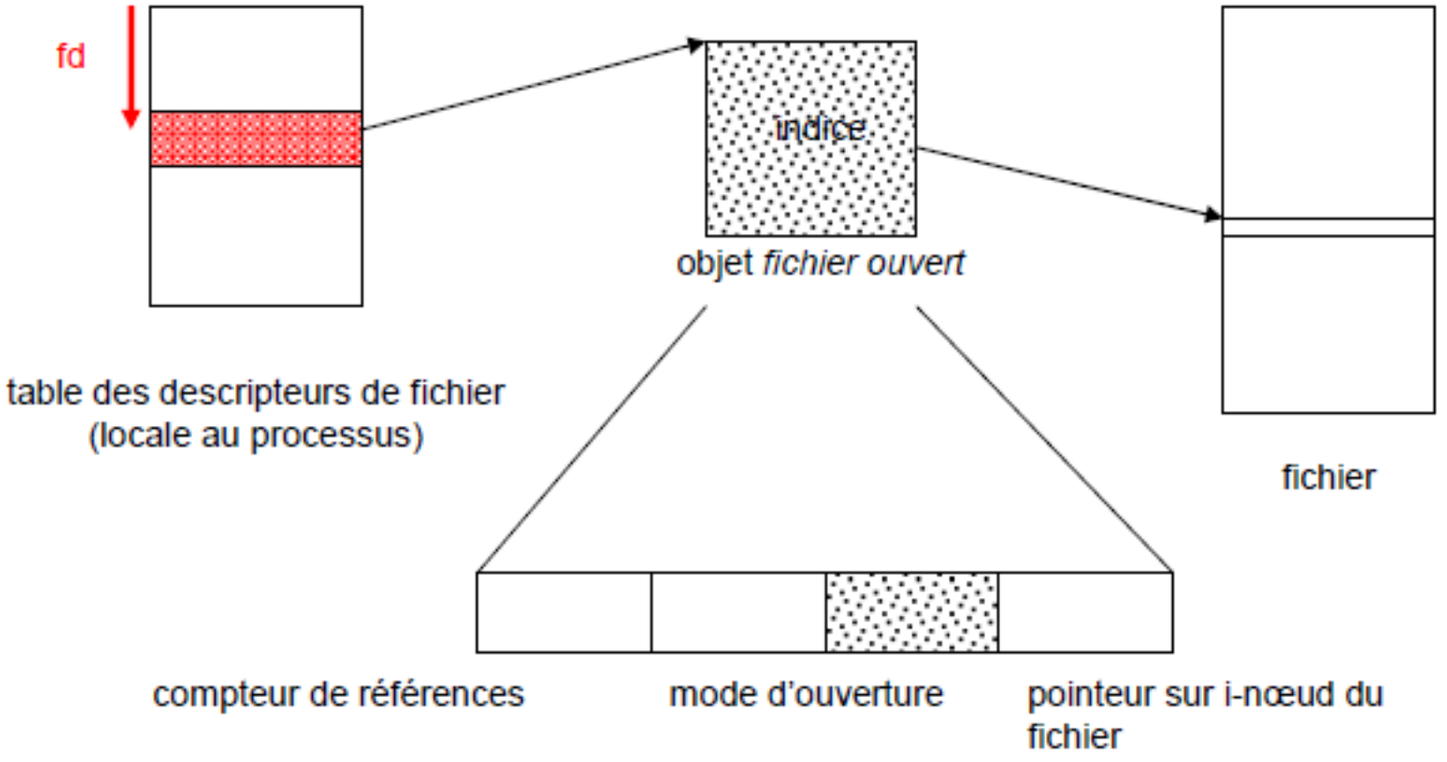
