Les fichiers UNIX

Luigi Santocanale

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

13 octobre 2004

Plan

Plan

Les fichiers « normaux »

- Fichiers réguliers :
  stockage de l'information sur disque.
- Répertoires :
  organisation logique de l'information sur disque.
- Liens symboliques : partage des ressources.

```
[lsantoca@localhost lecture2]$ touch a b
[lsantoca@localhost lecture2]$ ln -s b c
[lsantoca@localhost lecture2]$ mkdir d
[lsantoca@localhost lecture2]$ ls -pdl a b c d
-rw-r--r--    1 lsantoca lsantoca        0 sep  3 15:09 a
-rw-r--r--    1 lsantoca lsantoca        0 sep  3 15:09 b
lrwxrwxrwx    1 lsantoca lsantoca        1 sep  3 15:09 c -> b
drwxr-xr-x    2 lsantoca lsantoca     4096 sep  3 15:10 d/
```

(Ajourdui)

Les fichiers « normaux »

- Fichiers réguliers :
  stockage de l'information sur disque.
- Répertoires :
  organisation logique de l'information sur disque.
- Liens symboliques : partage des ressources.

```
[lsantoca@localhost lecture2]$ touch a b
[lsantoca@localhost lecture2]$ ln -s b c
[lsantoca@localhost lecture2]$ mkdir d
[lsantoca@localhost lecture2]$ ls -pdl a b c d
-rw-r--r--    1 lsantoca lsantoca        0 sep  3 15:09 a
-rw-r--r--    1 lsantoca lsantoca        0 sep  3 15:09 b
lrwxrwxrwx    1 lsantoca lsantoca        1 sep  3 15:09 c -> b
drwxr-xr-x    2 lsantoca lsantoca     4096 sep  3 15:10 d/
```

(Ajourdui)

Les fichiers « normaux »

- Fichiers réguliers :
  stockage de l'information sur disque.
- Répertoires :
  organisation logique de l'information sur disque.
- Liens symboliques : partage des ressources.

```
[lsantoca@localhost lecture2]$ touch a b
[lsantoca@localhost lecture2]$ ln -s b c
[lsantoca@localhost lecture2]$ mkdir d
[lsantoca@localhost lecture2]$ ls -pdl a b c d
-rw-r--r--    1 lsantoca lsantoca        0 sep  3 15:09 a
-rw-r--r--    1 lsantoca lsantoca        0 sep  3 15:09 b
lrwxrwxrwx    1 lsantoca lsantoca        1 sep  3 15:09 c -> b
drwxr-xr-x    2 lsantoca lsantoca     4096 sep  3 15:10 d/
```

(Ajourdui)

Les fichiers « normaux »

- Fichiers réguliers :
  stockage de l'information sur disque.
- Répertoires :
  organisation logique de l'information sur disque.
- Liens symboliques : partage des ressources.

```
[lsantoca@localhost lecture2]$ touch a b
[lsantoca@localhost lecture2]$ ln -s b c
[lsantoca@localhost lecture2]$ mkdir d
[lsantoca@localhost lecture2]$ ls -pdl a b c d
-rw-r--r--    1 lsantoca lsantoca        0 sep  3 15:09 a
-rw-r--r--    1 lsantoca lsantoca        0 sep  3 15:09 b
lrwxrwxrwx    1 lsantoca lsantoca        1 sep  3 15:09 c -> b
drwxr-xr-x    2 lsantoca lsantoca     4096 sep  3 15:10 d/
```

(Ajourdui)

Pipes (tubes)

But : communication entre processus.

- tube nommé :
  il lui correspond une référence, accessible à tous les processus,
- tube anonyme :
  communication entre processus de la même famille.

```
[lsantoca@localhost lecture2]$ ls -lp tube
prw-r--r--    1 lsantoca lsantoca        0 sep  3 15:17 tube|
```

(Plus tard dans ce cours)

Pipes (tubes)

But : communication entre processus.

- tube nommé :
  il lui correspond une référence, accessible à tous les processus,
- tube anonyme :
  communication entre processus de la même famille.

```
[lsantoca@localhost lecture2]$ ls -lp tube
prw-r--r--    1 lsantoca lsantoca        0 sep  3 15:17 tube|
```

(Plus tard dans ce cours)

Fichier spéciaux dev

But : gestion des périphériques.

- bloc : utilisation des caches noyau,
- caractère : écriture/lecture immédiate.

```
[lsantoca@localhost lecture2]$ ls -lL /dev/tty /dev/ram0
brw-------    1 root     root       1,   0 jan  1  1970 /dev/ram0
crw-rw-rw-    1 root     root       5,   0 sep  3 10:39 /dev/tty
[lsantoca@localhost lecture2]$ echo "Coucou" > /dev/tty
Coucou
```

Remarque : 1,5 nombres majeurs, 0 nombre mineur.

Fichier spéciaux dev

But : gestion des périphériques.

- bloc : utilisation des caches noyau,
- caractère : écriture/lecture immédiate.

```
[lsantoca@localhost lecture2]$ ls -lL /dev/tty /dev/ram0
brw-------    1 root     root       1,   0 jan  1  1970 /dev/ram0
crw-rw-rw-    1 root     root       5,   0 sep  3 10:39 /dev/tty
[lsantoca@localhost lecture2]$ echo "Coucou" > /dev/tty
Coucou
```

Remarque : 1,5 nombres majeurs, 0 nombre mineur.

Fichier spéciaux dev

But : gestion des périphériques.

- bloc : utilisation des caches noyau,
- caractère : écriture/lecture immédiate.

```
[lsantoca@localhost lecture2]$ ls -lL /dev/tty /dev/ram0
brw-------    1 root     root       1,   0 jan  1  1970 /dev/ram0
crw-rw-rw-    1 root     root       5,   0 sep  3 10:39 /dev/tty
[lsantoca@localhost lecture2]$ echo "Coucou" > /dev/tty
Coucou
```

Remarque : 1,5 nombres majeurs, 0 nombre mineur.

Fichier spéciaux dev

But : gestion des périphériques.

