TP4 : Manipulation de fichiers

Vincent Danjean

IUP L3 — Sytèmes

Résumé

Ce TP permet de se familiariser avec les fonctions C permettant de manipuler les fichiers.

1 Les deux API (Application Programming Interface)

Le langage `C` offre deux interfaces pour manipuler les fichiers.
La première, bas niveau, offre un accès direct aux appels systèmes du noyau du système d'exploitation. Chaque appel de fonction se traduira donc par l'exécution d'un appel système.
La seconde, de plus haut niveau, apporte deux fonctionnalités :

1. *elle permet de manipuler plus facilement des types de haut niveau (conversions entre formats de nombre en particulier) ;*
2. *elle utilise également un système de tampons (ou* buffers*) : les E/S ne sont généralement pas demandées immédiatement au noyau. Cela permet de diminuer le nombre d'appels systèmes.*

Quel est l'intérêt de diminuer le nombre d'appels systèmes ?
Classer les fonctions suivantes entre celles appartenant à l'interface de bas niveau et celles appartement à l'interface de haut niveau :

1. `int` **fflush**`(FILE *stream);`
2. `int` **fscanf**`(FILE *stream, const char *format, ...);`
3. `int` **fclose**`(FILE *fp);`
4. `ssize_t` **read**`(int fd, void *buf, size_t count);`
5. `size_t` **fread**`(void *ptr, size_t size, size_t nmemb, FILE *stream);`
6. `size_t` **fwrite**`(const void *ptr, size_t size, size_t nmemb, FILE *stream);`
7. `int` **fseek**`(FILE *stream, long offset, int whence);`
8. `ssize_t` **write**`(int fd, const void *buf, size_t count);`
9. `FILE *` **fopen**`(const char *path, const char *mode);`
10. `int` **creat**`(const char *pathname, mode_t mode);`
11. `int` **close**`(int fd);`
12. `int` **open**`(const char *pathname, int flags[, mode_t mode]);`
13. `FILE *` **fdopen**`(int fildes, const char *mode);`
14. `off_t` **lseek**`(int fildes, off_t offset, int whence);`
15. `int` **fprintf**`(FILE *stream, const char *format, ...);`

Si vous ne savez pas encore ce que font ces fonctions (et leurs paramètres), regarder brièvement dans leur page de manuel correspondante. La plupart ont été vues en cours. Regardez plus en détail la fonction `open`, ses paramètres et les constantes que l'on peut utiliser pour ses paramètres.

Note: Pour les curieux
Pour la fonction `open` (ligne 12), on observe qu'il y a parfois deux et parfois trois paramètres. Comment est-ce possible (étant donné qu'il n'y a qu'une seule fonction `open` dans la bibliothèque `C`) ?

2 Observation des E/S tamponnées (*bufferisées*)

La fonction `printf` (l'équivalent de la fonction `fprintf` avec `stdout`, ie la sortie standard, en premier paramètre) appartient à l'interface de haut niveau. Elle offre donc des E/S tamponnées. Cela signifie qu'un `printf` dans un programme ne sera pas nécessairement suivi d'un affichage à l'écran.
Il y a plusieurs manières de forcer `printf` à afficher tout ce qu'il y a en attente dans le tampon :

1. on peut utiliser la fonction `fflush()` (sur la sortie standard, ie `stdout`) ;
2. on peut aussi utiliser le caractère `\n` à la fin de la chaîne à faire afficher. Dans le cas où `printf` affiche dans un terminal *et pas dans un fichier ou un tube*, ce caractère force la bibliothèque `C` a tout afficher ce qu'il y a en attente dans le tampon.

Compiler et exécuter le programme `print_fork.c`. Comprendre ce qui se passe au niveau de l'affichage.
Ajouter le caractère `\n` à la fin de la chaîne du premier `printf`. Compiler, exécuter, comprendre.
Rediriger la sortie standard du programme précédent (avec le `\n`) dans un fichier ou dans un tube (au choix). Observer la sortie du programme. Que faire pour résoudre ce problème ?

3 Lire et écrire dans un fichier

Regarder les programmes `api1_write.c` et `api1_read.c` qui écrivent et lisent une série d'entiers dans le fichier `data1`. Compiler, tester.
Compléter les programmes `api2_write.c` et `api2_read.c` pour lire et écrire une série d'entiers dans un fichier.
Comparer les fichiers écrits ainsi que le code des programmes. Donner des avantages et inconvénients à chacune des deux API.
Changer les permissions de `data1` et/ou `data2` pour observer ce qui arrive en cas de droits non valides.