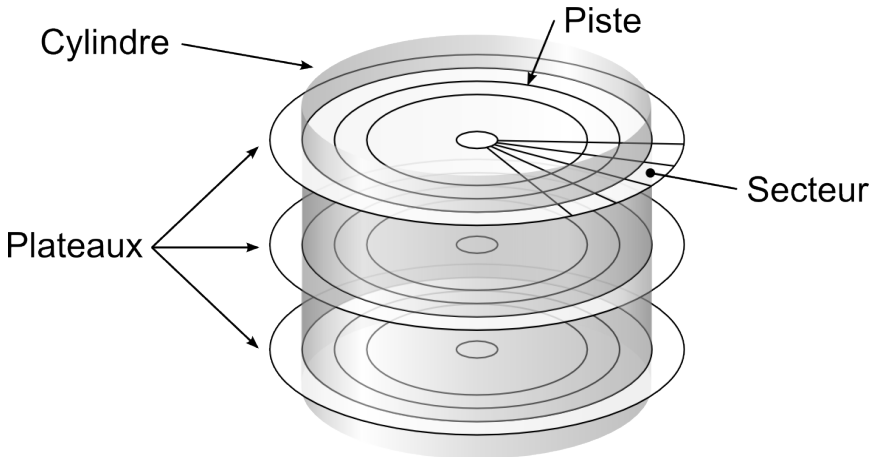
Chapitre 2

Système de Gestion de Fichiers (SGF)

