Fichiers et redirections

Thèmes

- Manipulation des fichiers
- Duplication de descripteurs et redirection

Ressources Pour ce TP, comme pour les autres, vous pourrez vous appuyer sur

- Le polycopié intitulé « Systèmes d'exploitation : Unix », qui fournit une référence généralement suffisante sur la sémantique et la syntaxe d'appel des différentes primitives de l'API Unix.
- Les pages du manuel en ligne (commande `man`), et plus particulièrement les sections 2 et 3.

Rendu à la fin de la séance

Comme pour les TPs précédents, vous fournirez sous Moodle le travail que vous avez effectué pendant la séance. Pour ce TP, il y aura 2 rendus :

- le programme `copier` réalisé à la section 2 ;
- l'archive du `minishell` qui contiendra votre progression en TP sur le projet Minishell.

Pour ce dernier, l'archive s'obtient en utilisant la commande `make archive`. Le fichier nommé `minishell-votreidentifiant.tar` ainsi que le programme `copier.c` sera alors à charger sous Moodle dans la section rendu, dans la zone qui correspond à votre groupe de TD.

1 Minishell : contrôle du fils par les commandes au clavier

Terminez les étapes ⑩ à ⑫ du TP3, si ce n'est pas déjà fait.

2 Manipulation des fichiers

Réalisation d'un programme `copier` Réaliser un programme `copier` qui devra

- définir un tableau de `BUFSIZE` caractères qui servira de tampon mémoire ;
- ouvrir les fichiers source et destination ;
- lire les données du fichier source fragment par fragment, chaque fragment lu étant stocké dans le tampon mémoire, avant d'être (immédiatement) écrit dans le fichier destination ;
- fermer les fichiers source et destination, une fois le dernier fragment transféré.

Le programme se lancera donc de la façon suivante :

```
copier source destination
```

Rendu 1 Chargez le fichier `copier.c` réalisé dans la section rendu, dans la zone qui correspond à votre groupe de TD.

3 Minishell : Redirection des entrées/sorties d'un processus

Lorsque l'utilisateur écrit `cmd > f`, l'exécution de `struct cmdline *commandes= readcmd();` effectue l'opération `commandes->out= f;`. Sinon, `commandes->out == NULL`. De la même manière, `cmd < f` implique que `commandes->in= f;`.

Etape 13 (Redirections) Complétez votre programme pour permettre d'associer l'entrée standard ou la sortie standard d'une commande à un fichier.
Exemple (classique) : `cat < f1 > f2` associe `f1` à l'entrée standard de `cat`, et `f2` à la sortie standard de `cat`.

4 Minishell : Manipulation des répertoires

La commande `cd` est une commande interne au shell. L'appel système `chdir` permet de se déplacer dans l'arborescence des répertoires. Cette primitive a l'interface suivante :

```
int chdir(const char *chemin);
```

Le répertoire de travail courant est alors remplacé par celui indiqué par `chemin`. La primitive renvoie `-1` en cas d'erreur.

Etape 14 (Déplacement dans l'arborescence) Ajoutez une fonction qui permet de changer le répertoire courant vers un répertoire indiqué en argument, en utilisant la primitive `chdir`. Lorsque l'argument de la fonction vaut `NULL`, le répertoire courant devient le répertoire racine de l'utilisateur, identifié par la variable d'environnement `HOME`. Modifiez la boucle principale du `minishell` de manière à lancer cette fonction lorsque l'utilisateur tape `cd` suivi d'un nom de répertoire.

Remarque :

- La primitive `char *getenv(const char *env_variable);` permet de récupérer la valeur de la variable d'environnement passée en argument. Le résultat est une chaîne de caractères. Par exemple, `char *shell= getenv("SHELL");` retourne la chaîne `shell == "/bin/shell"`.
- La primitive `char *getcwd(char buf[], size_t size);` retourne le répertoire courant et affecte ce répertoire à `buf` si le paramètre est présent et que la taille de `buf`, définie par `size`, est suffisante. Un exemple simple d'utilisation est :
  ```
  char *cwd= getcwd(NULL, 0);
  ```

Nous nous proposons ici de mettre en œuvre une commande du `minishell` qui liste le contenu d'un répertoire passé en argument : `dir [nom_repertoire]`. Pour afficher le contenu du répertoire, les primitives suivantes seront utilisées :

- `DIR *opendir(const char *repertoire);` ouvre le contenu du répertoire passé en argument et retourne un pointeur sur la première entrée du répertoire.
- `struct dirent *readdir(DIR *dir);`, à chaque appel, retourne un pointeur sur une structure `struct dirent` correspondant à la prochaine entrée du répertoire pointée par `dir` ou retourne `NULL` lorsque la dernière entrée du répertoire a été lue.

- `int closedir(DIR *dir);` retourne 0 en cas de succès ou -1 en cas d'erreur.

La structure `struct dirent` est défini de la façon suivante :

```
struct dirent {
    ino_t d_ino;     // Inode number
    char  d_name[]; // Null terminated filename
}
```

Etape 15 (Afficher le contenu d'un répertoire) Ajoutez une fonction qui affiche le contenu d'un répertoire passé en paramètre. Si la commande `dir` est exécutée sans argument, elle affiche le contenu du répertoire courant.