Entrées-Sorties: Stdio.h

Manipulation de fichiers

- Pour manipuler un fichiers:
- Plusieurs fonctions dispo dans stdio.h
- 4 types de fonctions:
  - Ouverture
  - Lectures
  - Écritures
  - Fermetures

Ouverture du fichier

- Pour pouvoir manipuler (lire ou écrire)
- Il faut “ouvrir” le fichier.
- Se fait avec une fonction: `fopen()`
- Permet d’obtenir un objet pour manipuler le fichier
- Avec stdio le type obtenu est un `FILE *`

`FILE *`

- L’accès au fichier se fait grâce à ce `FILE *`
- Cette structure fournit un index de lecture et/ou d’écriture sur le fichier.
- Il est possible d’ouvrir deux fois le fichier pour pouvoir accéder au fichier de deux manières différentes.

`FILE *` (Suite)

- Flux standards
- `stdin`: entrée standard (saisie terminal/clavier).
- `stdout`: affichage de sortie standard.
- `stderr`: affichage de sortie d’erreur.
- Ouvert par défaut pour chaque programme.
- Permettent d’avoir une écriture unifiée du programme que la lecture/écriture se fasse depuis/sur un vrai fichier ou depuis un terminal.

Structure générique

```
FILE * fd =
fopen(“Nom_fichier”,”r+) ;
…
//manipulation
…
fclose(fd) ;
```

fopen

- La fonction renvoie un pointeur sur une structure FILE; NULL si erreur.
- Premier paramètre: Nom de fichier (avec éventuellement le chemin du fichier).
- Second paramètre: Le mode d’ouverture.

Mode d’ouverture

- “r”: Lecture seule
- “w”: écriture
- “a”: écriture, curseur à la fin du fichier (a pour append)
- “rb”, “wb”, “ab” (même chose mais en binaire; pour les fichiers exécutables/compressés…)

Mode d’Ouverture (suite)

- “r+”: Lecture/écriture curseur au début
- “w+”: Lecture/écriture → crée le fichier s’il n’existe pas ET remet la taille du fichier à 0.
- “a+: Ouvre le fichier en ajout, place le curseur à la fin. Crée le fichier s’il n’existe pas.

Échec de `fopen()`

- La fonction `fopen` peut échouer.
- Les raisons sont multiples:
  - Fichier inexistant.
  - Droits incompatibles avec le mode demandé.
  - Ressources du systèmes indisponibles (trop de fichiers ouverts)

Affichage des erreurs.

- En cas d'échec `fopen` renvoie un pointeur NULL.
- Elle met également à jour, une variable globale: `errno`, pour "error number"
- On peut afficher l'erreur, si on appelle juste après la fonction `perror()`

Exemple `fopen`:

```
FILE * fd=fopen(“test.txt”,”r”) ;
if(fd==NULL){
 perror(“Problème d’ouverture de fichier”) ;
exit(-1) ;
}
// Manipulation normale du fichier
//...
```

Fermeture du fichier

- Une fois l'utilisation du fichier terminée on doit fermer le descripteur (`FILE *`).
- On utilise la fonction fclose
- Elle prend un seule paramètre : la variable `FILE *`
- Elle Renvoie 0 si Ok ; `EOF` (pour End Of File)
  si problème (p. ex essayer de fermer un descripteur déjà fermé).
- Permet de libérer le descripteur utilisé (ressource du système d'exploitation).

Notion de curseur

- Contrairement au terminal, la lecture dans un fichier nécessite une position:

  un curseur ou index.

- Le curseur est conservé dans la structure `FILE*`

Écritures dans les fichiers

Fonctions d’écritures

- Plusieurs fonctions d’écritures sont fournies.
- Chaque fonction a un usage qui lui est propre.
- Elles sont disponibles pour faciliter la programmation.

Fonction `frpintf`

- Elle est comme la fonction `printf`, elle permet un affichage formaté
- Elle quasiment les mêmes paramètres
- Signature:
  `int frpintf(FILE * stream, const char * format,…)`
- Renvoie le nombre d'octets écrits. Nombre négatifs en cas d'erreur. Erreur consultable avec `perror()`.

printf

- La fonction `printf(format,…)` est équivalent à :
- `fprintf(`**`stdout`**`,format,…)`
- `printf` est un cas particulier de `fprintf`

Distinction entre `stdout` et `stderr`

- Par défaut, l'affichage des deux flux arrivent dans le terminal.
- La distinction est utile pour séparer:
  - Les message d'erreur et de débogage (`stderr`).
  - La sortie normale du programme

Exemple

```
./programme > fichier.txt
```

- Permet de récupérer l'affichage de `stdout` dans fichier.txt.
- Si on veut récupérer l'affichage de `stderr` dans un fichier:

```
./programme 2> erreur.txt
```

Fonction `fputc`

```
int fputc(int char, FILE * stream)
```

- La fonction écrit un seul caractère dans un flux.
  - Le curseur d’écriture est avancée d’une position
  - Renvoie le caractère écrit si Ok, EOF sinon

Fonction `fputs`

```
int fputs(char * str, FILE * stream) ;
```

- Même chose que la précédente, mais écrit une chaîne de caractère `str`.
- `str` doit contenir un caractère `\0` pour marquer la fin.

