TD 1

Les fichiers

EXERCICE 1 ▶ Documentation de read() et write()

On donne ci-dessous un extrait de la page de manuel de la fonction `write()` :

```
NAME
   write - write to a file descriptor
SYNOPSIS
   #include <unistd.h>
   ssize_t write(int fd, const void *buf, size_t count);
DESCRIPTION
   write() writes up to count bytes from the buffer starting at buf to the file referred to by the file
   descriptor fd.

   The number of bytes written may be less than count if, for example, there is insufficient space on the
   underlying physical medium, [...], or the call was interrupted by a signal handler after having written
   less than count bytes.

   For a seekable file (for example, a regular file) writing takes place at the file offset, and the file
   offset is incremented by the number of bytes actually written. If the file was opened with O_APPEND,
   the file offset is first set to the end of the file before writing. The adjustment of the file offset
   and the write operation are performed as an atomic step. [...]
RETURN VALUE
   On success, the number of bytes written is returned (zero indicates nothing was written). It is not an
   error if this number is smaller than the number of bytes requested; this may happen for example because
   the disk device was filled.

   On error, -1 is returned, and errno is set appropriately.
NOTES
   The types size_t and ssize_t are, respectively, unsigned and signed integer data types specified by POSIX.1.

   If a write() is interrupted by a signal handler before any bytes are written, then the call fails with the
   error EINTR; if it is interrupted after at least one byte has been written, the call succeeds, and returns
   the number of bytes written. [...]
```

Voici un extrait similaire pour la fonction `read()` :

```
NAME
   read - read from a file descriptor
SYNOPSIS
   #include <unistd.h>
   ssize_t read(int fd, void *buf, size_t count);
DESCRIPTION
   read() attempts to read up to count bytes from file descriptor fd into the buffer starting at buf.

   On files that support seeking, the read operation commences at the file offset, and the file offset is
   incremented by the number of bytes read. If the file offset is at or past the end of file, no bytes are
   read, and read() returns zero.

   If count is zero, read() may detect the errors described below. In the absence of any errors, or if read()
   does not check for errors, a read() with a count of 0 returns zero and has no other effects. [...]
RETURN VALUE
   On success, the number of bytes read is returned (zero indicates end of file), and the file position is
   advanced by this number. It is not an error if this number is smaller than the number of bytes requested;
   this may happen for example because [...]

   On error, -1 is returned, and errno is set appropriately.
NOTES
   The types size_t and ssize_t are, respectively, unsigned and signed integer data types specified by POSIX.1.
```

1) Résumez, avec vos propres mots, comment utiliser la fonction `write()` ?

2) Résumez, avec vos propres mots, comment utiliser la fonction `read()` ?

3) On suppose qu'un programme initialise un tableau de caractères déclaré par `char tab[64]`. Comment appeler `write` pour écrire (au plus) tout le contenu de `tab` sur le descripteur de fichier `fd` ? Même question si le tableau a été déclaré par `long tab[64]`. Proposez des solutions avec ou sans `sizeof`.

EXERCICE 2 ▶ Écriture et lecture dans un fichier

Avec `open()`, `write()` et `read()`, on ne se préoccupe pas *a priori* du type des données que l'on stocke dans les fichiers : on écrit et on lit des octets dans les fichiers. Dans cet exercice, on réfléchit à la façon dont différentes données présentes en mémoire peuvent être rangées dans un fichier régulier.

On considère les deux extraits de programmes suivants, Listing 1.1 et Listing 1.2. On suppose que l'on arrive à les compiler (en ajoutant les fichiers d'en-têtes nécessaires) en deux exécutables `read-bin` et `write-bin`, et qu'ils s'exécutent et se terminent en retournant 0.