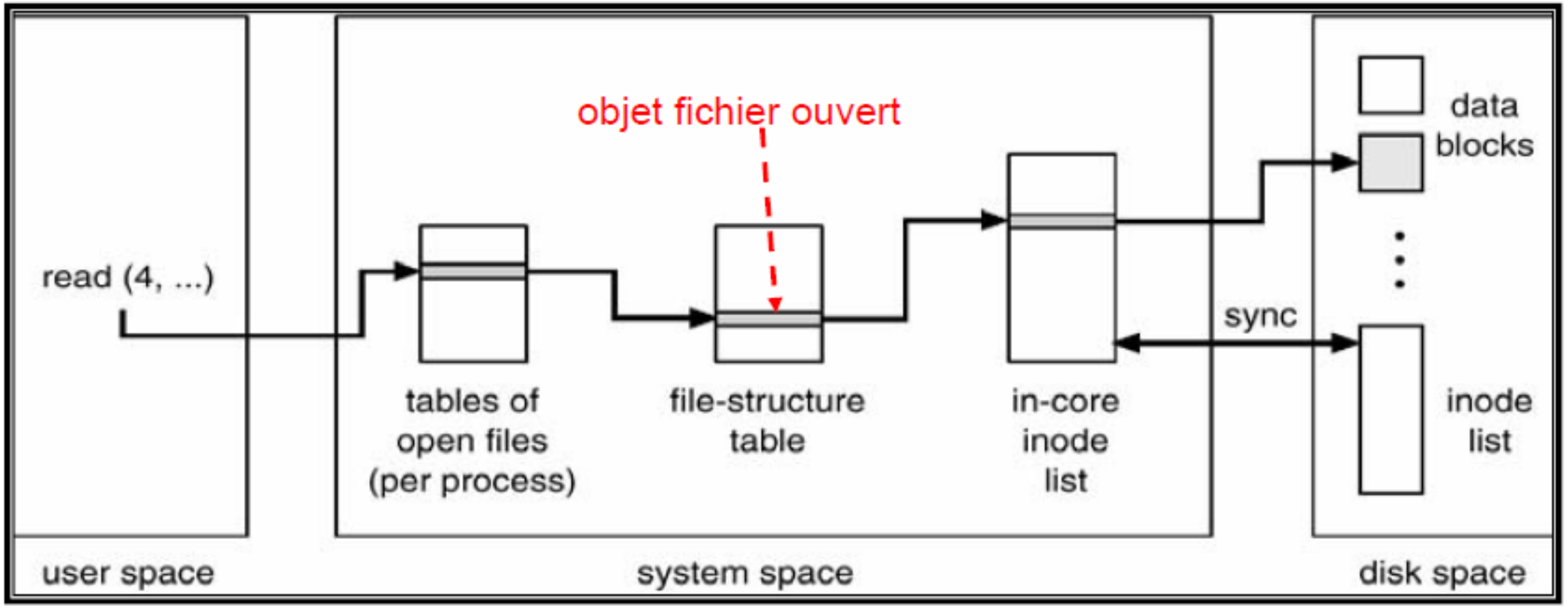
Fichier: séquence d'octets.