- bloc : utilisation des caches noyau,
- caractère : écriture/lecture immédiate.

```
[lsantoca@localhost lecture2]$ ls -lL /dev/tty /dev/ram0
brw-------    1 root     root       1,   0 jan  1  1970 /dev/ram0
crw-rw-rw-    1 root     root       5,   0 sep  3 10:39 /dev/tty
[lsantoca@localhost lecture2]$ echo "Coucou" > /dev/tty
Coucou
```

Remarque : 1,5 nombres majeurs, 0 nombre mineur.

Fichier spéciaux dev

But : gestion des périphériques.

- bloc : utilisation des caches noyau,
- caractère : écriture/lecture immédiate.

```
[lsantoca@localhost lecture2]$ ls -lL /dev/tty /dev/ram0
brw-------    1 root     root        1,   0 jan  1  1970 /dev/ram0
crw-rw-rw-    1 root     root        5,   0 sep  3 10:39 /dev/tty
[lsantoca@localhost lecture2]$ echo "Coucou" > /dev/tty
Coucou
```

Remarque : `1,5` nombres majeurs, `0` nombre mineur.

Sockets

But : communication sur un réseau

```
[lsantoca@localhost lecture2]$ ls -lp /tmp/agent.2416
srwxrwxr-x    1 lsantoca lsantoca     0 déc 15  2003 /tmp/agent.2416
```

(Cours réseaux année prochain)

Plan

Structure hiérarchique : première approximation

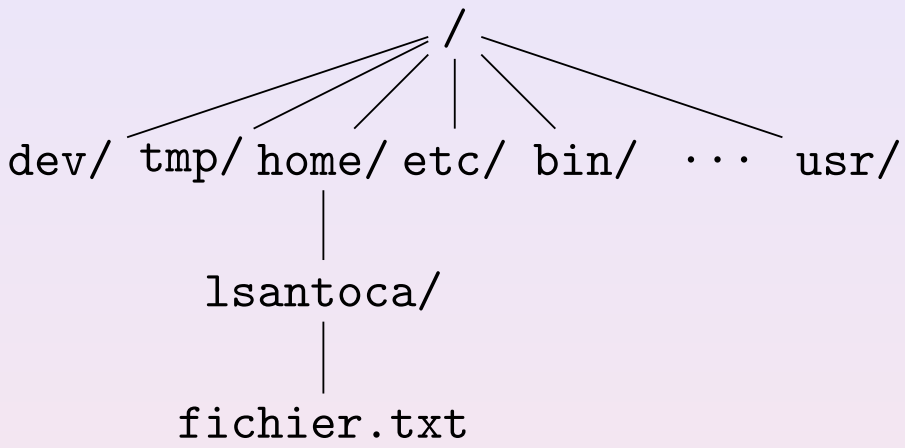
Les fichiers sont organisés en arbre :

- noeuds internes : répertoires,
- feuilles : fichiers réguliers, liens symboliques, tubes, sockets, fichiers dev.

Chaque fichier est dénoté par un chemin:

- à partir de la racine
- à partir du répertoire courante

Exemple




```
[lsantoca@localhost lsantoca]$ ls /home/lsantoca/fichier.txt
/home/lsantoca/fichier.txt
[lsantoca@localhost lsantoca]$ pwd
/home/lsantoca
[lsantoca@localhost lsantoca]$ ls fichier.txt
fichier.txt
```

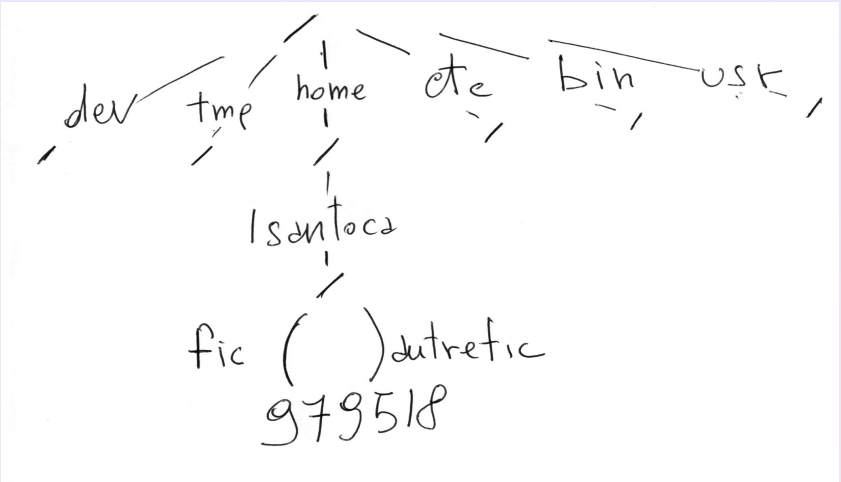
Structure hiérarchique : deuxième approximation

Les fichiers sont organisés en DAG (directed acyclig graph) :

- noeud interne : répertoire (i-noeud,contenu),
- arêtes : contenu d'un répertoire
- feuilles : fichiers sur disque (i-noeuds,contenu).

- Une feuille peut avoir plusieurs parents
- Plusieurs chemins aux même i-noeud

Les liens durs




```
[lsantoca@localhost lecture2]$ touch fic
[lsantoca@localhost lecture2]$ ln fic autrefic
[lsantoca@localhost lecture2]$ stat fic
...
Device: 308h/776d       Inode: 979518      Links: 2
...
[lsantoca@localhost lecture2]$ stat autrefic
...
Device: 308h/776d       Inode: 979518      Links: 2
...
```

Plan

Disques physiques et disques logiques

Disques physiques :

floppies, disques durs, etc.

Manipulation de :

cylindres, pistes, déplacement de la tête de lecture

gérée par le pilote de périphérique.

Disques logiques :

- disques « swap »:
  contiennent les processus endormis, hors de mémoire vive.
- disque logique de fichiers normaux.

Manipulation de :

noeuds d'information (i-node), blocs,
transformations de l'interface utilisateur

gérée par le système de gestion des fichiers (SGF).

Disques physiques et disques logiques

Disques physiques :

floppies, disques durs, etc.

Manipulation de :

cylindres, pistes, déplacement de la tête de lecture

gérée par le pilote de périphérique.

Disques logiques :

- disques « swap »:
  contiennent les processus endormis, hors de mémoire vive.
- disque logique de fichiers normaux.