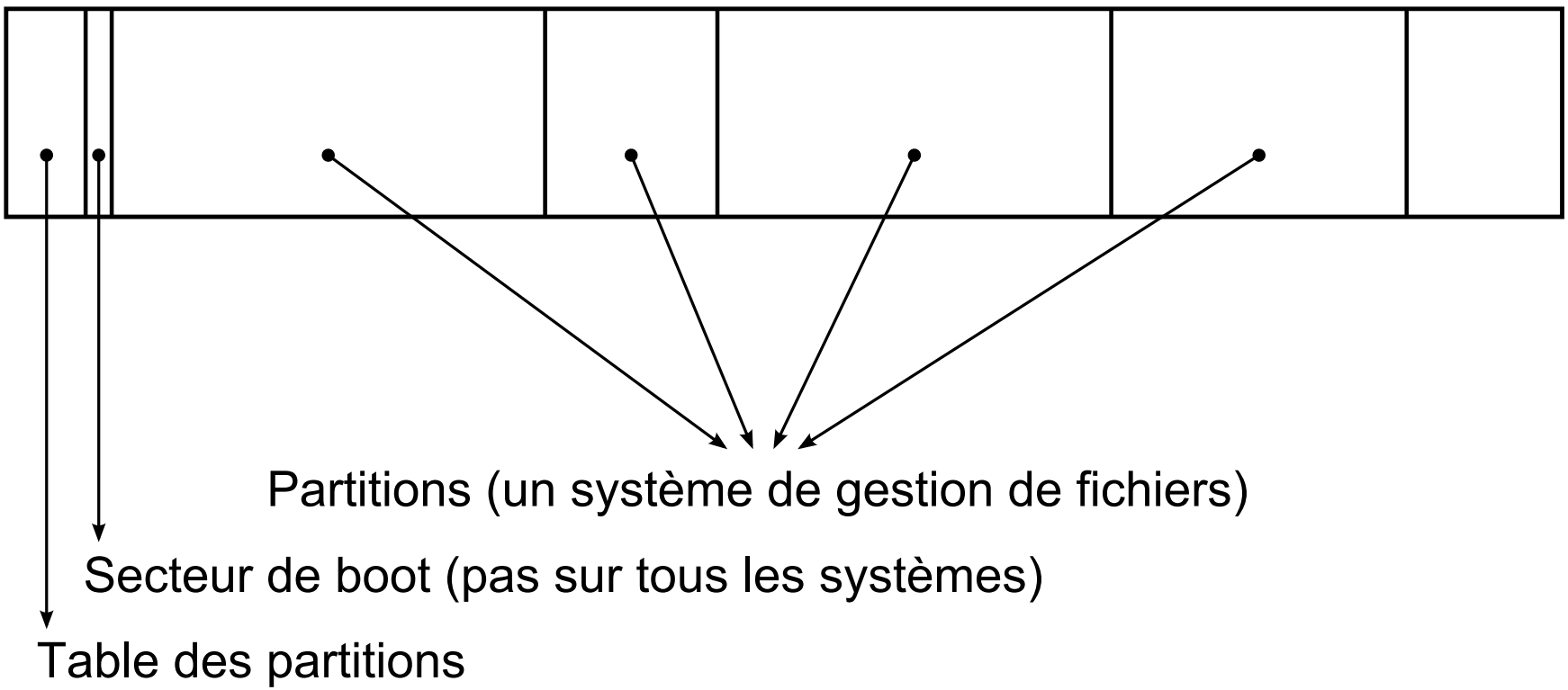
Géométrie d'un disque dur

- Disque physique :
  - plusieurs plateaux, 2 faces par plateau
  - découpage d'une face en pistes puis secteurs



- Linéarisation d'un disque (CHS $\rightarrow$ LBA (n_l)) :
  - Espace total : $\#_{faces} * \#_{cylindres} * \#_{secteurs/piste}$ secteurs
  - $n_l = n_s + n_c * \#_{secteurs/p} + n_f * \#_{cylindres} * \#_{secteurs/p}$
  - Retrouver coordonnées physiques :
    - $n_s = n_l \ \% \ \#_{secteurs/p}$
    - $n_c = (n_l \div \#_{secteurs/p}) \ \% \ \#_{cylindres}$
    - $n_f = (n_l \div (\#_{secteurs/p} * \#_{cylindres})$

