Mini-projet de programmation Système/Systèmes d'exploitation

Un interpréteur de commandes UNIX simplifié

José Ferro Pinto

28 janvier 2018

1 Introduction

Le but de ce document est de décrire l'état final de l'implémentation.

2 Le pré-bash

Avant de lancer l'interpréteur de bash, une condition minimale est qu'un fichier *profile* existe et qu'il contienne la définition de minimum deux variables d'environnement que sont *$HOME* et *$PATH*. Cette partie est implémentée entre les lignes 183 et 187.

Une deuxième vérification doit être faite si le shell est ouvert avec un paramètre. Ce paramètre doit être un fichier qui contient les commandes à lire par l'interpréteur. C'est la même chose que si on appelait le shell de la manière suivante :

```
./shell < mes_commandes.sh
```

Cette partie est implémentée entre les lignes 137 et 145. On commence par fermer le descripteur de fichiers 0 qui correspond à l'entrée standard (clavier) puis on essaie d'ouvrir le fichier. Le fichier prendra la place de l'entrée standard dans les descripteurs de fichiers.

3 Implémentation du bash

La première partie du travail consiste en une implémentation d'un interpréteur de commandes UNIX avec comme commandes de base :

— cd : pour *Change Directory*, qui change le répertoire de travail
— pwd : pour *Print working directory*, qui imprime le répertoire de travail
— alias : pour la création d'alias

Le gros du travail commence à la ligne 190 avec la lecture de l'entrée standard. Le traitement de la commande demandée par l'utilisateur se fait via la fonction *tokenize_ input* à la ligne 211. Elle sera expliquée avec plus de précision plus tard.

Entre les lignes 246 et 250, c'est la gestion de création de variables d'environnement, si le caractère = est trouvé dans la ligne entrée par l'utilisateur qui utilise la fonction du système UNIX : *setenv*.

Les lignes 251 et 262 servent à imprimer le contenu d'une variable d'environnement si la ligne de commande entrée par l'utilisateur commence par un $ et n'a pas d'autres arguments qui utilisent la fonction du système UNIX : *getenv*.

Les lignes 264 à 284 servent à lancer des commandes créées par moi-même. Dans cette partie, il faut faire une gestion de remplacement des alias, qui se fait à la ligne 266 à l'aide de la fonction *replace_ environment_ vars*. Elle sera aussi expliquée plus loin.

La gestion des commandes internes existantes se fait dans un pseudo mapping dans la structure *shell_ map* qui contient :

— name : correspond à la commande que l'utilisateur doit rentrer
— fct : un pointeur sur la fonction qui sera exécutée
— help : une description de la fonction
— argc : le nombre d'arguments max possible
— args : une description des arguments attendus si nécessaire

Aux lignes 264 à 272, la recherche d'une commande interne est faite et si elle est trouvée, alors on rentre dans le bloc 274 à 284.

Aux lignes 276 à 284, la vérification que le nombre d'arguments est correcte est faite et on appelle ensuite la fonction qui exécutera la commande.

Les lignes à partir de 285 jusqu'à 352 servent à la gestion des commandes non interne qui demandent directement au système pour l'exécution des commandes. Cette gestion est décrite dans la section suivante.

4 Gestion des redirections

Dans ce shell, il est possible de faire des redirections de flux standard. Cette gestion de redirection de flux est faite, pour l'ouverture, entre les lignes 213 et 244, et pour la fermeture du flux, entre les lignes 355 et 364.

4.1 Variables de contrôle de redirection

Il y a 6 variables qui servent à la gestion des différentes redirections :

— std{in,out}_redirected : Deux booléens qui disent si, respectivement, l'entrée standard et la sortie standard ont été redirigées.
— std{in,out}_copy : Une copie du descripteur de fichier qu'occupait, respectivement, l'entrée standard et la sortie standard, seulement si le descripteur en question a été redirigé.
— std{in,out}_tmp : Le registre qui contiendra le, respectivement, l'entrée standard temporaire et la sortie standard temporaire.

4.2 Redirection de la sortie

A l'encontre du caractère `>`, il faut gérer la redirection de la sortie standard vers un fichier donné en paramètre juste après.

Cette gestion est faire entre les lignes 219 et 230.

En fermant le descripteur de fichier 1 à l'aide de la ligne `close(1)`, il faut ensuite le remplacer par la nouvelle sortie en créant le fichier que contiendra la nouvelle sortie.

Cette création du fichier se fait à la ligne 226.

Les flags choisis sont `O_WRONLY | O_CREAT | O_EXCL` qui servent à limiter à la création d'un nouveau fichier et de ne pas écraser de fichier existant avec la combinaison `O_CREATE` et `O_EXCL`. Si la création du fichier à redirigé échoue, j'ai décidé de remettre la sortie standard pour laisser le programme tourner tout de même.

4.3 Redirection de l'entrée

A l'encontre du caractère `<`, il faut gérer la redirection de l'entrée standard à partir d'un fichier donné en paramètre juste après.

Cette gestion est faire entre les lignes 231 et 243.

En fermant le descripteur de fichier 0 à l'aide de la ligne `close(0)`, il faut ensuite le remplacer par la nouvelle entrée en ouvrant le fichier qui doit être lu.

