

Le système de gestion des fichiers, (II)

Luigi Santocanale

Laboratoire d'Informatique Fondamentale,
Centre de Mathématiques et Informatique,
39, rue Joliot-Curie - F-13453 Marseille

25 octobre 2005

1 Le SGF

- Représentation interne des fichiers, les i-noeuds
- Traduction de l'interface utilisateur
- Les tables des fichiers
- L'appel système open

2 Appels système POSIX

- Manipulation des i-noeuds
- Manipulation des répertoires

Plan

- 1 Le SGF
 - Représentation interne des fichiers, les i-noeuds
 - Traduction de l'interface utilisateur
 - Les tables des fichiers
 - L'appel système open
- 2 Appels système POSIX
 - Manipulation des i-noeuds
 - Manipulation des répertoires

i-noeud + contenu

Un fichier est un couple :

- noeud d'information (i-noeud),
avec sa location sur le disque logique,
- son contenu, ses données.

Contenu :

- Fichiers réguliers : notion de données usuelle,
- Répertoire : liste de couples de la forme
(`nom_de_fichier, numero_de_i-noeud`)
- Lien symbolique : chaîne de caractères
(i.e. chemin à un autre fichier).

i-noeud + contenu

Un fichier est un couple :

- noeud d'information (i-noeud),
avec sa location sur le disque logique,
- son contenu, ses données.

Contenu :

- Fichiers réguliers : notion de données usuelle,
- Répertoire : liste de couples de la forme
(`nom_de_fichier`, `numero_de_i-noeud`)
- Lien symbolique : chaîne de caractères
(i.e. chemin à un autre fichier).

(2 octets)

83	.
2	..
1798	init
1276	fsck
85	clri
1268	motd
1799	mount
88	mknod
2114	passwd
1717	umount
1851	checklist
92	fsdb1b
84	config
1432	getty
0	crash
95	mkfs
188	inittab

Session : liens durs

```
[lsantoca@localhost lecture5]$ touch a
[lsantoca@localhost lecture5]$ ls -l a
-rw-r--r--  1 lsantoca lsantoca 0 oct 24 21:31 a
[lsantoca@localhost lecture5]$ ln a b
[lsantoca@localhost lecture5]$ ls -l a b
-rw-r--r--  2 lsantoca lsantoca 0 oct 24 21:31 a
-rw-r--r--  2 lsantoca lsantoca 0 oct 24 21:31 b
[lsantoca@localhost lecture5]$ echo abc > a
[lsantoca@localhost lecture5]$ cat b
abc
[lsantoca@localhost lecture5]$ rm a
[lsantoca@localhost lecture5]$ ls -l a b
ls: a: No such file or directory
-rw-r--r--  1 lsantoca lsantoca 4 oct 24 21:25 b
[lsantoca@localhost lecture5]$ cat b
abc
```

Session : liens symboliques

```
[lsantoca@localhost lecture5]$ touch a
[lsantoca@localhost lecture5]$ ls -l a
-rw-r--r--  1 lsantoca lsantoca 0 oct 24 21:35 a
[lsantoca@localhost lecture5]$ ln -s a b
[lsantoca@localhost lecture5]$ ls -l a b
-rw-r--r--  1 lsantoca lsantoca 0 oct 24 21:35 a
lrwxrwxrwx  1 lsantoca lsantoca 1 oct 24 21:36 b -> a
[lsantoca@localhost lecture5]$ echo abc > a
[lsantoca@localhost lecture5]$ cat b
abc
[lsantoca@localhost lecture5]$ rm a
[lsantoca@localhost lecture5]$ ls -l a b
ls: a: No such file or directory
lrwxrwxrwx  1 lsantoca lsantoca 1 oct 24 21:36 b -> a
[lsantoca@localhost lecture5]$ cat b
cat: b: No such file or directory
```

L'accès aux i-noeuds

Accès au disque physique est coûteux,

solution : les caches.

compteur utilisation (si 0 alors libre)
numéro du i-noeud
verrouillé ?
est à jour ?
copie en mémoire

i-noeud sur disque

Les i-noeuds en mémoire

En tête :

- no périphérique, disque logique,
- indice de l'i-noeud dans le table des i-noeuds,
- nombre d'utilisation de l'i-noeud,
- état de l'i-noeud : à jour ou non,
- verrouillé ou non.

