Système de Fichiers

- **Généralités**
- Stockage d'un fichier
- Les catalogues
- Les fichiers partagés
- L'organisation de l'espace disque
- La cohérence du système de fichier
- Les primitives d'accès à un i-nœud
- Les primitives d'accès au répertoire

Le Système de Gestion des Fichiers (SGF)

Pour Le système Unix, tout est fichier. On distingue 3 types de fichier :

- Les fichiers ordinaires :
  - Fichiers de données : texte ou au format propriétaire
  - Fichiers programmes : instructions exécutables (binaire ou script)
- Les répertoires
- Les fichiers spéciaux : imprimantes, disques, cédéroms, … etc

Le système de gestion des fichiers (2)

Ces éléments sont regroupés dans une structure hiérarchisée représentée sous forme d'un arbre inversé. L'arbre comprend :

- Les nœuds : les répertoires
- Les feuilles : les fichiers ordinaires ou spéciaux

Le sommet de l'arbre est appelé racine (**root**)

Les caractéristiques d'un fichier

- Un fichier est désigné par son nom
- Accès au fichier se fait par :
  - Référence absolue
  - Référence relative
- L'ensemble des caractéristiques d'un fichier est contenues dans un **i-nœud** (nœud d'information).

Le nœud d'information

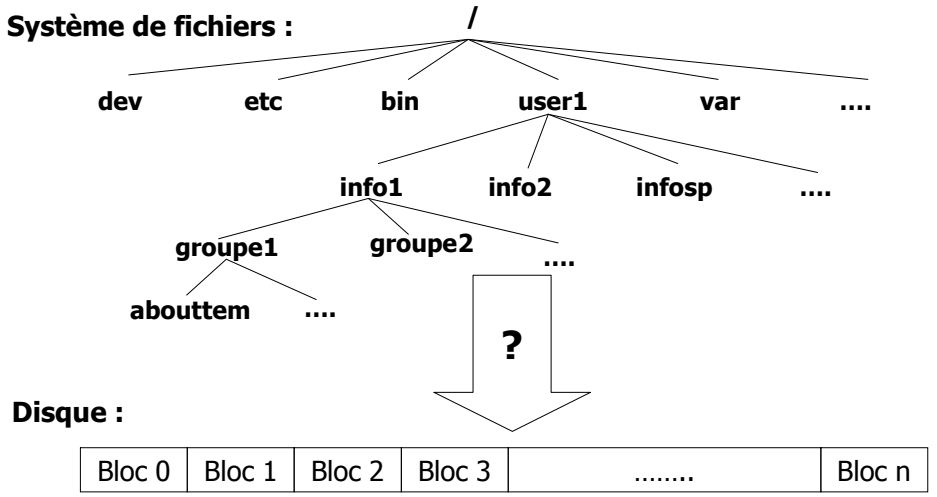
Un i-nœud contient les informations suivantes :

- Le type de fichier : fichier ordinaire, répertoire, …
- Les droits d'accès
- Le nombre de liens
- Le propriétaire
- Le groupe du propriétaire
- La taille du fichier en octets et en blocs
- Les dates d'accès et de modification
- … et le contenu du fichier

Système de Fichiers

- Généralités
- **Stockage d'un fichier**
- Les catalogues
- Les fichiers partagés
- L'organisation de l'espace disque
- La cohérence du système de fichier
- Les primitives d'accès à un i-nœud
- Les primitives d'accès au répertoire

Stockage d'un fichier ?



Les éléments à stocker

Les éléments du fichier à stocker sur le disque sont de 3 types :

- La référence du fichier (nom du fichier et le chemin d'accès)
- Les attributs du fichier (type, droits d'accès, le nombre de liens, …)
- Le contenu du fichier (une suite d'octets)

Stockage du fichier : allocation contiguë

Principe : stocker chaque fichier dans une suite de blocs consécutifs.