Cette ouverture du fichier se fait à la ligne 238.

Le flag choisi est `O_RDONLY` qui sert à limiter l'ouverture du fichier en lecture.

4.4 La remise en place de l'entrée et la sortie standard

Après avoir fait les redirections pour la commande demandée, il faut remettre en place l'entrée et la sortie standard si elles ont été utilisées.

Cette remise en place se fait entre les lignes 355 et 364 qui vérifient que l'entrée ou la sortie aient été changés, et si c'est le cas, la redirection temporaire est fermée et l'ancienne entrée ou sortie standard est remise en place

5 Commandes internes

Les différentes commandes seront décrites dans cette section.

5.1 help

La commande *help* n'était pas demandée explicitement dans le travail, mais une aide pour connaître les commandes implémentées est toujours la bienvenue ; et puisque que la structure du mapping des fonctions s'y portait bien, il a été simple de faire cet affichage. Cela consiste en une simple boucle qui traverse le mapping pour un affichage plutôt correct.

5.2 exit/quit

Ces deux commandes appellent la même fonction *do_ exit* qui ferme l'interpréteur de commandes.

5.3 pwd

La commande *Print working directory* est un wrapper qui appelle simplement la fonction du système UNIX : *getcwd*. L'apport du wrapper est la gestion d'un buffer qui peut contenir un path aussi grand que nécessaire (pour autant que la mémoire le permette) avec un agrandissement du buffer qui contiendra le path par puissance de 2 avec comme valeur initiale 32 caractères.

5.4 cd

La commande *change directory* est un wrapper qui appelle la fonction du système UNIX : *chdir*. Un traitement supplémentaire doit être fait lorsque la commande *cd* est appelée sans argument, ce qui a pour fonction de déplacer le répertoire de travail à la position de la variable d'environnement *$HOME*. J'ai décidé de rendre cette fonction la plus bavarde possible lorsqu'elle rencontre une erreur pour être le plus clair possible et car les erreurs retournées sont possiblement utiles pour l'utilisateur qui appelle *cd*.

5.5 alias

La commande *alias* est une implémentation complètement maison. J'ai décidé de créer et d'utiliser un arbre binaire pour la gestion des alias. La description de l'implémentation de l'arbre binaire sera faite plus tard. La commande alias a trois comportements distincts.

5.5.1 sans argument

Quand *alias* est appelée sans argument, la commande imprime tous les alias existants dans le système. Géré aux lignes 390 à 392.

5.5.2 un argument contenant le caractère =

Dans ce cas là, la commande va créer dans les systèmes un alias avec le contenu se trouvant à gauche de l'égalité pointant vers le contenu se trouvant à droite. Géré aux lignes 405 à 407.

5.5.3 un argument sans =

Ce dernier cas ne fait qu'imprimer, s'il existe, l'alias demandé avec ce vers quoi il pointe. Géré aux lignes 399 à 403.

6 Arbre binaire

Comme décrit plus tôt dans ce document, la gestion des alias se fait à l'aide d'un arbre binaire. Je me suis aidé d'un livre théorique d'algorithmie [1]. Toutes les fonctions liées à l'arbre binaire se trouvent dans le fichier *binary_tree.c*. L'arbre binaire a pour but d'accélérer la recherche dans un pool de données, s'il est totalement équilibré (pas implémenté pour l'instant) en $O(\log_2(n))$ au lieu de $O(n)$ dans un tableau non ordonné.

La spécificité de cet arbre binaire est qu'il devait travailler avec des paires de valeurs pour la gestion des alias avec la structure créée pour l'occasion, nommée *alias_t*. La clé de comparaison est le champ *var* de ladite structure.

7 Les fonctions supplémentaires

Certaines fonctions supplémentaires pour des traitements spécifiques ont dû être ajoutées pour la lisibilité du code.

7.1 tokenize_input

Cette fonction sert à parser la ligne qu'un utilisateur a entrée et la retourne sous forme d'un tableau qui contient tous les éléments. La partition de la ligne se fait entre chaque espace.

7.2 print_introduction

Cette fonction sert à imprimer la même introduction à chaque début de commande de l'utilisateur. Elle donne la date et l'heure actuelles.

7.3 extract_environment_var

Cette fonction sert à départager une chaîne de caractères contenant le caractère = et retourne les éléments à gauche et à droite dans une structure *alias_t*.

7.4 change_alias

Cette fonction est appelée avant la recherche d'une commande car elle remplace les alias dans la ligne de commande de l'utilisateur par leur contenu. Cette fonction tient compte des récursions et s'arrête dès qu'elle en aperçoit une à l'aide d'un arbre binaire interne à la fonction.

7.5 replace_environment_vars

Cette fonction sert à remplacer toutes les variables d'environnement (commençant par un $) existantes dans la commande entrée par l'utilisateur.

8 Conclusion

Ce travail est fait seul et non en binôme. Une amélioration à prévoir est de gérer les alias qui ont un contenu avec des espaces. Ceci devrait être fait dans la fonction *tokenize_ input* lors de la séparation de la ligne avec des espaces, il faut vérifier si elle ne contient pas le caractère " et, le cas échéant, utiliser le contenu entre " comme un seul et unique argument.

Références

[1] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Third Edition.* The MIT Press, 2009.