Informations copiés du disque:

- no. liens,
- utilisateur propriétaire, group propriétaire,
- droits sur le fichier,
- taille du fichier,
- date de dernier accès,
- date de dernière modification (contenu),
- date de dernière modification (info de l'i-noeud).

Les i-noeuds en mémoire

En tête :

- no périphérique, disque logique,
- indice de l'i-noeud dans le table des i-noeuds,
- nombre d'utilisation de l'i-noeud,
- état de l'i-noeud : à jour ou non,
- verrouillé ou non.

Informations copiés du disque:

- no. liens,
- utilisateur propriétaire, group propriétaire,
- droits sur le fichier,
- taille du fichier,
- date de dernier accès,
- date de dernière modification (contenu),
- date de dernière modification (info de l'i-noeud).

algorithme iget

entrée: numéro d'i-noeud dans un système de fichiers

sortie: i-noeud verrouillé

```
(
  while (non effectué)
  (
    if (i-noeud présent dans le cache des i-noeuds)
    (
      if (i-noeud verrouillé)
      (
        sleep(événement: i-noeud est déverrouillé);
        continue;
      )
      /* traitement spécial pour les points de montage */
      (voir chapitre 5)
      if (i-noeud dans la liste des i-noeuds libres)
        extraire l'i-noeud de la liste des i-noeuds libres;
      incrémenter le compte référence i-noeud;
      return (i-noeud);
    )
    /* i-noeud n'est pas dans le cache des i-noeuds */
    if (pas d'i-noeuds dans la liste des i-noeuds libres)
      return (erreur);
    extraire un nouvel i-noeud de la liste des i-noeuds libres;
    repositionner numéros d'i-noeud et de système de fichiers;
    extraire l'i-noeud de l'ancienne file, le placer dans la nouvelle;
    lire l'i-noeud depuis le disque (algorithme bread);
    initialiser l'i-noeud (compte référence à 1);
    return (i-noeud);
  )
)
```