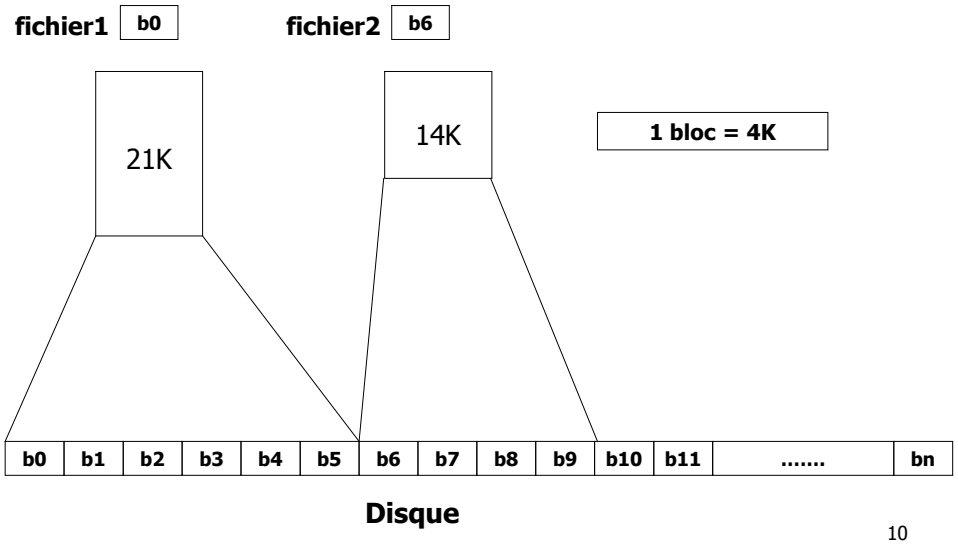
Avantages :

- Simple à mettre en œuvre (une seule adresse à mémoriser)
- Excellentes performances (le fichier peut être lu en une seule fois)

Inconvénients :

- Connaître la taille du fichier au moment de sa création
- Le disque est fragmenté (opération de compactage : coûteuse)

Exemple d'allocation contiguë



Stockage du fichier : allocation par liste chaînée

Principe : sauvegarder les blocs des fichiers dans une liste chaînée.

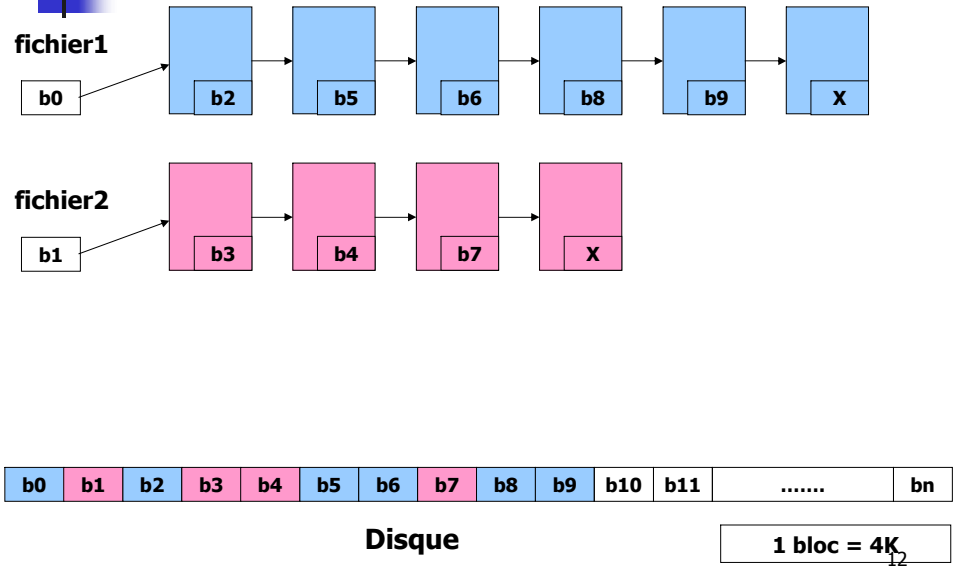
Avantages :

- Tous les blocs peuvent être utilisés (pas de fragmentation)
- Accès au fichier nécessite la connaissance de l'@ du 1er bloc.

Inconvénients :

- La lecture séquentielle est simple, l'accès aléatoire est extrêmement lent
- L'adresse du bloc suivant est contenue dans le bloc, un bloc ne contient pas que des données

Exemple d'allocation par liste chaînée



Stockage du fichier : allocation par liste chaînée indexée

Principe : retirer le pointeur de chaque bloc et le placer dans une table ou index en mémoire

Avantages :

- Le bloc ne contient que des données
- Accès aléatoire plus rapide : recherche se fait en mémoire
- Accès à un fichier : connaître le numéro du 1er bloc