Organisation du disque dur

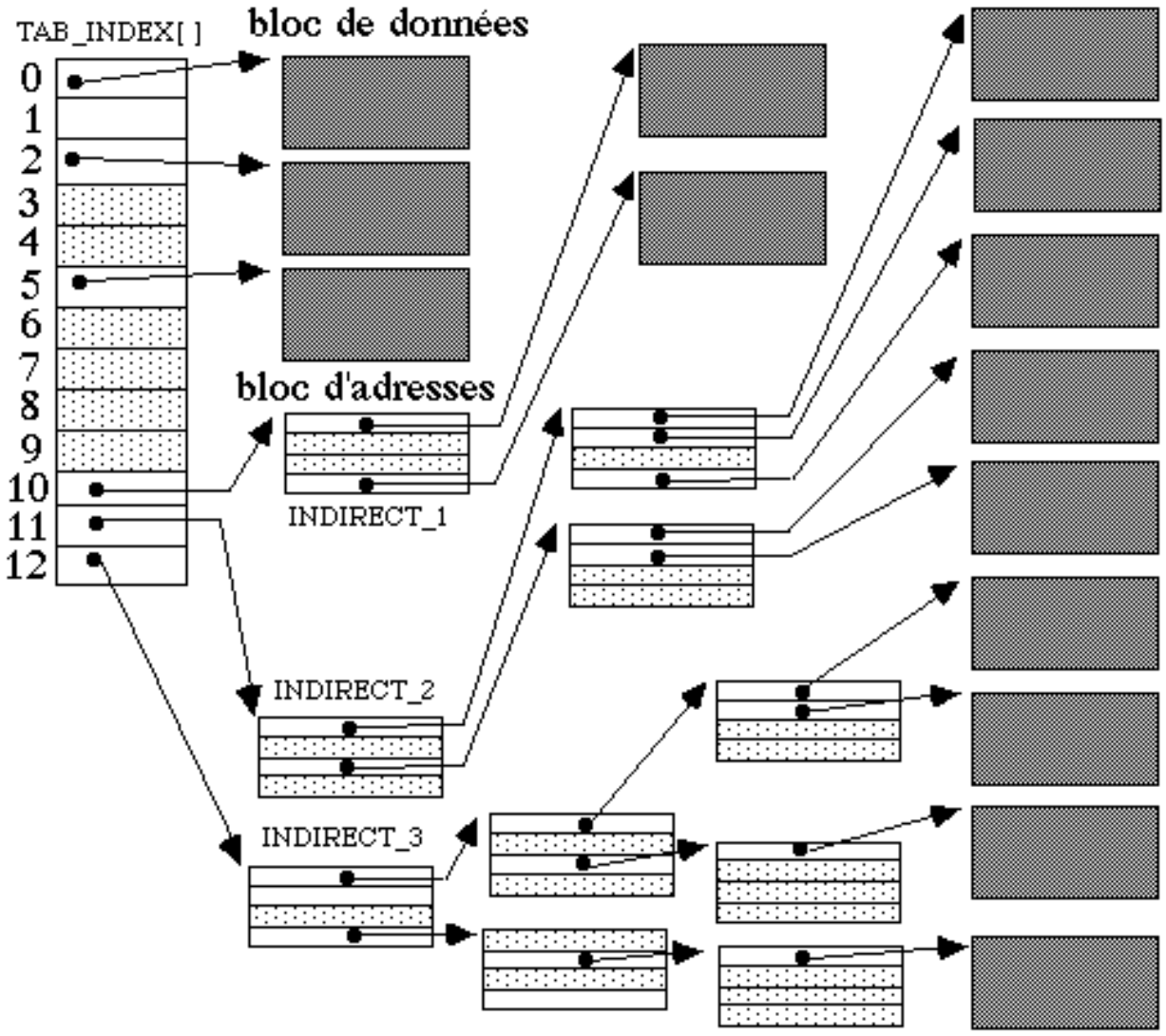


Le SGF (Système de Gestion de Fichiers)

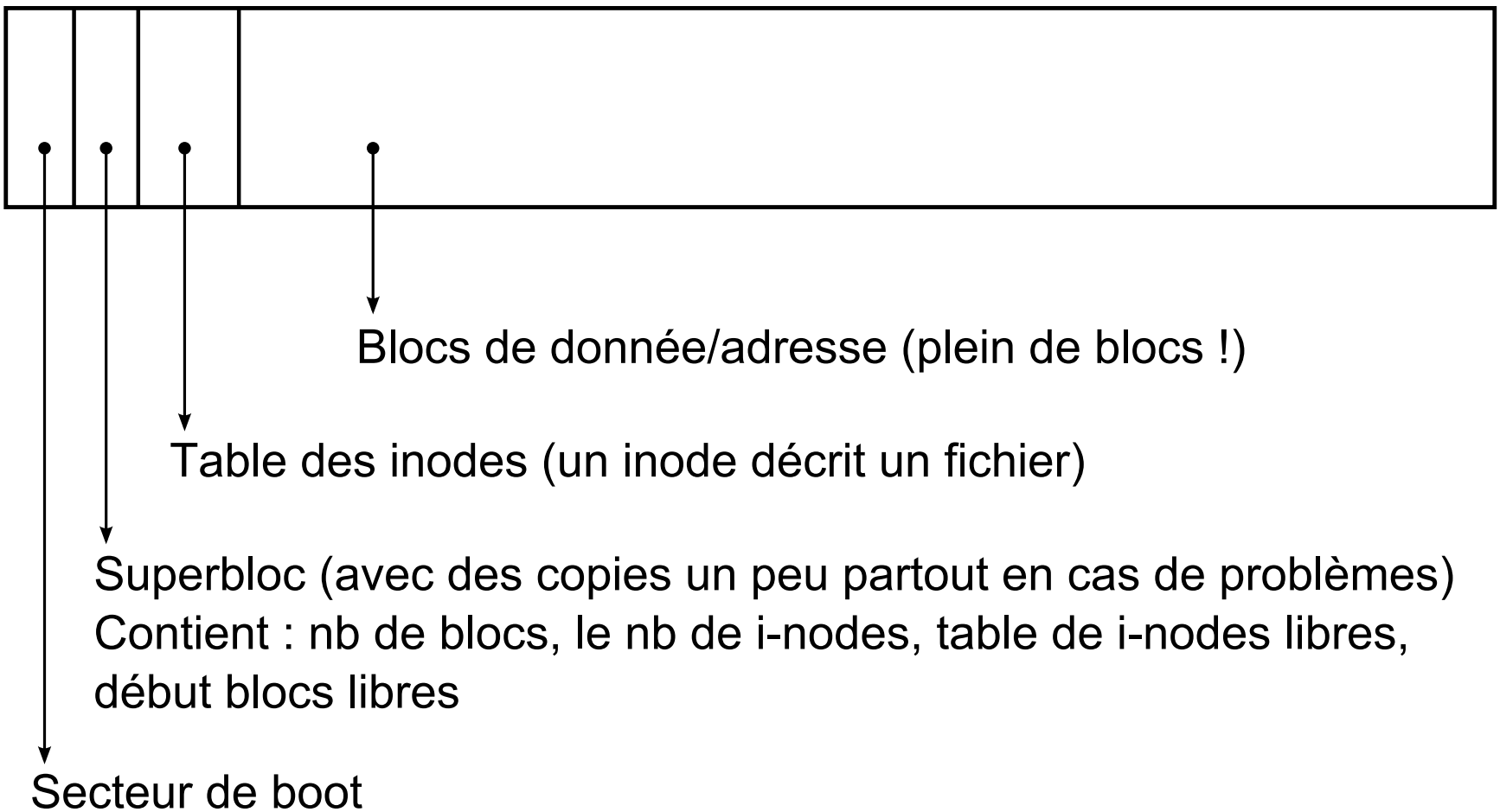
- But : masquer l'aspect « secteur » pour ne manipuler que des fichiers
- Les fichiers sont fragmentés sur le disque
- Le plus simple : FAT Dos
  - FAT : File Allocation Table
  - Grand tableau indiquant pour chaque bloc quel est son successeur ou si libre/défectueux/EOF
  - Inefficace :
    - Trop accès FAT (horreur si accés séquentiel)
    - FAT en mémoire (gros si gros disque)