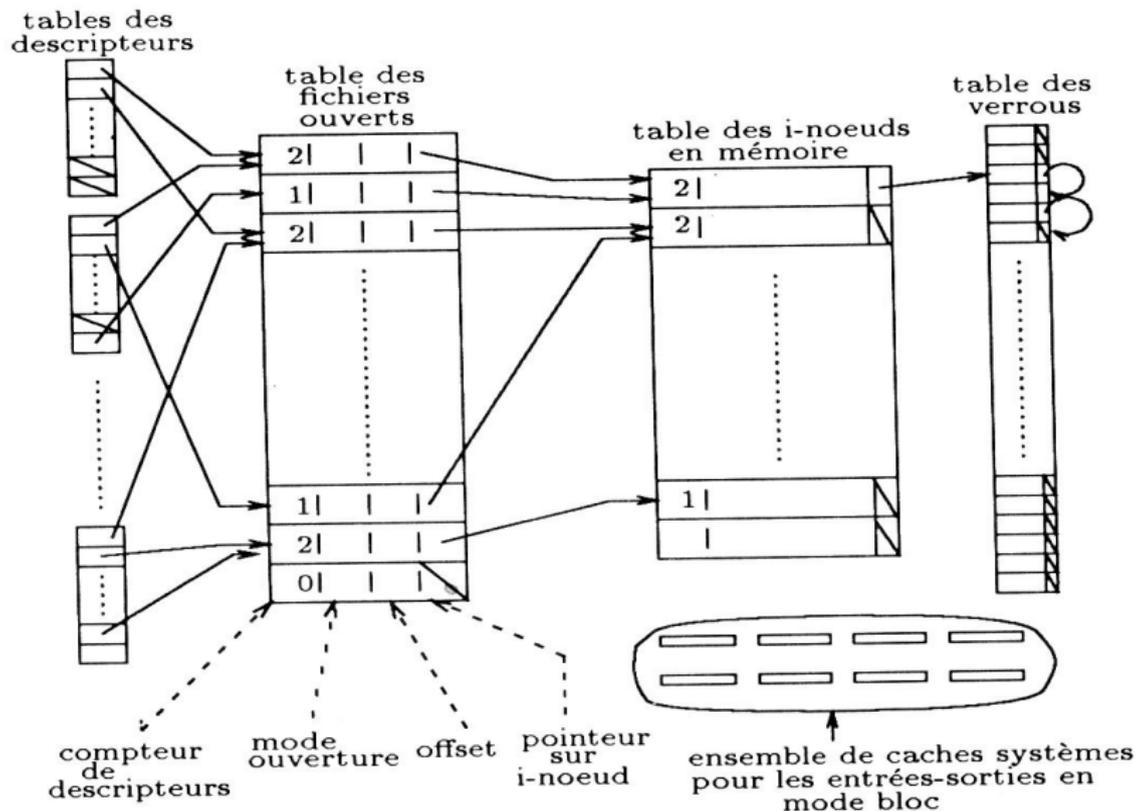
Plan

- 1 Le SGF
 - Représentation interne des fichiers, les i-noeuds
 - Traduction de l'interface utilisateur
 - Les tables des fichiers
 - L'appel système open
- 2 Appels système POSIX
 - Manipulation des i-noeuds
 - Manipulation des répertoires

```
algorithme namei /* conversion d'un chemin d'accès en un i-noeud */
entrée: chemin d'accès
sortie: i-noeud verrouillé
(
  if (chemin d'accès commence à la racine)
    i-noeud de travail = i-noeud racine (algorithme iget);
  else
    i-noeud de travail = i-noeud du répertoire courant
      (algorithme iget);
  while (chemin d'accès non épuisé)
  (
    lire la composante suivante du chemin d'accès
      passé en paramètre;
    vérifier que l'i-noeud de travail est celui d'un répertoire et
      que les permissions d'accès sont OK;
    if (i-noeud de travail est celui de la racine
      et que la composante est "..")
      continue;
    lire le répertoire (i-noeud de travail) par l'utilisation
      répétée des algorithmes bmap, bread et brelse;
    if (composante correspond à un élément de répertoire
      (i-noeud de travail))
    (
      accéder au numéro de l'i-noeud contenu dans l'élément;
      libérer l'i-noeud de travail (algorithme iput);
      i-noeud de travail = i-noeud de la composante (algorithme iget);
    )
    else /* la composante n'est pas dans le répertoire */
      return (pas d'i-noeud);
  )
)
return (i-noeud de travail);
```

Plan

- 1 Le SGF
 - Représentation interne des fichiers, les i-noeuds
 - Traduction de l'interface utilisateur
 - **Les tables des fichiers**
 - L'appel système open
- 2 Appels système POSIX
 - Manipulation des i-noeuds
 - Manipulation des répertoires



Les tables du système

- Tables des descripteurs, appartenant à un processus.

Descripteurs conventionnels :

- 0 STDIN_FILENO
- 1 STDOUT_FILENO
- 2 STDERR_FILENO

- Table des fichiers ouverts (appartenant au noyau) :

- compteur des descripteurs,
- mode d'ouverture,
- position courante,
- pointeur sur le i-noeud en mémoire.

- Cache des i-noeuds (noyau):

- nombre total d'ouvertures,
- id du disque logique,
- numéro de l'i-noeud sur ce disque,
- état de l'i-noeud.

Les tables du système

- Tables des descripteurs, appartenant à un processus.

Descripteurs conventionnels :

0 STDIN_FILENO
1 STDOUT_FILENO
2 STDERR_FILENO

- Table des fichiers ouverts (appartenant au noyau) :
 - compteur des descripteurs,
 - mode d'ouverture,
 - position courante,
 - pointeur sur le i-noeud en mémoire.
- Cache des i-noeuds (noyau):
 - nombre total d'ouvertures,
 - id du disque logique,
 - numéro de l'i-noeud sur ce disque,
 - état de l'i-noeud.

Les tables du système

- Tables des descripteurs, appartenant à un processus.

