Projet C : First Shell (FiSH)
Système et programmation système

Julien BERNARD
Université de Franche-Comté

Licence 2 Informatique
2015 – 2016

Le but de ce projet est de coder un interpréteur de commande (*shell*) et comme ce sera votre premier interpréteur de commande, il s'appellera *First Shell*, ou en abrégé `fish`.

Spécification

Le shell que vous allez réaliser est une simple boucle interactive qui demande une commande, éventuellement composée de tubes et/ou de redirections, puis l'exécute et recommence. La réalisation de ce shell sera progressive, en augmentant progressivement la difficulté des types de commandes à traiter.

Pour vous aider, vous avez à votre disposition une archive contenant une bibliothèque de lecture et d'analyse d'une ligne (dans les fichiers `cmdline.h` et `cmdline.c`), ainsi qu'un programme de démonstration qui vous servira de code de départ. Cette bibliothèque définit une structure `struct line`, qui représente une commande et qui contient en particulier un tableau de `struct cmd` pour représenter les commandes simples. Pour manipuler la structure `struct line`, vous avez trois fonctions :

— `line_init` : pour initialiser la structure ;
— `line_parse` : pour analyser une chaîne de caractères issue d'une saisie ;
— `line_reset` : pour remettre à zéro la structure après l'avoir utilisée.

Vous ne devriez pas avoir à modifier cette bibliothèque. Toutefois, si vous constatez un bug ou un manque, n'hésitez pas à en faire part à vos encadrants pour qu'une version corrigée puisse être mise à disposition.

Les cas que vous devrez prendre en charge pour les commandes sont :

— les commandes simples en avant-plan, avec ou sans redirections ;
— les commandes simples en arrière-plan, sans redirection ;
— les commandes avec tubes en avant-plan, avec ou sans redirection.

Consignes

Vous devrez suivre les consignes suivantes :

— Vous rédigerez votre code et les commentaires **en anglais**.
— Vous séparerez votre code en plusieurs fichiers si besoin.

— Vous lirez les pages de manuel de manière approfondie pour bien comprendre le comportement de chacune des fonctions utilisées.
— Vous vérifierez toutes les valeurs de retour des fonctions appelées.

Réalisation

Exercice 1 : Mise en route

Question 1.1 Compiler et exécuter les fichiers fournis. Vérifier que tout fonctionne correctement. Arrêter le programme à l'aide de `CTRL+C`.

Question 1.2 Lire le fichier `fish.c` pour comprendre comment fonctionne ce programme.

Exercice 2 : Commande simple

On suppose qu'il n'y a qu'une seule commande, c'est-à-dire aucun tube et aucune redirection. On suppose aussi qu'on attend la fin de la commande en cours avant de rendre la main au shell.

Question 2.1 Traiter le cas sans argument. On prendra garde à bien vérifier que l'exécution de la commande réussit ou pas, et d'envoyer un message d'erreur si la commande n'existe pas.

Question 2.2 Ajouter un message pour savoir si le programme s'est terminé normalement (en indiquant le code de retour, le cas échéant), ou s'il s'est terminé avec un signal (en indiquant le signal qui a terminé le processus). Indice : argument `status` de `wait(2)`.

Question 2.3 Traiter le cas avec arguments. On supposera que le tableau fourni par la bibliothèque termine bien par un `NULL`.

Exercice 3 : Commandes internes

Nous allons maintenant implémenter quelques commandes internes classiques. Pour cela, avant de lancer la commande, on vérifiera s'il s'agit d'une commande interne ou pas à l'aide de `strcmp(3)`.

Question 3.1 Implémenter la commande `exit` qui permet de quitter le shell. Cette commande est nécessaire pour pouvoir bien gérer les signaux dans la suite, et notamment `CTRL+C` (qui pour l'instant permet de sortir du programme).

Question 3.2 Implémenter la commande `cd` qui permet de changer de répertoire courant. Indice : `chdir(2)`. Changer le prompt du shell pour y inclure le répertoire courant. Indice : `getcwd(3)`.

Exercice 4 : Redirections

Pour bien gérer les redirections, il sera nécessaire d'ouvrir les fichiers concernés en lecture seule ou en écriture seule suivant le type de redirection.

Question 4.1 Gérer les redirections possibles de la commande, en entrée et en sortie.

Exercice 5 : Arrêt avec `CTRL+C`

Le shell va maintenant gérer l'arrêt du processus en cours à l'aide de la combinaison de touche `CTRL+C`, qui envoie le signal `SIGINT`. Actuellement, quand ce signal est envoyé, il déclenche le gestionnaire par défaut qui consiste à arrêter le processus. Il sera donc nécessaire de modifier ce gestionnaire de manière à arrêter le processus lancé par le shell et pas le shell lui-même.

Question 5.1 Gérer l'arrêt avec `CTRL+C` à l'aide de `sigaction(2)` (et pas `signal(2)`) et de `kill(2)`.

Exercice 6 : Commande en arrière-plan

Le shell va maintenant gérer les commandes en arrière-plan. Dans ce cas, le shell n'attend pas la fin de la commande mais continue. Cependant, il faut également gérer la fin des processus lancés. Lorsque qu'un processus fils finit, il envoie le signal `SIGCHLD` qu'il faudra donc gérer correctement de manière à terminer le processus correctement. Indice : `waitpid(2)`.

Question 6.1 Gérer les commandes en arrière-plan. On prendra garde à bien gérer les intéractions entre les commandes en arrière-plan et la commande en avant-plan, notamment quand on attend la terminaison de la commande en avant-plan.

Exercice 7 : Commandes et tubes

Le shell va maintenant gérer des tubes entre les commandes. Il faut veiller à créer le nombre de tubes suffisant, et à bien faire les redirections aux bons moments. Il faut aussi penser à fermer tous les descripteurs de fichiers inutilisés.

Question 7.1 Gérer les commandes avec un seul tube.

Question 7.2 Gérer les commandes avec un nombre arbitraire de tubes.

Exercice 8 : Bonus : Gestion des motifs

Cette partie est facultative, vous devez d'abord avoir fini tout le reste avant de vous attaquer à cette partie. Il s'agit ici pour le shell de gérer les motifs comme `*` ou `?` dans les noms de fichiers. Pour cela, il existe une fonction qui fait le plus gros du travail : `glob(3)`.

Question 8.1 Gérer les motifs à l'aide de `glob(3)`.

Travail à rendre

Le travail à rendre sera précisé pendant le projet.