# 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 2023-2024

# 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

$\Longrightarrow$ accès/modification à plusieurs niveaux :

- table des descripteurs *du processus*
- table des ouvertures de fichiers (*open files*) *du système*
- table des i-nœuds virtuels *du système* (en mémoire)
- tables des i-nœuds physiques (sur disque)

# 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*/);
```

$\Longrightarrow$ accès/modification à plusieurs niveaux :

- table des descripteurs *du processus*
- table des ouvertures de fichiers (*open files*) *du système*
- table des i-nœuds virtuels *du système* (en mémoire)
- tables des i-nœuds physiques (sur disque)

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`.

# Les différents modes d'ouverture

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

les `flags` indiquent le mode d'ouverture souhaité

un flag obligatoire pour le type d'accès : `O_RDONLY`, `O_WRONLY` ou `O_RDWR`

éventuellement combiné (par des « OU » bit-à-bit, `|`) avec d'autres flags tels que `O_CREAT`, `O_APPEND`, `O_EXCL`...

cas particulier de `O_CREAT` : `open` attend dans ce cas un 3e paramètre décrivant les droits d'accès à donner au fichier éventuellement créé

la réussite de l'ouverture dépend du mode d'ouverture demandé, de l'existence préalable du fichier, des droits d'accès au fichier (s'il existe)... consulter la variable `errno` en cas d'échec

# 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)*