières

Introduction

Dans ce rapport, nous présentons l'interpréteur de commandes, appelé **tesh**, que nous avons réalisé. Nous expliquons dans ce document les choix que nous avons effectué afin de satisfaire au maximum les exigences formulées par le responsable de ce projet. Nous remercions nos encadrants de travaux dirigés Lucas Nussbaum et Rémi Badonnel pour le savoir qu'ils nous ont transmis et qui a permis la réalisation de cet interpréteur de commandes.

1 Choix de conception

Une première phase de documentation sur les interpréteurs existants nous a permis de constater que le fonctionnement de base repose sur trois parties : la **lecture** de l'élément fourni à l'entrée standard, son **analyse** et son **exécution**. Ces trois parties constituent l'architecture de notre programme dans lequel nous avons associé à chacune de ces étapes une fonction.

Nous allons maintenant détailler davantage les trois parties évoquées ci-dessus dans le cadre d'un fonctionnement de base, c'est-à-dire, en **mode intéractif**. L'étape de **lecture** consiste à récupérer et stocker l'information soumise à l'entrée standard afin qu'elle puisse être analysée par la suite. L'étape d'**analyse** va permettre de distinguer dans l'information ce qui constitue un séparateur, une commande, un argument, etc.. C'est l'équivalent en compilation de l'analyseur syntaxique sur lequel nous reviendrons plus tard. La dernière étape, l'**exécution**, va, comme son nom l'indique, exécuter les commandes retournées par l'analyseur.

On retrouve ci-dessous un schéma récapitulatif des différentes étapes qui viennent ponctuer le trajet de l'information dans l'interpréteur de commandes tesh :

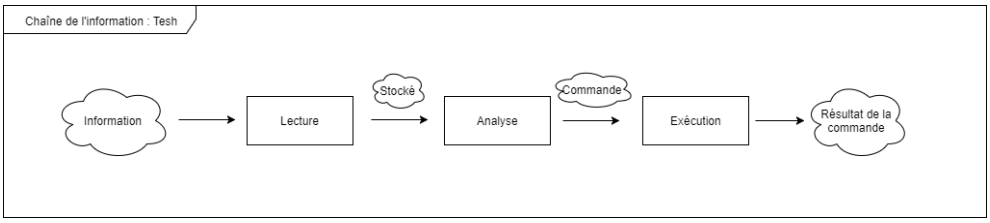

FIGURE 1 – Chaîne de l'information : Tesh

Après nous être renseignés sur les différentes exigences formulées dans le sujet, nous avons procédé à leur hiérarchisation et nous sommes convenus que le fonctionnement de base constituerait la plus importante partie du projet en terme de durée et c'est donc ce à quoi nous avons accordé le plus de temps.

2 Problèmes rencontrés

Au cours de ce projet, nous avons rencontré plusieurs difficultés que nous allons exprimer dans cette partie.

Les principaux problèmes que nous avons rencontrés ont eu lieu lors de l'implémentation de la fonctionnalité de base, notamment lors de l'étape d'exécution où nous avions du mal à définir les actions à effectuer en fonctions des opérateurs.

Une première version de notre fonctionnement de base passait avec succès certains de nos tests mais les tests blancs nous ont permis de remarquer une erreur que nous n'avions pas testé : notre programme ne gérait pas une commande composée, c'est-à-dire, un ensemble de commandes juxtaposées par des opérateurs. Cette erreur nous a permis de mettre en évidence une faiblesse de notre programme, celui-ci ne définissait pas de règles de priorités sur les opérateurs utilisés.

Une autre difficulté a été l'analyse de la grammaire utilisée par notre interpréteur de commande ainsi que la prise en main du Flex/Bison pour lequel il a fallu consacrer beaucoup de temps afin de se documenter.

Enfin, des problèmes mineurs sont venus ponctuer notre projet : allocation de la mémoire en C, les tableaux, les pointeurs.

3 Solutions proposées

Afin de corriger notre problème de priorités sur les opérateurs, nous avons étudié le **Shell Command Language** et sa grammaire comme détaillé sur le site suggéré par le responsable du projet. L'étude de cette grammaire nous a permis de comprendre comment définir les priorités sur les opérateurs, de produire un code beaucoup plus épuré et structuré. Il a été nécessaire de revoir le code dans sa quasi-totalité.