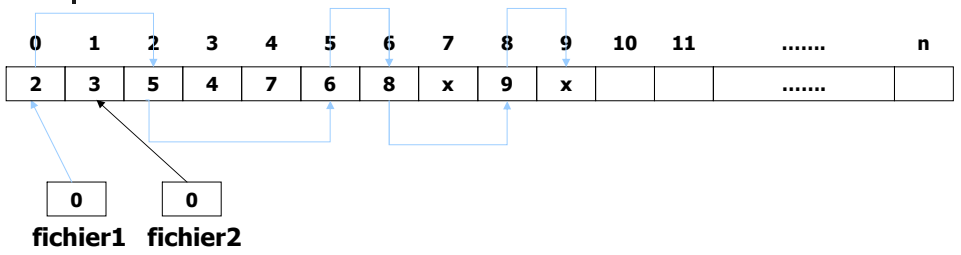
Inconvénient :

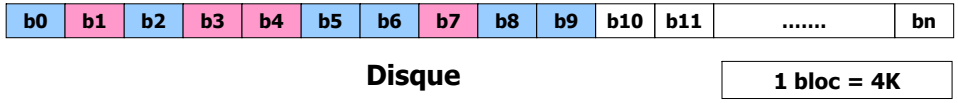
- La table des blocs doit être entièrement en mémoire en permanence.

  Exemple : disque de 10Go, bloc de 4K **à** 2 500 000 entrées. Une entrée occupe 3 octets **à** 7.5Mo pour la table de blocs

è Solution utilisée en Msdos

Exemple d'allocation par liste chaînée indexée





Stockage du fichier : les nœuds d'information

Principe : associer à chaque fichier une petite table appelé nœud d'information (i-nœud). La table contient :

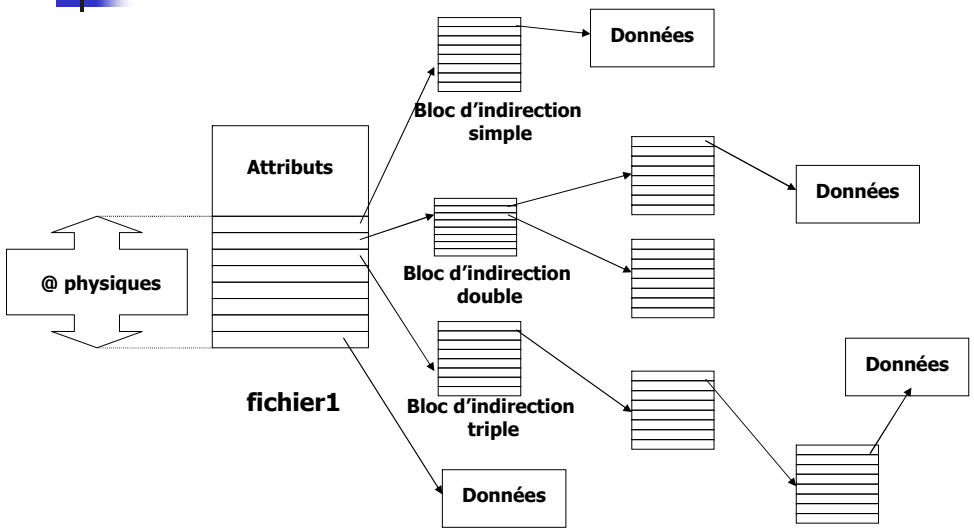
- Les attributs
- Les @ sur le disque des blocs du fichier.

Chaque entrée du i-nœud peut contenir :

- @ des données sur le disque
- @ sur un bloc d'indirection simple
- @ sur un bloc d'indirection double
- @ sur un bloc d'indirection triple

è Solution utilisée par Unix

Structure d'un nœud d'information



Système de Fichiers

- Généralités
- Stockage d'un fichier
- **Les catalogues**
- Les fichiers partagés
- L'organisation de l'espace disque
- La cohérence du système de fichier
- Les primitives d'accès à un i-nœud
- Les primitives d'accès au répertoire