- `fd = open (....)` : ouverture d'un fichier
- `open` crée un objet *fichier ouvert*, qui représente une instance de fichier ouvert. Contient les infos suivantes:
  - mode d'accès (lecture, lecture/écriture);
  - indice dans le fichier pour prochaine lecture/écriture (si ouvert en écriture);
- `open` retourne un descripteur de fichier (indice dans la table des descripteurs de fichiers du processus).

Ouverture d'un fichier par un utilisateur

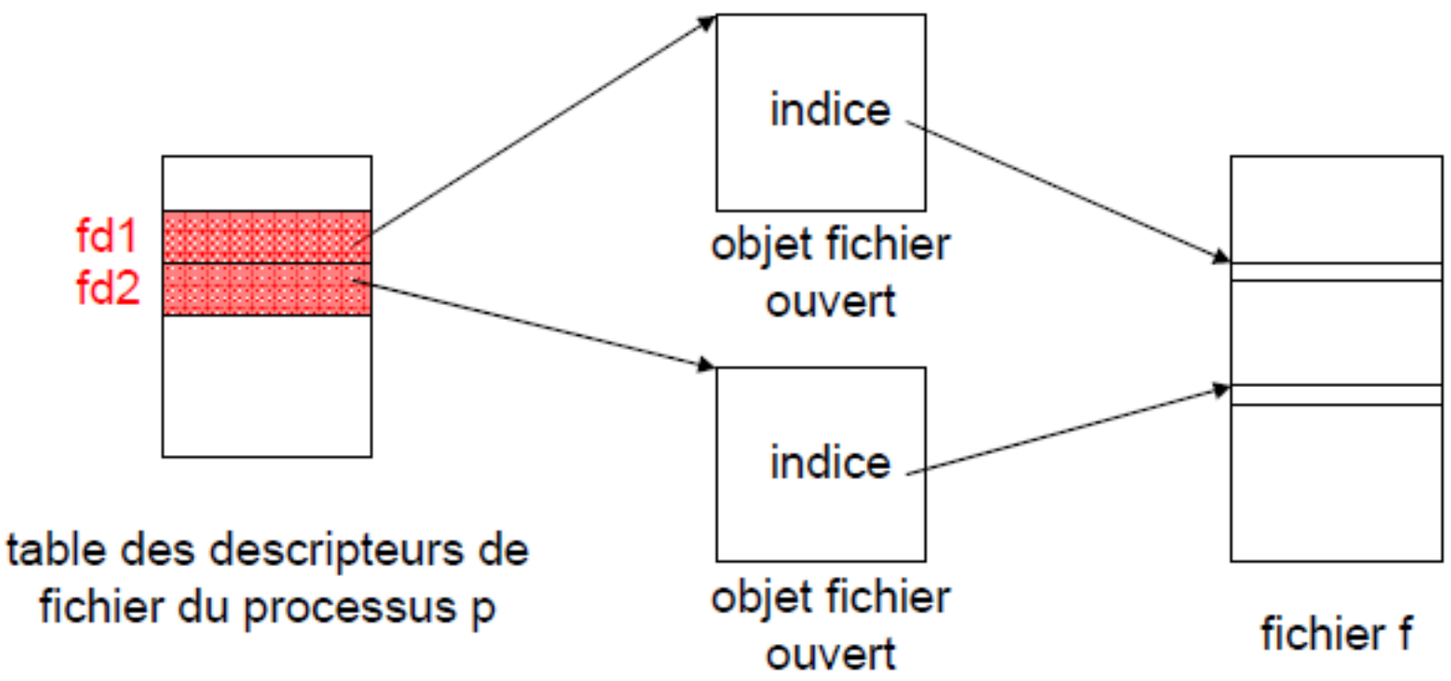


Accès au contenu du fichier

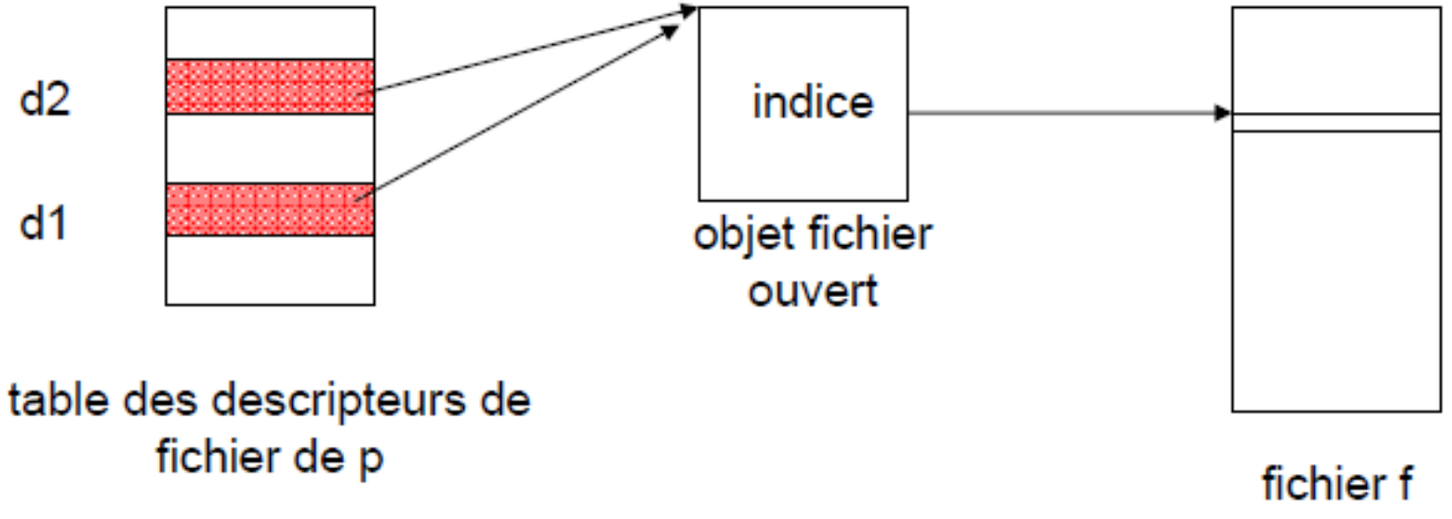


Silberschatz, Galvin and Gagne ©2002

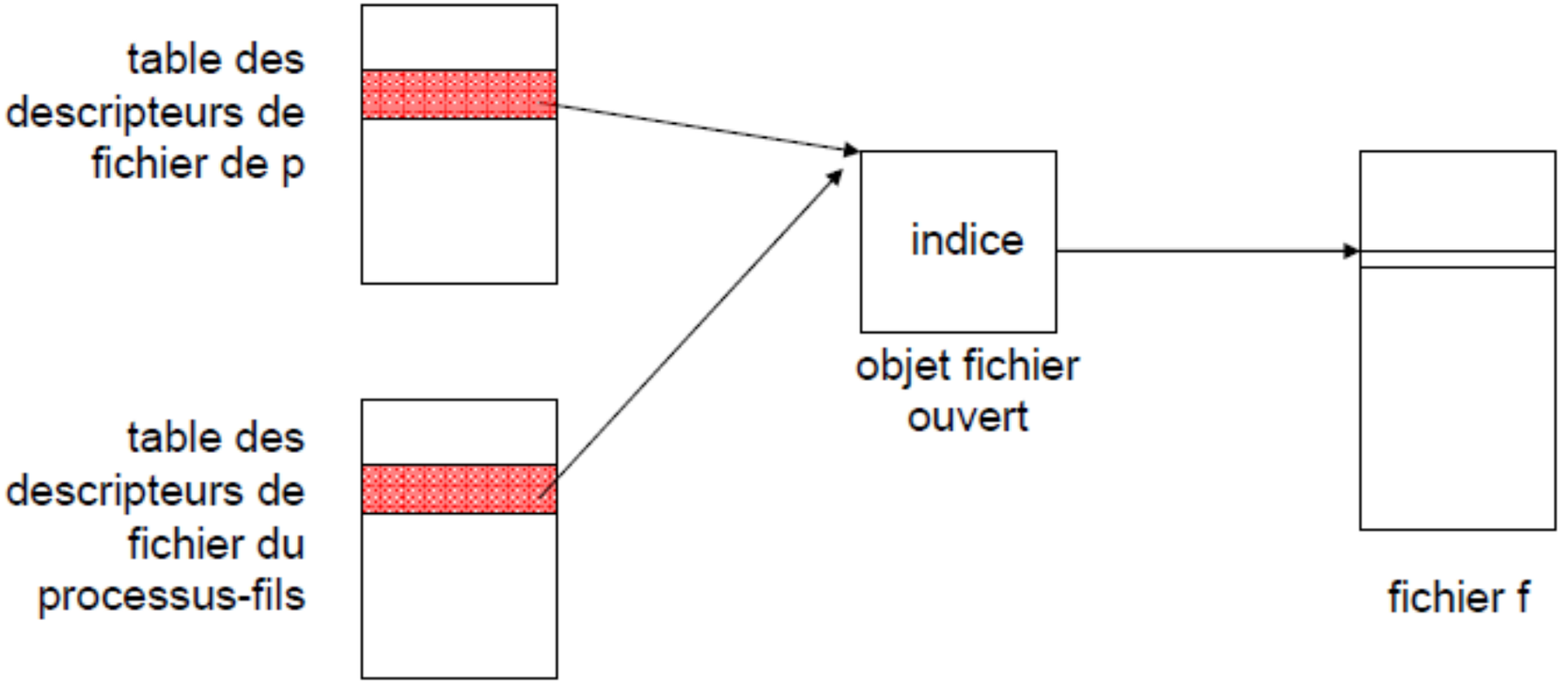
- **Exemple 1**: un processus p ouvre le fichier f deux fois:
  - fd1 = open(f, ...)
  - fd2 = open(f, ...)



