UNIVERSITÉ DE PAU ET DES PAYS DE L'ADOUR | COLLÈGE STEE SCIENCES ET TECHNOLOGIES POUR L'ÉNERGIE ET L'ENVIRONNEMENT

Techniques de programmation - TP nº 3 - Manipulation de fichiers en langage C

Samson Pierre <samson.pierre@univ-pau.fr>

03/10/2021

Ce TP a pour objectif de vous familiariser avec la manipulation de fichiers en langage C. Pour chaque exercice, sauf indication contraire, aucune fonction n'est autorisée et les mêmes options de compilation que celles du cours doivent être utilisées. Attention : travaillez exceptionnellement en dehors de `stockage` car le système de fichiers utilisé dans ce répertoire ne supporte pas les permissions nécessaires aux exercices qui suivent.

Exercice nº 1

Créez un programme `write.c` qui lit les paramètres passés en ligne de commande. Le premier paramètre est le chemin d'un fichier. Le deuxième paramètre est le texte à écrire dans le fichier. Affichez un message d'erreur dans le flux d'erreur standard et retournez le code d'erreur 1 si le nombre de paramètres passés en ligne de commande est différent de deux.

Ce programme doit écrire le texte spécifié dans le fichier dont le chemin est passé en paramètre. Si le fichier n'existe pas, il doit être créé. Affichez le texte écrit, le nombre d'octets écrits et le chemin du fichier. Affichez un message d'erreur dans le flux d'erreur standard et retournez le code d'erreur 1 si une erreur se produit lors de l'ouverture, l'écriture ou la fermeture du fichier.

Voici le résultat attendu :

```
$ ./write.out; echo $?
invalid number of arguments
1
$ ./write.out /file.txt; echo $?
invalid number of arguments
1
$ ./write.out /file.txt "hello world"; echo $?
unable to open the file
1
$ ./write.out file.txt "hello world"; echo $?
written: "hello world" (11 byte(s) to "file.txt")
0
$ cat file.txt
hello world$
```

Voici les fonctions autorisées pour cet exercice :

```
int close(int fd);
int fprintf(FILE *stream, const char *format, ...);
int open(const char *pathname, int flags, mode_t mode);
int printf(const char *format, ...);
size_t strlen(const char *s);
ssize_t write(int fd, const void *buf, size_t count);
```

Exercice nº 2

Créez un programme `read.c` qui lit un paramètre passé en ligne de commande. Ce paramètre est le chemin d'un fichier. Affichez un message d'erreur dans le flux d'erreur standard et retournez le code d'erreur 1 si le nombre de paramètres passés en ligne de commande est différent de un.

Ce programme doit lire le texte dans le fichier dont le chemin est passé en paramètre. La lecture doit se faire par tranches de 9 octets jusqu'à ce que la fin du fichier soit atteinte. Affichez le texte lu, le nombre d'octets lus et le chemin du fichier. Affichez un message d'erreur dans le flux d'erreur standard et retournez le code d'erreur 1 si une erreur se produit lors de l'ouverture, la lecture ou la fermeture du fichier.

Voici le résultat attendu :

```
$ ./read.out; echo $?
invalid number of arguments
1
$ ./read.out file.txt; echo $?
unable to open the file
1
$ echo -n "hello world" > file.txt
$ ./read.out file.txt; echo $?
read: "hello wor" (9 byte(s) from "file.txt")
read: "ld" (2 byte(s) from "file.txt")
read: "" (0 byte(s) from "file.txt")
0
$
```

Voici les fonctions autorisées pour cet exercice :

```
int close(int fd);
int fprintf(FILE *stream, const char *format, ...);
int open(const char *pathname, int flags);
int printf(const char *format, ...);
ssize_t read(int fd, void *buf, size_t count);
```

Exercice n° 3

Créez un programme `fwrite.c` similaire à celui de l'exercice n° 1. La seule différence concerne les fonctions autorisées.

