Principe des systèmes d'exploitation

TP 4

Avant propos. Ce TP sera noté ! Il faudra pour cela que vous rendiez le travail que vous avez fait sur elearning sous forme de fichier zippé. Votre fichier devra être sous le format `nom_prenom.zip`. Le rendu est possible jusqu'à 24 heures après la fin de la séance.

Exercice 1 : rappels sur la programmation C.

On rappelle que la déclaration d'une fonction *main* en programmation C se fait de la façon suivante : `main(int argc, char* argv[])`. Le paramètre `argc` désigne le nombre d'arguments passés en ligne de commande à l'exécution du programme et le paramètre `argv` est un tableau dans lequel sont stockés ces arguments.

1. En supposant qu'on ait un exécute un fichier `a.out` par la commande

```
> ./a.out arg1 arg2
```

   quelles sont les valeurs des paramètres `argc` et `argv` ?

2. Écrire un programme qui affiche sur la sortie standard les arguments passés en ligne de commande à la compilation. Vous effectuerez un retour à la ligne avant l'affichage d'un nouvel argument. En reprennant l'exemple précédent, l'affichage sur la sortie standard doit donner :

```
a.out
arg1
arg2
```

Exercice 2 : error, perror, read, write, ...

1. On rappelle qu'on accède à une page du *man* par la commande

```
> man nom-de-la-page
```

   De plus, si on souhaite accéder à la page d'une fonction lors d'un appel système, on utilise l'option `s2` :

```
> man -s2 nom-de-la-fonction
```

   Lire les pages du manuel consacrées à `error`, à `perror` ainsi qu'à la fonction `write` lors d'un appel système.

2. En utilisant la fonction `write`, écrire un programme qui affiche `Hello world!` sur la sortie standard. Pour cela, vous utiliserez comme buffer un tableau de 512 `char`. De plus, on rappelle que le *file descriptor* de la sortie standard est 1.

3. Refaire la question précédente en traitant les erreurs pouvant survenir lors de l'appel à `write` : en cas d'erreur, on souhaite afficher un message sur l'erreur standard en faisant appel à `perror` et on souhaite que le `main` retourne un code d'erreur (par exemple, 1). Testez votre programme en mettant la valeur du *file descriptor* à 3.

4. Lire les pages du *man* consacrées aux fonctions `open`, `read` et `close` lors d'un appel système. Écrire un programme auquel on donne en argument des noms de fichiers et qui affiche leurs contenus respectifs sur la sortie standard. Votre programme doit évidemment prendre en compte le traitement des erreurs éventuelles.

Exercice 3 : préliminaire avant archivage.

Consulter les pages du *man* consacrées aux fonctions `opendir`, `fopen`, `readdir`, `fprintf`, `fclose` et `closedir`. Écrire un programme listant l'ensemble des fichiers présents dans le répertoire courant et écrivant la taille des noms et les noms de ces fichiers. On rappelle que la taille d'une chaîne de caractère peut être récupérée par la fonction `strlen`[1]. Les tailles de noms et noms de fichiers devront être écrits dans un nouveau fichier dont le nom aura été passé en argument. Le stockage se fera sous la forme

`tailleNom1nom1tailleNom2nom2...tailleDernierNomdernierNom`

Un exemple de rendu de cette fonction sur un dossier contenant un fichier `tp1.c` et un autre fichier `a.out` serait

`2..1.5tp1.c5a.out`

(attention si vous allez jouer à regarder dans le fichier les chiffres risquent de ne pas avoir la bonne tête)

Exercice 4 : archivage et extraction simples.

Le but de cet exercice est d'écrire une programme d'archivage et d'extraction.

1. Créer la fonction d'archivage qui assemble tous les fichiers du répertoire courant de la façon suivante :

   `tailleNomFichier1nomFichier1tailleFichier1fichier1...`
   `tailleNomDernierFichiernomDernierFichiertailleDernierFichierdernierFichier`

   Voici quelques indications :

   (a) Il ne faut pas archiver plusieurs fichiers : ., .., l'archive elle-même et l'exécutable produit par le programme. Il faut donc que vous archiviez les fichiers qui ont un nom différent de ”.”, ”..”, de celui de l'archive et de l'exécutable produit (par exemple ”a.out”).

   (b) Pour tester que deux chaînes de carctères sont différentes, on rappelle que l'on peut comaprer deux chaînes de caractères en utilisant la fonction `strcmp`[2].

   (c) Pour parcourrir les fichiers présents dans le répertoire, connaître leurs noms (et donc leurs tailles) vous utiliserez la même méthode que dans l'exercice précédent.

   (d) Pour connaître la taille d'un fichier, vous pouvez utiliser son *statut* : si un fichier a pour statut `stat`, alors sa taille peut être récupérée par la commande `stat.st_size`. Vous pouvez récupérer le statut d'un fichier en utilisant la fonction `stat` lors d'un appel système[3].

   (e) Lors de l'appel de la fonction `write`, faites bien attentionà recopier uniquement le nombre de caractères que vous avez lus et pas tout ce qui est contenu dans le buffer.

   (f) Pour pouvoir lire et écrire sur l'archive, il faut avoir les permissions requises. Afin d'avoir ces permissions, le paramètre `mode` lors de l'appel à la fonction `open` peut être fixé à `00664`.

   (g) Pensez à traiter les erreurs éventuelles en affichant des messages permettant de localiser facilement l'endroit de l'erreur. Par exemple, lorsque vous voulez traiter des erreur lors de l'appel de la fonction `open` pour l'ouverture de l'archive, vous pouvez utiliser l'instruction `perror("open de l'archive");`.

2. Définir la fonction d'extraction associée : vous devez, à partir du fichier d'archive, récupérer les noms des fichiers et leurs contenus. Pour pouvoir tester vos fonctions, vous pouvez utiliser le répertoire Test sur elearning !

1. Faites appel au *man* si vous voulez connaître le comportement de cette fonction.
2. Pour connaître le comportement d'une fonction, on vous rappelle qu'il faut consulter la page du *man* associée !
3. Utilisez le *man*.