Partie C

Programmation Système (débutant)

IUT A

Cours n° C.1

Entrées/Sorties bas niveau

Principes

Rappel du cours de 1re annee (système de gestion de fichiers) :

Au niveau noyau un fichier est une suite non-structurée d'octets (*byte stream*)

Le système permet d'accéder directement à cette suite :

- ☞ Sans formatage
- ☞ Sans conversion

➥ Seuls des **octets** sont lus et écrits

- ☞ Sans tampon d'entrées/sorties au niveau du programme

➥ Les octets sont lus et écrits **directement** et **sans intermédiaire**

➠ **Les accès sont dits de bas niveau**

Accès aux fichiers

Rappel du cours de 1re annee (processus) :

> Tous les processus gère une table stockant le nom des différents fichiers qu'ils utilisent. Chaque index de cette table est appelé un *descripteur de fichiers*.

Pour accéder aux données stockées dans un fichier il faut utiliser des appels systèmes pour :

1. Ouvrir le fichier ..... `open`
2. Manipuler le contenu du fichier ..... `read` ou `write`
3. Fermer le fichier ..... `close`

- **Les fichiers seront repérés par leur descripteur**
- **Tous les fichiers se manipulent de la même manière**
- **Le nombre de descripteurs disponibles pour chaque processus est limité**

Ouverture de fichier (1)

```
int open(const char * pathname, int flags);
```

- ☞ `pathname` → chemin (relatif ou absolu) du fichier à ouvrir
- ☞ `flags` → mode d'accès au fichier (lecture, écriture, etc.)
  - `O_RDONLY`
  - `O_WRONLY`
  - `O_RDWR`
  - `O_CREAT`
  - ...

  } combinable avec l'opérateur «|»

- ➠ retourne le descripteur du fichier ou -1 en cas d'erreur

plus de détails dans le manuel : `open(2)`

Ouverture de fichier (2)

```
int open(const char *pathname, int flags, mode_t mode);
```

- ☞ `pathname` → chemin (relatif ou absolu) du fichier à ouvrir
- ☞ `flags` → mode d'accès au fichier (lecture, écriture, etc.)
- ☞ `mode` → droits d'accès au fichier s'il doit être créé
- ➠ retourne le descripteur du fichier ou -1 en cas d'erreur

➥ Les droits effectifs sont calculés en retirant la valeur du masque (`umask`)

plus de détails dans le manuel : `open(2)`

Exemple

```c
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdio.h>


int main(int argc, char **argv)
{
    int fd;

    fd = open("/tmp/toto", O_WRONLY | O_CREAT, 0666);

    if (fd == -1)
        printf(" Cration du fichier /tmp/toto impossible");
    else
        close(fd);

    return 0;
}
```

Traitement des erreurs

- ☞ Lorsqu'une erreur survient lors d'un appel système la variable globale `errno` est fixée
- ☞ La fonction `void perror(const char * s);` permet d'exploiter la valeur de `errno` pour afficher un message expliquant la dernière erreur système survenue

```
...
if (open("/monfichier", O_WRONLY) == -1)
{
  perror("/monfichier");
  exit(1);
}
...
```

```
bash$ ./essai
/monfichier: No such file or directory
```

À chaque utilisation d'un appel système vous **devrez** traiter le cas d'erreur

Lecture

```
int read(int fd, char * buf, int count);
```

- `fd` → un descripteur d'un fichier ouvert au moins en lecture
- `buf` → l'adresse d'une zone mémoire où les octets lus seront stockés par la fonction. La zone pointée **doit avoir été réservée** pour le programmeur (allocation statique ou dynamique)
- `count` → le nombre d'octets que l'on veut lire
- retourne :
  - -1 si une erreur s'est produite
  - 0 si la fin du fichier est atteinte
  - le nombre d'octets lus sinon