Voici les fonctions autorisées pour cet exercice :

```
int fclose(FILE *stream);
int ferror(FILE *stream);
FILE *fopen(const char *path, const char *mode);
int fprintf(FILE *stream, const char *format, ...);
size_t fwrite(const void *ptr, size_t size, size_t nmemb, FILE *stream);
int printf(const char *format, ...);
size_t strlen(const char *s);
```

Exercice n° 4

Créez un programme `fread.c` similaire à celui de l'exercice n° 2. La seule différence concerne les fonctions autorisées.

Voici les fonctions autorisées pour cet exercice :

```
int fclose(FILE *stream);
int ferror(FILE *stream);
FILE *fopen(const char *path, const char *mode);
int fprintf(FILE *stream, const char *format, ...);
size_t fread(void *ptr, size_t size, size_t nmemb, FILE *stream);
int printf(const char *format, ...);
```

Exercice n° 5

Créez un programme `fputs.c` similaire à celui de l'exercice n° 1. La seule différence concerne les fonctions autorisées.

Voici les fonctions autorisées pour cet exercice :

```
int fclose(FILE *stream);
int ferror(FILE *stream);
FILE *fopen(const char *path, const char *mode);
int fprintf(FILE *stream, const char *format, ...);
int fputs(const char *s, FILE *stream);
int printf(const char *format, ...);
size_t strlen(const char *s);
```

Exercice n° 6

Créez un programme `fgets.c` similaire à celui de l'exercice n° 2. La seule différence concerne les fonctions autorisées.

Voici les fonctions autorisées pour cet exercice :

```
int fclose(FILE *stream);
int ferror(FILE *stream);
char *fgets(char *s, int size, FILE *stream);
FILE *fopen(const char *path, const char *mode);
int fprintf(FILE *stream, const char *format, ...);
int printf(const char *format, ...);
size_t strlen(const char *s);
```

Exercice n° 7

Créez un programme `umask.c` qui lit les paramètres passés en ligne de commande. Le premier paramètre est le chemin d'un fichier. Le deuxième paramètre est le `umask` (une valeur octale). Affichez un message d'erreur dans le flux d'erreur standard et retournez le code d'erreur 1 si le nombre de paramètres passés en ligne de commande est différent de deux.

Ce programme doit ouvrir le fichier dont le chemin est passé en paramètre. Si le fichier n'existe pas, il doit être créé avec les permissions calculées à partir du `umask`. Affichez l'ancien et le nouveau `umask`. Affichez un message d'erreur dans le flux d'erreur standard et retournez le code d'erreur 1 si le `umask` ne peut pas être converti (exemple : s'il contient des chiffres qui n'existent pas en base octale). Affichez un message d'erreur dans le flux d'erreur standard et retournez le code d'erreur 1 si une erreur se produit lors de l'ouverture ou la fermeture du fichier.

Voici le résultat attendu :

```
$ ./umask.out; echo $?
invalid number of arguments
1
$ ./umask.out /file.txt; echo $?
invalid number of arguments
1
$ ./umask.out /file.txt 89a; echo $?
invalid umask
1
$ ./umask.out /file.txt 022; echo $?
unable to open the file
1
$ ./umask.out file.txt 022; echo $?
old umask = 022
new umask = 022
0
$ ls -l file.txt
-rw-r--r-- 1 spierre spierre [...] file.txt
$ rm file.txt
$ ./umask.out file.txt 077; echo $?
old umask = 022
new umask = 077
0
$ ls -l file.txt
-rw------- 1 spierre spierre [...] file.txt
$
```

Voici les fonctions autorisées pour cet exercice :

```
int fclose(FILE *stream);
FILE *fopen(const char *path, const char *mode);
int fprintf(FILE *stream, const char *format, ...);
int printf(const char *format, ...);
long int strtol(const char *nptr, char **endptr, int base);
mode_t umask(mode_t mask);
```