1 Projet : Création d'un shell en C

1.1 Compilation et exécution du shell

Le projet utilise un `Makefile` pour compiler correctement. De ce fait, pour compiler le projet, il suffit d'exécuter la commande `make` dans le fichier courant du projet.
Une fois la compilation du projet faite, un exécutable `shell` est dans le projet, il suffit de l'exécuter avec la commande `./shell` pour que shell se lance.

1.2 Réponses aux questions

1. Pour exécuter une commande, le shell utilise la fonction `execvp` car cette fonction prend 2 arguments :
- L'exécutable (ou la commande) qu'on souhaite exécuter à la place de notre programme
- Un tableau de chaîne de caractères représentant les arguments de l'appel

Ces arguments étant sous cette forme dans la structure `struct cmd`, cette fonction est la plus appropriée

2. Le symbole pour une séquence de commandes est `";"`
La seconde commande est exécutée quoiqu'il arrive, ce qui n'est pas le cas de l'opérateur `&&` :
- `mkdir . ; echo "test"` : cette commande va échouer pour créer un fichier . (car il existe déjà par défaut), mais ensuite va afficher le mot "test"
- `mkdir . && echo "test"` : cette commande va échouer pour créer un fichier ., et la seconde commande ne sera donc pas exécutée

4. Oui la commande s'exécute dans bash.
Par exemple avec la commande
- `(echo "un message" && echo "un second message") | wc` va compter le nombre de mots et de caractères sur l'entrée donnée par les deux echo. Ainsi les parenthèses sont nécessaires
- `echo "un message" && echo "un second message" | wc` va afficher "un message" puis compter le nombre de mots et caractères du second echo (ce qui est différent de la commande sans parenthèses)

5. Sans rien faire, CTRL-C va envoyer un signal SIGTERM au shell qui par défaut ferme le programme. De ce fait, CTRL-C ferme le shell. Pour empêcher cela, le shell va modifier le handler appelé par défaut lors de la réception du SIGTERM avec la fonction `signal` (on va en plus résoudre les bugs graphiques généré par cet appel dans le nouvel handler).

6. Sans rien faire, la commande `ls > dump` exécute et affiche dans la sortie standard `ls` (ce qui n'est pas ce qui est attendu)

7. Si on utilise `dup2` à la place de `pipe`, on va devoir rediriger la sortie dans l'entrée, et dans ce cas on ne créer pas de file-descriptor (on va juste rediriger le file-descriptor). Ceci peut amener à des résultats différents. La commande `ls | wc` avec `dup2` on va afficher dans la sortie standard le résultat de `ls`, puis le terminal laissera l'entrée à l'utilisateur pour écrire ce qui sera ensuite utilisé par `wc` (alors qu'on veut seulement lire la sortie de `ls`). De ce fait, la création d'un file-descriptor est nécessaire.

1.3 Exemples

- `cat file` : l'exécution de cette commande va créer un processus fils qui va exécuter `cat` avec les bons arguments. Une fois l'exécution terminée, le processus va mourir et le processus père (qui est le processus principal du shell) va continuer et attendre une prochaine fonction
- `echo "1" ; echo "2"` (ou avec `&& ||`) : l'exécution de cette commande va exécuter la commande `echo "1"`. Puis une fois fini on exécute la commande `echo "2"`. La différence avec `&&` ou `||`, le shell va regarder la valeur de retour de la première commande pour savoir si elle à réussi ou non et donc exécuter la seconde commande si la valeur de retour est correcte.
- `ls > dump` : l'exécution de cette commande va créer un processus fils qui va d'abord mettre à jour ses file-descriptor : il va créer un file-descriptor qui correspondra à l'écriture dans le fichier `dump`, il va ensuite remplacer le file-descriptor de la sortie standard par ce file-descriptor à l'aide de `dup2`. Ce changement ne posera pas de problème par la suite car la modification n'est présente que dans le processus fils (et donc lors de la fin de ce processus la modification n'existera plus). Finalement, le programme exécute la commande ls dans le processus fils. (le shell fonctionne pareil avec les opérateurs `>>` et `<`).

— `(echo "1" && ls) | wc` : l'exécution de cette commande va d'abord mettre à jour les file-descriptor pour le pipe. Ensuite, le shell exécutera les commandes dans la parenthèse. Une fois ces commandes exécutée, le shell exécutera la commande `wc` avec les bons file-descriptor pour lire la sortie des premières commandes. Ainsi la prioriété des parenthèse est respectée.