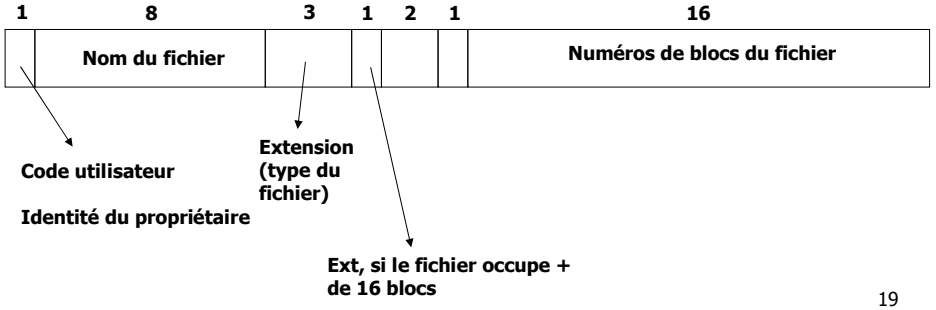
La mise en œuvre des catalogues

Les catalogues de CP/M

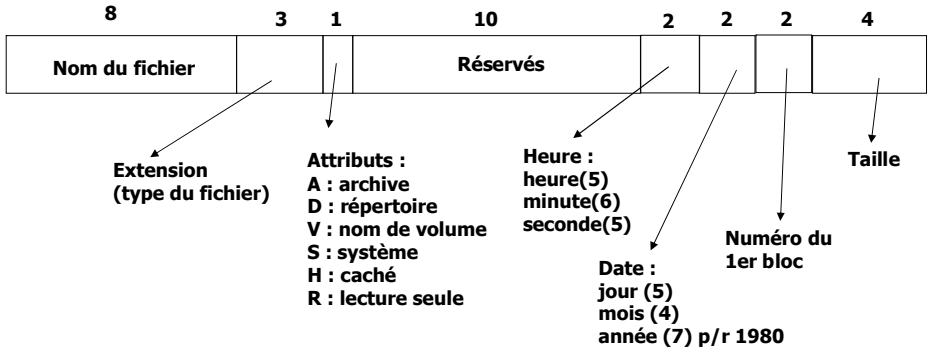
Le système comporte qu'un seul répertoire. Une entrée du répertoire a une taille de 32 octets

Un bloc peut ne pas être plein, le système connaît uniquement la taille du fichier en nombre de blocs.

Si la taille du fichier dépasse > 16 blocs, le système lui alloue une entrée supplémentaire.

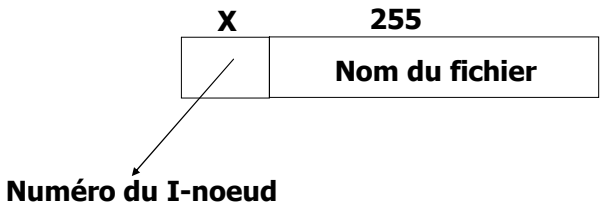


Les catalogues de MSDOS



Remarque : Le lien logique n'est pas possible sous MSDOS.

Les catalogues Unix



Les catalogues Unix : exemple

Recherche du répertoire : /user1/home/info1

Le bloc x contient le répertoire racine

1	.
1	..
4	bin
7	dev
	. . .
14	user1

I-nœud 14 /user1

type mode user group taille dates ....
132
....

Le bloc 132 contient le répertoire /user1

14	.
1	..
15	home
17	prof
	. .

I-nœud 15 /user1/home

type mode user group taille ....
156
....

Le bloc 156 contient le répertoire /user1/home

15	.
14	..
24	info1
29	info2
	. .

I-nœud 24 /user1/home/info1

type mode user group taille ....
176
....

Organisation du disque

I-nœud 1	I-nœud 2	.....	I-nœud m	bloc1	bloc2	bloc3	.....	bloc N

Système de Fichiers

- Généralités
- Stockage d'un fichier
- Les catalogues
- **Les fichiers partagés**
- L'organisation de l'espace disque
- La cohérence du système de fichier
- Les primitives d'accès à un i-nœud
- Les primitives d'accès au répertoire

Les fichiers partagés sous dos

Difficile à réaliser :

- Créer une entrée supplémentaire ayant les mêmes données physiques du fichier
- A chaque modification du fichier, il faut répercuter la modification des attributs à toutes les entrées pointant sur ce fichier

è Solution complexe à mettre en oeuvre

Les fichiers partagés sous unix

Il existe 2 solutions :

- Utiliser un i-nœud : lien physique
- Créer une entrée de répertoire du type lien. Ce fichier contient le chemin d'accès du fichier partagé : lien symbolique

Fichiers partagés : lien physique

Le bloc 156 contient le répertoire /user1/info1

23	.
19	..
77	fichier1
80	fichier2
	. .

Le bloc 132 contient le répertoire /user1/info2

55	.
70	..
77	f_partagé
90	symbol
	. .

I-nœud 77 /user1/info1/fichier1

type mode user group Taille lien = 2
180
....

Fichiers partagés : lien symbolique

Le bloc 156 contient le répertoire /user1/info1

23	.
19	..
77	fichier1
80	fichier2
	. .

