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 2021-2022

Rappel : les slides du cours sont sur le dépôt `git` ; ils ne contiennent pas grand chose – ce contenu doit absolument être connu sur le bout des doigts avant d'aller en TP.

Organisation du système de fichiers (suite)

Où et comment stocker les fichiers ?

un fichier, c'est...

- du contenu
- des attributs ou méta-données (type, permissions, dates...)

plusieurs solutions :

- stockage contigu (CD-ROM)
- liste chaînée de blocs
- liste chaînée séparée (*File Allocation Table*)
- table d'i-nœuds, regroupant les attributs et les adresses des blocs
- ...

Consultation des i-nœuds

```
int stat(const char *pathname, struct stat *statbuf);
int lstat(const char *pathname, struct stat *statbuf);
int fstat(int fd, struct stat *statbuf);
```

remplissent une `struct stat` avec les caractéristiques de l'i-nœud et retournent 0, ou -1 en cas d'erreur, précisée par `errno` (`ENOENT` ou `EACCESS` par exemple)

le type `struct stat` contient (entre autres) les champs suivants :

```
struct stat {
  dev_t     st_dev;       /* ID of device containing file */
  ino_t     st_ino;       /* Inode number */
  mode_t    st_mode;      /* File type and mode */
  uid_t     st_uid;       /* User ID of owner */
  off_t     st_size;      /* Total size, in bytes */
  blksize_t st_blksize;   /* Block size for filesystem I/O */
  blkcnt_t st_blocks;     /* Number of 512B blocks allocated */
  /* ... */
```

Consultation des i-nœuds

Par exemple, pour déterminer un numéro d'i-nœud :

```
struct stat st;  /* déclaration préalable d'une struct stat */
if (stat("toto", &st)==-1) perror("stat toto");
else printf("inoeud numéro : %ld\n", st.st_ino);
```

champ donnant le numéro dans la table des i-noeuds

Consultation des i-nœuds

Par exemple, pour déterminer un numéro d'i-nœud :

```
struct stat st;  /* déclaration préalable d'une struct stat */
if (stat("toto", &st)==-1) perror("stat␣toto");
else printf("inoeud␣numéro␣:␣%ld\n", st.st_ino);
```

Mais d'autres champs sont plus compliqués à manipuler :

- `st.st_atime`, `st.st_ctime`, `st.st_mtime` sont des `struct timespec`

Consultation des i-nœuds

Par exemple, pour déterminer un numéro d'i-nœud :

```
struct stat st;  /* déclaration préalable d'une struct stat */
if (stat("toto", &st)==-1) perror("stat␣toto");
else printf("inoeud␣numéro␣:␣%ld\n", st.st_ino);
```

Mais d'autres champs sont plus compliqués à manipuler :

- `st.st_atime`, `st.st_ctime`, `st.st_mtime` sont des `struct timespec`
- `st.st_uid` et `st.st_gid` sont les identifiants de l'utilisateur et du groupe propriétaires ; pour déterminer leurs noms, il faut se référer au fichier des mots de passe, par exemple à l'aide de :
  `struct passwd *getpwuid(uid_t uid);`

Consultation des i-nœuds

Par exemple, pour déterminer un numéro d'i-nœud :

```
struct stat st;  /* déclaration préalable d'une struct stat */
if (stat("toto", &st)==-1) perror("stat␣toto");
else printf("inoeud␣numéro␣:␣%ld\n", st.st_ino);
```

