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 2023-2024

Organisation du système de fichiers (suite)

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);
int fstatat(int fd, const char *path, struct stat *buf, int flag);
```

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 */
  /* ... */
};
```

Modification des i-nœuds

pour changer les droits :

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

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);
```

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

Structuration du système de fichiers

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

organisation hiérarchique $\Longrightarrow$ répertoires ou dossiers

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

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 :

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`

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`

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`

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 peut associer à chaque `nom`, l'adresse où le contenu du fichier `nom` est stocké, ou le numéro de son 1^{er} bloc, ou son numéro d'i-nœud, ou d'enregistrement MFT....
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 en lecture le répertoire, alloue un objet `DIR` et en renvoie l'adresse, ou `NULL` en cas d'échec (et `errno` est renseignée)

```
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 :

```
DIR *dirp = opendir(dirname);
struct dirent *entry;
while ((entry = readdir(dirp))/* != NULL */) {
  /* traitement de l'entrée de répertoire
     basé en particulier sur (l)stat(ref),
     où ref est une référence valide de l'entrée */
}
closedir(dirp);
```

Consultation des répertoires

Schéma d'un parcours de répertoire :

```
DIR *dirp = opendir(dirname);
struct dirent *entry;
while ((entry = readdir(dirp))/* != NULL */) {
  /* traitement de l'entrée de répertoire
     basé en particulier sur (l)stat(ref),
     où ref est une référence valide de l'entrée */
}
closedir(dirp);
```

Autres fonctions liées au parcours de répertoire :

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

Algorithme de recherche d'un fichier

donnée : une référence `ref`

initialisation :

- si `ref` est une référence absolue : i-nœud courant = i-nœud racine
- sinon, i-nœud courant = i-nœud du répertoire de travail courant

tant que `ref` est non vide,

- vérifier que l'i-nœud courant est un répertoire avec les bons droits d'accès,
- lire la composante suivante de `ref`
- parcourir les entrées de l'i-nœud courant ; si une entrée coïncide avec la composante, i-nœud courant = i-nœud de la composante
- sinon, échec

retourner l'i-nœud courant

(attention, ceci occulte toute la gestion mémoire des i-nœuds)

Effets des droits sur les répertoires

droit en lecture : nécessaire pour utiliser `opendir` et `readddir`

droit en exécution : nécessaire pour utiliser la correspondance *nom – numéro d'i-nœud*, donc utiliser le répertoire dans des références (relatives ou absolues)

droit en écriture : nécessaire pour modifier le contenu du répertoire, *i.e.* la liste de ses entrées

Apparté sur la gestion des droits

chaque utilisateur dispose d'un `umask`, qui s'applique à chaque création de fichiers, ce qui lui permet de se protéger contre la création intempestive de fichiers accordant trop de droits aux autres utilisateurs

à chaque création (avec `open`, `mkdir`...) avec un paramètre `mode`, les droits effectivement accordés sont `mode & ~umask` pour les répertoires, et `mode & ~umask & 0666` pour les fichiers ordinaires.

```
mode_t umask(mode_t mask);
```

permet de régler l'umask, et de récupérer l'ancienne valeur

pour accorder des droits supplémentaires, il faut un appel explicite à `chmod`

Modification des répertoires

création d'une entrée de répertoire :

- avec création d'i-nœud :

```
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);
int symlink(const char *target, const char *linkpath);
int mkfifo(const char *pathname, mode_t mode);
```

Modification des répertoires

création d'une entrée de répertoire :

- avec création d'i-nœud :

```
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);
int symlink(const char *target, const char *linkpath);
int mkfifo(const char *pathname, mode_t mode);
```

- sans création d'i-nœud :

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

Modification des répertoires

création d'une entrée de répertoire :

- avec création d'i-nœud :

```
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);
int symlink(const char *target, const char *linkpath);
int mkfifo(const char *pathname, mode_t mode);
```

- sans création d'i-nœud :

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

suppression d'une entrée de répertoire

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

modification d'une entrée de répertoire

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

Modification des répertoires

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

- `oldpath` est une référence valide de fichier autre qu'un répertoire (sauf si utilisateur privilégié)
- `newpath` ne correspond à aucun lien existant,
- `dirname(newpath)` désigne un répertoire sur le même disque que `oldpath`

- crée un nouveau lien physique `basename(newpath)` dans le répertoire `dirname(newpath)` vers l'i-nœud désigné par `oldpath`,
- incrémente le compteur de liens de l'i-nœud,
- retourne 0, ou -1 en cas d'échec.

Modification des répertoires

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

où `pathname` est une référence valide de fichier *autre que répertoire*,

- supprime le lien correspondant dans `dirname(pathname)`,
- décrémente le compteur de liens de l'i-nœud correspondant,
- si ce compteur est nul (et si le nombre d'ouvertures du fichier est nul), le fichier est supprimé,
- retourne 0, ou -1 en cas d'échec.

Modification des répertoires

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

où `pathname` est une référence valide de fichier *autre que répertoire*,

- supprime le lien correspondant dans `dirname(pathname)`,
- décrémente le compteur de liens de l'i-nœud correspondant,
- si ce compteur est nul (et si le nombre d'ouvertures du fichier est nul), le fichier est supprimé,
- retourne 0, ou -1 en cas d'échec.

Pour la suppression des répertoires *vides* :

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

Modification des répertoires

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

- `oldpath` est une référence valide de fichier autre que `.` et `..`
- si `newpath` correspond à un lien existant, il doit être de même type que `oldpath`

- remplace, *de manière atomique*, le lien (déduit de) `oldpath` par le lien (déduit de) `newpath`,
- si ce lien existait déjà, il est supprimé (cf `unlink` et `rmdir`)
- retourne 0, ou -1 en cas d'échec.