Manipulation de :

noeuds d'information (i-node), blocs,
transformations de l'interface utilisateur

gérée par le système de gestion des fichiers (SGF).

Disques physiques et disques logiques

Disques physiques :

floppies, disques durs, etc.

Manipulation de :

cylindres, pistes, déplacement de la tête de lecture

gérée par le pilote de périphérique.

Disques logiques :

- disques « swap »:
  contiennent les processus endormis, hors de mémoire vive.
- disque logique de fichiers normaux.

Manipulation de :

noeuds d'information (i-node), blocs,
transformations de l'interface utilisateur

gérée par le système de gestion des fichiers (SGF).

Disques physiques et disques logiques

Disques physiques :

floppies, disques durs, etc.

Manipulation de :

cylindres, pistes, déplacement de la tête de lecture

gérée par le pilote de périphérique.

Disques logiques :

- disques « swap »:
  contiennent les processus endormis, hors de mémoire vive.
- disque logique de fichiers normaux.

Manipulation de :

noeuds d'information (i-node), blocs,
transformations de l'interface utilisateur

gérée par le système de gestion des fichiers (SGF).

Disques physiques et disques logiques

Disques physiques :

floppies, disques durs, etc.

Manipulation de :

cylindres, pistes, déplacement de la tête de lecture

gérée par le pilote de périphérique.

Disques logiques :

- disques « swap »:
  contiennent les processus endormis, hors de mémoire vive.
- disque logique de fichiers normaux.

Manipulation de :

noeuds d'information (i-node), blocs,
transformations de l'interface utilisateur

gérée par le système de gestion des fichiers (SGF).

Disques physiques et disques logiques

Disques physiques :

floppies, disques durs, etc.

Manipulation de :

cylindres, pistes, déplacement de la tête de lecture

gérée par le pilote de périphérique.

Disques logiques :

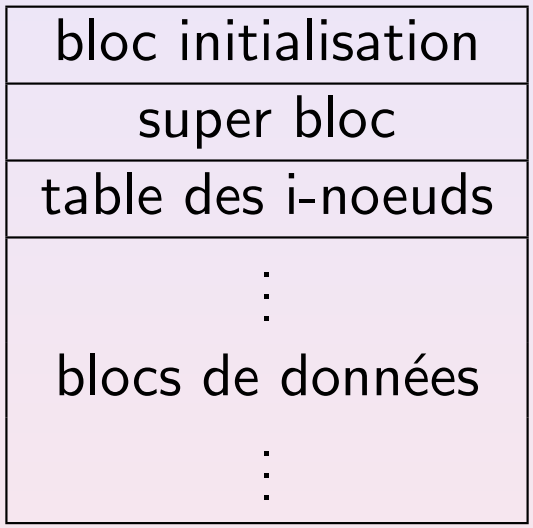
- disques « swap »:
  contiennent les processus endormis, hors de mémoire vive.
- disque logique de fichiers normaux.

Manipulation de :

noeuds d'information (i-node), blocs,
transformations de l'interface utilisateur

gérée par le système de gestion des fichiers (SGF).

Les SGF System V



utilisé au chargement du système

informations générales sur le SGF

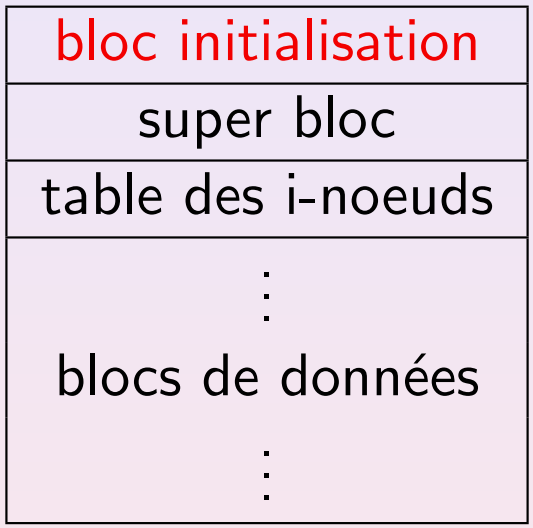
chaque i-noeud:

- type et droits
- id du propriétaire et du groupe
- nombre lien physiques
- taille
- dates dernière consultation/modification des données, modification du noeud
- 10 adresses de blocs directes, 1 indirecte simple, 1 indirecte double, 1 indirecte triple

Répertoires : enregistrements taille fixe 16 caractères, liens 14 caractères

chaque bloc : 512/1024/2048/4096 octets

Les SGF System V



utilisé au chargement du système

informations générales sur le SGF

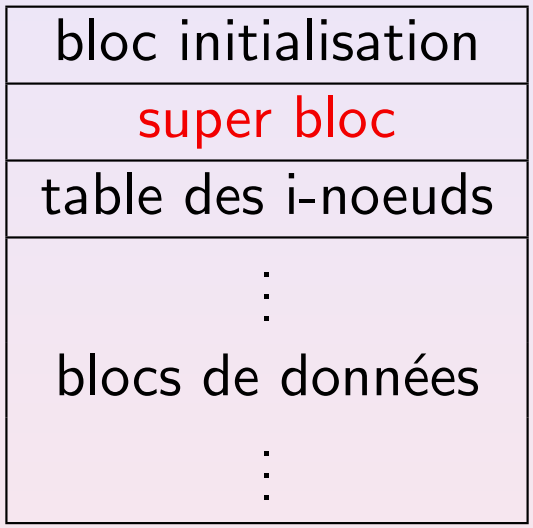
chaque i-noeud:

- type et droits
- id du propriétaire et du groupe
- nombre lien physiques
- taille
- dates dernière consultation/modification des données, modification du noeud
- 10 adresses de blocs directes, 1 indirecte simple, 1 indirecte double, 1 indirecte triple