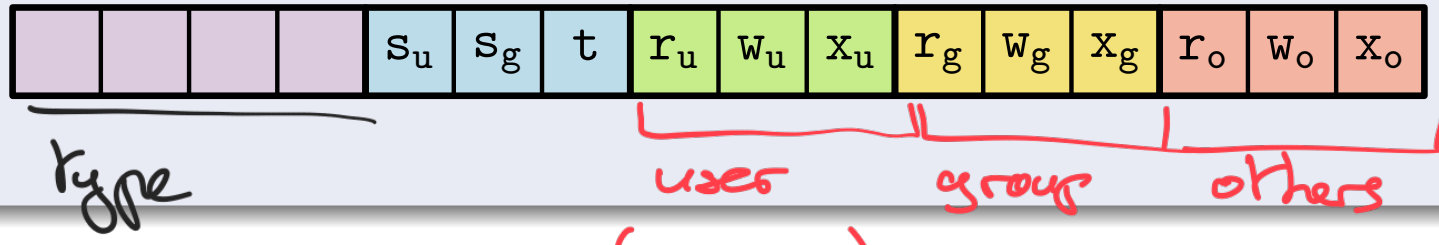
Mais d'autres champs sont plus compliqués à manipuler :

- `st.st_atime`, `st.st_ctime`, `st.st_mtime` sont des `struct timespec`
- `st.st_uid` et `st.st_gid` sont les identifiants de l'utilisateur et du groupe propriétaires ; pour déterminer leurs noms, il faut se référer au fichier des mots de passe, par exemple à l'aide de :
  `struct passwd *getpwuid(uid_t uid);`
- `st.st_mode` est un entier qui agrège deux informations : le type et les droits du fichier

Interprétation du champ st_mode

le champ st_mode est constitué de 2 octets, soit 16 bits :

4 pour le type, puis 4 fois 3 pour les droits :

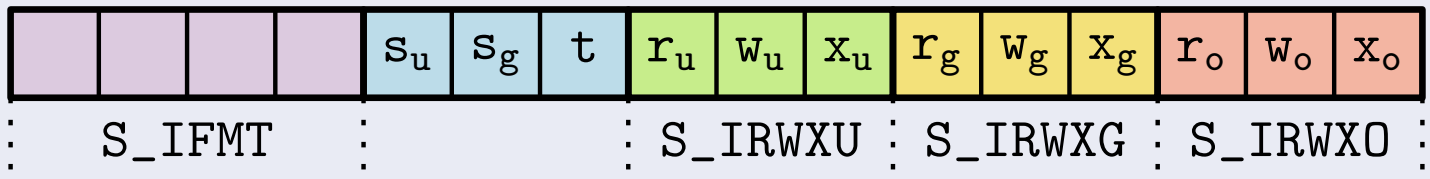


(ex 7)

Interprétation du champ st_mode

le champ st_mode est constitué de 2 octets, soit 16 bits :

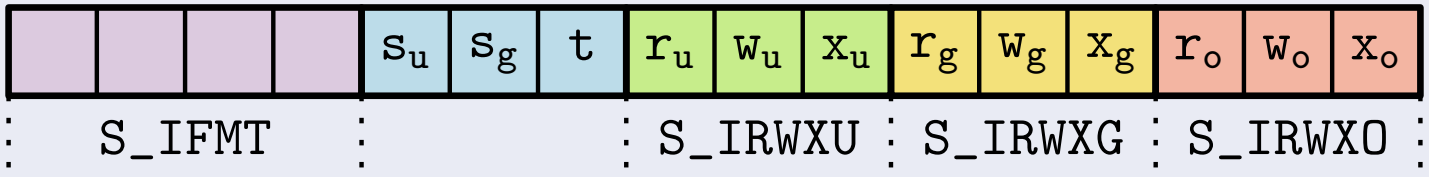
4 pour le type, puis 4 fois 3 pour les droits :



l'interprétation passe par des combinaisons logiques bit à bit avec des masques, comme par exemple S_IFMT=0170000, c'est-à-dire 4 bits 1 suivis de 12 bits 0, ou S_IWOTH=02, c'est-à-dire un unique 1 en avant-dernière position

Interprétation du champ st_mode

le champ st_mode est constitué de 2 octets, soit 16 bits :
4 pour le type, puis 4 fois 3 pour les droits :



l'interprétation passe par des combinaisons logiques bit à bit avec des masques, comme par exemple S_IFMT=0170000, c'est-à-dire 4 bits 1 suivis de 12 bits 0, ou S_IWOTH=02, c'est-à-dire un unique 1 en avant-dernière position

Exemple, pour tester si un fichier est un répertoire :

```
struct stat st;
if (stat("toto", &st)==-1) perror("stat toto");
if ((st.st_mode & S_IFMT) == S_IFDIR) { /* ... */ }
```

ou de manière équivalente :

```
if (S_ISDIR(st.st_mode)) { /* ... */ }
```

A RETENIR : stat("...", &st) permet d'avoir les informations d'un fichier (ex n° i-noeud, type, droits ...)

