COMPÈRE Nicolas
LE ROUX Yannick
481C

Licence 2 Maths-Info
X22I030

Système d'exploitation
Rapport de projet

UNIVERSITÉ DE NANTES

Sommaire

Rappel du sujet

Le but du projet est de réaliser une version succincte d'un shell utilisant uniquement les commandes de Bash appelé mush. De la même façon qu'un shell classique, le programme doit demander en boucle des commandes à exécuter, rentrées par l'utilisateur. Il doit être capable d'exécuter ces commandes en séquentielle si elles sont séparées par un point virgule (`;`) ou en parallèle si elles sont séparées par un double slash (`//`). Le programme peut être appelé avec un nom de fichier en paramètre, ce qui aura pour effet de lire le fichier et d'exécuter les commandes ce trouvant dans celui-ci, puis quitter le programme mush et rendre la main. Le shell doit aussi être capable de reconnaitre les expressions de globbing. Pour quitter le programme, il faut rentrer la commande "`quit`", qui retourne au terminal en rendant la main à l'utilisateur.

Structures de données

Le programme est découpé en plusieurs fichiers :

- `execution.cpp` et `execution.hpp`

Contient la procédure d'exécution. Gère le `fork()` et le globbing.

- `lancement.cpp` et `lancement.hpp`

Contient la procédure `lancement()` qui découpe la saisie de l'utilisateur en commandes composées en utilisant un `vector<vector<vector<char *>>>`. Cette structure à été implémentée car elle était plus intuitive pour le reste du programme.

– `parsing.cpp` et `parsing.hpp`

Contient toutes les fonctions relatives au parsing.

Algorithmes utilisés

– `void execution(vector<vector<char*>> commandes)`

La procédure exécute les commandes simples contenues dans le `vector commandes`. La procédure reconnait si les commandes sont à exécuter en séquentielle ou en parallèle. Autant de fils sont créer qu'il y a de commandes simples dans chaque commande composée. Le globbing s'effectue sur tout les paramètres de toutes les commandes. Le processus père attend que tout les fils aient terminés.

– `void lancement(string saisie, bool &quitter)`

La procédure reçoit la saisie de l'utilisateur ou les commandes contenues dans le fichier passé en paramètre qui sont au préalable stockées dans la variable saisie de type chaîne de caractères et découpe ces commandes à l'aide des fonctions du parser. Ensuite, le programme passe chaque commande composée à la fonction `execution()` sauf si la commande "`quit`" est présente en séquentielle.

– `glob(char *, GLOB_NOCHECK, nullptr, &g)`

Fonction de globbing.

– `execvp(char[0]*, char[]*)`

Fonction permettant d'exécuter les commandes internes de bash dans le langage C++.

– `fork()`

Fonction permettant de créer un processus fils.

Difficultés du projet

Difficultés anticipées

- La gestion de commandes en parallèle

La principale difficulté à été la gestion de la fin de tout les fils créés avant que le père ne reprenne la main.
Pour résoudre ce problème, il a fallut faire une boucle pour exécutant `wait(nullptr)` autant de fois qu'il y avais de fils.

- Le globbing

Les fonctions de globing ont été difficiles à implémenter car elles nécessitaient de modifier la liste des commandes passées en paramètre dans la fonction d'exécution.

- Apprentissage de la classe vector

Nos connaissances pour la classe vector était assez limitées puisque c'est l'une des première fois que nous y avions affaire. Cependant, cette classe s'est avérée plutôt simple à appréhender et très efficace à l'utilisation.

Difficultés rencontrées

- Compréhension du parser

La fonction de parser étant fournie, il a fallu d'abord commencé par comprendre comment elle fonctionnait avant de pouvoir l'utiliser correctement dans le programme.

- Gestion du `quit`

La commande quit n'étant pas une commande de bash, il fallait l'implémenter en plus dans le programme. Des choix ont du être effectués concernant la gestion de la commande par rapport aux autres commandes saisies en séquentielles et en parallèles.

- Utilisation de la fonction `execvp()`

Cette fonction n'accepte que des tableaux de tableaux de caractères ce qui rendait son utilisation compliquée car il fallait parfois au préalable convertir des chaines de caractères vers des tableaux de caractères et vice-versa.
La gestion d'une commande non valide à été gérée en ré-exécutant la commande `execvp()` en lui passant en paramètres un echo du message d'erreur.

Restrictions du programme

Le programme possède quelques restrictions :

- La commande “`quit`” ne fonctionne qu’en séquentielle.

- Si plusieurs arguments sont passés en paramètres, seul le premier sera lu, les autres seront ignorés.

- La commande interne “`cd`” ne peut pas être exécutée par le programme pour se déplacer dans l’arborescence.

Manuel du programme

La compilation du programme se fait grâce au `Makefile` inclus dans le dossier en tapant la commande “`make`”.

Le programme se lance avec la saisie de la commande “`./main`”.

Pour lancer le programme mush avec un fichier, il faut le passer en paramètre en tapant “`./main nom_du_fichier`”. Celui-ci doit se trouver dans le même répertoire que le programme. Autrement, il faut passer le chemin d’accès relatif ou absolu vers le fichier. Un seul fichier peut être ouvert par le programme. S’il y a plus d’un argument passé en paramètre, seul le premier sera lu et les autres seront ignorés.

Les différents messages d’erreurs apparaissent si :

- Le fichier est vide,
- Aucun nom de fichier passé en paramètre ne correspond à l’argument.

Une fois le programme lancé, il est possible d’écrire les commandes classiques de bash. Pour exécuter les commandes en séquentielle, il faut les séparer par un point virgule (`;`). Pour les exécuter en parallèle, il faut les séparer par un double slash (`//`).

Le résultat d’exécution de chaque commande composée est séparé par un trait.

Pour quitter le programme, il faut utiliser la commande “`quit`”. Attention, celle-ci ne peut être utilisée qu’en séquentielle. Si “`quit`” est écrit en parallèle avec d’autres commandes, celles-ci seront exécutées mais un message d’erreur apparait pour indiquer que le programme n’a pas été arrêté.

Conclusion

Ce projet nous aura permis de consolider les connaissances que l'on pouvait avoir dans ce module et de bien comprendre comment fonctionnaient les fonctions `execvp()`, `fork()` et `glob()`. Il nous aura également permis d'améliorer notre niveau de programmation pour le langage C++. Le découpage d'un programme en plusieurs fichiers est relativement nouveau pour nous avec l'utilisation d'un `.hpp` pour chaque fonctions/ procédures. De plus, nous n'avions jamais utilisé de `Makefile` auparavant, ce qui s'est avéré très utile dans la compilation de plusieurs fichiers entre eux nécessaire pour ce projet.

Le travail a été très formateur pour nous et les connaissances acquises nous serviront très certainement par la suite.