Système de gestion des fichiers

Système d'exploitation

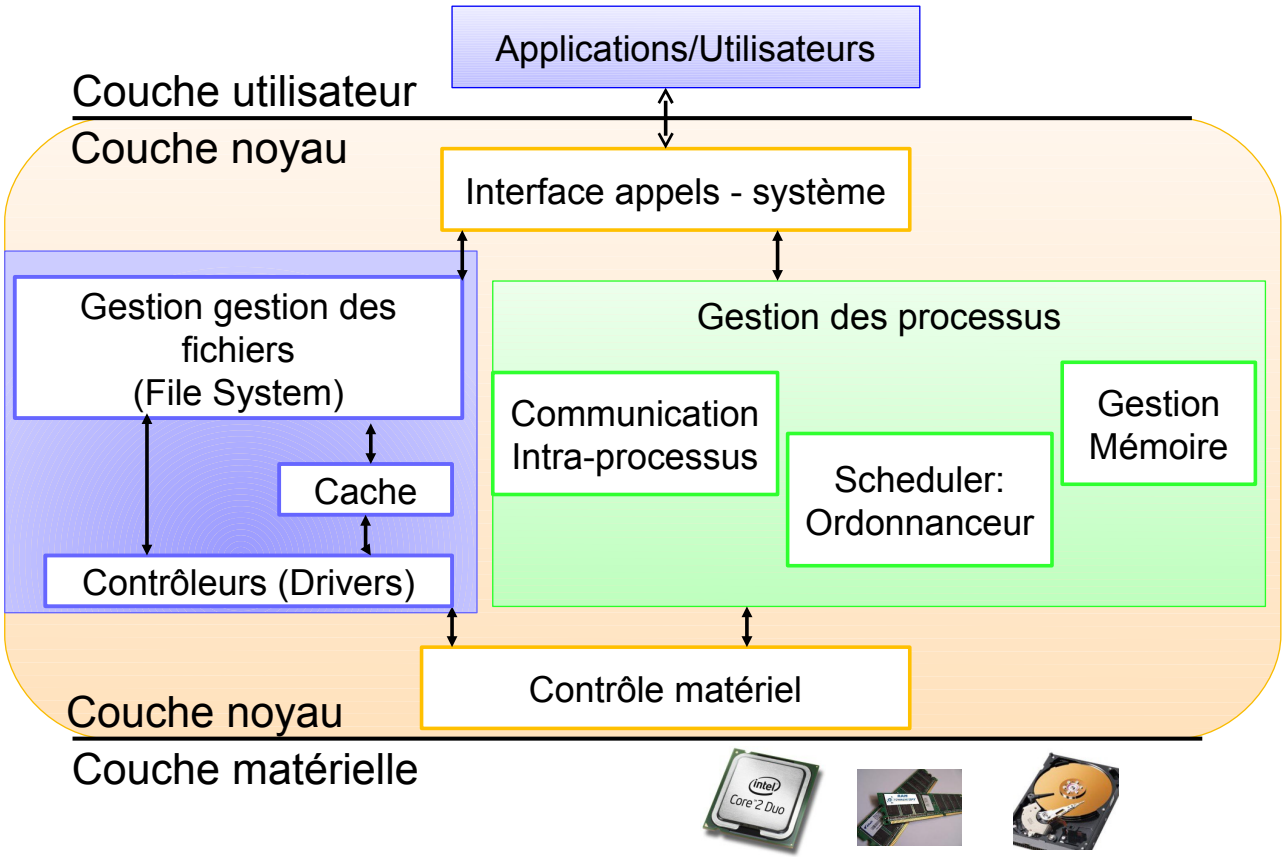
Sylvain Chevallier/Nicoleta Preda

IUT de Vélizy
Département IGI

Définition : Système d'exploitation

- Interface entre l'utilisateur et l'ordinateur
- Allocateur et gestionnaire des ressources

L'architecture du noyau



Plusieurs système(s) de fichiers sur la même machine

- Chaque disque est partagé en partitions (ou volumes)
- A chaque partition est associée un système de fichiers : e.g. MSDOS, ext2, ext3, HFS, NTFS, FAT32, ISO9660, UDF)
- Le système de fichiers peut être disponible (monté dans l'arborescence) ou indisponible

Nota bene :

Il y a une seule arborescence (une racine) par machine UNIX !

