Module SY5 – Systèmes d'Exploitation

Dominique Poulalhon
dominique.poulalhon@irif.fr

Université de Paris (Diderot)
L3 Informatique & DL Bio-Info, Jap-Info, Math-Info
Année universitaire 2021-2022

Gestion des entrées/sorties (suite)

Lecture et écriture

```
ssize_t read(int fd, void *buf, size_t count);
ssize_t write(int fd, const void *buf, size_t count);
```

- `fd` est un descripteur
- `count` est la taille des données à lire ou écrire
- `buf` est l'adresse d'un emplacement mémoire pour stocker les données lues ou lire les données à écrire

la valeur de retour `nb` ($\leqslant$`count`) est le nombre d'octets effectivement lus ou écrits – ou -1 en cas d'erreur ; voir `errno` dans ce cas !

effet de bord : la position courante (*offset*) de la tête de lecture/écriture avance de `nb` octets

en particulier, un appel à `read` avec un pointeur à la fin d'un fichier ordinaire (ou au delà) renvoie 0

Lecture dans des fichiers

Plus précisément :

```
ssize_t read(int fd, void *buf, size_t count);
```

- renvoie -1 notamment si `fd` n'est pas un descripteur ouvert en lecture (`O_RDONLY` ou `O_RDWR`) ou si l'adresse `buf` est invalide ;
- si la tête de lecture n'a pas atteint la fin du fichier, lit dans le fichier au plus `count` octets (sans dépasser la fin du fichier), les copie à l'adresse `buf` et renvoie le nombre d'octets lus ; l'offset augmente en conséquence ;
- si l'offset est supérieur à la taille du fichier, renvoie 0.

Lecture dans des fichiers

Plus précisément :

```
ssize_t read(int fd, void *buf, size_t count);
```

- renvoie -1 notamment si `fd` n'est pas un descripteur ouvert en lecture (`O_RDONLY` ou `O_RDWR`) ou si l'adresse `buf` est invalide ;
- si la tête de lecture n'a pas atteint la fin du fichier, lit dans le fichier au plus `count` octets (sans dépasser la fin du fichier), les copie à l'adresse `buf` et renvoie le nombre d'octets lus ; l'offset augmente en conséquence ;
- si l'offset est supérieur à la taille du fichier, renvoie 0.

Exemple d'une boucle qui calcule la taille d'un fichier :

```
int taille = 0;
int fd = open("toto", O_RDONLY); /* tête au début du fichier */
/* boucle jusqu'à la fin du fichier */
while ((nb = read(fd, buf, count)) > 0) taille += nb;
```

(attention, ce n'est pas une bonne manière d'obtenir cette information...)

Écriture dans des fichiers

```
ssize_t write(int fd, void *buf, size_t count);
```

- renvoie -1 notamment si `fd` n'est pas un descripteur ouvert en écriture (`O_WRONLY` ou `O_RDWR`) ou si l'adresse `buf` est invalide ;
- si `fd` est ouvert en `O_APPEND`, la tête est déplacée en fin de fichier ;
- (au plus) `count` octets lus à l'adresse `buf` sont copiés à partir de la position de la tête ; l'offset augmente en conséquence ;
- la valeur renvoyée est le nombre d'octets correctement écrits ; si elle est strictement inférieure à `count`, cela signifie qu'il y a eu une erreur (disque plein par exemple).

Écriture dans des fichiers

```
ssize_t write(int fd, void *buf, size_t count);
```

- renvoie -1 notamment si `fd` n'est pas un descripteur ouvert en écriture (`O_WRONLY` ou `O_RDWR`) ou si l'adresse `buf` est invalide ;
- si `fd` est ouvert en `O_APPEND`, la tête est déplacée en fin de fichier ;
- (au plus) `count` octets lus à l'adresse `buf` sont copiés à partir de la position de la tête ; l'offset augmente en conséquence ;
- la valeur renvoyée est le nombre d'octets correctement écrits ; si elle est strictement inférieure à `count`, cela signifie qu'il y a eu une erreur (disque plein par exemple).

Attention au paramètre `count` *!* il doit correspondre à la quantité de données qu'on souhaite réellement copier, qui n'est pas nécessairement la taille du buffer utilisé ; s'il a été rempli par une lecture `nb = read(fd, buf, size)`, le nombre d'octets pertinents est `nb`, qui vaut au plus `size`, mais peut être strictement inférieur.

