

Fichiers

François Trahay



Contents

Entrées-sorties bufferisées	1
Ouverture/fermeture	1
Primitives d'écriture	2
binaire vs. ascii	2
Sortie d'erreur	3
Exemple	3
Primitives de lecture	4
Curseur	5

Entrées-sorties bufferisées

- Le système¹ fournit des primitives d'entrées/sorties (E/S) bufferisées
 - permet d'accéder au contenu de fichiers
 - *bufferisées*: les E/S sont d'abord groupées en mémoire, puis exécutées sur le périphérique
 - permet d'améliorer les performances
 - * exemple: 1024 écritures de 1 octet sont regroupées en une écriture de 1ko (donc gain important en performances)
 - **FILE***: type "opaque" désignant un fichier ouvert

¹ Pour être exact, il s'agit de la bibliothèque standard (la libc)

Plus précisément, **FILE*** désigne un *flux*. Ce flux peut être un fichier, mais également des *flux standards* (`stdin`, `stdout`, ou `stderr`), des tubes, des sockets, etc.

Ouverture/fermeture

- **FILE*** `fopen(char* fichier, char* mode);`

– `mode`: mode d'ouverture

- * `"r"` : lecture
- * `"w"` : écriture. Curseur au début du fichier
- * `"a"` : ajout. Curseur à la fin du fichier
- * `"r+", "w+", "a+"` : lecture et écriture

- `int fclose(FILE* f);` * Complète les opérations et ferme le fichier

Pour plus de détails concernant les modes d'ouverture, `man fopen` indique:

- `r`: Ouvre le fichier en lecture. Le pointeur de flux est placé au début du fichier.
- `r+`: Ouvre le fichier en lecture et écriture. Le pointeur de flux est placé au début du fichier.
- `w`: Ouvre le fichier en écriture. Le fichier est créé s'il n'existait pas. S'il existait déjà, sa longueur est ramenée à 0. Le pointeur de flux est placé au début du fichier.
- `w+`: Ouvre le fichier en lecture et écriture. Le fichier est créé s'il n'existait pas. S'il existait déjà, sa longueur est ramenée à 0. Le pointeur de flux est placé au début du fichier.
- `a`: Ouvre le fichier en ajout (écriture à la fin du fichier). Le fichier est créé s'il n'existait pas. Le pointeur de flux est placé à la fin du fichier.
- `a+`: Ouvre le fichier en lecture et ajout (écriture en fin de fichier). Le fichier est créé s'il n'existait pas. Le pointeur de flux est placé à la fin du fichier.

Après appel à la fonction `fclose`, `f` (le `FILE*`) devient inutilisable : le pointeur pointe vers une zone mémoire qui a peut être été libérée par `fclose`. Il convient donc de ne plus utiliser le fichier !

Primitives d'écriture

- `int fprintf(FILE* f, char* format, ...);`
 - similaire à `printf`, mais écrit dans le fichier `f`
 - écrit une chaîne de caractères dans un fichier
- `size_t fwrite(void* ptr, size_t size, size_t nmemb, FILE* f);`
 - écrit les `size×nmemb` octets situés à l'adresse `ptr` dans `f`

binaire vs. ascii

Quelle est la différence entre `printf` et `fwrite` ? La fonction `fprintf` écrit un ensemble de caractères ASCII dans le fichier, alors que `fwrite` écrit un ensemble de bits.

Ainsi, pour écrire la valeur 12 dans un fichier en ASCII, on peut exécuter:

```
fprintf(f, "%d", 12);
```

Le fichier contiendra donc les octets 0x31 (le caractère '1'), et 0x32 ('2').

Pour écrire la valeur 12 en binaire, on peut exécuter:

```
int n=12;
fwrite(&n, sizeof(int), 1, f);
```

Le fichier contiendra donc les octets 0x0C 0x00 0x00 0x00, c'est à dire les 4 octets d'un int dont la valeur est 12.

Sortie d'erreur

Puisqu'un FILE* désigne en fait un flux, on peut utiliser fprintf pour écrire sur la sortie standard d'erreur :

```
fprintf(stderr, "Warning: flux capacitor overflow !\n");
```

Exemple

Voici un programme montrant l'utilisation de primitives de lecture et d'écriture:

```
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>

int main(int argc, char**argv) {
    if (argc != 3) {
        fprintf(stderr, "USAGE = %s fichier_source fichier_destination\n", argv[0]);
        return EXIT_FAILURE;
    }

    /* open the input and output files */
    FILE*f =fopen(argv[1], "r");
    assert(f);
    FILE*f_out =fopen(argv[2], "w");
    assert(f_out);
    char line[1024];
    int lineno = 1;
    /* read the input file line by line */
    while(fgets(line, 1024, f)) {
        /* write the line to the output file */
        fprintf(f_out, "%s", line);
        lineno++;
    }

    /* close the files */
    fclose(f);
    fclose(f_out);
}
```

```
    return 0;
}
```

Primitives de lecture

- `int fscanf(FILE* f, char* format, ...);`
 - similaire à `scanf`, mais lit depuis le fichier `f`
- `size_t fread(void* ptr, size_t size, size_t nmemb, FILE* f);`
 - lit `nmemb×size` octets et les stocke à l'adresse `ptr`
 - retourne le nombre d'items lus
 - `fread` renvoie une valeur `< nmemb` si la fin du fichier (EOF) est atteinte
- `char* fgets(char* s, int size, FILE* f);`
 - lit au plus `size` caractères et les stocke dans `s`
 - * arrête la lecture avant `size` si lit `\n` ou EOF

Ces 3 fonctions sont généralement utilisées chacune dans un cas précis:

- `fscanf` est utilisée pour lire des valeurs et les stocker dans des variables.
Par exemple:

```
int a;
float b;
fscanf(f, "%d\t%f\n", &a, &b);
```

- `fread` est utilisée pour charger le contenu d'une (ou de plusieurs) structure(s):

```
struct s {
    int a;
    float b;
    char c;
};
struct s tab[10];
fread(tab, sizeof(struct s), 10, f);
```

- `fgets` est utilisée pour lire un fichier ligne par ligne:

```
char line[1024];
int lineno = 1;
while(fgets(line, 1024, f)) {
    printf("line %d: %s\n", lineno, line);
    lineno++;
}
```

Curseur

- Position dans le fichier à laquelle la prochaine opération aura lieu
 - Initialisé à 0 (le début du fichier) si le fichier est ouvert en mode `r`, `r+`, `w`, `w+`
 - Positionné à la fin du fichier si ouvert en mode `a` ou `a+`
- Avance à chaque opération de lecture/écriture
- `long ftell(FILE *stream);`
 - Indique la position courante (en nombre d'octets depuis le début du fichier)
- `int fseek(FILE *f, long offset, int whence);`
 - déplace le curseur de `offset` octets depuis
 - * le début du fichier (si `whence` vaut `SEEK_SET`)
 - * la position courante (si `whence` vaut `SEEK_CUR`)
 - * la fin du fichier (si `whence` vaut `SEEK_END`)