C TD/TP 3: Pointeurs

Exercice : Arborescence de fichiers

Le but de cet exercice est d'implémenter un système de gestion de fichiers basique. Dans notre implémentation, un fichier possède seulement un nom et un contenu. On représente un fichier par la structure suivante :

```
typedef struct {
   char *nom;
   char *contenu;
} fichier;
```

Les fichiers peuvent être regroupés en répertoires. Les répertoires peuvent contenir des fichiers, ainsi que d'autres répertoires. Cette organisation s'appelle "arborescence de fichiers" et peut donc facilement être représentée par un arbre. Le répertoire racine de l'arborescence est appelé "/". Un répertoire est représenté par la structure suivante :

```
struct repertoire_s {
   char *nom;
   int nb_repertoires;
   struct repertoire_s *sous_repertoires;
   int nb_fichiers;
   fichier *fichiers;
}
typedef struct repertoire_s repertoire;
```

Le répertoire racine sera défini par la variable globale `repertoire *racine`.

Le système de gestion de fichiers supporte les opérations suivantes :

1. Créer répertoire
2. Afficher arborescence
3. Ajouter fichier
4. Lire fichier
5. Supprimer fichier
6. Supprimer répertoire (supprime également tous les fichiers et sous-répertoires)

Le programme doit accepter des commandes saisies par l'utilisateur [1]. Chaque commande représente une opération sur le système de fichiers, et respecte la syntaxe suivante :
`nom_operation argument1 argument2 ....`
Comme dans l'arborescence linux, la localisation des fichiers et répertoires se fera à l'aide de la notation "/[repertoire]*/fichier" Par exemple, si l'utilisateur saisit la commande :
`ajouter_fichier /mon_repertoire/mon_sous_repertoire/fichier_toto toto`

1. La fonction `main` du programme est donc une boucle infinie qui contient un appel à la fonction `scanf`.

Le système de gestion de fichiers créera le fichier “fichier_toto” avec le contenu “toto” dans le sous répertoire “mon_sous_repertoire”.

Question 1

Coder la fonction `char **analyser_commande(char *commande, int *taille)` qui prend en paramètre la chaine de caractères saisie par l'utilisateur, et retourne un tableau de chaines de caractères dont le premier élément est le nom de la commande, et les autres éléments sont les arguments de la commande.

Question 2

Coder la fonction `char **analyser_chemin(char *chemin, int *taille_tableau)` qui prend en paramètre le chemin du fichier (ou du répertoire) et qui retourne un tableau de chaines de caractères, dont chaque élément est un nom de répertoire situé sur le chemin.

Par exemple,

`analyser_chemin(/mon_repertoire/mon_sous_repertoire/fichier_toto, &taille)`

renverra le tableau {`mon_repertoire, mon_sous_repertoire, fichier_toto`} et la variable taille vaudra 3.

Question 3

Coder la fonction `void creer_repertoire(char **chemin)` qui crée le repértoire qui se situe à la position indiquée par le tableau de caractères `chemin` passé en paramètre. Si le tableau `chemin` contient un seul élément, le répertoire est ajouté à la racine de l'arbre.

Question 4

Coder la fonction `void afficher_arborescence()` qui affiche toute l'arborescence de fichiers à partir de la racine (similaire à la commande linux ls /).

Question 5

Coder la fonction `void ajouter_fichier(char **chemin, char *contenu)` qui crée un fichier dans l'arborescence.

Question 6

Coder la fonction `void lire_fichier(char **chemin)` qui retourne le contenu du fichier situé à la position `chemin`, et null si le fichier n'existe pas.

Question 7

Coder la fonction `void supprimer_fichier(char **chemin)` qui supprime de l'arborescence des fichiers le fichier donné en argument.

Question 8

Coder la fonction `void supprimer_repertoire(char **chemin)` qui supprime de l'arborescence des fichiers le répertoire donné en argument, ainsi que tous les fichiers et eventuels sous répertoires qu'il contient.