

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 2022-2023

Modification d'emploi du temps pour le *groupe 2* :

- TD/TP de Systèmes d'exploitation le lundi de 8h30 à 10h30 (salles 2031-2036)
- TP de Programmation fonctionnelle le lundi de 10h45 à 12h45 (salles 2031-2032)

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` ($\leq \text{count}$) 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)`)

ET LA BIBLIOTHÈQUE STANDARD ?

elle définit également des fonctions d'entrée-sortie, manipulant des **flots** de type **FILE *** : c'est une **surcouche** au-dessus des descripteurs et des appels système les manipulant

un **FILE** contient :

- un **descripteur** (donc un accès à une entrée de la table des ouvertures de fichiers, avec entre autres un mode d'ouverture et une tête de lecture/écriture)
- un **tampon** pour limiter le nombre d'appels système **read** et **write**
- les données nécessaires pour le manipuler : sa capacité, le nombre de caractères présents, et un pointeur vers la position courante dans ce tampon

DUPLICATION DE DESCRIPTEUR ET REDIRECTION

il est possible de définir un *synonyme* `fd2` d'un descripteur `fd1`, c'est-à-dire de faire en sorte que `fd2` pointe sur la même ligne de la table des ouvertures de fichiers que `fd1` :

```
int dup2(int fd1, int fd2);  
int dup(int fd1); /* variante dans laquelle fd2 est choisi par le  
système parmi les descripteurs non alloués */
```

- `fd1` doit être un descripteur valide
- `fd2` est un descripteur quelconque, alloué ou non

si `fd2` est déjà alloué, le système le ferme (comme un `close(fd2)`) avant la duplication

la valeur de retour est `fd2`, ou -1 en cas d'échec

en particulier, `fd1` et `fd2` partagent alors la même position courante (*offset*)

ATOMICITÉ

problématique omniprésente : lorsque deux actions sont nécessaires, comment s'assurer que rien ne s'est produit entre les deux qui annihile les effets de la 1^{re} ?

certaines appels système offrent cette garantie d'atomicité, i.e. de non interruption

ouverture en `O_CREAT` | `O_EXCL`

vs test puis création (en deux appels système)

ouverture en `O_APPEND` \implies déplacement en fin de fichier avant chaque écriture de manière atomique

vs `lseek` puis `write`

duplication avec `dup2`

vs `close` puis `dup`