Descripteurs conventionnels :

0 STDIN_FILENO
1 STDOUT_FILENO
2 STDERR_FILENO

- Table des fichiers ouverts (appartenant au noyau) :
 - compteur des descripteurs,
 - mode d'ouverture,
 - position courante,
 - pointeur sur le i-noeud en mémoire.
- Cache des i-noeuds (noyau):
 - nombre total d'ouvertures,
 - id du disque logique,
 - numéro de l'i-noeud sur ce disque,
 - état de l'i-noeud.

Plan

- 1 Le SGF
 - Représentation interne des fichiers, les i-noeuds
 - Traduction de l'interface utilisateur
 - Les tables des fichiers
 - L'appel système open
- 2 Appels système POSIX
 - Manipulation des i-noeuds
 - Manipulation des répertoires

open

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
int
open(const char * fic, int mode, mode_t droits);
```

fic : le nom du fichier à ouvrir

mode : disjonction bit-à-bit de :
O_RDONLY, O_WRONLY, O_RDWR,
O_APPEND, O_TRUNC,
O_CREAT, O_EXCL,

droits : voir page suivante.

Retourne : erreur -1, descripteur de fichier sinon.

open

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
int
open(const char * fic, int mode, mode_t droits);
```

fic : le nom du fichier à ouvrir

mode : disjonction bit-à-bit de :
O_RDONLY, O_WRONLY, O_RDWR,
O_APPEND, O_TRUNC,
O_CREAT, O_EXCL,

droits : voir page suivante.

Retourne : erreur -1, descripteur de fichier sinon.

open

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
int
open(const char * fic, int mode, mode_t droits);
```

fic : le nom du fichier à ouvrir

mode : disjonction bit-à-bit de :
O_RDONLY, O_WRONLY, O_RDWR,
O_APPEND, O_TRUNC,
O_CREAT, O_EXCL,

droits : ... voir page suivante.

Retourne : erreur -1, descripteur de fichier sinon.

open

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
int
open(const char * fic, int mode, mode_t droits);
```

fic : le nom du fichier à ouvrir

mode : disjonction bit-à-bit de :
O_RDONLY, O_WRONLY, O_RDWR,
O_APPEND, O_TRUNC,
O_CREAT, O_EXCL,

droits : voir page suivante.

Retourne : erreur -1, descripteur de fichier sinon.

open

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
int
open(const char * fic, int mode, mode_t droits);
```

fic : le nom du fichier à ouvrir

mode : disjonction bit-à-bit de :
O_RDONLY, O_WRONLY, O_RDWR,
O_APPEND, O_TRUNC,
O_CREAT, O_EXCL,

droits : voir page suivante.

Retourne : erreur -1, descripteur de fichier sinon.

algorithme open

entrée: (1) nom de fichier

(2) type d'ouverture

(3) permissions fichier

sortie: descripteur de fichier

{

convertir le nom du fichier en un i-noeud (algorithme namei);

if (fichier inexistant ou accès non permis)

return (erreur);

attribuer à l'i-noeud un élément de la table des fichiers,

initialiser le compte et le déplacement;

attribuer un descripteur de fichier utilisateur,

positionner un pointeur vers l'élément associé

de la table des fichiers;

if (type d'ouverture spécifie une troncature du fichier)

libérer tous les blocs du fichier (algorithme free);

déverrouiller l'i-noeud; /* verrouiller dans namei */

return (descripteur de fichier);

}

Plan

- 1 Le SGF
 - Représentation interne des fichiers, les i-noeuds
 - Traduction de l'interface utilisateur
 - Les tables des fichiers
 - L'appel système open
- 2 Appels système POSIX
 - Manipulation des i-noeuds
 - Manipulation des répertoires

stat

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

struct stat {
    dev_t    st_dev;        /* Device. */
    ino_t    st_ino;       /* File serial number. */
    mode_t   st_mode;      /* File mode. */
    nlink_t  st_nlink;     /* Link count. */
    uid_t    st_uid;       /* User ID of the file's owner. */
    gid_t    st_gid;       /* Group ID of the file's group.*/
    off_t    st_size;      /* Size of file, in bytes. */
    time_t   st_atime;     /* Time of last access. */
    time_t   st_mtime;     /* Time of last modification. */
    time_t   st_ctime;     /* Time of last status change. */
};
```

stat, lstat, fstat

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

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

```
int stat(const char * ref, struct stat * pstat);
```

ref : chemin au fichier

pstat : pointeur à une structure stat à remplir

Retourne : 0 succès, -1 échec

```
int lstat(const char * ref, struct stat * pstat);
```

