

Programmation Système

Les attributs de fichiers

Emmanuel Bouthenot (emmanuel.bouthenot@u-bordeaux.fr)

Licence professionnelle ADSILLH - Université de Bordeaux

2018-2019

Définition

Les **attributs de fichiers** sont toutes les informations dont le système dispose sur un fichier (ou répertoire)

- taille du fichier
- permissions d'accès
- propriétaire et groupe
- horodatages

Définition

Un système de fichiers ou système de gestion de fichiers (**FS** pour File System) est une façon de stocker les informations et de les organiser dans des fichiers sur une mémoire de masse (disque dur/ssd, clé usb, etc.)

Parmi les systèmes de fichiers plus utilisés :

- ext2,3,4, btrfs (Linux)
- Yaffs (Android)
- NTFS FAT (Windows)
- ZFS, UFS (Unix)
- APFS (Apple)
- SMB, NFS (réseau)
- Ceph, GlusterFS (Cluster, Distribué)

Figure 4.13. Disk drive, partitions, and a file system

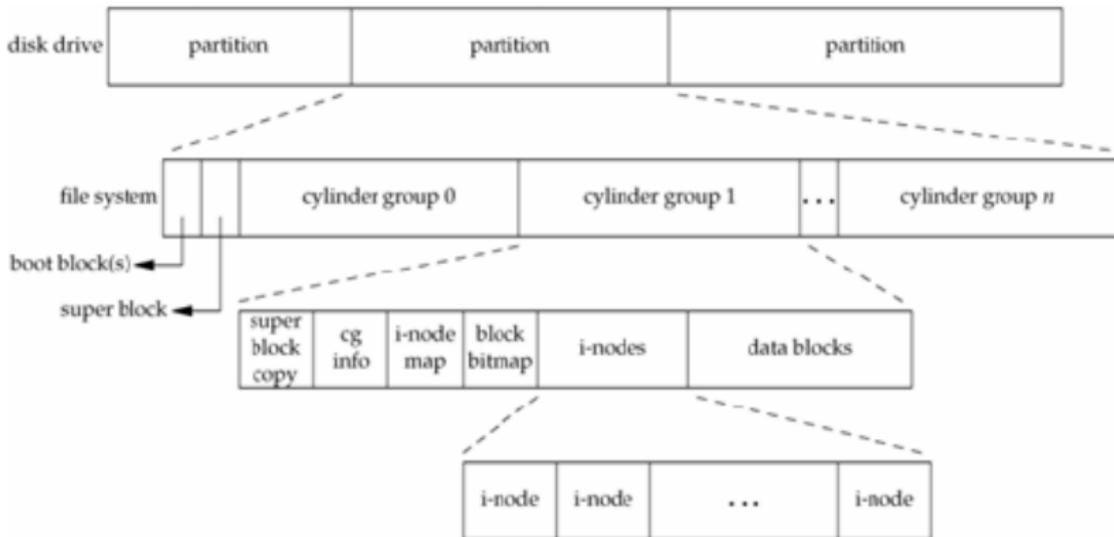


Figure: Disques, partitions et système de fichiers (**APUE** 4.13)

Système de fichiers - inodes

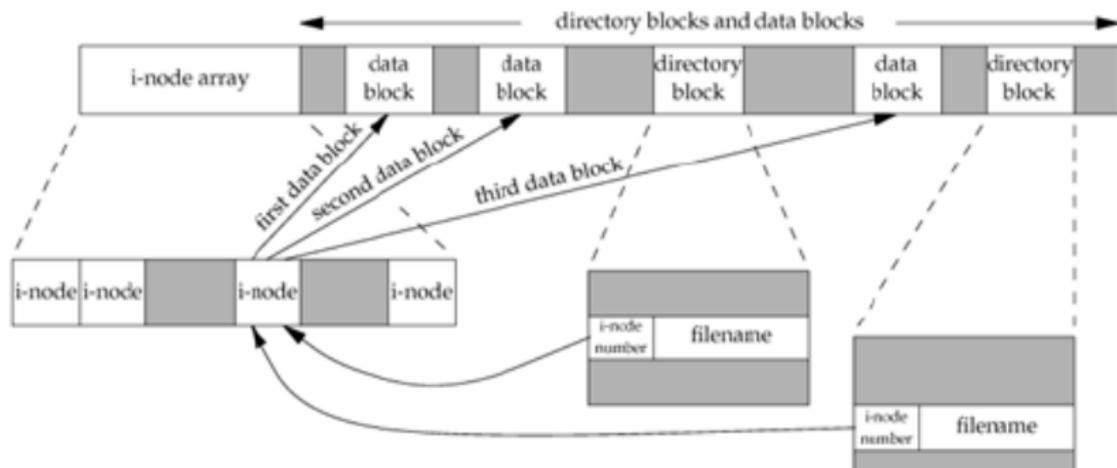


Figure: Inodes et blocs de données (**APUE** 4.14)