SGF Unix

- Blocs de données/blocs d'adresses
- Permet stockage gros fichiers
- Pas de FAT en mémoire



Structure d'une partition



Association partition/répertoire

- Notion de montage
- Sens plus large que partition
  - Systèmes distants (NFS, . . .)
  - Volume logique (union de partitions)
  - Montage associé à un driver (noyau)
    - `/proc` (proc)
    - `/proc/bus/usb` (usbdevfs)

Un inode (index node)

- Décrit et identifie un fichier (on ne manipule plus des noms de fichier)
  - gid/uid, type, droits, dates (amc), nb hard link, taille
  - Pointeur sur bloc de données
- Association nom/inode
  - Chaque enregistrement
    - Numéro d'inode
    - Nom relatif
  - Taille des noms variable ($<$ 256 sous Linux)

Liens *durs*

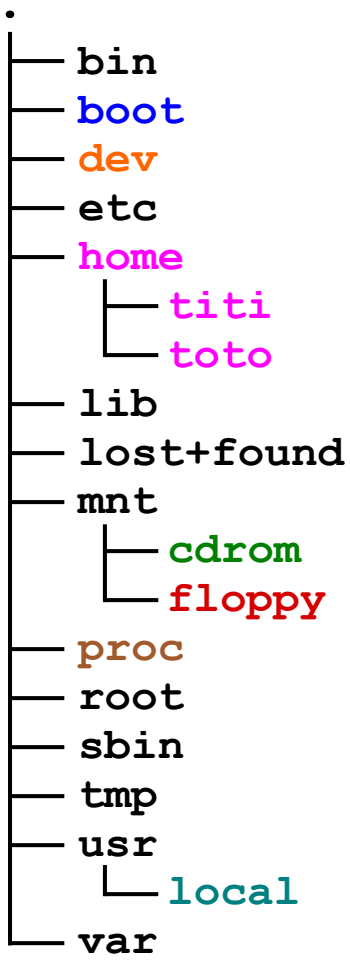
- Lien dur
  - Deux entrées dans la table nom/inode avec le même inode et des noms différents
  - La source et la destination doivent être sur la même partition
- Créer un lien dur en shell
  - `ln` **sans** l'option `-s`
  - `ln sourceExistante nomDuLienDur`
- Supprimer un fichier = décrémenter le compteur de liens durs
  - Suppression effective ssi le nombre de liens durs = 0