Le bloc 132 contient le répertoire /user1/info2

55	.
70	..
77	f_partagé
90	symbol
	. .

Le bloc 190 contient le contenu de /user1/info2/symbol

/user1/info1/fichier2

I-nœud 80 /user1/info1/fichier2

type mode user .... lien = 1
180
....

I-nœud 90 /user1/info2/symbol

type mode user .... lien = 1
190
....

Système de Fichiers

- Généralités
- Stockage d'un fichier
- Les catalogues
- Les fichiers partagés
- **L'organisation de l'espace disque**
- La cohérence du système de fichier
- Les primitives d'accès à un i-nœud
- Les primitives d'accès au répertoire

L'organisation de l'espace disque

Support de sauvegarde des fichiers est un disque. Le disque est découpé en blocs.

La taille d'un bloc doit être judicieusement déterminée

- Si le bloc est trop grand **à** perte de l'espace disque
- Si le bloc est trop petit **à** fractionnement du fichier, augmente le nombre d'accès au disque

En pratique, la taille est de 4K.

Mémorisation des blocs libres (1)

2 solutions sont possibles :

- Liste chaînée : utilise une liste chaînée de blocs du disque, chaque nœud contient des numéros de bloc libres.

  bloc = 4Ko, une @ de bloc sur 32bits

  1 bloc de 4Ko contient 1024 @ de blocs libres

  Un disque de 10 Go possède 2 500 000 blocs

  nécessite 2500 blocs pour la sauvegarde

  à 10 Mo est nécessaire pour mémoriser les @ de blocs

Mémorisation des blocs libres (2)

- Table des bits : une table de bits (0 pour occupé et 1 pour libre). Un disque de n blocs nécessite une table de n bits.

  Un disque de 10 Go possède 2 500 000 blocs

  à nécessite une table de 2 500 000 bits

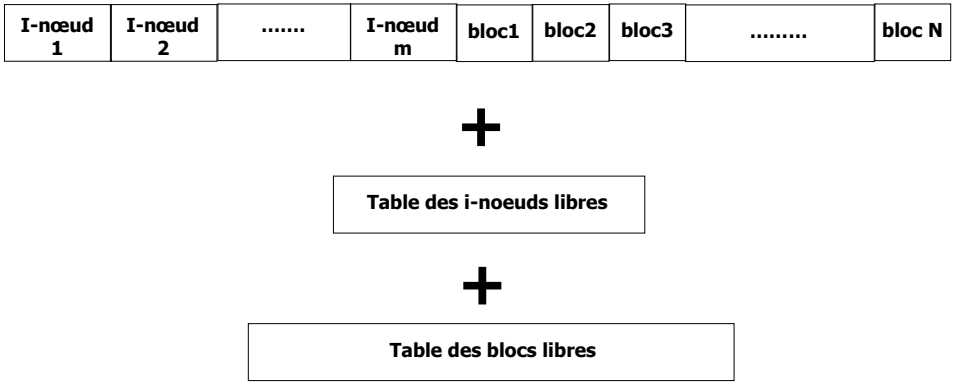
  à 312 500 octets

  à 77 blocs !!

Le choix dépend de la mémoire centrale :

- Ne peut contenir qu'un seul bloc : solution 1
- Peut contenir plusieurs blocs : solution 2

Organisation du disque



Système de Fichiers

- Généralités
- Stockage d'un fichier
- Les catalogues
- Les fichiers partagés
- L'organisation de l'espace disque
- **La cohérence du système de fichier**
- Les primitives d'accès à un i-nœud
- Les primitives d'accès au répertoire

La cohérence du système de fichier

La mise à jour des fichiers nécessite :

- La lecture d'un bloc
- La modification
- La ré-écriture

Si une panne survient entre la modification et la ré-écriture, le système peut se trouver dans un état incohérent

Des opérations de vérifications sont nécessaire pour maintenir la cohérence

2 types de vérifications sont utilisées :

- Vérification des blocs
- Vérification des fichiers

Vérification des blocs (1) : algorithme

L'opération utilise 2 tables :

- Une table des blocs occupés
- Une table des blocs libres

Algorithme de vérification :

Tant qu'il y a des fichiers

Lister les blocs utilisés par le fichier courant

Incrémenter les compteurs correspondants de la table 1

Fin de tant que

Parcourir la liste des blocs libres et mettre à jour la table 2

Le système est dans un état cohérent, si chaque bloc est soit occupé (1 dans la table 1), soit libre (1 dans la table 2)