- **Exemple 2**: un processus p ouvre le fichier f puis exécute l'appel dup2: `int dup2(int d1, int d2);` (provoque implicitement `close(d2)` si nécessaire)



- **Exemple 3**: un processus p ouvre le fichier f puis exécute l'appel fork:



Organisation d'un disque:

- bloc de bootstrap
- super bloc (contient des informations permettant de gérer l'espace disque)
- liste de i-noeuds, un i-noeud par fichier (i = index)
- blocs de données (contenu des fichiers)

Format du super-bloc

- taille (en blocs) du système de fichiers
- taille (en blocs) de la liste des i-noeuds
- liste de blocs libres
- liste de i-noeuds libres

Format d'un i-noeud sur disque

- type du fichier (normal, répertoire,…)
- droits
- nombre de liens "durs" (hard links) sur le fichier
- UID du propriétaire
- GID du propriétaire
- taille du fichier (nb d'octets)
- adresses des blocs du fichier, sur 13 entrées
- numéro d'incarnation (incrémenté chaque fois que le i-noeud est réutilisé pour un autre fichier)
- heure du dernier accès au fichier
- heure de la dernière modification au fichier
- heure de la dernière modification au i-noeud (autre que *heure accès* / *heure modification*)

Adressage du fichier dans un i-nœud

- les blocs 0 à 9 sont adressés directement
- les 256 blocs suivants sont adressés avec une indirection
- les 65'536 blocs suivants sont adressés avec une double indirection
- les 16'777'216 blocs suivants sont adressés avec une triple indirection

Permet d'accéder efficacement à des fichiers de petite taille (la plupart des fichiers sont petits).

Entrées-sorties standards
Bibliothèque ANSI C

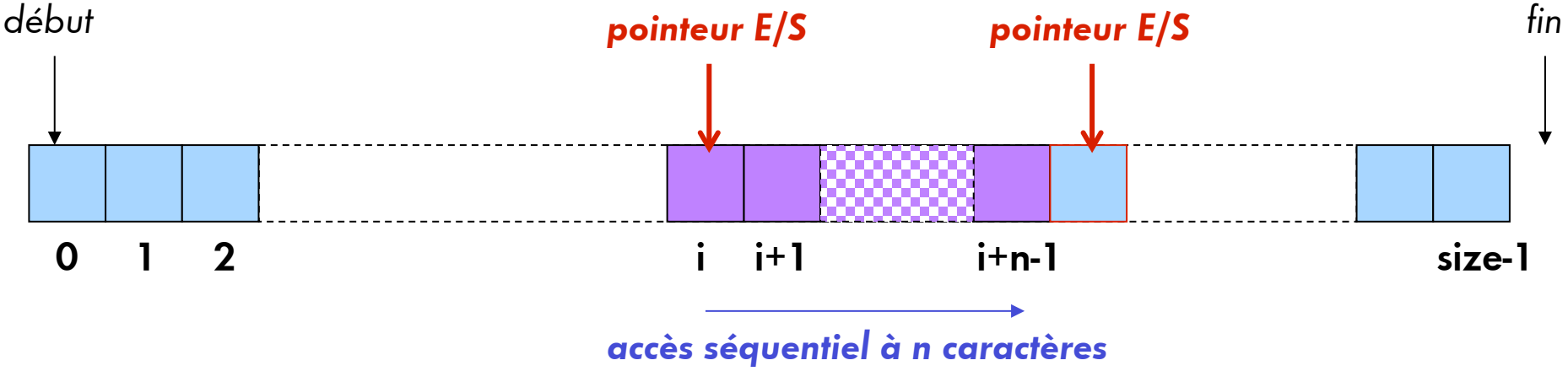
- **POSIX.1 contient l'ensemble de la bibliothèque d'entrée-sortie standard de C (`<stdio.h>`)**
- **Les E/S on lieu avec « bufferisation »**
  - sauf si le support est un terminal
  - la fonction **`fflush(FILE *)`** vide le buffer
- **En fait ces fonctions d'E/S sont réalisées avec des fonctions primitives qui constituent l'API d'E/S de base de POSIX**

