Programmation système : i-nœuds

Juliusz Chroboczek

24 Novembre 2009

1 I-nœuds

L'i-nœud est une des structures fondamentales du système de fichiers Unix ; conceptuellement, l'i-nœud « c'est » le fichier.

Structure d'un i-nœud sur disque Un i-nœud sur disque contient, entre autres :
- le type du fichier qu'il représente (fichier ordinaire ou répertoire) ;
- le nombre de liens sur l'i-nœud ;
- une indication du propriétaire du fichier ;
- la taille des données du fichier ;
- les temps de dernier accès et de dernière modification des données du fichier, et le temps de dernière modification de l'i-nœud.
- une structure de données (typiquement un arbre) qui permet de retrouver les données du fichier.

Structure d'un i-nœud en mémoire Lorsqu'il est chargé en mémoire, un i-nœud contient en outre les données nécessaires pour retrouver l'i-nœud sur disque :
- le numéro du périphérique dont il provient ;
- le numéro d'i-nœud sur le périphérique.

2 Lecture des i-nœuds

Un i-nœud contient un certain nombre de données qui peuvent intéresser le programmeur. L'appel système `stat` et son cousin `fstat` permettent de les consulter.

Ces appels système ont les prototypes suivants :

```
int stat(char *path, struct stat *buf);
int fstat(int fd, struct stat *buf);
```

Ils différent par la façon de spécifier l'i-nœud à consulter : `stat` y accède par un nom de fichier, tandis que `fstat` utilise un descripteur de fichier[1]. Ils remplissent le tampon `buf` avec les informations de l'i-nœud, puis retournent 0 en cas de succès, -1 en cas d'échec (et alors la variable `errno` est positionnée).

2.1 La structure stat

La structure `stat` retournée par `stat` et `fstat` contient au moins les champs suivants :

```
struct stat {
    dev_t st_dev
    ino_t st_ino
    mode_t st_mode
    nlink_t st_nlink
    uid_t st_uid
    gid_t st_gid
    off_t st_size
    time_t st_atime
    time_t st_mtime
    time_t st_ctime
    blkcnt_t st_blocks
};
```

Les types opaques ci-dessus (`dev_t`, `ino_t` etc.) sont souvent des synonymes de `int`, sauf `off_t`, qui est normalement un synonyme de `long`.

Localisation de l'i-nœud Les champs `st_dev` et `st_ino` contiennent, respectivement, le numéro du périphérique où se trouve l'i-nœud et le numéro de l'i-nœud. Ils ont la propriété qu'à un moment donné la paire (`st_dev`, `st_ino`) identifie l'i-nœud de façon unique sur le système, ce qui permet par exemple de vérifier que deux descripteurs de fichier pointent sur le même fichier.

Notons cependant qu'un numéro d'i-nœud peut être réutilisé après qu'un fichier a été supprimé, ce qui implique qu'il n'est pas possible de se servir de ces champs pour identifier des fichiers à des moments différents.

Comptage des références Le champ `st_nlinks` contient le compte des références à l'i-nœud. Nous en parlons en plus de détail ce-dessous.

Type et permissions Le champ `st_mode` contient 16 bits qui servent à identifier le type de fichier et en décrire les permissions d'accès.

Les 12 bits d'ordre bas de `st_mode` décrivent les permissions.

Les 4 bits d'ordre haut décrivent le type de fichier. Les valeurs possibles pour ces bits incluent (en octal) :

– 0100000 ou `S_IFREG` : fichier ordinaire ;

1 Unix ne permet pas d'accéder directement à un i-nœud.

– 0040000 ou S_IFDIR : répertoire ;

Outre les macros S_IF... ci-dessus, le fichier d'entête <sys/stat.h> définit des macros nommées S_ISREG et S_ISDIR qui permettent de tester ces valeurs.

Propriétaire Les champs st_uid et st_gid définissent le propriétaire du fichier. Il s'agit normalement des champs euid et egid du processus qui a créé l'i-nœud.

Taille Le champ st_size contient la taille du fichier, comptée en octets.

Temps Les champs st_atime, st_mtime et st_ctime contiennent les temps du fichier, comptés en secondes depuis l'Époque (même unités que l'appel système time vu auparavant). Ils ont la sémantique suivante :

– st_atime est le *temps du dernier accès* : il est mis à jour à chaque fois que des données du fichier sont lues (par exemple par l'appel système read) ;
– st_mtime est le *temps de dernière modification* : il est mis à jour à chaque fois que des données sont écrites dans le fichier ;
– st_ctime est le *temps de dernière modification du i-nœud* : il est mis à jour à chaque fois que le i-nœud est modifié (par exemple par les appels système du paragraphe 3 ci-dessous). Ce champ est parfois appelé *temps de création du fichier*.

Normalement, c'est st_mtime qui est intéressant ; l'auteur de ces lignes avoue qu'il ne s'est jamais encore servi des deux autres temps.

Les fonctions localtime(3) et strftime(3) permettent de convertir le temps Unix en temps local et de formater le résultat.

Taille des données Le champ st_blocks indique la place occupée sur disque par les données. L'unité dans laquelle est exprimée cette valeur n'est pas définie par POSIX (elle est de 512 octets sous la 7e édition, sous Unix BSD et sous Linux, mais elle vaut 1024 octets sur certains Unix Système V). Bien qu'il ne soit pas fiable sur les systèmes de fichiers modernes, ce champ permet parfois de détecter les fichiers à trous.

