Rapport technique - Projet Mini-Shell

Architecture

Le projet est séparé en deux parties : interpréteur et commandes (respectivement les répertoires `interpreter/` et `cmd/`).

interpreter/

interpreter.c

Contient la partie principale du projet, le main, accompagné du prompt ainsi que le parseur.

Lorsque l'utilisateur rentre des commandes, on récupère le contenu de l'entrée standard en séparant les "mots" (suite de caractères sans espace) et on les met dans un tableau.

Ce tableau est ensuite traité : on va récupérer les commandes et les chaîner si il y a présence d'un pipe, d'une redirection ou encore de caractères de chaînage conditionnel (`||` et `&&`).

Appel de la fonction `call()` pour ensuite exécuter chacunes des commandes.

call.c

Ce fichier contient la fonction `call()` faisant office d'interface entre l'interpréteur et l'appel des commandes. On y trouve ainsi le code pour l'exécution des commandes sous les 3 formes requises : exécutable, fonction intégrée et librairie.

redirection.c

Fichier comportant l'ensemble du code associé aux redirections de flux (pipe).

Le pipe `|`, permet de faire communiquer deux programmes et de les exécuter en parallèle. La sortie du premier programme pourra être récupérée par le second. Pour mettre en place le pipe, il faut utiliser l'appel système `pipe()`. Cela nous permettra de manipuler l'entrée standard et la sortie standard des processus. On fork le processus, pour exécuter les deux commandes (de chaque côtés du caractère de redirection) pour les exécuter en parallèle. Il ne reste plus qu'à fermer une entrée dans l'un des processus et fermer la sortie dans l'autre pour avoir un pipe fonctionnel. Si on veut que le processus fils envoie des données vers l'entrée standard du père, il faut rediriger la sortie du fils vers l'entrée standard du père.

Les autres redirections `<`, `<<`, `>` et `>>` sont des dérivées du pipe classique `|`. Pour les redirections de sorties `>` et `>>`, on exécute la commande dans le processus fils, puis le processus père lit l'entrée standard et écrit le résultat dans un fichier. Pour les redirections de sortie `<` et `<<`, c'est l'inverse, on lit le fichier ou les entrées clavier dans le processus fils, puis le processus père exécute la commande qui va traiter le contenu.

Nous avons essayé de factoriser au maximum cette partie, puisque nous nous sommes très vite rendus compte que c'est presque la même chose pour chaque type de redirection. Ainsi, il n'y a que 5 fonctions (que l'on appelle interfaces), avec un appel quasiment similaire à `myPipe()` :

- `redirectionCommandeVersCommande()` pour `|`

- `redirectionCommandeVersFichierEnAjout()` pour `>>`
- `redirectionCommandeVersFichierEnEcrasant()` pour `>`
- `redirectionFichierVersCommande()` pour `<`
- `redirectionClavierVersCommande()` pour `<<`

Les autres fonctions ne sont utilisées et utilisables que depuis le fichier `redirection.c`.

cmd/

cmd/src/

Fichiers sources `.c` de nos commandes.

cmd/include/

Fichiers header `.h` contenant les prototypes des fonctions de nos commandes.

cmd/utils/macro_main.h

On y trouve la définition de la macro qui permet de créer un main dans chaque fichier source des commandes lors de la création des exécutables indépendants. Ajouter `-D CREATE_MAIN` lors de la compilation.

cmd/bin/

Emplacement des fichiers exécutables des commandes, générés avec `make`.

cmd/lib/

Emplacement des librairies dynamiques (fichiers `.so`) générées avec `make`.

cmd/cmd.h et cmd/cmd.c

Sorte d'interface contenant la liste des fonctions de commandes en dur. Ceci est utilisé pour la forme "fonctions intégrées". Ainsi, la fonction `call()` peut simplement récupérer une fonction en indiquant simplement son nom.

Commandes disponibles

cat

Utilisation : `cat [fichier]`

Affiche le contenu d'un fichier s'il est renseigné en paramètre, sinon la commande cat lit et affiche en continue le contenu de l'entrée standard.

cd

Utilisation : `cd [nomDestination]`

Changer le répertoire de travail courant vers le répertoire donné en paramètre. Par défaut, le répertoire destination est le HOME de l'utilisateur.

> Cette commande fonctionne (on arrive bien à changer de répertoire), mais elle est inutilisable au sein du shell à cause des forks, puisque le processus fils ne peut changer le répertoire courant du père.

echo

Utilisation : `echo [message ...]`

Affiche la ou les chaînes de caractères passées en paramètre.

ls

Utilisation : `ls [cible]`

Affiche l'ensemble des fichiers d'un répertoire cible de façon détaillée : type de fichier, droits, propriétaire, groupe, taille, date de modification, nom. Par défaut, la cible est le répertoire courant. Si l'utilisateur indique le nom/chemin vers un fichier, ls affiche simplement le nom du fichier.

cp

Utilisation : `cp nomSource nomDestination`

Permet de faire une copie d'un fichier vers la destination souhaitée. Si un fichier existant porte le même nom que le fichier cible, il sera écrasé.

mv

Utilisation : `mv nomSource nomDestination`

Permet de déplacer un fichier. On peut aussi se servir de cette commande pour renommer un fichier par exemple.

pwd

Utilisation : `pwd`

Affiche le chemin d'accès absolu du répertoire de travail courant.

rm

Utilisation : `rm nom`

Supprime le fichier ou le répertoire passé en paramètre.

makedir

Utilisation : `makedir nom`

Crée un répertoire à l'emplacement choisi.

Utilisation de l'interpréteur

Génération des exécutables

Pour utiliser notre application, il faut d'abord générer les exécutables grâce à la commande `make`.

`make` génère tous les exécutables et toutes les librairies, cependant, il est possible de générer seulement les parties souhaitées :

- `make as_executable` génère tous les exécutables des commandes, ainsi que l'exécutable de l'interpreteur correspondant
- `make as_integrated_function` génère l'interpreteur de commande avec les fonctions de commandes intégrées
- `make as_library` génère les librairies (fichiers ".so"), ainsi que l'exécutable de l'interpreteur correspondant

Exécution

Pour executer le Mini-Shell, choisissez l'une des 3 commandes suivantes :

> Tip : le répertoire courant est la racine du projet

- `./asExecutable`
- `./asIntegratedFunction`
- `./asLibrary`

Pour quitter le Mini-Shell, il suffit de rentrer la commande `exit`.

`make clean` afin d'effacer tous les fichiers générés (exécutables + librairies dynamiques).