Sommaire : Comme stat, sauf pour les

liens symboliques : les informations portent sur les liens.

```
int fstat(int desc, struct stat * pstat);
```

desc : descripteur du fichier

stat, lstat, fstat

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

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

```
int stat(const char * ref, struct stat * pstat);
```

ref : chemin au fichier

pstat : pointeur à une structure stat à remplir

Retourne : 0 succès, -1 échec

```
int lstat(const char * ref, struct stat * pstat);
```

Sommaire : Comme stat, sauf pour les

liens symboliques : les informations portent sur les liens.

```
int fstat(int desc, struct stat * pstat);
```

desc : descripteur du fichier

pstat : pointeur à une structure stat à remplir

stat, lstat, fstat

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

```
int stat(const char * ref, struct stat * pstat);
```

ref : chemin au fichier

pstat : pointeur à une structure stat à remplir

Retourne : 0 succès, -1 échec

```
int lstat(const char * ref, struct stat * pstat);
```

Sommaire : Comme stat, sauf pour les

liens symboliques : les informations portent sur les liens.

```
int fstat(int desc, struct stat * pstat);
```

desc : descripteur du fichier

pstat : pointeur à une structure stat à remplir

stat, lstat, fstat

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

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

```
int stat(const char * ref, struct stat * pstat);
```

ref : chemin au fichier

pstat : pointeur à une structure stat à remplir

Retourne : 0 succès, -1 échec

```
int lstat(const char * ref, struct stat * pstat);
```

Sommaire : Comme stat, sauf pour les

liens symboliques : les informations portent sur les liens.

```
int fstat(int desc, struct stat * pstat);
```

desc : descripteur du fichier

pstat : pointeur à une structure stat à remplir

Test du type fichier

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

type fichier	MACRO
régulier	S_ISREG
périphérique bloc	S_ISBLK
périphérique caractère	S_ISCHR
répertoire	S_ISDIR
tube	S_ISFIFO
lien symbolique	S_ISLNK
socket	S_ISSOCK

Exemple :

```
struct stat fic_info;  
assert( (ret = stat("nomfichier",&fic_info)) == 0 );  
if ( IS_REG(fic_info.st_mode) )  
    desc = open("nomfichier",O_RDONLY);
```

Programme : estrep.c

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <fcntl.h>
4 #include <errno.h>
5
6 extern int errno;
7 int main(int argc, char *argv[])
8 {
9     if(argc != 2) exit(EXIT_FAILURE);
10    if(open(argv[1], O_WRONLY) == -1
11        && errno == EISDIR)
12        printf("Répertoire.\n");
13    else
14        printf("Pas un répertoire.\n");
15    exit(EXIT_SUCCESS);
16 }
```

Programme : estrep2.c

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <sys/types.h>
4 #include <sys/stat.h>
5
6 int main(int argc, char *argv[])
7 {
8     struct stat buf;
9
10    if(argc != 2) exit(EXIT_FAILURE);
11    if(stat(argv[1], &buf) == -1)
12        { perror("stat"); exit(EXIT_FAILURE); }
13
14    if( S_ISDIR(buf.st_mode) )
15        printf("Répertoire.\n");
16    else
17        printf("Pas un répertoire.\n");
18    exit(EXIT_SUCCESS);
19 }
```

access

```
#include <unistd.h>  
int access(char * ref, int acces);
```

ref : chemin au fichier

acces : disjonction bit-à-bit de types d'accès suivants :

R_OK : accès en lecture

W_OK : accès en écriture

X_OK : accès en exécution

F_OK : fichier existe ?

Sommaire : Teste les droits par rapport au propriétaire/groupe réels

access

```
#include <unistd.h>  
int access(char * ref, int aces);
```

ref : chemin au fichier

aces : disjonction bit-à-bit de types d'accès suivants :

R_OK : accès en lecture

W_OK : accès en écriture

X_OK : accès en exécution

F_OK : fichier existe ?