Appels systèmes : stat, fstat et lstat

La lecture des attributs d'un fichier passent par un appel système

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

int stat(const char *pathname, struct stat *statbuf);
// récupère l'état du fichier pointé par path et remplit
// le tampon statbuf

int fstat(int fd, struct stat *statbuf);
// identique à stat(), sauf que le fichier dont l'état est
// donné est celui référencé par le descripteur de fichier fd

int lstat(const char *pathname, struct stat *statbuf);
// identique à stat(), sauf que dans le cas où path est
// un lien symbolique, il donne l'état du lien lui-même
// plutôt que celui du fichier visé.

// En cas de succès, renvoi 0. En cas d'échec, renvoi -1 et
// positionne errno en conséquence
```

Structure stat

```
struct stat {
    dev_t      st_dev;      /* Périphérique */
    ino_t      st_ino;     /* Numéro d'i-nœud */
    mode_t     st_mode;    /* Protection */
    nlink_t    st_nlink;   /* Nombre de liens physiques */
    uid_t      st_uid;     /* UID du propriétaire */
    gid_t      st_gid;     /* GID du propriétaire */
    dev_t      st_rdev;    /* Type de périphérique */
    off_t      st_size;    /* Taille totale en octets */
    blksize_t  st_blksize; /* Taille de bloc pour E/S */
    blkcnt_t   st_blocks;  /* Nombre de blocs de 512 o alloués */

    struct timespec st_atim; /* Heure dernier accès */
    struct timespec st_mtim; /* Heure dernière modification */
    struct timespec st_ctim; /* Heure dernier changement état */
};
// st_rdev, st_blksize, et st_blocks ne sont pas requises par
// POSIX.1 mais définies comme extensions dans SUS
```

Structure stat expliquée

Nom	Signification
st_mode	Permissions d'accès au fichier ainsi que le type de ce dernier (répertoire, socket, fichier normal, etc.)
st_ino	Numéro de référence du fichier (SUSv4), identifiant unique d'accès au contenu du fichier, plus communément sous UNIX : numéro d' i-inœud
st_dev	Numéro du périphérique qui contient le système de fichier auquel se rapporte le numéro d' i-inœud
st_nlink	Nombre de liens physiques sur i-inœud (un fichier peut avoir plusieurs noms). L'appel système unlink() décrémente cette valeur jusqu'à 0 alors le fichier sera réellement supprimé
st_uid	User ID du propriétaire du fichier
st_gid	Group ID du propriétaire du fichier
st_size	Taille du fichier mesurée en octets (uniquement utile pour les fichiers normaux)

Structure stat expliquée

Nom	Signification
st_atime	Date du dernier accès (mise à jour lors de chaque lecture ou écriture)
st_ctime	Date du dernier changement de status (lecture, écriture, propriétaire, groupe, permissions, etc.)
st_mtime	Date de la dernière modification du contenu du fichier (insensible au changement de propriétaire, groupe, permissions, etc.)
st_rdev	Pour un fichier représentant un périphérique, numéros d'identification majeur et mineur
st_blksize	Taille optimale (en octets) pour les entrées-sorties sur ce fichier
st_blocks	Taille (en octets) effectivement allouée pour le fichier (telle qu'elle est mesurée par l'utilitaire du (champ évalué en nombre de blocs)

Macros pour les types de fichiers

```
if (S_ISREG(statbuf.st_mode)) {
    printf "Regular file";
}
if (S_ISDIR(statbuf.st_mode)) {
    printf "Directory file";
}
if (S_ISCHR(statbuf.st_mode)) {
    printf "Character special file";
}
if (S_ISBLK(statbuf.st_mode)) {
    printf "Block special file";
}
if (S_ISFIFO(statbuf.st_mode)) {
    printf "Pipe or Fifo";
}
if (S_ISLNK(statbuf.st_mode)) {
    printf "Symbolic link";
}
if (S_ISSOCK(statbuf.st_mode)) {
    printf "Socket";
}
```

Type de fichier - exemple

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>

int main(int argc, char *argv[]) {
    int i;
    struct stat statbuf;

    for (i = 1; i < argc; i++) {
        printf("%s: ", argv[i]);
        if (lstat(argv[i], &statbuf) < 0) {
            perror("Unable to get stats");
            continue;
        }
        if (S_ISREG(statbuf.st_mode)) {
            printf("regular\n");
        }
        else if (S_ISDIR(statbuf.st_mode)) {
            printf("directory\n");
        }
        else if (S_ISCHR(statbuf.st_mode)) {
            printf("character special\n");
        }
    }
}
```

