TP 4 : implémentation d'un mini-shell

Partie 1 : fonctionnalités basiques

Écrivez un programme **`shell`** implémentant les fonction basiques d'un interpréteur de commande (« shell »). Ce programme boucle sur les trois actions suivantes :

1. affichage d'un message d'invite,
2. saisie d'une commande,
3. exécution de la commande saisie,

jusqu'à ce que l'utilisateur saisisse le caractère fin de fichier (CTRL+D).

Remarques :

- Votre programme devra créer un nouveau processus pour chaque commande à exécuter.
- Votre programme ne devra pas créer de zombie.
- Si la commande saisie n'existe pas, votre programme devra afficher un message d'erreur approprié.
- Si la commande est vide (l'utilisateur a simplement tapé « entrée »), alors votre programme devra simplement ré-afficher l'invite de saisie, sans afficher de message d'erreur.
- Pour saisir une commande, et la découper en « mots », vous pouvez utiliser la fonction **`lis_ligne()`** du fichier **ligne_commande.c**
- Pour exécuter un autre programme, vous utiliserez la fonction **`execvp`** qui s'utilise directement avec ce que retourne la fonction **`lis_ligne()`**
- Pour comparer deux chaînes de caractères, vous pouvez utiliser la fonction **`strcmp`** de la librairie **`string.h`**

Variables d'environnement :

Faîtes en sorte que votre shell gère la création et la modification de variables d'environnement. Cela consiste à détecter et gérer les instructions du type

```
export TOTO=777
```

où « export » est un mot réservé que vous détecterez, TOTO est le nom de la variable d'environnement (à créer ou à modifier), et 777 est sa valeur. Afin de séparer "TOTO" de "777", vous pourrez utiliser la fonction separe_egal de ligne_commande.c

Bonus

Faîtes en sorte que votre shell gère l'exécution de commandes en arrière plan, grâce au caractère "&" placé en fin de commande.

Par exemple :

```
gedit toto.txt &
```

lance le programme gedit (éditeur de texte) avec son interface graphique en arrière plan : le shell vous redonne immédiatement la main pour saisir une autre commande. Sans le caractère "&", il faudrait terminer le processus lancé (c'est à dire fermer gedit) afin d'en saisir une autre.

Partie 2

Gestion des redirections

Vous ferez en sorte de gérer la redirection de sortie **>**. Par exemple, la commande

```
ls -l > toto.txt
```

exécute la commande **ls -l**, mais, au lieu d'écrire le résultat sur la sortie standard, elle l'écrit dans le fichier "toto.txt"

On rappelle que le principe d'une redirection, telle que la redirection **>**, est de changer le descripteur de fichier 1 (qui pointe d'habitude vers la sortie standard) pour qu'il corresponde au descripteur de fichier d'un fichier donné.

Pour cela, utilisez la fonction **dup2(fd1, fd2)** qui fait les choses suivantes :

- ferme le descripteur de fichier **fd2** s'il était ouvert
- ré-ouvre le descripteur de fichier **fd2** et le fait pointer vers la même chose que **fd1**

Exemple : si **fd1** pointe vers **fichier1.txt** en écriture, et **fd2** pointe vers **fichier2.txt** en écriture, alors, après un appel à **dup2(fd1, fd2)**, une écriture dans **fd2** écrira désormais vers **fichier1.txt**

Amélioration

Faîtes en sorte que votre shell gère les redirections **>>**, **2>, 2>>**, ainsi que **<**