Sommaire : Teste les droits par rapport au propriétaire/groupe réels

access

```
#include <unistd.h>  
int access(char * ref, int aces);
```

ref : chemin au fichier

aces : disjonction bit-à-bit de types d'accès suivants :

R_OK : accès en lecture

W_OK : accès en écriture

X_OK : accès en exécution

F_OK : fichier existe ?

Sommaire : Teste les droits par rapport au propriétaire/groupe réels

access

```
#include <unistd.h>  
int access(char * ref, int aces);
```

ref : chemin au fichier

aces : disjonction bit-à-bit de types d'accès suivants :

R_OK : accès en lecture

W_OK : accès en écriture

X_OK : accès en exécution

F_OK : fichier existe ?

Sommaire : Teste les droits par rapport au propriétaire/groupe réels

Droits d'exécution sur les répertoires

```
[lsantoca@localhost lecture2]$ mkdir rep
[lsantoca@localhost lecture2]$ echo "abc" > rep/fic
[lsantoca@localhost lecture2]$ chmod u-x rep
[lsantoca@localhost lecture2]$ cat rep/fic
cat: rep/fic: Permission denied
[lsantoca@localhost lecture2]$ ls rep/
ls: rep/fic: Permission denied
[lsantoca@localhost lecture2]$ ls -d rep
rep/
```

Droits de lecture sur les répertoires

```
[lsantoca@localhost lecture2]$ chmod u+x rep
[lsantoca@localhost lecture2]$ chmod 300 rep
[lsantoca@localhost lecture2]$ ls -ld rep
d-wx-----  2 lsantoca lsantoca      4096 oct  8 12:10 rep
[lsantoca@localhost lecture2]$ ls rep/
ls: rep/: Permission denied
[lsantoca@localhost lecture2]$ cat fic/rep
cat: fic/rep: No such file or directory
[lsantoca@localhost lecture2]$ cat rep/fic
abc
```

link

```
#include <unistd.h>  
int link(char * ref1, char * ref2);
```

ref1 : chemin à un fichier existante.

ref2 : chemin à la référence à créer.

Remarques : on obtient un erreur si une de ces conditions s'avère :

- *ref1* est un répertoire,
- *ref2* existe déjà,
- *ref2* se trouve sur un disque logique autre que celui de *ref1*.

link

```
#include <unistd.h>  
int link(char * ref1, char * ref2);
```

ref1 : chemin à un fichier existante.

ref2 : chemin à la référence à créer.

Remarques : on obtient un erreur si une de ces conditions s'avère :

- *ref1* est un répertoire,
- *ref2* existe déjà,
- *ref2* se trouve sur un disque logique autre que celui de *ref1*.

link

```
#include <unistd.h>  
int link(char * ref1, char * ref2);
```

ref1 : chemin à un fichier existante.

ref2 : chemin à la référence à créer.

Remarques : on obtient un erreur si une de ces conditions s'avère :

- *ref1* est un répertoire,
- *ref2* existe déjà,
- *ref2* se trouve sur un disque logique autre que celui de *ref1*.

link

```
#include <unistd.h>  
int link(char * ref1, char * ref2);
```

ref1 : chemin à un fichier existante.

ref2 : chemin à la référence à créer.

Remarques : on obtient un erreur si une de ces conditions s'avère :

- *ref1* est un répertoire,
- *ref2* existe déjà,
- *ref2* se trouve sur un disque logique autre que celui de *ref1*.

unlink, rename

```
#include <unistd.h>
```

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

ref : chemin à un fichier existante.

Sommaire : détruit un lien

Remarques : le i-node n'est pas détruit tant que:

- le nombre de liens physiques sur ce inode est > 0 , ou
- le nombre d'ouverture de ce fichier est > 0 .

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

old : vieux chemin

new : nouveau chemin

Remarques : atomique.

unlink, rename

```
#include <unistd.h>
```

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

ref : chemin à un fichier existante.