Vérification des blocs (2) : erreurs

- Un bloc est manquant : 0 dans les 2 tables.
  - à Ajouter le bloc manquant dans la table des blocs libres
- Un bloc est libre et occupé à la fois : 1 dans les 2 tables.
  - à Retirer le bloc de la table des bloc libres
- Un bloc appartient à 2 fichiers différents : le compteur du bloc occupé concerné est à 2.
  - à Allouer un bloc libre, recopier le contenu du bloc et insérer ce dernier dans l'un des 2 fichiers. Le système contiendra des erreurs, mais retrouve un état stable

Vérification des fichiers (1) : algorithme

Comptabilisé le nombre de fois où chaque i-neoud est référencé et comparer avec le compteur de liens du i-nœud

Le système est dans un état cohérent, si les 2 valeurs sont identiques.

Vérification des fichiers (2) : erreurs

- Compteur de liens > le nombre d'entrées du catalogue pointant sur le i-nœud
  - PB : La suppression du i-nœud n'efface pas le i-nœud !
  - à Mettre à jour le compteur de liens à la bonne valeur
- Compteur de liens < le nombre d'entrées du catalogue pointant sur le i-nœud
  - PB : La suppression du i-nœud n'efface pas le i-nœud
  - à Mettre à jour le compteur de liens à la bonne valeur

Autres vérifications

- Le numéro d'un i-nœud doit être inférieur au numéro max des i-nœuds
- Le propriétaire d'un fichier doit avoir plus de droits que les autres.
- Les fichiers root situés dans les répertoires utilisateurs ne doivent pas avoir le bit SETUID positionné.

Vérifications : Système de fichiers journalisés

- Principe :
  - Mémoriser toutes les modifications dans un journal
  - Une modification ne sera effective que si elle a été répercutée sur le disque.
  - En cas de panne du système, il suffit de répercuter toutes les modifications non effectives du journal.
- Systèmes existants : Reiserfs, xfs, jfs, ext3fs.

Système de Fichiers

- Généralités
- Stockage d'un fichier
- Les catalogues
- Les fichiers partagés
- L'organisation de l'espace disque
- La cohérence du système de fichier
- **Les primitives d'accès à un i-nœud**
- Les primitives d'accès au répertoire

Le nœud d'information : rappel

Un i-nœud contient les informations suivantes :

- Le type de fichier : fichier ordinaire, répertoire, …
- Les droits d'accès
- Le nombre de liens
- Le propriétaire
- Le groupe du propriétaire
- La taille du fichier en octets et en blocs
- Les dates d'accès et de modification
- … et le contenu du fichier

Les primitives d'accès au status

SYNOPSIS

```
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
int stat(const char * file_name,struct stat *buf );
int fstat(int filedes,struct stat *buf );
int lstat(const char * file_name,struct stat *buf );
```

DESCRIPTION

Ces fonctions renvoient des informations à propos du fichier indiqué. Vous n'avez besoin d'aucun droit d'accès au fichier pour obtenir les informations, mais vous devez avoir le droit de parcours de tous les répertoires mentionnés dans le chemin menant au fichier.

Les primitives d'accès au status (2)

- **stat** récupère le statut du fichier pointé par *file_name* et remplit le buffer *buf* .
- **lstat** est identique à stat,sauf qu'il donne le statut d'un lien lui-même et non pas du fichier pointé par ce lien.
- **fstat** est identique à stat,sauf que le fichier ouvert est pointé par le descripteur *filede*s,obtenu avec *open(3*).

La structure stat

Les trois fonctions retournent une structure *stat* contenant les champs suivants :

```
struct stat {
dev_t st_dev;              /* Périphérique */
ino_t st_ino;              /* Numéro i-noeud */
mode_t st_mode;            /* Protection */
nlink_t st_nlink;          /* Nb liens matériels */
uid_t st_uid;              /* UID propriétaire */
gid_t st_gid;              /* GID propriétaire */
dev_t st_rdev;             /* Type périphérique */
off_t st_size;             /* Taille totale en octets */
unsigned long st_blksize;  /* Taille de bloc pour E/S */
unsigned long st_blocks;   /* Nombre de blocs de 512 octets alloués  */
time_t st_atime;           /* Heure dernier accès */
time_t st_mtime;           /* Heure dernière modification */
time_t st_ctime;           /* Heure dernier changement */ };
```


Le type de fichier (st_mode)

Les macros POSIX suivantes sont fournies pour manipuler les statuts d'un fichier, avec m du type mode_t.