Listing 1.1 – Le programme write-bin

```
1  int main(void) {
2    int ret, n, fd = open("test.dat", O_CREAT | O_TRUNC | O_WRONLY, S_IRUSR | S_IWUSR);
3    if(fd == -1) return -1;
4
5    char msg[] = "value";
6    n = (strlen(msg) + 1) * sizeof(char);
7    ret = write(fd, msg, n);
8    if(ret != n) return -1;
9
10   unsigned int x = 42;
11   n = sizeof(unsigned int);
12   ret = write(fd, &x, n);
13   if(ret != n) return -1;
14
15   unsigned int tab[3] = {1, 2, 0x12345678};
16   n = 3 * sizeof(unsigned int);
17   ret = write(fd, tab, n);
18   if(ret != n) return -1;
19
20   ret = close(fd);
21   if(ret == -1) return -1;
22
23   return 0;
24 }
```

Listing 1.2 – Le programme read-bin

```
1  int main(void) {
2    int ret, fd = open("test.dat", O_RDONLY);
3    if(fd == -1) return 1;
4
5    do {
6      char c;
7      ret = read(fd, &c, 1);
8      if(ret == 1) printf("%02x ", c); // affiche c en hexadécimal
9    } while(ret == 1);
10   printf("\n"); // affiche un retour à la ligne
11   if(ret == -1) return -1;
12
13   ret = close(fd);
14   if(ret == -1) return -1;
15
16   return 0;
17 }
```

1) En exécutant le programme `read-bin` ou `write-bin` depuis un shell, comment faire pour vérifier qu'il se termine en retournant 0 ? Que nous apprend cette valeur de retour sur les opérations d'écriture et de lecture qui ont été exécutées par le programme ?

2) À la ligne 7 du programme `write-bin`, combien d'octets ont écrits dans le fichier `test.dat` (on attend une explication et une valeur numérique) ?

3) Questions préliminaires : combien de chiffres hexadécimaux sont nécessaires pour représenter un octet ? Quels sont les écritures en hexadécimal des entiers 42, 3, 2 et 1 (écrit en décimal) ?

4) Que fait la boucle des lignes 5 à 9 dans le programme `read-bin` ?

5) Après avoir exécuté `write-bin` pour créer `test.dat`, le programme `read-bin` affiche :

```
76 61 6c 75 65 00 2a 00 00 00 01 00 00 00 02 00 00 00 78 56 34 12
```

— Quel est le code hexadécimal du caractère `'e'` codé en ASCII ?

— Quelle est la taille d'un entier de type `unsigned int` sur ce système ?

— Comment est codé en mémoire l'entier 42 sur ce système ?

EXERCICE 3 ▶ Écritures sur le disque avec write

On a un très grand tableau de `n` caractères en mémoire, et on dispose d'un pointeur de type `char*` vers le premier caractère de ce tableau : on veut écrire ce tableau dans un fichier régulier, *via* un descripteur de fichier `fd`. Donnez une fonction `writech` pour cela, en procédant **caractère par caractère**. Donnez un exemple d'appel de votre fonction.

EXERCICE 4 ▶ Arborescence — Droits sous unix

Sur le système considéré, il y a 4 utilisateurs :

— `fontaine` qui fait partie du groupe `prof` et `user` ;

— `elise` qui fait partie des groupes `etu` et `user` ;

— `hippolyte` qui fait partie du groupe `prof`.

— `root` qui est l'administrateur et fait partie du groupe `root`

```
drwxr-x--x 27      root  user   /bin/
-rwsr-xr-x  1      root   etu   /bin/visioneurPDF
drwxr-xr-x 80      root  user   /home/
drwxr-x--x 10  fontaine  prof   /home/fontaine/
drwx--x---  4  fontaine  prof   /home/fontaine/prive/
-rw-r-x---  1  fontaine  prof   /home/fontaine/sujet.pdf
-rw-r--r--  1 hippolyte  prof   /home/fontaine/prive/correction.pdf
-rw-rw----  1  fontaine  prof   /home/fontaine/prive/notes.ods
```

Attention, pour les fichiers vous devez tenir compte des droits des répertoires et sous répertoires. Comme son nom l'indique, `visioneurPDF` est un logiciel pour visualiser les PDF…

1) Représenter l'arborescence.

2) Quelle commande peut faire `hippolyte` pour avoir une copie du fichier `correction.pdf` à la racine de son compte ? En a-t-il le droit ?

3) Représenter les possibilités d'accès des 4 utilisateurs aux fichiers `visioneurPDF`, `sujet.pdf`, `correction.pdf` et `notes.ods`, par un tableau à double entrée.

4) Élise peut-elle copier le fichier `correction.pdf` ? Peut-elle visualiser ce fichier ?