Projet de programmation système

Mini-Shell

DOMMARTIN Jules

GUEZE Lucas

Sommaire

Table des matières

**Choix des fonctionnalités1**

Tapez le titre du chapitre (niveau 2)1

**Organisation du répertoire et des fonctions2**

Tapez le titre du chapitre (niveau 2)2

**Problèmes rencontrés3**

Tapez le titre du chapitre (niveau 2)3

**Le rendu4**

Tapez le titre du chapitre (niveau 2)4

1. Choix des fonctionnalités

Les commandes internes

Le choix des fonctionnalités de notre Mini-Shell s’est fait en plusieurs étapes. En effet, nous voulions tout d’abord réaliser un Mini-Shell avec des fonctionnalités  « basiques », comme l’exécution de commandes dites « internes » (echo, cd), mais qui fonctionne correctement avec un traitement de toutes les erreurs.

Les commandes externes

Après avoir réalisé cette première étape, nous nous sommes décidés à travailler sur les commandes dites « externes », dans la même dynamique de réalisation, avec le traitement des erreurs. Par exemple, entrer une ligne de commande vide ne fait rien (ce n’est pas le cas si l’on ne traite pas cette entrée, bash renvoie une erreur). L’exécution de commandes externes se fait en créant un processus fils qui va exécuter la commande. Voir partie II.

Les tubes

Nous avons voulu pouvoir combiner plusieurs commandes grâce au caractère « | », que l’on nommera « pipe ». La réalisation de tubes s’est faite en plusieurs étapes. Nous avons dû modifier les fonctions déjà existantes et créer deux nouveaux types structurés (que l’on a appelé job\_t et ensemble\_job\_t) pour la réalisation de cette fonctionnalité. Elle est fonctionnelle et permet d’enchaîner jusqu’à trois commandes. Nous avons décidé arbitrairement de limiter le nombre de commandes en pipe à trois, mais nous pouvons le changer très facilement puisqu’il s’agit en réalité d’une variable interne au programme.

La mise en arrière-plan

Une commande peut-être lancée en arrière-plan grâce au caractère « & » placée à la fin de la commande. Ainsi, le Mini-Shell n’attend pas la mort du processus fils créé pour continuer son exécution.

Les signaux

En dernière fonctionnalité, nous avons ajouté le traitement des signaux. Ainsi, Ctrl+D arrête le Mini-Shell, Ctrl+C ne fait rien dans le cas où aucun processus en premier plan n’est lancé (alors qu’il devrait arrêter le Mini-Shell s’il n’était pas traité), sinon il arrête le processus en premier-plan en cours. NB : Deux processus lancés en pipe s’arrêtent tous les deux avec un Ctrl+C.

1. Organisation du répertoire et des fonctions

Le répertoire contient plusieurs fichiers « .c » et « .h ». L’ensemble des fonctions définies dans ces fichiers suivent une logique précise de programmation.

Le fichier mon\_shell.c

La fonction main (le point d’entrée du programme) se trouve dans le fichier  « mon\_shell.c ». Elle contient plusieurs variables et fonctions d’initialisation du programme :

* Une variable « sigact » qui est une structure de contrôle représentant les différents signaux.
* Une variable « ligne » qui est une structure de contrôle représentant une ligne de commande.
* Une fonction d’initialisation des jobs qui initialise une structure de contrôle pour un pointeur ensemble\_job\_t passé en paramètre.
* Une fonction d’initialisation des signaux, qui suspend le signal « SIGINT » (Ctrl+C) dans le programme.

Cette fonction lance ensuite une boucle infinie sur trois fonctions :

1. affiche\_invite() qui affiche une ligne contenant le répertoire courant suivi du caractère « > ». NB : Cette fonction utilise la fonction get\_current\_dir\_name() qui a besoin de « #define \_GNU\_SOURCE » pour fonctionner.
2. lit\_ligne() qui reçoit la ligne de commande entrée et la traite pour la stocker dans la variable ligne. Elle est définie dans le fichier « ligne.h ».
3. execute\_ligne() qui exécute la ou les commandes contenues dans la variable ligne.

Le programme ne s’arrête pas tant qu’il n’a pas reçu son signal d’arrêt.

1. Problèmes rencontrés
2. Le rendu