3 Modification des i-nœeuds

Un i-nœud peut être modifié *implicitement*, par un appel système qui manipule les données du disque ou la structure de répertoires, ou *explicitement* par un appel système dont le rôle principal est de modifier l'i-nœud.

3.1 Modification implicite

Toute extension ou troncation d'un fichier modifie les champs st_size et st_mtime d'un fichier. Toute écriture modifie st_mtime.

Toute lecture modifie st_atime.

Toute création ou suppression de liens modifie st_nlinks.

Toute création ou suppression de liens, et tout changement de propriétaire ou de permissions modifie le champ `st_ctime`.

3.2 Modification explicite

Changement de permissions Les permissions d'un fichier (soit les 12 bits bas du « mode » de l'i-nœud) peuvent être modifiés à l'aide des deux appels système suivants :

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

Comme dans le cas de `stat` et `fstat`, `chmod` et `fchmod` diffèrent dans la façon dont est identifié l'i-nœud à modifier : `chmod` prend un nom de fichier, `fchmod` un descripteur. Ces appels retournent 0 en cas de succès, -1 en cas d'erreur.

Changement de propriétaire Le propriétaire d'un fichier peut être changé avec les appels système suivants :

```
int chown(char *path, int uid, int gid);
int fchown(int fd, int uid, int gid);
```

Sous tous les systèmes, l'utilisateur `root` (euid = 0) a le droit de changer le propriétaire de tout fichier. Sous Système V, un utilisateur a en outre le droit de « donner » un fichier qui lui appartient à un autre utilisateur ; sous Unix BSD et Linux, ceci n'est pas autorisé pour éviter de contourner le système de quotas.

Dans certaines circonstances, un utilisateur a aussi le droit de changer le groupe d'un fichier qui lui appartient.

Changement de temps Les temps d'un fichier peuvent être changés à l'aide de l'appel système suivant :

```
int utime(char *filename, struct utimbuf *buf);
```

La structure `utimbuf` est définie comme suit :

```
struct utimbuf {
    int actime;
    int modtime;
};
```

Le champ `actime` contient le nouveau temps de dernier accès, et le champ `modtime` contient le nouveau temps de dernière modification. (Le temps de dernère modification de l'i-nœud est mis à jour par cet appel.)

4 Liens

Comme nous l'avons vu, un lien à un i-nœud est normalement créé par un appel à `open` avec le *flag* `O_CREAT` positionné.

4.1 Création de liens supplémentaires

L'appel système `link` crée un nouveau nom et un nouveau lien vers un i-nœud existant. Cet appel permet donc de faire apparaître le même fichier à deux endroits de la hiérarchie de fichiers, ou sous deux noms différents.

L'appel `link` a le prototype suivant :

```
int link(char *old, char *new);
```

où `oldpath` est le nom de fichier existant, et `new` le nom créé. Si `new` existe déjà, l'appel `link` échoue avec `errno` valant `EEXIST`.

4.2 Suppression de liens

L'appel `unlink` supprime un nom de fichier et le lien associé ; s'il s'agissait du dernier lien pointant sur un i-nœud, le fichier lui-même (i-nœud et données) est supprimé aussi. Cet appel a le prototype suivant :

```
int unlink(char *filename);
```

4.3 Renommage et déplacement de fichiers

Sous la 7e édition, un fichier était renommé ou déplacé en créant simplement un nouveau lien puis en supprimant l'ancien :

```
int old_rename(char *old, char *new) {
    int rc;
    unlink(new);
    rc = link(old, new); if(rc < 0) return -1;
    rc = unlink(old); if(rc < 0) return -1;
    return 0;
}
```

Cette approche avait un défaut sérieux : comme `old_rename` est composé de plusieurs actions dont chacune peut échouer, il est possible de se retrouver après une erreur (par exemple de permissions) avec un renommage partiellement effectué, par exemple où `new` n'existe plus tandis que `old` n'a pas été renommé. De plus, comme `link` n'est pas autorisé sur les répertoires pour un utilisateur ordinaire, seul `root` pouvait renommer les répertoires.

Aujourd'hui, Unix implémente un appel système `rename` :

```
int rename(char *old, char *new);
```

Cet appel système ne peut pas échouer partiellement. À la différence de `link`, il supprime le nouveau nom s'il existait déjà.

5 Comptage des références

Lorsque le dernier lien vers un i-nœud est supprimé, l'i-nœud lui-même et le contenu du fichier sont libérés, et l'espace peut être reutilisé pour d'autres fichiers. Intuitivement, l'appel système `unlink` sert à supprimer les fichiers.

Pour ce faire, le système maintient un champ entier `st_nlink` dans chaque i-nœud, son *compte de références*, qui est égal au nombre de liens existant sur cet i-nœud. Ce champ vaut initialement 1, il est incrémenté à chaque appel `link`, et décrémenté à chaque appel `unlink`. Lorsqu'il atteint 0, l'i-nœud et son contenu sont supprimés.

Le comptage de références, qui est une des techniques classiques de gestion de la mémoire, ne fonctionne pas en présence de cycles dans le graphe de références. Unix évite la présence de cycles dans le système de