Déplacement de la tête de lecture/écriture

- `open` positionne la tête au début du fichier (offset égal à 0) ;
- chaque lecture ou écriture entraîne un déplacement de cette tête ;
- en mode `O_RDWR`, la même tête sert pour les lectures et les écritures.

Exemple (sans gestion des erreurs) :

```
int fd, nb;
char buf[3];
fd = open("toto", O_WRONLY | O_CREAT | O_TRUNC, 0600);
write(fd, "abcdefghi", 9);
close(fd);
fd = open("toto", O_RDWR);
nb = read(fd, buf, 3); write(1, buf, nb);
write(fd, "DEF", 3);
nb = read(fd, buf, 3); write(1, buf, nb);
close(fd);
```

Déplacement de la tête de lecture/écriture

pour les fichiers ordinaires, les lectures/écritures ne sont pas nécessairement séquentielles ; il est possible de changer de position courante :

```
off_t lseek(int fd, off_t offset, int whence);
```

- `fd` est un descripteur
- `whence` est une position de référence (`SEEK_SET`, `SEEK_CUR`, `SEEK_END`)
- `offset` est un décalage par rapport à cette position de référence

la valeur de retour est la nouvelle position courante, ou -1 en cas d'erreur

Déplacement de la tête de lecture/écriture

pour les fichiers ordinaires, les lectures/écritures ne sont pas nécessairement séquentielles ; il est possible de changer de position courante :

```
off_t lseek(int fd, off_t offset, int whence);
```

- `fd` est un descripteur
- `whence` est une position de référence (`SEEK_SET`, `SEEK_CUR`, `SEEK_END`)
- `offset` est un décalage par rapport à cette position de référence

la valeur de retour est la nouvelle position courante, ou -1 en cas d'erreur

(ce qui permet de manière indirecte de connaître la position courante d'une ouverture grâce à l'appel `lseek(fd, 0, SEEK_CUR)`, ou la taille du fichier par `lseek(fd, 0, SEEK_END)`)

Où et comment stocker les fichiers ?

un fichier, c'est...

- du contenu
- des attributs ou méta-données (type, permissions, dates...)

plusieurs solutions :

- stockage contigu (CD-ROM)

Où et comment stocker les fichiers ?

un fichier, c'est...

- du contenu
- des attributs ou méta-données (type, permissions, dates...)

plusieurs solutions :

- stockage contigu (CD-ROM)
- liste chaînée de blocs

Où et comment stocker les fichiers ?

un fichier, c'est...

- du contenu
- des attributs ou méta-données (type, permissions, dates...)

plusieurs solutions :

- stockage contigu (CD-ROM)
- liste chaînée de blocs
- liste chaînée disjointe des blocs eux-mêmes (*File Allocation Table*)

Où et comment stocker les fichiers ?

un fichier, c'est...

- du contenu
- des attributs ou méta-données (type, permissions, dates...)

plusieurs solutions :

- stockage contigu (CD-ROM)
- liste chaînée de blocs
- liste chaînée disjointe des blocs eux-mêmes (*File Allocation Table*)
- table d'i-nœuds, regroupant les attributs et les adresses des blocs

Consultation des i-nœuds

```
int stat(const char *pathname, struct stat *statbuf);
int fstat(int fd, struct stat *statbuf);
```

remplissent une `struct stat` avec les caractéristiques de l'i-nœud et retournent 0, ou -1 en cas d'erreur, précisée par `errno` (`ENOENT` ou `EACCESS` par exemple)

le type `struct stat` contient (entre autres) les champs suivants :

```
struct stat {
  dev_t     st_dev;      /* ID of device containing file */
  ino_t     st_ino;      /* Inode number */
  mode_t    st_mode;     /* File type and mode */
  uid_t     st_uid;      /* User ID of owner */
  off_t     st_size;     /* Total size, in bytes */
  blksize_t st_blksize;  /* Block size for filesystem I/O */
  blkcnt_t st_blocks;    /* Number of 512B blocks allocated */
  /* ... */
};
```

Modification des i-nœuds

```
int chmod(const char *path, mode_t mode);
int fchmod(int fd, mode_t mode);

int chown(const char *path, uid_t owner, gid_t group);
int fchown(int fildes, uid_t owner, gid_t group);

int utimes(const char *path, const struct timeval times[2]);
int futimes(int fildes, const struct timeval times[2]);

int truncate(const char *path, off_t length);
int ftruncate(int fd, off_t length);
```