Les Fichiers en C sous UNIX

Michael Mrissa

Université de Pau
et des Pays de l'Adour

Note de l'enseignant

Cours initialement créé par A. Aoun, A. Benzekri, J.-M. Bruel.
Supports repris de Nicolas Belloir, merci à lui.

Librairies, Entrées et Sorties

Librairies concernées

La librairie <stdio.h> contient les prototypages de toutes les fonctions d'entrées/sorties standards, la structure FILE et quelques macros utiles (stdin, stdout, stderr, EOF, NULL, ...).

Définition

Le langage C ne comporte **aucune** instruction d'entrée/sortie (E/S) ! Toute opération de ce type est réalisée :

- soit par un **appel système** (famille open, read, write) ;
- soit par une fonction de la **bibliothèque standard** (famille fopen, fread, fwrite) .

Remarque

Certaines de ces fonctions sont macro-définies (getc, getchar, ...).

Définition

Le système d'E/S d'UNIX est divisé en deux sous-systèmes :

- en mode **bloc** (ou mode structuré) pour les opérations disque ;
- en mode **caractère** (ou non structuré) pour les E/S sur terminaux.

Remarque

Au niveau des appels systèmes, les fichiers sont identifiés par un descripteur (ou nom symbolique) alors que les fonctions de la bibliothèque standard ont comme résultat des pointeurs sur des objets de type FILE.

Définition

La bibliothèque présente 2 avantages :

- nombre d'appels au moniteur minimisés ;
- niveau élevé des fonctions (E/S formatées, etc.).

Remarque

Par contre, en cas d'interruption de processus, puisque les fonctions de la bibliothèque travaillent avec un tampon mémoire, le transfert mémoire/fichier physique peut ne pas s'être réalisé et générer des problèmes. D'où l'utilisation des fonctions de vidage de *buffer* (fflush).

Définition

En UNIX, les droits d'accès à un fichier sont mémorisés sur 12 bits :

- 11, 10, 9 : set_uid, set_gid et "sticky bit", d'utilisation particulière ;
- 8, 7, 6 : accès lecture, écriture, exécution pour le propriétaire (owner) ;
- 5, 4, 3 : accès lecture, écriture, exécution pour le groupe (group) ;
- 2, 1, 0 : accès lecture, écriture, exécution pour les autres (others).

Remarque

La bibliothèque fcntl.h est parfois nécessaire à ces fonctions.

Les entrées sorties formatées clavier

La fonction *printf()*

Format

```
printf( <formats> [,<paramètres>]);
```

Fonctionnement

La fonction printf() est une fonction qui permet d'afficher sur la console des messages suivant le format donné par une chaîne de caractère, et de convertir à l'intérieur de cette chaîne les valeurs des paramètres.

Example

```
a=65;
printf("%d",a);
    /* 65 */
printf("La valeur de a est %d \n",a);
    /* La valeur de a est 65 */
printf("%d\n%c\n%5.2f\n",a,a,a);
    /*  65
        A
        65.00 */
```

La fonction *scanf()*

Format

```
scanf( < formats > , < adresse des paramètres >);
```

Fonctionnement

La fonction scanf() est une fonction qui permet d'entrer à partir de la console des valeurs qui seront stockées à l'intérieur des paramètres, en effectuant la conversion indiquée dans la chaîne des formats.

Example

```
scanf("%d",&a); /* lecture d'un entier */
scanf("%d␣%c␣%f",&a,&c,&f);
/* Lecture d'un entier, d'un caractère et d'un réel */
```

Remarque

Pour utiliser les fonctions scanf() et printf() ajoutez en entête de votre fichier la directive :

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

Remarque

Dans les chaînes de formats on utilise :

%d	pour convertir en un entier
%s	pour convertir en une chaîne de caractères
%f	pour convertir en un réel
%lf	pour convertir en un réel double
%ld	pour convertir en un entier long
%x	pour convertir en un hexadécimal
%c	pour convertir en un seul caractère
\n	pour introduire un saut de ligne
\t	pour une marque de tabulation
\r	pour introduire un retour chariot
\\	pour le caractère \
\"	pour le caractère "

Création et destruction d'un fichier

La fonction creat()

BUT : Création d'un fichier en précisant l'accès autorisé.

DECLARATION

```
int creat(nom_fichier, autorisation);
    char *nom_fichier;
    int autorisation;
```

PARAMETRES

autorisation : nombre représentant les différents droits d'accès (rwxrwxrwx) comme tout fichier UNIX.

RETOUR

Succès : numéro de descripteur de fichier

Échec : -1 (et positionnement de errno)

Example

EXEMPLE[a]

```
int num;
num = creat("fichier.dta",0777);
```

---

a. Le code octal 0777 représente : tous les droits pour tout le monde (on remarque que le umask est appliqué).

Remarque

- Si le fichier existe il est écrasé.
- Les fichiers standards ont un descripteur prédéfini :
  - stdin 0
  - stdout 1
  - stderr 2

La fonction unlink()

BUT : Destruction de fichier.

DECLARATION

```
int unlink(nom_fichier);
char *nom_fichier;
```

RETOUR

Succès : 0

Échec : -1 (et positionnement de errno)

Example

```
retour = unlink("fichier.dat");
```

Ouverture et fermeture d'un fichier

La fonction open()

BUT : Ouverture d'un fichier dans le mode d'accès spécifié. [a]

DECLARATION

```
int open (nom_fichier, mode[,autorisation]);
char *nom_fichier;
int mode;
[int autorisation;]
```

RETOUR

Succès : numéro de fichier

Échec : -1 (et positionnement de errno)

a. Pas nécessaire après un creat.