Répertoires : enregistrements taille fixe 16 caractères, liens 14 caractères

chaque bloc : 512/1024/2048/4096 octets

Les SGF System V



utilisé au chargement du système

informations générales sur le SGF

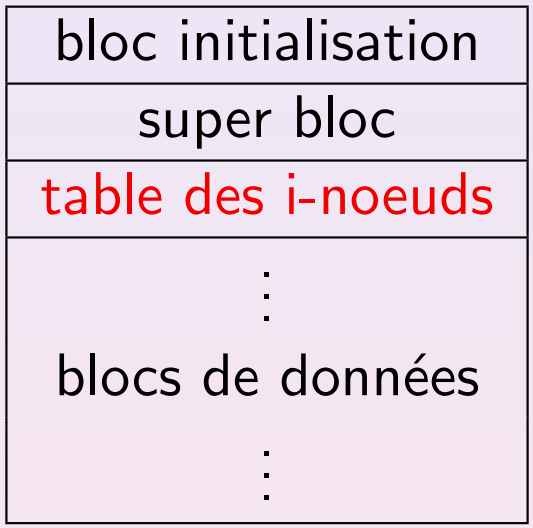
chaque i-noeud:

- type et droits
- id du propriétaire et du groupe
- nombre lien physiques
- taille
- dates dernière consultation/modification des données, modification du noeud
- 10 adresses de blocs directes, 1 indirecte simple, 1 indirecte double, 1 indirecte triple

Répertoires :
enregistrements taille fixe 16 caractères, liens 14 caractères

chaque bloc : 512/1024/2048/4096 octets

Les SGF System V



utilisé au chargement du système

informations générales sur le SGF

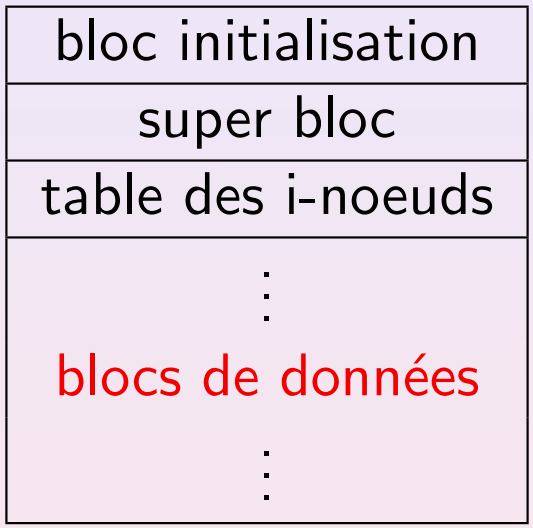
chaque i-noeud:

- type et droits
- id du propriétaire et du groupe
- nombre lien physiques
- taille
- dates dernière consultation/modification des données, modification du noeud
- 10 adresses de blocs directes, 1 indirecte simple, 1 indirecte double, 1 indirecte triple

Répertoires : enregistrements taille fixe 16 caractères, liens 14 caractères

chaque bloc : 512/1024/2048/4096 octets

Les SGF System V



utilisé au chargement du système

informations générales sur le SGF

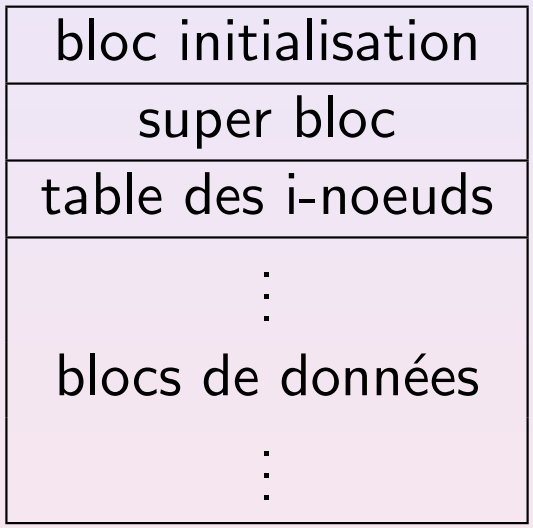
chaque i-noeud:

- type et droits
- id du propriétaire et du groupe
- nombre lien physiques
- taille
- dates dernière consultation/modification des données, modification du noeud
- 10 adresses de blocs directes, 1 indirecte simple, 1 indirecte double, 1 indirecte triple

Répertoires : enregistrements taille fixe 16 caractères, liens 14 caractères

chaque bloc : 512/1024/2048/4096 octets

Les SGF System V



utilisé au chargement du système

informations générales sur le SGF

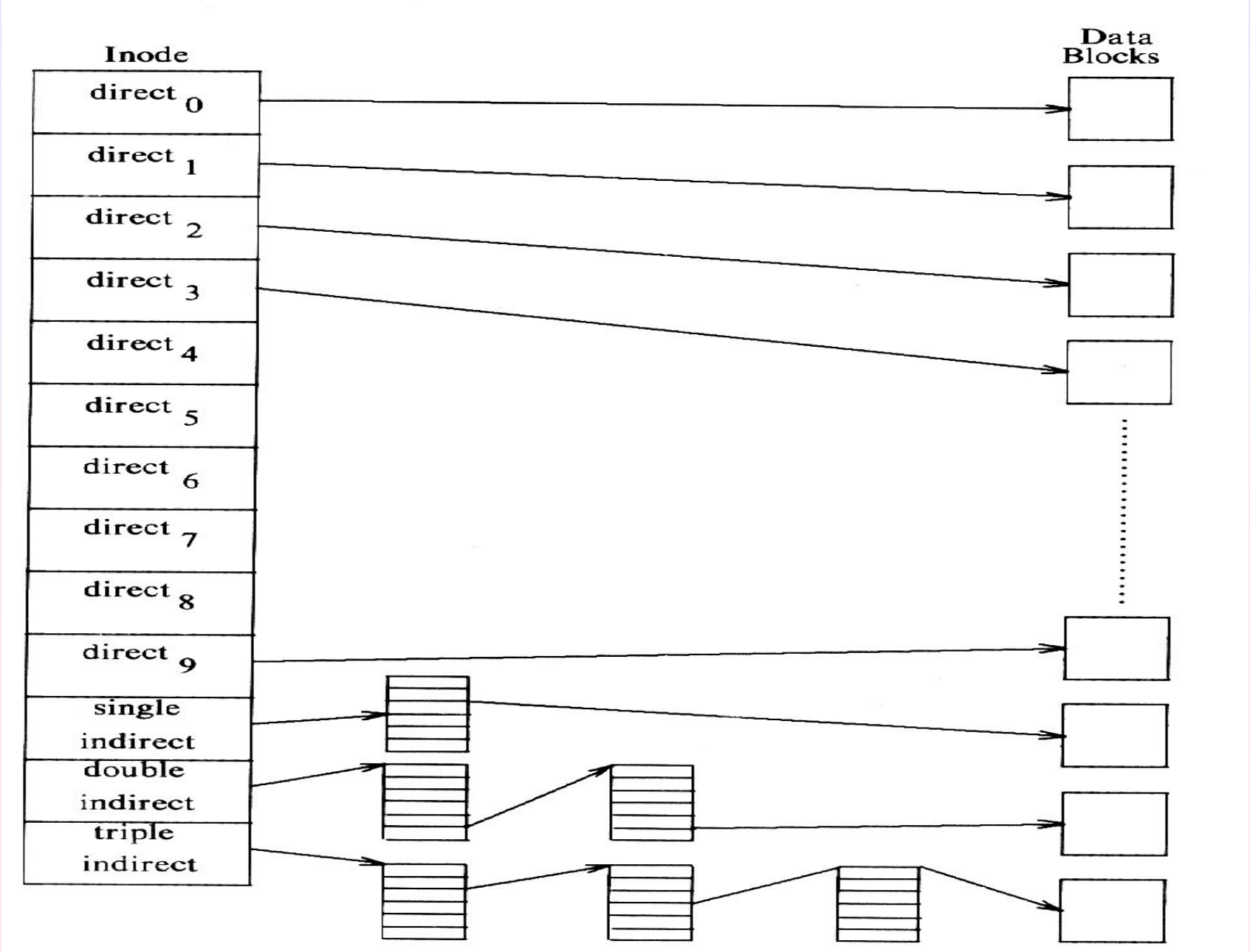
chaque i-noeud:

- type et droits
- id du propriétaire et du groupe
- nombre lien physiques
- taille
- dates dernière consultation/modification des données, modification du noeud
- 10 adresses de blocs directes, 1 indirecte simple, 1 indirecte double, 1 indirecte triple

Répertoires :
enregistrements taille fixe 16 caractères, liens 14 caractères

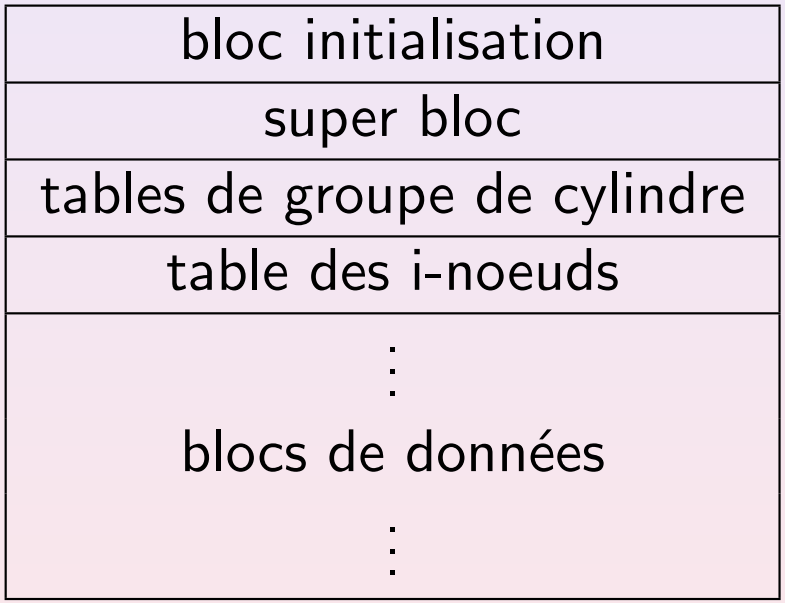
chaque bloc : 512/1024/2048/4096 octets

Adresses de blocs indirectes (Bach 1986)



Plan

Les SGF ffs/ufs (BSD)



optimisation allocation des blocs
par rapport à la tête du disque

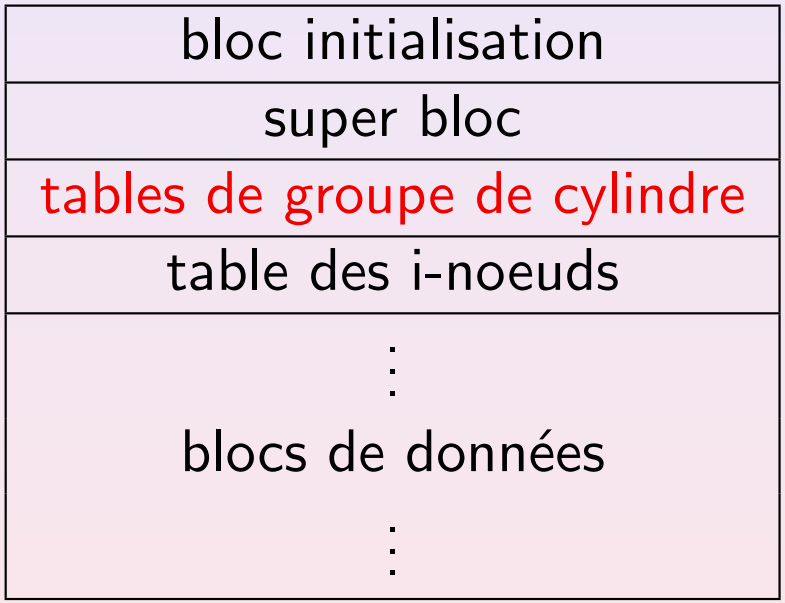
chaque i-noeud:

- 12 adresses directes, 1 indirecte simple, 2 indirectes doubles

blocs de taille 4K ou 8K,
fragmentation des blocs

Répertoires :
enregistrements taille variable,
liens taille max 255 caractères

Les SGF ffs/ufs (BSD)



optimisation allocation des blocs
par rapport à la tête du disque

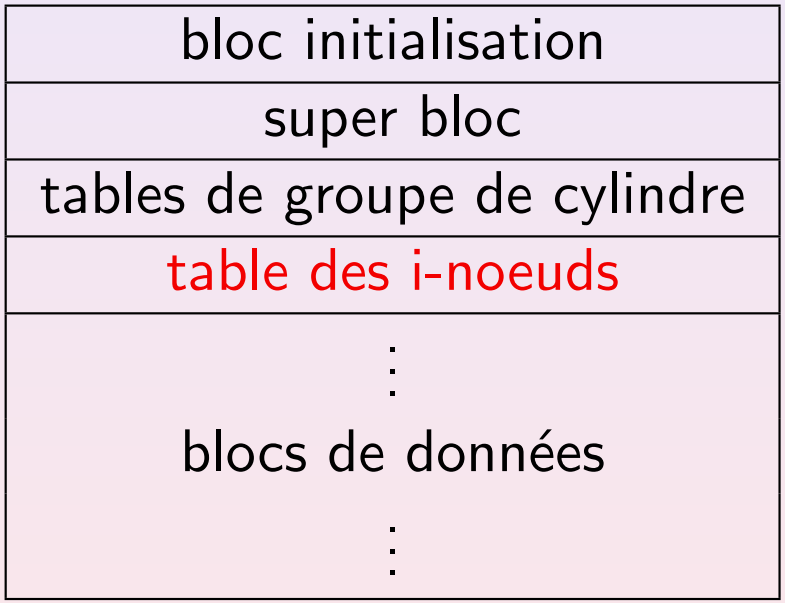
chaque i-noeud:

- 12 adresses directes, 1 indirecte simple, 2 indirectes doubles

blocs de taille 4K ou 8K,
fragmentation des blocs

Répertoires :
enregistrements taille variable,
liens taille max 255 caractères

Les SGF ffs/ufs (BSD)



optimisation allocation des blocs
par rapport à la tête du disque

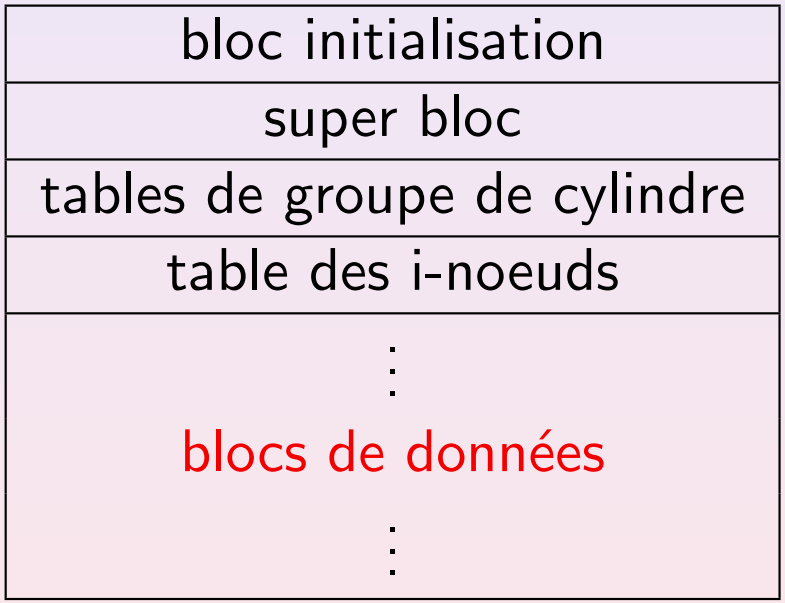
chaque i-noeud:

- 12 adresses directes, 1 indirecte simple, 2 indirectes doubles

blocs de taille 4K ou 8K,
fragmentation des blocs

Répertoires :
enregistrements taille variable,
liens taille max 255 caractères

Les SGF ffs/ufs (BSD)



optimisation allocation des blocs
par rapport à la tête du disque

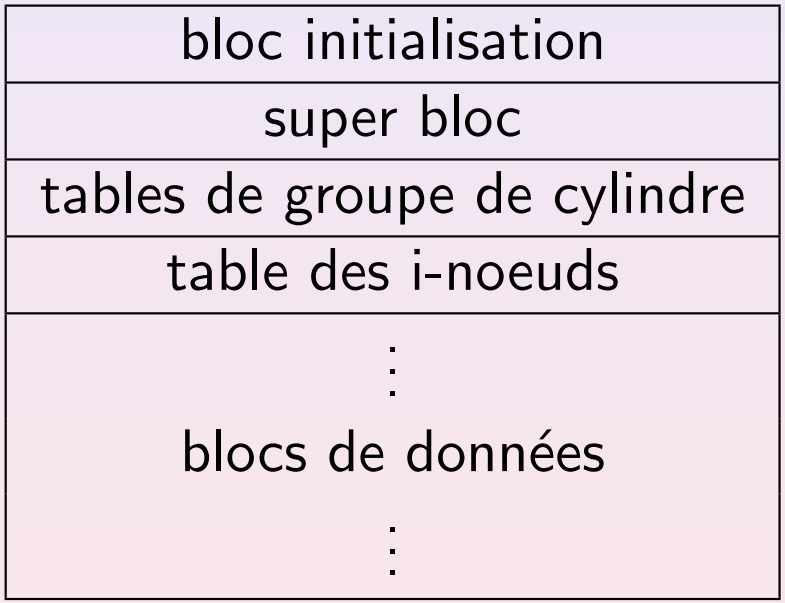
chaque i-noeud:

- 12 adresses directes, 1 indirecte simple, 2 indirectes doubles

blocs de taille 4K ou 8K,
fragmentation des blocs

Répertoires :
enregistrements taille variable,
liens taille max 255 caractères

Les SGF ffs/ufs (BSD)