Système de fichiers

Assure :

- L'accès facile aux données
- Le passage à l'échelle

L'unité de base : le fichier

Plan

1. Le fichier
   - Types de fichiers
   - Organisation générale
   - Manipulation de fichiers
   - Le dossier
2. Gestionnaire du cache
3. La structure du système de fichiers

Le fichier

L'unité fondamentale d'un système de fichiers

Dans les SE classiques il y a deux abstractions :

- Le fichier pour les données
- Le processus pour le actions

Dans le monde UNIX : tout est fichier (ou presque)

- Les processus & les fichiers :
  mêmes interface, mêmes opérations

Le système de fichier /proc

- Ne correspond pas à un emplacement physique
- Interface avec le noyau
- Accès aux données sous forme de fichiers virtuels

```
$ ls /proc
1       951          config.gz  execdomains  irq
1039    957          cpuinfo    fb           kallsyms
10605   998          crypto     filesystems  kcore
10917   acpi         devices    fs           kmsg
11      asound       diskstats  i8k          kpagecount
```

- Il y a un répertoire par processus en cours d'exécution
- Contient la table des fichiers ouverts, la zone d'espace mémoire, ...

Plan

1. Le fichier
   - Types de fichiers
   - Organisation générale
   - Manipulation de fichiers
   - Le dossier
2. Gestionnaire du cache
3. La structure du système de fichiers

Exemples des fichiers

- Fichiers simples (regular files) : ascii, binaires, archives
- Répertoires (directories)
- Liens (links)
- Tuyaux (pipelines/FIFOS)
- Dispositifs type caractère/block (char/block devices)
- Sockets

Tuyaux FIFO & Sockets

Tuyaux FIFO (pipe)

- mécanisme de communication intra-processus
- un processus écrit, et un autre processus lit

```
$ ls -l  some-pipe
prw-r--r-- 1 bob bob 0 2008-03-03 21:14 some-pipe
```

Sockets

- mécanisme de communication intra-processus
- un socket a associé une entré dans le système de fichiers

```
$ ls -l /var/run/cups/cups.sock
srwxrwxrwx 1 root root 0 2008-03-03 11:10 /var/run/cups/cups.sock
```

Dispositifs type caractère/block :

Type caractère (char device)

- Accès séquentiel
- Associé aux dispositifs série e.g. souris, clavier

```
$ ls -l /dev/ttyS0
crw-rw---- 1 root dialout 4, 64 Feb 26 2005 /dev/ttyS0
```

Type bloc (block device)

- Accès direct (aléatoire)
- Disques, CD-ROM, USB Flash Drive

```
$ ls -l
/dev/sda brw-rw---- 1 root disk 8, 0 Feb 26 2005 /dev/sda
```

Les fichiers spéciaux dans /dev

Chaque périphérique ou partition a associé un fichier dans /dev

Plan

1. Le fichier
   - Types de fichiers
   - Organisation générale
   - Manipulation de fichiers
   - Le dossier
2. Gestionnaire du cache
3. La structure du système de fichiers

SF : deux points de vue

P. d. V. Utilisateur :

- Structure hiérarchique des répertoires & fichiers
- Interface d'accès à l'information

P. d. V. SF :

- Structures de données sur le disque
- L'unité de base : le bloc
- Translation ficher $\leftrightarrow$ support physique

  1 fichier logique = inode + des blocs de données

inode

A tout fichier est associé une structure de données (sur disque) appelée inode (information node)
Dans un inode se trouvent différentes informations :

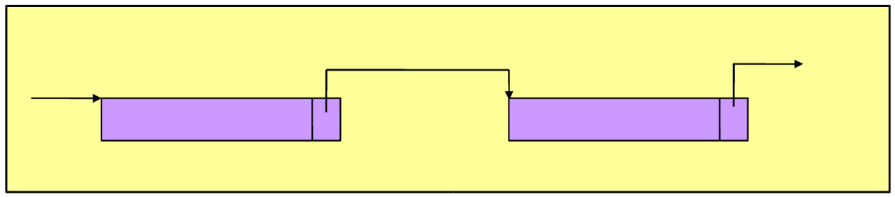
- le type de fichier (fichier standard, lien symbolique, répertoire)
- les droits d'accès
- le propriétaire (UID), le groupe propriétaire (GID)
- le nombre de références
- la taille
- la date de dernier accès en écriture
- la date de dernier accès en lecture
- l'adresse sur le disque du premier bloc de données