Fonction `fputs`

- La fonction n’écrit pas le `\0` .
- Renvoie le nombre d’octets écrits EOF sinon.
- Déplace le curseur d’écriture du nombre d’octets écrits.

Fonction `fwrite()`

- La fonction `fwrite` écrit le contenu d'un buffer sans distinction de type
- Fonction très générale
- Ne formate pas.
- Utile pour écrire des données brutes.
  (p. Ex un tableau d'entiers)
  - Il faut faire attention pour les structures à plusieurs dimensions.

Fonction `fwrite()`

- Signature:
  `size_t fwrite(const void * ptr, size_t size, size_t nmemb, FILE* stream)`
- `ptr` est le pointeur sur la zone mémoire à écrire
- `size` est la taille de chaque élément (1 pour un caractère, 4 pour un entier,…)
- `nmemb` correspond au nombre d'élements dans la zone
- `stream` est le flux dans lequel écrire

Lecture depuis un fichier

Fonctions de lectures: stdio

- La librairie stdio propose plusieurs fonctions de lectures
- Les fonctions à utiliser dépendent de l’usage

Fonction `fscanf`

- La fonction `fscanf` se comporte comme `scanf` classique.
- La signature est très similaire
  `int fscanf(FILE * fd, const char * format,… );`
- Les paramètres dans l'ellipse sont les pointeurs sur les variables à affecter.
- Renvoie le nombre de variable correctement affectée

Fonction `fscanf`

- Exemple :

```
int a,b,c ;
int k=fscanf(fd,''%d%d%d'',&a,&b,&c) ;
```

- Renverra 3 si il parvient à lire 3 entiers.
- Si le contenu dans le fichier est ‘’12 271 100”
  les 3 entiers seront correctement lu.
- Si le contenu dans le fichier est ‘’256 test 999”
  Seul le premier entier sera lu.

Fonction `fscanf`

- La fonction avance dans le flux.
- Les espaces et les retours à la ligne sont des séparateurs pour `scanf/fscanf`
- Exemple :

```
char * chaine= … ;
fscanf(‘’%s’’, chaine) ;
```

- ‘’bonjour tout le monde’’
- Seul le mot ‘’bonjour’’ sera placé dans la chaîne.

Utilité de `scanf/fscanf`

- Ces fonctions sont très utiles pour faire du *parsing:* à savoir faire des conversions de chaînes de caractères vers un entier ou un flottant.
- Fonctions déconseillées pour copier l’intégralité d’un fichier.

Fonction `fgetc`

```
unsigned int fgetc(FILE *  stream) ;
```

- Lit un seul caractère depuis le flux.
- Avance d’une position dans le flux.
- Renvoie le caractère lu ou EOF en cas d’erreur.
- Un caractère est compris entre 0 et 255. (ce qui fait 256 valeurs).
- Le type de retour est un entier pour pouvoir prendre en compte l’erreur.

Fonction `fgets`

```
char * fgets(char * str,int n,
FILE* stream) ;
```

- `str` :est un tableau de caractères.
  Le tableau doit déjà être créé.
- `N:` est la taille du tableau
  (en comptant le `'\0'`)
- `stream`: le fichier sur lequel lire.

Fonction `fgets`

- La fonction s’arrête si :
  - n-1 caractères ont été lus (est ajouté par la fonction `‘\0’`).
  - Une nouvelle ligne est rencontrée (*i.e.* `‘\n’`).
  - La fin du fichier est atteinte.
- La fonction renvoie un pointeur sur la chaîne si Ok.
- Renvoie `NULL` si erreur ou fin de fichier atteinte.
  Le contenu de `str` n’est pas changé dans ce cas.

Fonction `fread`

- La fonction `fread` est la version en lecture de `fwrite`.
- La lecture est non formatée (contrairement aux précédentes)
- Signature :

```
size_t fread(void * ptr, size_t size, size_t nmemb, FILE * fd) ;
```

Fonction `fread`

- `ptr` est la zone mémoire dans laquelle écrire.
- `size` la taille d'une donnée (i.e. taille d'un case du tableau).
- `nmemb` le nombre d'éléments dans le tableau.
- `fd` le fichier depuis lequel lire.

Fonction `fread`

- Exemple:

```
int *  T =
malloc(100*sizeof(int)) ;
…
int k=fread(T,sizeof(int),100,fd) ;
```

- La fonction va lire 400 octets et va placer ces 400 octets dans le tableau d’entiers.