Entrées-sorties de base
Descripteurs de fichiers

- Un « file descriptor », fd, est un numéro d'unité logique qui permet de référencer un fichier
- fd est un entier positif ou nul
  - 0, 1 et 2 correspondent respectivement à l'entrée, la sortie et l'erreur standards
- Les « file descriptors » sont alloués par processus
- Les fonctions qui retournent un descripteur de fichier (e.g., open) retournent en général la plus petite valeur disponible dans le processus courant

Entrées-sorties de base
Modèle de base des fichiers

- **Tableau de caractères**
  - indexé à partir de 0
- **Pointeur d'E/S**
  - index courant
  - manipulable directement (accès direct)
- **Lecture/écriture séquentielle**
  - à partir de l'index courant



Entrées-sorties de base
Ouverture et création de fichiers *(1)*

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

int open(const char *path, int oflag
         [, mode_t mode]);
```

- Flag **`oflag`**

`O_RDONLY, O_WRONLY, O_RDWR`	mode d'ouverture
`O_APPEND`	place le pointeur d'E/S en fin de fichier
`O_CREAT`	crée le fichier s'il n'existe pas
`O_EXCL`	si **`O_CREAT`** et si le fichier existe, erreur
`O_TRUNC`	tronquer le fichier s'il existe

etc.

Entrées-sorties de base
Ouverture et création de fichiers (2)

- **Troisième argument de open (mode)**
  - utilisé seulement si **O_CREAT**
  - positionne les droits d'accès au nouveau fichier
    - filtrage par le **UMASK**
- **Exemple**

```
if (open("foo", O_WRONLY|O_CREAT|O_TRUNC,
         S_IRUSR|S_IWUSR|S_IRGRP) < 0) {
  perror("open");
}
```

Entrées-sorties de base
Masque de création de fichiers (cmask)

```
#include <sys/types.h>
#include <sys/stat.h>
mode_t umask(mode_t mode);
```

- **mode** ne doit contenir que des droits d'accès
- la fonction retourne la valeur précédente du masque
- mode contient les droits que l'on veut ***dénier***
  après **`fd = open("foo",…|O_CREAT, m);`**
  le mode est : **`m & ~cmask`**
- le masque est un attribut du processus

❒ **Exemple**

```
oldmask = umask(S_IWGRP|S_IWOTHR);
```

Entrées-sorties de base
Lecture/écriture séquentielles *(1)*

```
#include <unistd.h>
ssize_t read(int fd, void *buf, size_t n);
ssize_t write(int fd, void *buf, size_t n);
```

- **E/S séquentielles**
  - Le pointeur d 'E/S est avancé du nombre de caractères effectivement transférés
  - Par défaut, **`read()`** est bloquant ; **`write()`** peut l'être
- **Valeur de retour**
  - Nombre de caractères effectivement transférés
  - Pour **`read()`**, valeur de retour 0 ⇒ fin de fichier

Entrées-sorties de base
Lecture/écriture séquentielles (2)

- Exemple : une version rustique et partielle de copie de fichiers

```
char buffer[MAX];
int n;
int fdin = open("foo", O_RDONLY|O_EXCL);
int fdout =
  open("bar", O_WRONLY|O_TRUNC|O_CREAT, …);

while((n = read(fdin, buffer, MAX)) != 0)
    write(fdout, buffer, n);
```