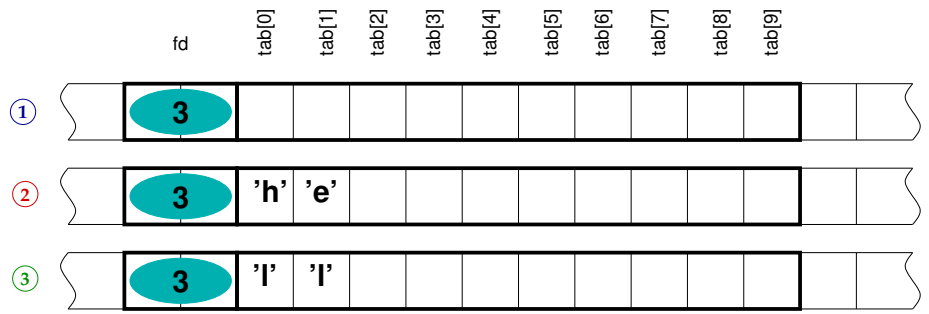
Les lectures se font de manière séquentielle

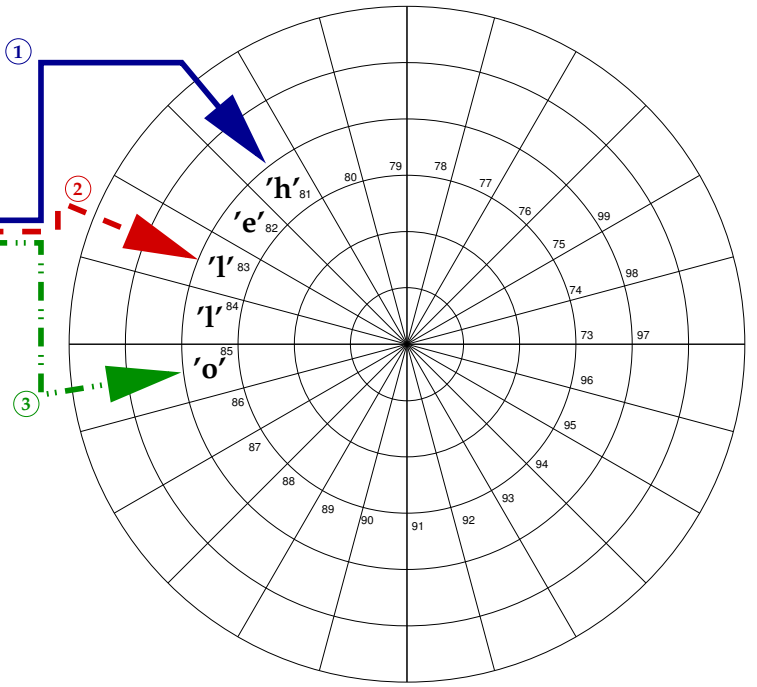
plus de détails dans le manuel : `read(2)`

```
char tab[10];
int fd;

fd = open("unfichier", O_RDONLY);
(1)
read(fd, tab, 2);
(2)
read(fd, tab, 2);
(3)
read(fd, tab, 2);
```

0	/dev/stdin	
1	/dev/stdout	
2	/dev/stderr	
3	unfichier	





Écriture

```
int write(int fd, const char * buf, int count);
```

- `fd` → un descripteur d'un fichier ouvert au moins en écriture
- `buf` → l'adresse d'une zone mémoire où les octets à écrire sont stockés et doivent être lus
- `count` → le nombre d'octets que l'on veut écrire
- retourne :
  - -1 si une erreur s'est produite
  - le nombre d'octets écrits sinon

Les écritures se font de manière séquentielle

plus de détails dans le manuel : `write(2)`

Fermeture

```
int close(int fd);
```

- ☞ `fd` $\rightarrow$ un descripteur d'un fichier ouvert
- ➠ retourne :
  - ☛ -1 si une erreur s'est produite
  - ☛ 0 sinon

plus de détails dans le manuel : `close(2)`

➥ Les fichiers ouverts par un programme sont automatiquement fermés par le système lorsque le processus qui l'exécute se termine

Entrées/Sorties de données quelconques

On peut lire/écrire autre chose que des octets (caractères), en respectant quelques règles :

- ☞ contrôle du formatage des données
- ☞ prendre garde à la taille des données à manipuler
- ☞ attention à **l'alignement** et à **l'endianness** !

```
struct point
{
  int x;
  int y;
};
struct point a;
...
write(f, &a, sizeof(a));
...
read(f, &a, sizeof(struct point));
```