- S_ISLNK(m) un lien symbolique ?
- S_ISREG(m) un fichier régulier ?
- S_ISDIR(m) un répertoire ?
- S_ISCHR(m) un périphérique en mode caractère ?
- S_ISBLK(m) un périphérique en mode blocs ?
- S_ISFIFO(m) une FIFO ?
- S_ISSOCK(m) une socket ?

Les droits d'accès (st_mode)

S_ISUID	0004000	bit Set-UID
S_ISGID	0002000	bit Set-Gid
S_IRWXU	00700	droits de lecture/écriture/exécution du propriétaire
S_IRUSR	00400	le propriétaire a le droit de lecture
S_IWUSR	00200	le propriétaire a le droit d'écriture
S_IXUSR	00100	le propriétaire a le droit d'exécution
S_IRWXG	00070	droits de lecture/écriture/exécution du groupe
S_IRGRP	00040	le groupe a le droit de lecture
S_IWGRP	00020	le groupe a le droit d'écriture
S_IXGRP	00010	le groupe a le droit d'exécution
S_IRWXO	00007	droits de lecture/écriture/exécution des autres
S_IROTH	00004	les autres ont le droit de lecture
S_IWOTH	00002	les autres ont le droit d'écriture
S_IXOTH	00001	les autres ont le droit d'exécution

Valeur renvoyée

Ces fonctions retournent zéro si elles réussissent. En cas d'echec -1 est renvoyé, et *errno* contient le code d'erreur.

Code d'erreur :

- EBADF — *filedes* est un mauvais descripteur.
- ENOENT — Un composant de *file_name* n'existe pas, ou il s'agit d'une chaîne vide.
- ENOTDIR — Un composant du chemin d'accès n'est pas un répertoire.
- ELOOP — Trop de liens symboliques rencontrés dans le chemin d'accès.
- EFAULT — Un pointeur se trouve en dehors de l'espace d'adressage.
- EACCES — Autorisation refusée.
- ENOMEM — Pas assez de mémoire pour le noyau.
- ENAMETOOLONG — Nom de fichier trop long

Exemples : main

```
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
#include <iostream.h>
main (int argc, char **argv){
    struct stat buf;
    if (argc==2) {
        if (lstat (argv[1],&buf)!=0)
                cout << "Le fichier n'existe pas";
        else {
                afficher_type(buf.st_mode);
                afficher_droits(buf.st_mode);
                nom_uid(buf.st_uid);
                nom_gid(buf.st_gid);
        }
    } else cout << "\n\n format de la commande : ls nom \n\n";
}
```


Exemples : afficher_type

```
int affiche_type(mode_t m) {
 /* Cette fonction affiche le type du fichier*/
    if (S_ISLNK(m)) cout << "l";
    else if (S_ISREG(m)) cout << "-";
    else if (S_ISDIR(m)) cout << "d";
    else if (S_ISCHR(m)) cout << "c";
    else if (S_ISBLK(m)) cout << "b";
    else if (S_ISFIFO(m)) cout << "p";
    else if (S_ISSOCK(m)) cout << "s";
    else cout << "?";
}
```


Exemples : afficher_droits

```
int affiche_droits(mode_t mode) {
 /* Cette fonction affiche les droits du fichier*/
    cout << mode & S_IRUSR ? 'r' :'-';
    cout << mode & S_IWUSR?'w':'-';
    cout << mode & S_ISUID?'S':mode & S_IXUSR?'x':'-';
    cout << mode & S_IRGRP?'r':'-';
    cout << mode & S_IWGRP?'w':'-';
    cout << mode & S_ISGID?'S':mode & S_IXGRP?'x':'-';
    cout << mode & S_IROTH?'r':'-';
    cout << mode & S_IWOTH?'w':'-';
    cout << mode & S_ISVTX?'T':mode & S_IXOTH?'x':'-';
}
```

Le nom d'un utilisateur : getpwuid

#include <pwd.h>

#include <sys/types.h>

struct passwd *getpwuid (uid_t *uid*);

La fonction **getpwuid()** renvoie un pointeur sur une structure contenant les divers champs de l'enregistrement de */etc/passwd* correspondant au à l'ID d'utilisateur ***uid.*** La structure ***passwd*** est définie dans <*pwd.h*> ainsi :

```
struct passwd {
    char *pw_name;      /* Nom d'utilisateur */
    char *pw_passwd;    /* Mot de passe */
    uid_t pw_uid;       /* ID de l'utilisateur */
    gid_t pw_gid;       /* ID du groupe de l'utilisateur */
    char *pw_gecos;     /* Nom réel de l'utilisateur */
    char *pw_dir;       /* Répertoire de connexion */
    char *pw_shell;     /* Programme Shell de connexion */
};
```