optimisation allocation des blocs
par rapport à la tête du disque

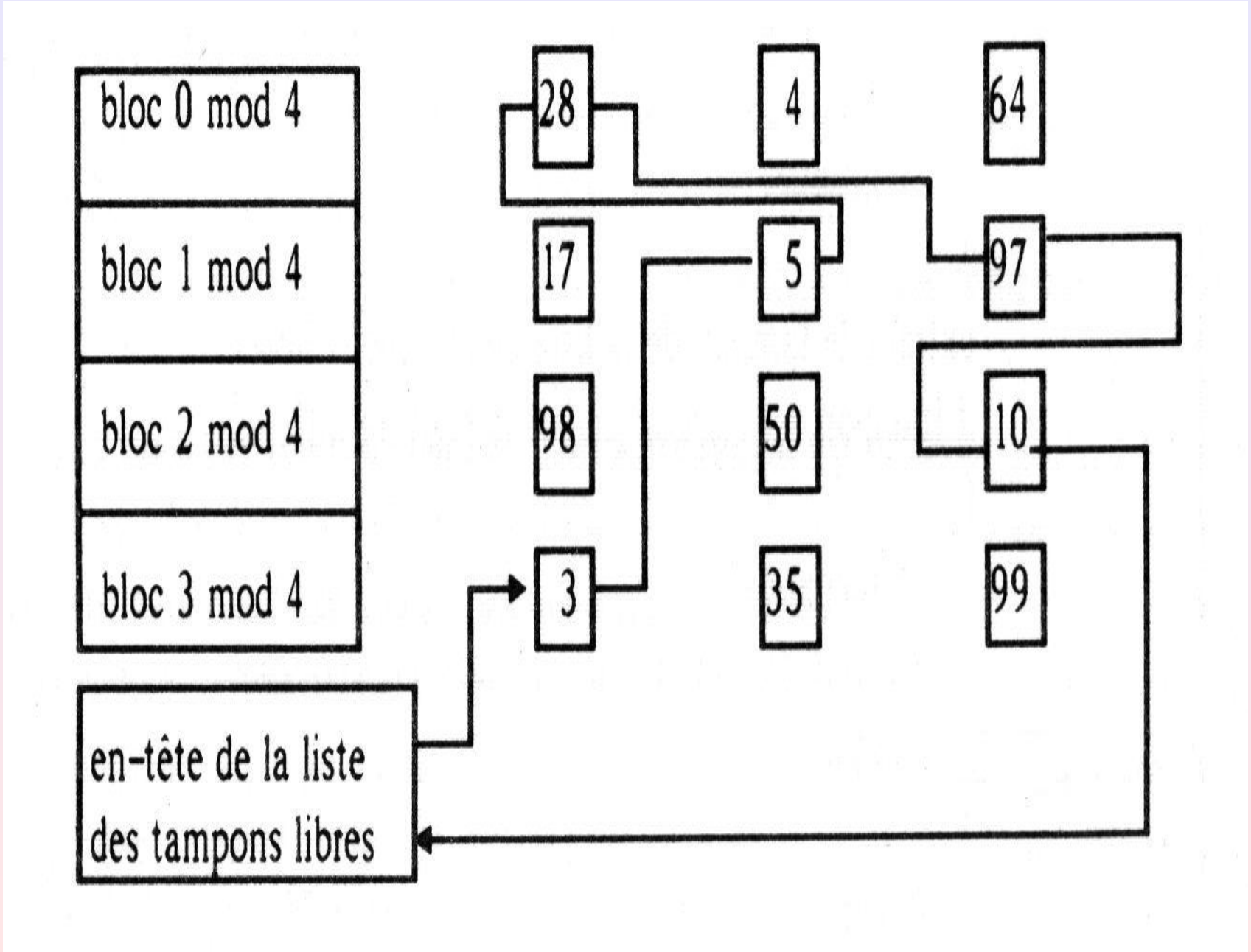
chaque i-noeud:

- 12 adresses directes, 1 indirecte simple, 2 indirectes doubles

blocs de taille 4K ou 8K,
fragmentation des blocs

Répertoires :
enregistrements taille variable,
liens taille max 255 caractères

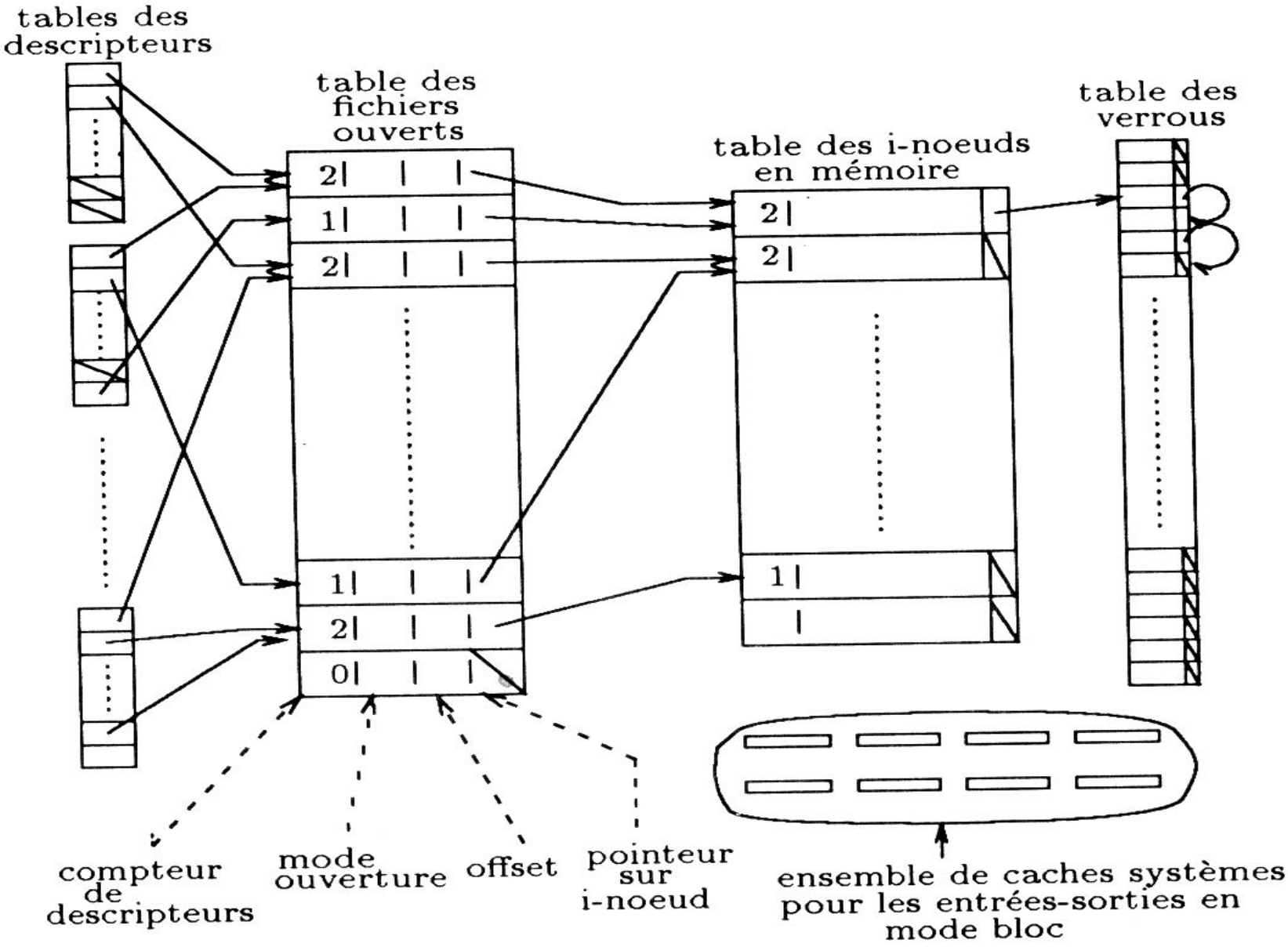
Les caches (Bach 1986)