Une autre solution que nous avons mise en place consiste en la décomposition de l'exécution en sous-fonctions traitant chacune un niveau de priorité. Ainsi, les règles de priorité que l'on a pu définir à l'aide du **Shell Command Language** nous donnent la hierarchie suivante :

```
— ; &
   — && ||
      — |
         — < > >>
```

L'utilisation de l'analyseur lexical Flex et de l'analyseur syntaxique Bison nous a permis de simplifier la création d'une structure qui représente la commande tout en respectant la hierarchie des opérateurs.

4 Gestion de projet

Au début du projet nous travaillons ensemble sur le fonctionnement de base jusqu'à avoir des résultats positifs. Une fois cette étape importante du projet presque fonctionnelle, nous avons décidé de séparer les tâches afin d'essayer d'utiliser Flex/Bison. L'un s'occupait principalement de la mise en place du Flex/Bison tandis que l'autre implémentait les fonctionnalités mineures telles que la sortie sur erreur, le mode non interactif, etc...

On retrouve ci-dessous le tableau récapitulatif de la répartition du travail :

Etapes	Najib Aghenda	Syméon Carle
Fonctionnement de base	14h	10h
Commande interne : cd	2h	x
Affichage d'une invite de commande	1h	x
Enchaînement conditionnel de commandes	3h	4h
Redirections d'entrées et sorties	2h	2h
Mode interactif et non interactif	2h	x
Sortie sur erreur	2h	x
Lancement de commandes en arrière plan	1h	3h
Edition de la ligne de commande du shell avec readline	x	3h
Flex & Bison	x	15h
Ecriture du rapport	2h	x
Total	29h	36h

5 Bilan

Nous allons faire ici le bilan de notre projet.

5.1 Notions vues au cours du projet

Au cours de ce projet, nous avons revu de nombreuses notions développées dans les cours de Réseaux et Systèmes de deuxième année et de C de première année dont nous avions besoin pour réaliser nos objectifs :

- Processus
- Fichiers et entrées/sorties
- Exécution des programmes
- Mémoire en C

5.2 Comparaison avec les objectifs établis

Nous avons atteint l'objectif principal qui était de concevoir un interpréteur de commande fonctionnel qui puisse intégrer la plupart des fonctionnalités demandées par le responsable du projet.

5.3 Le rendu final

Nous avons implémanté les fonctionnalités suivantes :

- Fonctionnalités de tesh
  - Sortie sur erreur
  - Edition de ligne
  - Mode non interactif
  - Affichage de prompt
  - Redirections
  - Enchaînements opérationnels
  - Commande interne : cd
  - Bibliothèque readline liée dynamiquement
  - Flex/Bison

6 Conclusion

En conclusion, ce projet nous a permis de comprendre l'importance des tests car sans ces derniers, nous aurions peut-être vu nos erreurs trop tard sans avoir le temps de les modifier. Réaliser un interpréteur de commandes n'a pas été une tâche évidente mais nous sommes globalement satisfaits du travail que nous avons réalisé car nous avons validé un bon nombre des critères demandés et nous avons essayé d'aller un peu plus loin en proposant l'utilisation de Flex/Bison afin d'être plus tolérants sur la syntaxe. Bien qu'imparfait, notre interpréteur de commande est fonctionnel et nous avons beaucoup appris en le réalisant.

7 Annexe

Sources

Source pour les fonctionnalités demandées :

- Cours de Réseaux et Systèmes (2A Telecom Nancy) de Lucas Nussbaum
- Cours de C (1A ESIAL) de Marion Chardon
- Documentation des commandes linux (Man)
- https ://brennan.io/2015/01/16/write-a-shell-in-c/
- http ://pubs.opengroup.org/onlinepubs/009695399/utilities/xcu_chap02.html
- http ://man7.org/linux/man-pages/man2/

Pour Flex/Bison :

- https ://gnuu.org/2009/09/18/writing-your-own-toy-compiler/
- https ://www.lemoda.net/c/reentrant-parser/
- http ://www.enib.fr/ harrouet/Data/Courses/Flex_Bison.pdf/
- http ://westes.github.io/flex/manual/