Fonction `fread`

- Renvoie k le nombre d’éléments correctement lus.
- Si k != nmemb, soit erreur soit fin du fichier atteinte.

Déplacement dans le flot de fichier

Déplacement.

- Un fichier peut être vu comme un flux de donnée.
- Chaque opération de lecture ou d’écriture déplace le curseur de lecture/écriture.
- Il peut être nécessaire de se déplacer à l’intérieur du flux.

Deux types de fonctions

- Accesseur de consultation:
  - Savoir où on se trouve dans le fichier.
- Accesseur de déplacement:
  - Pour déplacer le curseur de lecture/écriture.

Accesseur de consultation

- Deux fonctions disponibles
  - `ftell`
  - `fgetpos`
- La seconde est plus *compliquée* à utiliser.
- La première peut ne pas être disponible sur tous les systèmes.

Fonction `ftell`

- Signature:
  `long int ftell(FILE * fd) ;`
- Renvoie la position courante dans le fichier. 0 correspond au début du fichier.
- Renvoie `-1L` si erreur.
- Usage: s’appelle quand on veut stocker une position particulière pour pouvoir y revenir ensuite.

Fonction `fgetpos`

- Signature :
  ```
  int fgetpos(FILE * fd, fpos_t *pos) ;
  ```
- Elle renseigne la structure passée en paramètre `pos`.
- On ne peut pas faire d'arithmétique sur la position.
- S'utilise ensuite avec `setpos`.
- Renvoie 0 si Ok : sinon valeur différente de 0.

Accesseur de modification

- Trois fonctions :
  - `fseek` (analogue de `ftell`).
  - `setpos` (analogue de `getpos`).
  - `rewind`.

Fonction `fseek`

- Signature :

```
int fseek(FILE * fd, long int offset, int whence) ;
```

- `fd` : Le fichier sur lequel se déplacer.
- `whence` est la position de référence :
  -

Fonction `fseek`

- `whence` : Trois valeurs possibles :
  - `SEEK_SET` : début du fichier.
  - `SEEK_CUR` : position courante du curseur.
  - `SEEK_END` : fin du fichier (attention ça peut changer).
- offset : décalage par rapport à whence.
  la valeur peut être négative (si `whence = SEEK_CUR ou SEEK_END`)

Fonction `fseek`

- La fonction renvoie 0 si Ok. Diff. de 0 sinon.
- Exemples :
  - `fseek(fd,0,SEEK_SET) ;`
    positionne le curseur au début du fichier.
  - `fseek(fd,-100,SEEK_CUR) ;`
    revient en arrière de 100 octets à partir de la position courrante du curseur.
  - `fseek(fd,10000000,SEEK_END) ;`
    Rajoute 1 000 000 d'octets à la fin du fichier.

Fonction `fsetpos`

- Signature :
  ```
  int fsetpos(FILE * fd, const fpos_t * pos) ;
  ```
- `fd` le fichier sur lequel se déplacer.
- `pos` la structure de position :
  on ne peut pas faire d'arithmétique comme avec `fseek`. Il faut enregistrer des positions dans des variables.
- Renvoie 0 si Ok.

Fonction `rewind`

- Signature:
  ```
  void rewind(FILE * fd);
  ```
- La fonction `rewind` permet de remettre le curseur au début du fichier.
- Équivalent à:
  ```
  fseek(stream, 0L, SEEK_SET)
  ```

Autres fonctions Utiles

Fonctions `fflush` et `feof`

- Pour savoir si on a atteint la fin d'un fichier.

  On peut utiliser feof()
- Signature:
  ```
  int feof(FILE * fd) ;
  ```
- Renvoie 0 si à la fin du fichier. Diff de 0 sinon.

`fflush`

- Les entrées/sorties avec stdio sont *“bufferisées”.*
- Quand on demande une écriture (`fprintf`,`fwrite`...), elle n’est pas faite tout de suite.
- Ça peut poser des problèmes quand on cherche des erreurs : déterminer où se situe l’erreur à l’aide d’affichage successifs dans le programme.

`fflush` et `flush`

- Les fonctions `flush` et `fflush` permettent de déclencher les écritures.
- Signature :
  ```
  int fflush(FILE * fd) ;
  ```
- Renvoie 0 si Ok.

Modifications des fichiers

- Les fichiers peuvent être modifiées par plusieurs programmes en même temps.
- Si tout le monde est en lecture : pas de problème.
- Si un ou plusieurs programme sont en écriture. Ça peut poser des problèmes de cohérence des données sur le fichier.

Modifications des fichiers

- La librairie stdio fournit des fonctions de blocage entre plusieurs threads mais pas entre processus.
- Pour « bloquer » un fichier il faut avoir recours à des fonctions systèmes non traitées dans ce cours.