Toutes les informations fournies par ls -l, sauf le nom

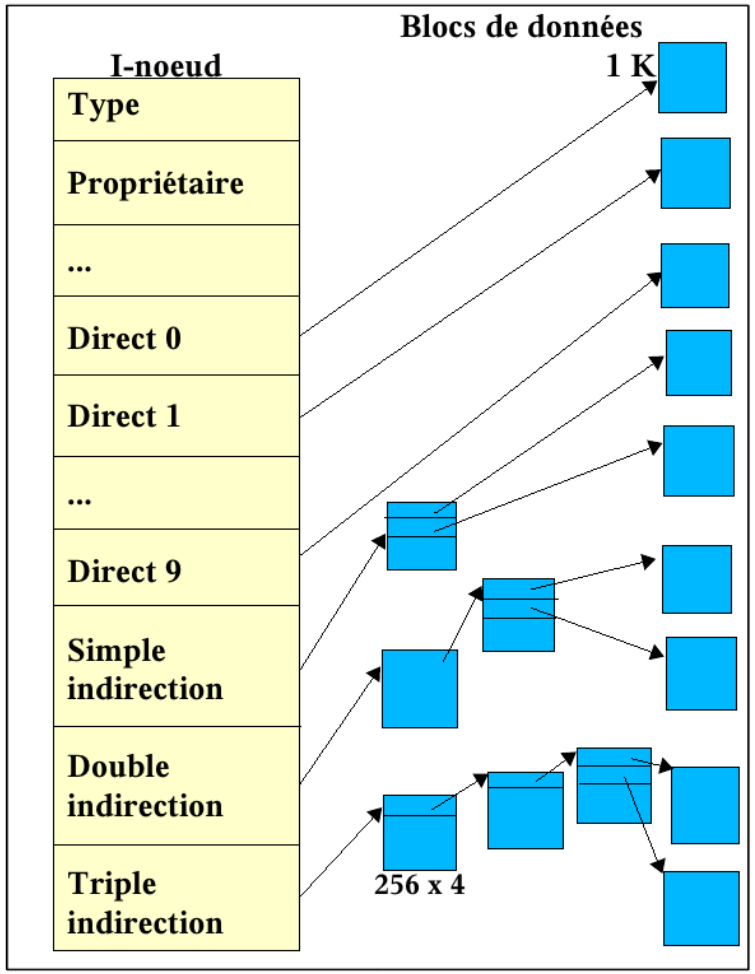
Le fichier sur le disque

- 1 fichier logique = inode + des blocs de données.
- Le nom de fichier se trouve dans le répertoire père !

- Le inode se trouve dans un bloc qui ne contient que des inodes.
- Chaque inode a un numéro unique (inode number).
- Les blocs de données sont répartis aléatoirement sur le disque.
- Ils sont référencés par :
  - Liste chaînée



  - Table d'index



Récupération des informations de l'inode

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

ou `fd` : le descripteur de fichier (voir les transparents suivants)

La structure stat contient :

`dev_t st_dev`	N° du périphérique contenant le fichier.
`ino_t st_ino`	N° d'inode.
`mode_t st_mode`	Type du fichier et droits d'accès.
`nlink_t st_nlink`	Nombre de liens (links).
`uid_t st_uid`	Identificateur de l'utilisateur du fichier.
`gid_t st_gid`	Identificateur de groupe du fichier.
`off_t st_size`	Taille en octets du fichier.
`time_t st_atime`	Heure du dernier accès.
`time_t st_mtime`	Heure de la dernière modification.
`time_t st_ctime`	Heure de la dernière modification de l'état,

Plan

1. Le fichier
   - Types de fichiers
   - Organisation générale
   - Manipulation de fichiers
   - Le dossier
2. Gestionnaire du cache
3. La structure du système de fichiers

Manipulation de fichiers : toujours dans un processus !

Opérations élémentaires : ouverture, lecture, écriture, ...

Shell
Bibliothèque C ISO/ANSI (générique) : `stdio.h`
Bibliothèque système (Unix/Linux) : `fcntl.h`, `unistd.h`

- Ouvrir un fichier avec `open` ou `creat`
- Réaliser des E/S avec `read` et `write`
- Se positionner avec `lseek`
- Fermer un fichier avec `close`

Création d'un fichier

Shell

touch /path/to/file
Observation : les commandes qui écrivent le fichier le crée également

ISO/ANSI C

```
FILE *f = fopen ("/path/to/file", "rw") ;
```

Système Unix/Linux

```
int fd = open ("/path/to/file", O_CREAT | O_EXCL, 0644) ;
```

Windows

```
HANDLE fileHandle = CreateFile ("/path/to/file",...,CREATE_NEW) ;
```

Ouverture d'un fichier

ISO/ANSI CFILE

*f = fopen("/path/to/file", "rt") ;

Unix

int fd = open("/path/to/file", `O_RDONLY`) ;

Windows

HANDLE f = CreateFile("/path/to/file", ..., `OPEN_EXISTING`) ;

Appel systèmes Unix : open

```
int open(const char *pathname, int flags);
int open(const char *pathname, int flags, mode_t mode);
```

pathname Nom du fichier (avec ou sans chemin)

flags Type d'ouverture + options

mode Droits d'accès du fichier crée

valeur de retour Descripteur du fichier (`int`)

Flags :