Type de fichier - exemple

```
        printf("character special\n");
    }
    else if (S_ISBLK(statbuf.st_mode)) {
        printf("block special\n");
    }
    else if (S_ISFIFO(statbuf.st_mode)) {
        printf("fifo\n");
    }
    else if (S_ISLNK(statbuf.st_mode)) {
        printf("symbolic link\n");
    }
    else if (S_ISSOCK(statbuf.st_mode)) {
        printf("socket\n");
    }
    else {
        printf("*unknown*\n");
    }
}

exit(EXIT_SUCCESS);
}
```

Type de fichier - demo

```
./stat-file /etc /etc/passwd /dev/null /dev/sda  
/etc: directory  
/etc/passwd: regular  
/dev/null: character special  
/dev/sda: block special
```

Macros pour les permissions de fichiers

```
if (statbuf.st_mode & S_IRUSR) { printf "User can read"; }  
if (statbuf.st_mode & S_IWUSR) { printf "User can write"; }  
if (statbuf.st_mode & S_IXUSR) { printf "User can execute"; }  
if (statbuf.st_mode & S_IRGRP) { printf "Group can read"; }  
if (statbuf.st_mode & S_IWGRP) { printf "Group can write"; }  
if (statbuf.st_mode & S_IXGRP) { printf "Group can execute"; }  
if (statbuf.st_mode & S_IROTH) { printf "Others can read"; }  
if (statbuf.st_mode & S_IWOTH) { printf "Others can write"; }  
if (statbuf.st_mode & S_IXOTH) { printf "Others can execute"; }
```

Changer les permissions d'accès sur un fichier existant

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

int chmod(const char *pathname, mode_t mode);
// modifie les permissions du fichier indiqué dont le nom
// est fourni dans path, qui est déréférencé s'il s'agit
// d'un lien symbolique

int fchmod(int fd, mode_t mode);
// modifie les permissions du fichier référencé par le
// descripteur de fichier ouvert fd

// En cas de succès, renvoi 0. En cas d'échec, renvoi -1 et
// positionne errno en conséquence
```

Appels systèmes : chmod, fchmod - options

Le **mode** est un masque de bit comme suit

Mode	Valeur	Signification
S_IRUSR	00400	Accès en lecture pour le propriétaire
S_IWUSR	00200	Accès en écriture pour le propriétaire
S_IXUSR	00100	Accès en exécution/parcours par le propriétaire
S_IRGRP	00040	Accès en lecture pour le groupe
S_IWGRP	00020	Accès en écriture pour le groupe
S_IXGRP	00010	Accès en exécution/parcours pour le groupe
S_IROTH	00004	Accès en lecture pour les autres
S_IWOTH	00002	Accès en écriture pour les autres
S_IXOTH	00001	Accès en exécution/parcours pour les autres
...		S_ISUID, S_ISGID, S_ISVTX, cf. man 2 chmod

*Note : **parcours** s'applique aux répertoires, et signifie que le contenu du répertoire est accessible*

Modifier l'appartenance d'un fichier existant

```
#include <unistd.h>

int chown(const char *path, uid_t owner, gid_t group);
// modifie l'appartenance du fichier indiqué dont le nom
// est fourni dans path, qui est déréférencé s'il s'agit
// d'un lien symbolique

int fchown(int fd, uid_t owner, gid_t group);
// modifie l'appartenance du fichier référencé par le
// descripteur de fichier ouvert fd

int lchown(const char *path, uid_t owner, gid_t group);
// lchown() est comme chown(), mais ne déréférence pas les liens sy

// En cas de succès, renvoi 0. En cas d'échec, renvoi -1 et
// positionne errno en conséquence
```

-  **[APUE]** Advanced Programming in the UNIX Environment.
W. Richard Stevens and Stephen A. Rago.
Addison-Wesley Professional, 2005.
-  **[TLPI]** The Linux Programming Interface.
Michael Kerisk.
No Starch Press, 2010.