Plan

Tables du système

(Rifflet 1999)



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.
- Table des i-noeuds en cache (noyau):
  - nombre total d'ouvertures,
  - id du disque logique,
  - numéro du noeud sur ce disque,
  - état du 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.
- Table des i-noeuds en cache (noyau):
  - nombre total d'ouvertures,
  - id du disque logique,
  - numéro du noeud sur ce disque,
  - état du 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.
- Table des i-noeuds en cache (noyau):
  - nombre total d'ouvertures,
  - id du disque logique,
  - numéro du noeud sur ce disque,
  - état du 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.
- Table des i-noeuds en cache (noyau):
  - nombre total d'ouvertures,
  - id du disque logique,
  - numéro du noeud sur ce disque,
  - état du 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.
- Table des i-noeuds en cache (noyau):
  - nombre total d'ouvertures,
  - id du disque logique,
  - numéro du noeud sur ce disque,
  - état du 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.
- Table des i-noeuds en cache (noyau):
  - nombre total d'ouvertures,
  - id du disque logique,
  - numéro du noeud sur ce disque,
  - état du 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.
- Table des i-noeuds en cache (noyau):
  - nombre total d'ouvertures,
  - id du disque logique,
  - numéro du noeud sur ce disque,
  - état du 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.
- Table des i-noeuds en cache (noyau):
  - nombre total d'ouvertures,
  - id du disque logique,
  - numéro du noeud sur ce disque,
  - état du 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.
- Table des i-noeuds en cache (noyau):
  - nombre total d'ouvertures,
  - id du disque logique,
  - numéro du noeud sur ce disque,
  - état du 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.
- Table des i-noeuds en cache (noyau):
  - nombre total d'ouvertures,
  - id du disque logique,
  - numéro du noeud sur ce disque,
  - état du 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.
- Table des i-noeuds en cache (noyau):
  - nombre total d'ouvertures,
  - id du disque logique,
  - numéro du noeud sur ce disque,
  - état du noeud.

Plan

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, fstat

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

int stat (const char * *ref*, struct stat * *ptr_stat*)

ref : chemin au fichier

ptr_stat : pointeur à une structure stat à remplir

Retourne : Toutes ces primitives retournent 0 si succès, -1 échec

int fstat (int *desc*, struct stat * *ptr_stat*)

desc : descripteur du fichier

ptr_stat : pointeur à une structure stat à remplir

stat, fstat

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

int stat (const char * *ref*, struct stat * *ptr_stat*)

ref : chemin au fichier

ptr_stat : pointeur à une structure stat à remplir

Retourne : Toutes ces primitives retournent 0 si succès, -1 échec

int fstat (int *desc*, struct stat * *ptr_stat*)

desc : descripteur du fichier

ptr_stat : pointeur à une structure stat à remplir

stat, fstat

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

int stat (const char * *ref*, struct stat * *ptr_stat*)

ref : chemin au fichier

ptr_stat : pointeur à une structure stat à remplir

Retourne : Toutes ces primitives retournent 0 si succès, -1 échec

int fstat (int *desc*, struct stat * *ptr_stat*)

desc : descripteur du fichier

ptr_stat : 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 && errno == EISDIR)
11 :     printf("Répertoire.\n");
12 :   else
13 :     printf("Pas un répertoire.\n");
14 :   exit(EXIT_SUCCESS);
15 : }
```

Programme : estrep2.c

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

access

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

ref : chemin au fichier

type_acces : |de `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 type_acces)
```

ref : chemin au fichier

type_acces : |de

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 type_acces)
```

ref : chemin au fichier

type_acces : |de

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 :

- ref1 n'est pas un repertoire,
- ref2 n'existe pas,
- ref2 corresponde au même disque logique que 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 :

- ref1 n'est pas un repertoire,
- ref2 n'existe pas,
- ref2 corresponde au même disque logique que 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 :

- ref1 n'est pas un repertoire,
- ref2 n'existe pas,
- ref2 corresponde au même disque logique que 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 :

- ref1 n'est pas un repertoire,
- ref2 n'existe pas,
- ref2 corresponde au même disque logique que 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 lien physiques sur ce inode est > 0, ou
- le nombre d'ouverture de ce fichier est > 0.

int rename (`char * vieux, char * nouveau`)

vieux : vieux chemin

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

int rename (`char * vieux, char * nouveau`)

vieux : vieux chemin

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

int rename (`char *` *vieux*, `char *` *nouveau*)

vieux : vieux chemin

nouveau : nouveau chemin

Remarques : atomique.

chmod, chown

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

int chmod (char * *ref*, mode_t *mode_acces*)

ref : chemin au fichier

mode_acces : mode d'acces

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_acces*)

ref : chemin au fichier

mode_acces : mode d'acces

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_acces*)

ref : chemin au fichier

mode_acces : mode d'acces

int chown (char * *ref*, uid_t *id_proprietaire*, gid_t *id_groupe*)

ref : chemin au fichier

Plan

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 : Pointer, 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 : Pointer, 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 : Pointer, 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 : Pointer, 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 : Pointer, 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 :   }
12 :
```

Programme : repertoires.c

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