La fonction open()

PARAMETRES

mode	O_RDONLY	lecture uniquement
	O_WRONLY	écriture uniquement
	O_RDWR	lecture et écriture
	O_APPEND	ajouts en fin de fichier
	O_CREAT	création de fichier
autorisation	(comme creat())	
	O_TRUNC	effacement du contenu du fichier
	O_TEXT	mode translaté
	O_BINARY	mode binaire

Example

```
int num;
num= open("fichier.dta",O_WRONLY|O_CREAT,0777);
/* ouverture en écriture, s'il n'existe pas alors */
/* création en lecture/écriture */
```

La fonction fopen()

BUT : Ouverture d'un fichier dans le mode d'accès spécifié avec accès tampon.

DECLARATION

```
FILE *fopen (nom_fichier, type);
char *nom_fichier;
char *type;
```

PARAMETRES

type :

r	lecture	r+	lecture/écriture
w	écriture/création/effacement	w+	lecture/écriture/création/efface
a	création/ajout eof	a+	lecture/création/ajout eof

RETOUR

Succès : pointeur vers une structure FILE

Échec : pointeur NULL

Example

```
FILE *ptr;
ptr =  fopen("fichier.dta","w+");
```

Remarque

Il y a des cas où des fichiers sont ouverts sans utiliser d'open : les pipes.

La fonction close()

BUT : Fermeture du fichier ouvert avec open().

DECLARATION

```
int close(num);
int num;
```

RETOUR

Succès : 0

Échec : -1 (et positionnement de errno)

La fonction fclose()

BUT : Fermeture du fichier ouvert avec fopen().

DECLARATION

```
int fclose(ptr);
FILE *ptr;
```

RETOUR

Succès : 0

Échec : -1 (et positionnement de errno)

Lecture et écriture dans un fichier

La fonction read()

BUT : Lecture d'un certain nombre d'octets d'un fichier ouvert avec open().

DECLARATION

```
int read(numero, buffer, nombre);
int numero;
char *buffer;
unsigned nombre;
```

PARAMETRES

numero	l'entier retourné par open()
buffer	pointeur sur la zone de réception
nombre	nombre d'octets à lire

RETOUR

Succès :	nombre d'octets réellement lus
Échec :	-1 (et positionnement de errno)

La fonction write()

BUT : Écriture d'un certain nombre d'octets d'un fichier ouvert avec open().

DECLARATION

```
int write(numero, buffer, nombre);
```

RETOUR

Succès : nombre d'octets réellement écrits

Échec : -1 (et positionnement de errno)

La fonction fread()

BUT : Lecture d'un certain nombre d'éléments regroupant un certain nombre d'octets.

DECLARATION

```
size_t fread(buffer, taille, nbelem, ptr);
void *buffer;
unsigned taille;
unsigned nbelem;
FILE *ptr;
```

PARAMETRES

buffer :	pointeur sur la zone de réception
taille :	nombre d'octets d'un élément
nbelem :	nombre d'éléments à lire
ptr :	pointeur renvoyé par fopen().

La fonction fread()

RETOUR

Succès :	nombre d'éléments lus
Échec :	nombre d'éléments lus inférieur à celui désiré. Utiliser feof() ou ferror() pour savoir si c'est vraiment un échec ou si c'est la fin du fichier.

La fonction fwrite()

BUT : Ecriture d'un certain nombre d'éléments dans un fichier ouvert avec fopen().

DECLARATION

```
size_t fwrite(buffer, taille, nbelem, ptr);
```

RETOUR

Succès : nombre d'éléments réellement écrits

Échec : nombre d'éléments écrits inférieur à celui désiré

La fonction fgets()

BUT : Lit une chaîne dans un fichier.

DECLARATION

```
char *fgets(line, ptr);
char *line;
FILE *ptr;
```

PARAMETRES

line	pointeur sur la zone de réception
maxline	nombre de caractères à lire
ptr	pointeur de fichier.

RETOUR

Succès :	pointeur vers une chaîne de caractères
Échec :	pointeur NULL

REMARQUE

Au plus maxline-1 caractères seront lus et la ligne se termine par \0.

La fonction fputs()

BUT : Écrit une chaîne dans un fichier.

DECLARATION

```
int fputs(line, maxline, ptr);
char *line;
int maxline;
FILE *ptr;
```

RETOUR

Succès : dernier caractère écrit

Échec : -1

Positionnement dans un fichier

La fonction lseek()

BUT : Positionnement[a] dans un fichier ouvert avec open().

DECLARATION

```
long lseek(numero, deplacement, origine);
int numero;
long deplacement;
int origine;
```

RETOUR

Succès : nouvelle position dans le fichier

Échec : -1L (et positionnement de errno)

a. il existe une fonction seek() identique à lseek() mais avec int deplacement.

La fonction lseek()

PARAMETRES

numero	l'entier retourné par open()
deplacement	valeur du déplacement en octets
origine	spécifie le départ pour le déplacement :
	SEEK_SET début de fichier
	SEEK_CUR position courante
	SEEK_END fin de fichier.

Example

```
long deplac;
deplac = lseek(num,30,SEEK_SET);
```

La fonction fseek()

BUT : Positionnement dans un fichier ouvert avec fopen().

DECLARATION

```
int fseek(ptr, deplacement, origine);
```

RETOUR

Succès : 0

Échec : valeur non nulle