Sommaire : détruit un lien

Remarques : le i-node n'est pas détruit tant que:

- le nombre de liens physiques sur ce inode est > 0 , ou
- le nombre d'ouverture de ce fichier est > 0 .

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

old : vieux chemin

new : nouveau chemin

Remarques : atomique.

unlink, rename

```
#include <unistd.h>
```

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

ref : chemin à un fichier existante.

Sommaire : détruit un lien

Remarques : le i-node n'est pas détruit tant que:

- le nombre de liens physiques sur ce inode est > 0 , ou
- le nombre d'ouverture de ce fichier est > 0 .

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

old : vieux chemin

new : nouveau chemin

Remarques : atomique.

chmod, chown

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

```
int chmod(char * ref, mode_t mode);
```

ref : chemin au fichier

mode : mode d'accès

```
int
```

```
chown(char * ref,
```

```
uid_t id_proprietaire, gid_t id_groupe);
```

ref : chemin au fichier

chmod, chown

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

```
int chmod(char * ref, mode_t mode);
```

ref : chemin au fichier

mode : mode d'accès

```
int
```

```
chown(char * ref,
```

```
uid_t id_proprietaire, gid_t id_groupe);
```

ref : chemin au fichier

chmod, chown

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

```
int chmod(char * ref, mode_t mode);
```

ref : chemin au fichier

mode : mode d'accès

```
int
```

```
chown(char * ref,
```

```
uid_t id_proprietaire, gid_t id_groupe);
```

ref : chemin au fichier

Plan

- 1 Le SGF
 - Représentation interne des fichiers, les i-noeuds
 - Traduction de l'interface utilisateur
 - Les tables des fichiers
 - L'appel système open
- 2 Appels système POSIX
 - Manipulation des i-noeuds
 - Manipulation des répertoires

stat

```
#include <dirent.h>

struct dirent {
    ...
    char d_name[NAME_MAX];
    ...
};
```

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

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

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

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

int mkdir(const char * ref, mode_t mode);

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

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

int mkdir(const char * ref, mode_t mode);

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

opendir, readdir, rewinddir, closedir

```
#include <sys/types.h>
#include <dirent.h>
```

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

Retourne : Le flot ouvert en lecture, et NULL si échec.

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

Retourne : Pointeur, et NULL si fin du répertoire ou échec.

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

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

Retourne : 0/-1.

opendir, readdir, rewinddir, closedir

```
#include <sys/types.h>  
#include <dirent.h>
```

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

Retourne : Le flot ouvert en lecture, et NULL si échec.

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

Retourne : Pointeur, et NULL si fin du répertoire ou échec.

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

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

Retourne : 0/-1.

opendir, readdir, rewinddir, closedir

```
#include <sys/types.h>  
#include <dirent.h>
```

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

Retourne : Le flot ouvert en lecture, et NULL si échec.

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

Retourne : Pointeur, et NULL si fin du répertoire ou échec.

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

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

Retourne : 0/-1.

opendir, readdir, rewinddir, closedir

```
#include <sys/types.h>
#include <dirent.h>
```

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

Retourne : Le flot ouvert en lecture, et NULL si échec.

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

Retourne : Pointeur, et NULL si fin du répertoire ou échec.

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

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

Retourne : 0/-1.

opendir, readdir, rewinddir, closedir

```
#include <sys/types.h>  
#include <dirent.h>
```

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

Retourne : Le flot ouvert en lecture, et NULL si échec.

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

Retourne : Pointeur, et NULL si fin du répertoire ou échec.

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

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

Retourne : 0/-1.

Programme : repertoires.c

```
1 #include <dirent.h>
2 #include <stdio.h>
3
4 int main(int argc, char *argv[])
5 {
6     DIR *flot = opendir(argv[1]);
7     struct dirent *entree;
8
9     if (flot == NULL){
10         perror(argv[1]); return 0;
11     }
```

Programme : repertoires.c(II)

```
13     for (;;) {
14         entree = readdir(flott);
15         if (entree == NULL)
16             {
17                 closedir(flott);
18                 break;
19             }
20         printf("%s\n", entree->d_name);
21     }
22
23     close(flott);
24     return 0;
25 }
```