Le nom d'un utilisateur : exemple

```
#include <pwd.h>
#include <sys/types.h>
…..
void nom_uid (uid_t uid) {
  /* procedure permettant d'afficher le nom d'utilisateur en fonction de
     son uid */
  struct passwd *pass;

  pass=getpwuid(uid);
  cout << pass->pw_name;
}
```


Le nom d'un group : getpwuid

#include <grp.h>

#include <sys/types.h>

struct group *getgrgid (gid_t *gid*);

La fonction **getgrgid()** renvoie un pointeur sur une structure contenant l'enregistrement issus de */etc/group* pour le groupe correspondant a l'ID *gi*d. La structure *group* est définie dans <*grp.h*> ainsi :

```
struct group {
    char *gr_name;      /* Nom du groupe. */
    char *gr_passwd;    /* Mot de passe du groupe. */
    gid_t gr_gid;       /* ID du groupe. */
    char **gr_mem;      /* Membres du groupe. */
};
```


Le nom d'un group : exemple

```
#include <grp.h>
#include <sys/types.h>
.....
void nom_gid (gid_t gid) {
  /* procedure permettant d'afficher le nom du groupe en fonction de
     son gid */
  struct group *gp;

  gp=getgrgid(gid);
  cout << gp->gr_name;
}
```

Système de Fichiers

- Généralités
- Stockage d'un fichier
- Les catalogues
- Les fichiers partagés
- L'organisation de l'espace disque
- La cohérence du système de fichier
- Les primitives d'accès à un i-nœud
- **Les primitives d'accès au répertoire**

Les primitives d'accès à un répertoire

Un répertoire contient un ensemble de fichiers et/ou de répertoire. On peut considérer un répertoire comme un fichier.

14	.
1	..
15	home
17	prof
	⋮

Les primitives d'accès au répertoire sont :

- Ouverture du répertoire : opendir
- Lecture d'une entrée du répertoire : readdir
- Fermeture du répertoire : closedir

Ouverture d'un répertoire

SYNOPSIS

#include <sys/types.h>

#include <dirent.h>

DIR *opendir (const char *name);

DESCRIPTION

La fonction **opendir()** ouvre un flux répertoire correspondant au répertoire *name*,et renvoie un pointeur sur ce flux. Le flux est positionné sur la première entrée du répertoire.

Ouverture d'un répertoire (2)

VALEUR RENVOYÉE

La fonction **opendir()** renvoie un pointeur sur le flux répertoire ou NULL si une erreur se produit.

ERREURS

- **EACCESS** Accès interdit.
- **EMFILE** Trop de descripteurs de fichiers pour le processus en cours.
- **ENFILE** Trop de fichiers ouverts simultanément sur le système.
- **ENOENT** Le répertoire n'existe pas, ou *name* est une chaîne vide.
- **ENOMEM** Pas assez de mémoire.
- **ENOTDIR** *name* n'est pas un répertoire

Accès à une entrée

SYNOPSIS

#include <sys/types.h>

#include <dirent.h>

struct dirent *readdir (DIR *dir);

DESCRIPTION

La fonction readdir() renvoie un pointeur sur une structure dirent représentant l'entrée suivante du flux répertoire pointé par *dir*.Elle renvoie NULL a la fin du répertoire, ou en cas d'erreur.

Structure d'une entrée

La structure *dirent* est déclarée comme suit :

```
struct dirent {
    long d_ino;
        /* un numéro d'i-node */
    off_t d_off;
        /*la distance entre le début du répertoire et cette structure
        dirent */
    unsigned short d_reclen;
        /* la longueur d_name, sans compter le caractère nul final */
    char d_name [NAME_MAX+1];
        /* le nom de fichier terminé par un caractère nul. */
}
```


Exemple : contenu d'un répertoire

```
#include <sys/types.h>
#include <dirent.h>
#include <iostream.h>
main (int argc, char **argv){
    DIR  *rep;
    struct dirent * ent;
    if (argc==2) {
        if ((rep = opendir(argv[1])==NULL)
                cout << "Le répertoire n'existe pas";
        else {
                while (ent=readdir(rep)) {
                    cout << ent->d-name;
                }
                closedir(rep);
        }
    } else cout << "\n\n format de la commande : dir nom \n\n";
}
```