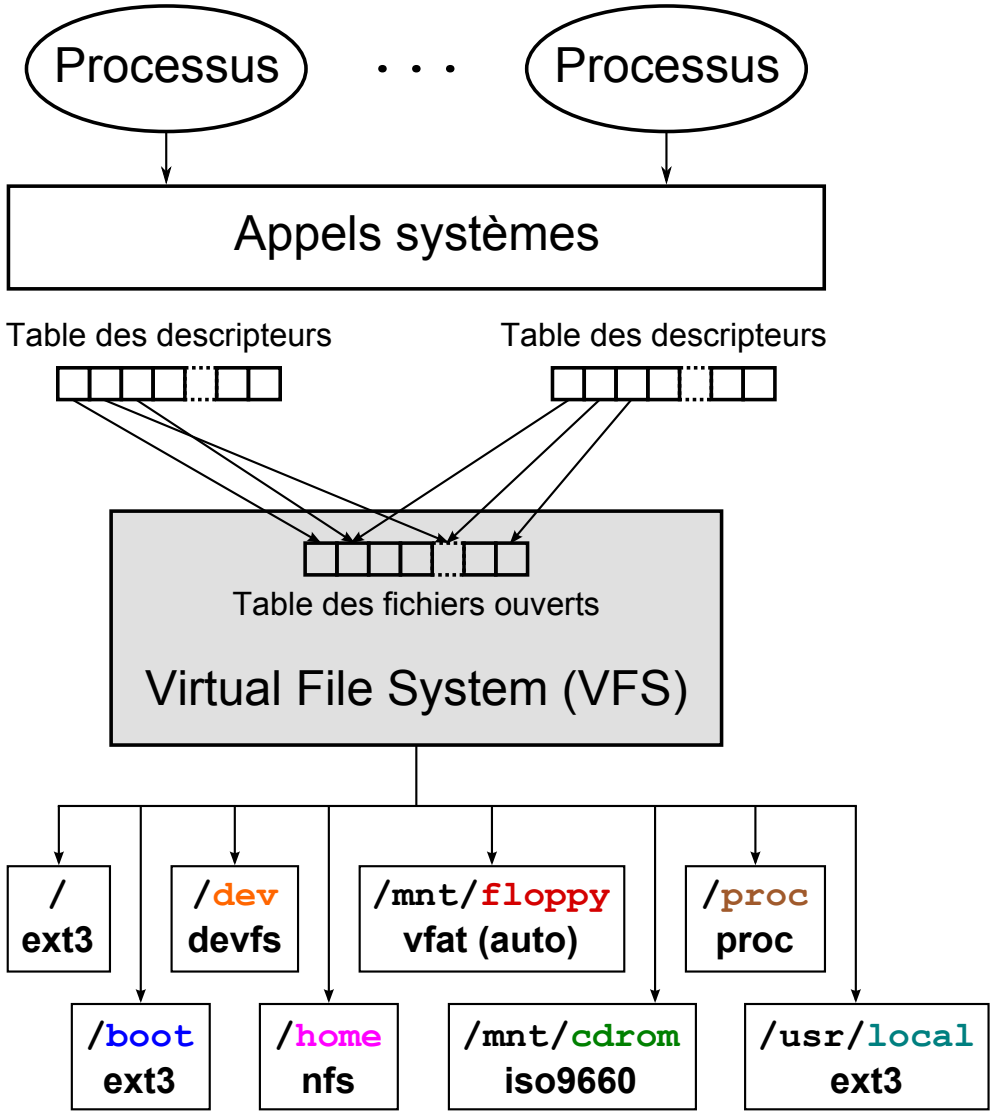
Types de fichiers

```
$ ls -l
total 4
brw-rw----   1 root  floppy    2,   0 mar  3  2002 fd0
lrwxrwxrwx   1 root  root          3 nov 17 08:43 lien -> fd0
-rw-r--r--   1 root  root          0 nov 17 08:44 regulier
drwxr-xr-x   2 root  root       4096 nov 17 08:44 répertoire
srwx------   1 root  root          0 nov 16 22:01 socket
crw-rw----   1 root  dialout   4,  64 mar  3  2002 ttyS0
prw-r--r--   1 root  root          0 nov 17 08:40 tube
```

