Cours de Programmation Impérative
Lecture et écriture dans les fichiers texte

Julien David

A101 - david@lipn.univ-paris13.fr

Résumé

Jusqu'ici nos programmes peuvent

- lire et écrire de l'information située dans la RAM
- interagir avec un terminal (`printf`, `scanf`)

On peut en réalité faire beaucoup plus

- lire et écrire sur d'autres support (disque dur, clés usb)
- interagir avec d'autres programmes
- interagir avec d'autres périphériques (carte réseau, carte graphique, carte son, imprimante, . . .)

Résumé

Jusqu'ici nos programmes peuvent

- lire et écrire de l'information située dans la RAM
- interagir avec un terminal (`printf`, `scanf`)

On peut en réalité faire beaucoup plus

- lire et écrire sur d'autres support (disque dur, clés usb)
- interagir avec d'autres programmes
- interagir avec d'autres périphériques (carte réseau, carte graphique, carte son, imprimante, . . .)

Disque Dur

Le disque dur

Organisation des répertoires sur le disque dur

- Les répertoires sont organisée en forme d'**arborescence**.
- Comme tout les arbres, celui ci possède une racine.
- Sous linux, on la note /

Observons l'organisation sur cet ordinateur. . .

Systèmes de fichiers

- /**bin** : éxécutables commun aux utilisateur (`ls, cd, rm, ...`).
- /**boot** : ensemble de fichiers nécéssaires lors du démarrage de l'ordinateur.
- /**etc** : fichiers de configuration de programme installés.
- /**home** : répertoires des utilisateurs.
- /**proc** : informations sur le processeur.
- /**tmp** : fichiers temporaires créés par dess processus en cours d'exécution.
- /**usr** : ensemble de librairies, programmes, fichiers communs aux utilisateurs (`stdio.h, gedit, emacs, gcc, ...`).

Répertoire courant

Répertoire courant

- Comme son nom l'indique, il s'agit du répertoire dans lequel on est en train de travailler.
- Pour changer de répertoire courant, on utilise la commande `cd`

Chemins

Chemin absolu

Le chemin absolu est le chemin qui mène de la racine / au fichier/répertoire auquel on souhaite accéder.

- Pour connaître le chemin absolu du répertoire courant, on utilise la commande `pwd`

Chemin relatif

Le chemin absolu est le chemin qui mène du répertoire courant au fichier auquel on souhaite accéder.

Les droits

Les différents droits

- Lecture : *r*
- Écriture : *w*
- Exécution : *x*

Les différents utilisateurs

- L'utilisateur : u (user),
- Son groupe : g (group),
- Les autres : o (other).

Les droits

Observer les droits

Pour connaître les droits des fichiers du répertoire courant, on utilise la commande `ls -l`

Changer les droits

Pour changer les droits d'un fichier/répertoire, on utilise la commande `chmod`

Les fichiers en C

Les fichiers en C

L'idée

Tout comme pour les interactions avec le terminal :

- le programme n'interagit pas directement avec le disque dur.
- c'est le système d'exploitation qui se charge de la communication entre les différents composants.
- pour qu'un programme accède à un fichier, il demande au système de lui donner un accès.
- le système va alors créer un espace mémoire ou le programme pourra lire et/ou écrire.
- Cet espace mémoire est appelé **tampon**, ou **buffer**.
- On va voir aujourd'hui comment manipuler cet espace mémoire.

Note : voir le programme `terminal`

Les fichiers : un nouveau type de variable

Le type `FILE`

- le type `FILE` permet de manipuler des fichiers.
- il s'agit d'une structure qui contient l'adresse du tampon,
- la position de la `tête de lecture/écriture` dans le fichier.
- les droits que le programme possède sur le fichier.

En pratique

On utilise toujours un `FILE *`

Les fonctions : ouverture d'un fichier

Ouvrir un fichier

- `FILE *fopen(const char *path, const char *mode);`

Arguments

- `const char *path` : chemin relatif ou absolu par rapport au répertoire courant.
- `const char *mode` : mode d'ouverture du fichier.

Les fonctions : ouverture d'un fichier

Commmande	Mode	Position	Fichier inexistant
r	lecture	Début	renvoit NULL
r+	lecture+écriture	Début	renvoit NULL
w	écriture	Début	Crée le fichier.
w+	lecture+écriture	Début	Crée le fichier.
a	lecture+écriture	Fin	Crée le fichier.
a+	lecture+écriture	R : Début W :Fin	Crée le fichier.

Note : programme `fichier.c`

Les fonctions : ouverture d'un fichier

Valeur de retour de `fopen`

- `fopen` renvoit donc `NULL` si la fonction n'a pas réussi à ouvrir le fichier.
- il faut TOUJOURS tester la valeur de retour de cette fonction. Sinon le programme a de forte chance de planter.

Les fonctions : fermeture d'un fichier

La fonction `fclose`

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

- La fonction `fopen` effectue un `malloc` pour créer le tampon/buffer qui va servir à interagir avec le disque dur.
- La fonction `fclose` permet de libérer ce bloc mémoire et si besoin, force le système à écriture les données du tampon qui n'ont pas encore été transférée sur le disque dur.

Lecture et écriture dans un fichier

La fonction `fprintf`

Fonctionne comme la fonction `printf`, excepté que le premier argument de la fonction est le fichier dans lequel on souhaite écrire.

