Module SY5 – Systèmes d'Exploitation

Dominique Poulalhon
dominique.poulalhon@irif.fr

Université Paris Cité
L3 Informatique & DL Bio-Info, Jap-Info, Math-Info
Année universitaire 2022-2023

Gestion des entrées/sorties

Ouverture et fermeture de fichiers

l'accès à un fichier est une action critique $\Longrightarrow$ appel système

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

que fait cet appel ?

- il teste si `pathname` est une référence valide et accessible
  dans le cas contraire, il renvoie -1 avec positionnement de la variable `errno` selon les valeurs indiquées dans `man 2 open`
- il met en place (en mémoire) les structures nécessaires pour accéder simplement au contenu du fichier
  le descripteur renvoyé est le point d'accès du processus à ces structures

Ouverture et fermeture de fichiers

l'accès à un fichier est une action critique $\Longrightarrow$ appel système

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

que fait cet appel ?

- il teste si `pathname` est une référence valide et accessible
  dans le cas contraire, il renvoie -1 avec positionnement de la variable `errno` selon les valeurs indiquées dans `man 2 open`
- il met en place (en mémoire) les structures nécessaires pour accéder simplement au contenu du fichier
  le descripteur renvoyé est le point d'accès du processus à ces structures

- table des descripteurs *du processus*
- table des fichiers ouverts *du système*
- table des i-nœuds *du système* (en mémoire)

Ouverture et fermeture de fichiers

l'accès à un fichier est une action critique $\Longrightarrow$ appel système

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

- table des descripteurs *du processus*
- table des fichiers ouverts *du système*
- table des i-nœuds *du système* (en mémoire)

pour libérer les ressources correspondantes :

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

Ouverture et fermeture de fichiers

vous connaissez déjà des descripteurs :

- 0 est le descripteur associé à l'entrée standard ;
- 1 est le descripteur associé à la sortie standard ;
- 2 est le descripteur associé à la sortie erreur standard.

ils sont (en général) *hérités* du processus père et ne nécessitent pas d'ouverture ; nous verrons plus tard comment *changer* les fichiers ouverts associés à ces descripteurs

note : ces trois descripteurs sont aussi définis par des macros dans `unistd.h` : `STDIN_FILENO`, `STDOUT_FILENO` et `STDERR_FILENO`.

Lecture et écriture dans des fichiers

```
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

. *(les types `size_t` et `ssize_t` sont des entiers respectivement non signés et signés pour POSIX.1. Ils servent essentiellement à conserver la compatibilité entre les différentes versions de POSIX)*

Lecture et écriture dans des fichiers

```
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 !

. *(les types `size_t` et `ssize_t` sont des entiers respectivement non signés et signés pour POSIX.1. Ils servent essentiellement à conserver la compatibilité entre les différentes versions de POSIX)*

Lecture et écriture dans des fichiers

```
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

. *(les types `size_t` et `ssize_t` sont des entiers respectivement non signés et signés pour POSIX.1. Ils servent essentiellement à conserver la compatibilité entre les différentes versions de POSIX)*

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.

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 (ainsi éventuellement que la taille du fichier) ;
- 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 (ainsi éventuellement que la taille du fichier) ;
- 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)`)