- `-` Fichier régulier
- `d` Répertoire
- `l` Lien symbolique
- `p` Tube nommé
- `b` Périphérique (mode bloc)
- `c` Périphérique (mode caractère)
- `s` Socket nommée

Arborescence de fichiers





Appels systèmes relatifs aux fichiers I

chmod, fchmod – Modifier les permissions d'accés à un fichier.

```
int chmod(const char *pathname, mode_t mode);
int fchmod(int fildes, mode_t mode);
```

chown, fchown, lchown – Modifier l'appartenance d'un fichier.

```
int chown(const char *path, uid_t owner, gid_t group);
int fchown(int fd, uid_t owner, gid_t group);          (Non POSIX)
int lchown(cost char *path, uid_t owner, gid_t group);
```

close – Fermer un descripteur de fichier.

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

dup, dup2 – Dupliquer un descripteur de fichier.

```
int dup(int oldfd);
int dup2(int oldfd, int newfd);
```

Appels systèmes relatifs aux fichiers II

fcntl – Manipuler un descripteur de fichier.

```
int fcntl(int fd, int cmd);
int fcntl(int fd, int cmd, long arg);
int fcntl(int fd, int cmd, struct flock *lock);
```

fsync – Synchroniser un fichier en mémoire avec le disque.

```
int fsync (int fd);
```

ioctl – Controler les périphériques.

```
int ioctl(int d, int requete, ...);
```

(Non POSIX)

link – Crée un nouveau nom pour un fichier.

```
int link (const char *oldpath, const char *newpath);
```

lseek – Positionner la tête de lecture/écriture dans un fichier.

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

mknod – Créer un noeud du système de fichier. (POSIX si FIFO)

```
int mknod(const char *pathname, mode_t mode, dev_t dev);
```

Appels systèmes relatifs aux fichiers III

mount, umount – Monter/démonter des systèmes de fichiers.

```
int mount(const char *specialfile, ...
int umount(const char *dir);
```

(Non POSIX)

open – Ouvrir ou créer éventuellement un fichier.

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

pipe – Créer un tube.

```
int pipe(int filedes[2]);
```

poll – Attendre un évènement concernant un descripteur de fichier.

```
int poll(struct pollfd *ufds, unsigned int nfds, int delai);
```

(Non POSIX)

read – Lire le contenu d'un fichier.

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

Appels systèmes relatifs aux fichiers IV

readlink – Lire le contenu d'un lien symbolique.

```
ssize_t readlink(const char *path, char *buf, size_t bufsiz);
```

rename – Changer le nom ou l'emplacement d'un fichier.

```
int rename(const char *oldpath, const char *newpath);
```

select, pselect – Multiplexage d'entrées/sorties synchrones.

(Non POSIX)

```
int select(int n, fd_set *readfds,
           fd_set *writefds, fd_set *exceptfds,
           struct timeval *timeout);
int pselect(int n, fd_set *readfds, fd_set *writefds,
            fd_set *exceptfds, const struct timespec *timeout,
            sigset_t * sigmask);
```

socket – Créer un point de communication.

```
int socket(int domain, int type, int protocol);
```

Appels systèmes relatifs aux fichiers V

stat, fstat, lstat – Obtenir le statut d'un fichier (file status).

```
int stat(const char *file_name, struct stat *buf);
int fstat(int filedes, struct stat *buf);                    (Non POSIX)
int lstat(const char *file_name, struct stat *buf);          (Non POSIX)
```

symlink – Créer un nouveau nom pour un fichier.

```
int symlink(const char *cible, const char *nom);
```

umask – Fixer le masque de création de fichiers.

```
int umask(int mask);
```

unlink – Détruire un nom et éventuellement le fichier associé.

```
int unlink(const char *pathname);
```

write – Écrire dans un descripteur de fichier.