La fonction `fscanf`

Fonctionne comme la fonction `scanf`, excepté que le premier argument de la fonction est le fichier dans lequel on souhaite lire.

Lecture et écriture dans un fichier

Les limites de l'écriture

On peut écrire dans le fichier tant qu'il y a de la place sur le disque dur.

- on s'autorise à considérer que ça n'arrivera pas.

Les limites de la lecture

On lit dans le fichier jusqu'à ce qu'il soit terminé.

- la fonction `int feof(FILE *F)` permet de tester si on est à la fin du fichier.

Note : voir programme `ecriture` puis `lecture`

Les différents types de fichiers

Les deux familles de fichiers

- les fichiers textes.
- les fichiers binaires.

Les différents types de fichiers

Les fichiers texte

- peuvent être manipulée directement par un éditeur.
- on peut utiliser les fonctions `fprintf` et `fscanf`

Les formats de fichiers

Formattage des fichiers

Lorsqu'un programme lit les informations dans un fichier :

- on veut pouvoir lire (accéder) **rapidement** l'information sans avoir à la traiter.
- par rapidement on entend que l'écriture du programme soit rapide mais aussi que l'exécution le soit.
- on a donc besoin que l'information contenue dans le fichier soit formatée au maximum.

Exemple

Le relevé de notes

- On veut faire un programme qui manipule des notes d'étudiants.
- Chaque étudiant a un fichier à son nom, qui contient
  - Le nom des matières.
  - Des notes.
  - Des commentaires.
- On veut pouvoir stocker toute ces informations après les avoir lues.

```
1 struct matiere_s{
2   char * nom;
3   double  note;
4   char * commentaire;
5 };
6 typedef struct matiere_s matiere_t;
```

Exemple

Le relevé de notes

- On veut faire un programme qui manipule des notes d'étudiants.
- Chaque étudiant a un fichier à son nom, qui contient
  - Le nom des matières.
  - Des notes.
  - Des commentaires.
- On veut pouvoir stocker toute ces informations après les avoir lues.

```
1 struct matiere_s{
2   char * nom;
3   double  note;
4   char * commentaire;
5 };
6 typedef struct matiere_s matiere_t;
```

Un premier fichier

Programmation Impérative 20 Très bien
Logique 15
Algèbre 2 08 étudiant hostile à Bourbaki
ISDL 18

De nombreux problèmes

- Quel espace mémoire réserver pour stocker les chaînes de caractères ?
- Comment savoir que 08 est la note de ”Algèbre 2” ?
- Comment gère t’on les lignes où il n’y a pas de commentaires ?

Un premier fichier

Programmation Impérative 20 Très bien
Logique 15
Algèbre 2 08 étudiant hostile à Bourbaki
ISDL 18

De nombreux problèmes

- Quel espace mémoire réserver pour stocker les chaînes de caractères ?
- Comment savoir que 08 est la note de ”Algèbre 2” ?
- Comment gère t’on les lignes où il n’y a pas de commentaires ?

Solution 1 : les caractères séparateur

```
Programmation Impérative :20 :Très bien :
Logique :15 : :
Algèbre 2 :08 :étudiant hostile à Bourbaki :
ISDL :18 : :
```

Découpage de l'information

- On décide d'un ou plusieurs caractères (ici " :") qui vont délimiter les différents champs.
- Inconvénient : il faut faire le découpage à la main.
- Avantage : ne prends pas excessivement plus de place en mémoire.

Solution 2 : les champs de taille fixe

Programmation Impérative	20	Très bien
Logique	15	
Algèbre 2	08	étudiant hostile à Bourbaki
ISDL	18	

Découpage de l'information

- On décide d'un ou plusieurs caractères (ici " ;") qui vont délimiter les différents champs.
- Inconvénient : le fichier prends plus de place sur le disque dur.
- Avantage : on va voir que c'est beaucoup plus facile à lire

Une fonction de plus

`fread`

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

- Lit dans le fichier `stream` un nombre d'octets *size* $*$ *nmemb*.
- Copie le contenu à l'adresse pointée par `ptr`.

Cette fonction permet de charger et de découper très rapidement le contenu d'un fichier.

Note : voir le programme `lecture_rapide`.

Les en-têtes

- Un en-tête, ou `header` est un ensemble d'informations que l'on place au début du fichier et qui donne des informations sur :
  - le format dans lequel est écrit le fichier.
  - les informations qu'il contient.

Par exemple

- on peut indiquer combien de lignes le fichier contient, ce qui va faciliter l'écriture du programme.
- pour une image ou une vidéo, on précise ses dimensions, le format (bmp,jpeg,gif,png,mpeg,avi).

Solution avec en-tête

4		
Programmation Impérative	20	Très bien
Logique	15	
Algèbre 2	08	étudiant hostile à Bourbaki
ISDL	18	

Découpage de l'information

- La première ligne constitue l'en-tête.
- On sait donc dès le début combien d'espace mémoire il va falloir allouer et comment utiliser l'appel à `fread`.

Note : voir le programme `lecture_propre`.

Conclusion

Pour que les interactions fichiers/programmes soient optimales

- Il faut que le format du fichier soit réfléchi, que la disposition de l'information soit organisée, normalisée.
- On a tout juste entrevue quelques possibilités de format de fichiers.
- il nous reste à voir les fichiers binaires.
- pour cela, il faut d'abord comprendre comment on code l'information en binaire.