Modification des i-nœuds

pour changer les droits :

```
int chmod(const char *path, mode_t mode);
int fchmod(int fd, mode_t mode);
```

→ prend l'i-noeud au lieu du path (mieux)

pour changer les propriétaires :

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

→ idem

pour changer les dates :

```
int utimes(const char *path, const struct timeval times[2]);
int futimes(int fd, const struct timeval times[2]);
```

pour changer la taille :

```
int truncate(const char *path, off_t length);
int ftruncate(int fd, off_t length);
```

Structuration du système de fichiers

manifestement, une organisation « à plat » n'est pas viable

organisation hiérarchique $\Longrightarrow$ répertoires ou dossiers

une référence d'un fichier est la description d'un chemin menant au fichier – chemin absolu s'il part de la racine de l' arborescence, relatif s'il part du répertoire de travail courant

Exemples :

- sous Windows : `\quel\beau\chemin`
- sous UNIX : `/quel/beau/chemin`
- sous MULTICS : `>quel>beau>chemin`

un répertoire est un fichier structuré permettant de retrouver tous les blocs des fichiers qu'il contient : selon les cas, le répertoire associe à chaque `nom`, soit l'adresse où le fichier `nom` est stocké, soit le numéro de son 1^{er} bloc, soit son numéro d'i-nœud.
dans les deux premiers cas, il contient aussi les attributs du fichier

Consultation des répertoires

trop d'organisations physiques différentes
$\Longrightarrow$ normalisation des accès à travers le type `DIR`

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

→ ouvre un dossier. et renvoie le dossier là.

ouvre en lecture le répertoire, alloue un objet `DIR` et en renvoie l'adresse, ou `NULL` en cas d'échec (et `errno` est renseignée)

NULL si erreur (pas là, n'existe pas, etc)

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

libère la ressource

Consultation des répertoires

les entrées de répertoire sont représentées par des `struct dirent` qui contiennent au moins :

```
  struct dirent {
    ino_t   d_ino;       /* Inode number */
    char    d_name[];    /* Null-terminated filename */
    /* ... */
};
```

pour passer d'une entrée à la suivante, il faut utiliser :

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

qui lit l'entrée courante, décale le curseur de lecture à l'entrée suivante, et renvoie un pointeur vers la `struct dirent` remplie ; renvoie `NULL` lorsque la lecture est terminée, ou en cas d'erreur (et dans ce cas, `errno` est renseignée)

Consultation des répertoires

Schéma d'un parcours de répertoire :

```c
DIR *dirp = opendir(dirname);
struct dirent *entry;
while ((entry = readdir(dirp))) {
  /* traitement d'entry */
}
closedir(dirp);
```

if (!dirp)
perror("dossier non lu")

Consultation des répertoires

Schéma d'un parcours de répertoire :

```
DIR *dirp = opendir(dirname);
struct dirent *entry;
while ((entry = readdir(dirp))) {
  /* traitement d'entry */
}
closedir(dirp);
```

Autres fonctions liées au parcours de répertoire :

```
void rewinddir(DIR *dirp);
long telldir(DIR *dirp);
void seekdir(DIR *dirp, long loc);
```

Modification des répertoires

création et suppression d'un répertoire

```
int mkdir(const char *pathname, mode_t mode);
int rmdir(const char *pathname);
```

modification des entrées d'un répertoire

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

mais aussi...

```
int creat(const char *pathname, mode_t mode);
int open(const char *pathname, int flags, mode_t mode); /* en O_CREAT *
int mkdir(const char *pathname, mode_t mode);
```

crée si n'existe pas