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

opendir – Ouvrir un répertoire pour le parcourir.

```
DIR *opendir(const char *name);
```

Appels systèmes relatifs aux fichiers VI

closedir – Fermer un répertoire (après un opendir).

```
int closedir(DIR *dir);
```

readdir – Lire une entrée du répertoire.

```
struct dirent *readdir(DIR *dir);
```

struct dirent – Type pour une entrée de répertoire.

```
struct dirent {
  long           d_ino;     /* inode number */
  off_t          d_off;     /* offset to the next dirent */
  unsigned short d_reclen;  /* length of this record */
  unsigned char  d_type;    /* type of file */
  char           d_name[?]; /* filename */
};
```

rewinddir – Faire revenir le répertoire à sa première entrée.

```
void rewinddir(DIR *dir);
```

Options de open

- `int open(const char *pathname,`
  `int flags`
  `[, mode_t mode]);`
- Flags (combinés avec « | ») :
  - `O_RDONLY, O_WRONLY, O_RDWR`
  - `O_CREAT [ | O_EXCL]`
  - `O_TRUNC, O_APPEND`
  - `O_NONBLOCK, O_SYNC, O_NOFOLLOW`
- Mode : indiquer le mode (`chmod`-like) en octal ou macro

L'appel système `read`

- `ssize_t read(int fd, void *buf, size_t count)`
- Read reçoit :
  - Le file descriptor designant le fichier dans lequel on lit
  - L'adresse de la zone dans laquelle on met les octets lus
  - Le nombre d'octets qu'on voudrait lire
- Read renvoit :
  - Le nombre d'octets réellement lus, ou :
  - 0 en cas de fin de fichier
  - `-1` en cas d'erreur (cf. `errno`)
- `write` fonctionne de la même façon que `read`

Détails de stat

```
int stat(const char *file_name, struct stat *buf);
```

```
struct stat {
  dev_t         st_dev;     /* Peripherique                 */
  ino_t         st_ino;     /* Numero i-noeud               */
  mode_t        st_mode;    /* Protection                   */
  nlink_t       st_nlink;   /* Nb liens materiels           */
  uid_t         st_uid;     /* UID proprietaire             */
  gid_t         st_gid;     /* GID proprietaire             */
  dev_t         st_rdev;    /* Type peripherique            */
  off_t         st_size;    /* Taille totale en octets      */
  unsigned long st_blksize; /* Taille de bloc pour E/S      */
  unsigned long st_blocks;  /* Nombre de blocs alloues      */
  time_t        st_atime;   /* Heure dernier acces          */
  time_t        st_mtime;   /* Heure derniere modification  */
  time_t        st_ctime;   /* Heure dernier changement     */
};
```

Des fonctions fichiers de la libC

```
FILE *fdopen(int fildes, const char *mode);
```

- Transforme un File Descriptor en FILE*

```
FILE *freopen(const char *path, const char *mode,
              FILE *stream);
```

- Ouvre le fichier path et l'associe au flux stream
- L'ancien fichier ouvert par stream est fermé
- Exemple :
  - `freopen("tmp/errlog", "w+", stderr);`
  - Redirige le flux stderr vers le fichier "/tmp/errlog"

Gestion de chaînes et mémoire (libC)

- `char *strstr(const char *meule_de_foin, const char *aiguille);`
- `char *strncpy(char *dest, const char *src, size_t n);`
- `char *strncat (char *dest, const char *src, size_t n);`
- `int strncmp (const char *s1, const char *s2, size_t n);`
- `void *memcpy (void *dest, const void *src, size_t n);`
- `void *memmove (void *dest, const void *src, size_t n);`
  - Chevauchement autorisé
- `void *memset (void *s, int c, size_t n);`
- `int sprintf (char *str, const char *format, ...);`
- `atof, atoi, atol`
- + variables statiques

Exercice

- Écrire le programme `mycp` qui fait une copie de fichier.
- `mycp` accepte deux arguments : le nom du fichier source, et le nom du fichier destination.
- Il va lire le fichier source, par blocs de `4kiB`, et copier chacun de ces blocs dans le fichier destination, qu'il aura créé ou tronqué au besoin.