- Types d'ouverture : `O_RDONLY`, `O_WRONLY`, ou `O_RDWR`
- `O_APPEND` Positionnement en fin de fichier
- `O_CREAT` Création du fichier si besoin
- `O_TRUNC` Écrasement du fichier

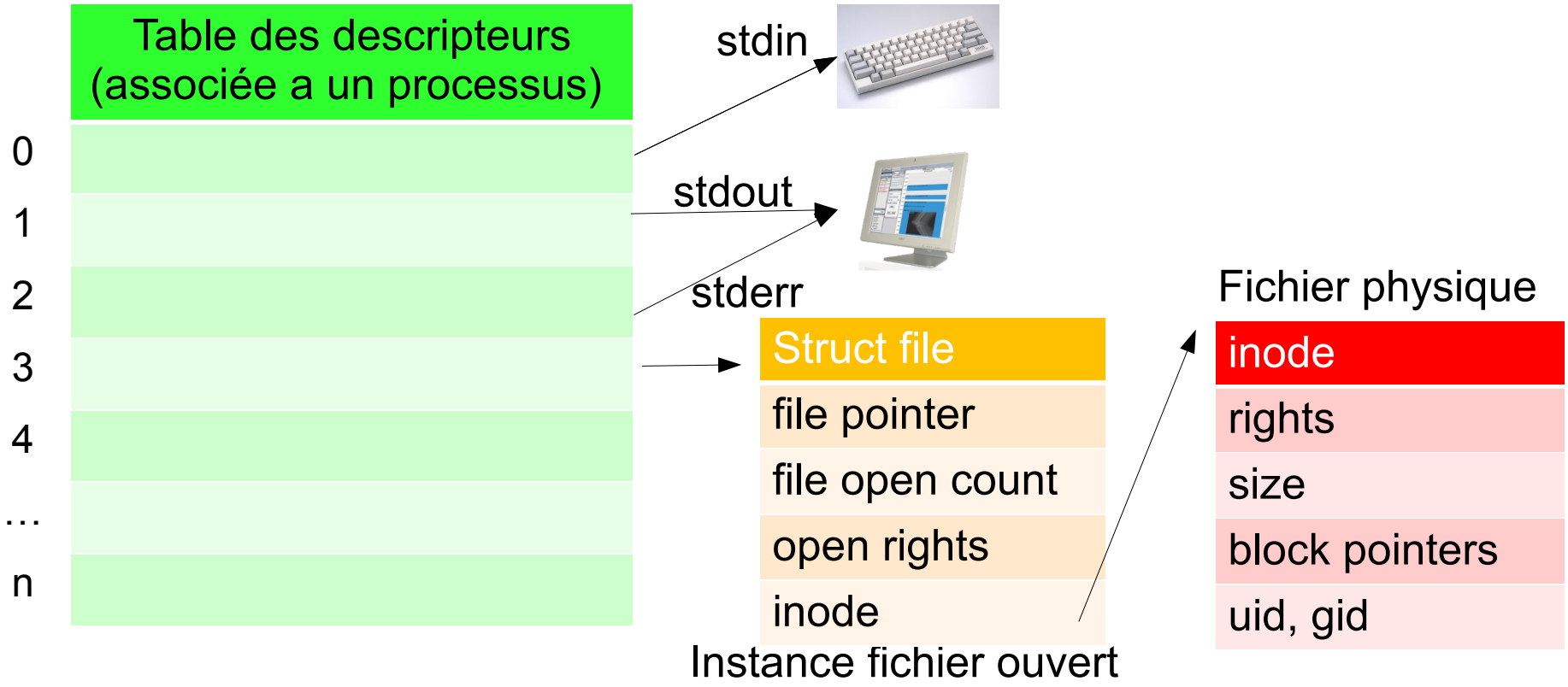
Exemple d'utilisation d'open

```c
#include <stdio.h>
#include <fcntl.h>

int main (int argc, char **argv) {
    int fd ;
    if (argc < 2) {
        fprintf(stderr," ERREUR : aucun paramètre\n");
        return 1;
    }
    fd=open(argv[1],O_WRONLY|O_CREAT|O_TRUNC,0644);
    if (fd == -1) {
        perror ("Ouverture");
        return 2;
    }
}
```

Descripteur de fichier

- L'ouverture d'un fichier se fait dans un processus
- Chaque processus a une table de descripteurs (de fichiers)
- descripteur de fichier = indice dans la table de descripteurs



Descripteurs spéciaux de fichier

Chaque descripteur de fichier est propre à un processus
Il existe 3 descripteurs spéciaux :

- `0` : entrée standard (stdin)
- `1` : sortie standard (stdout)
- `2` : sortie d'erreur (stderr)

Lors de la création d'un processus (`fork`), le fils hérite de la table de descripteurs du père.

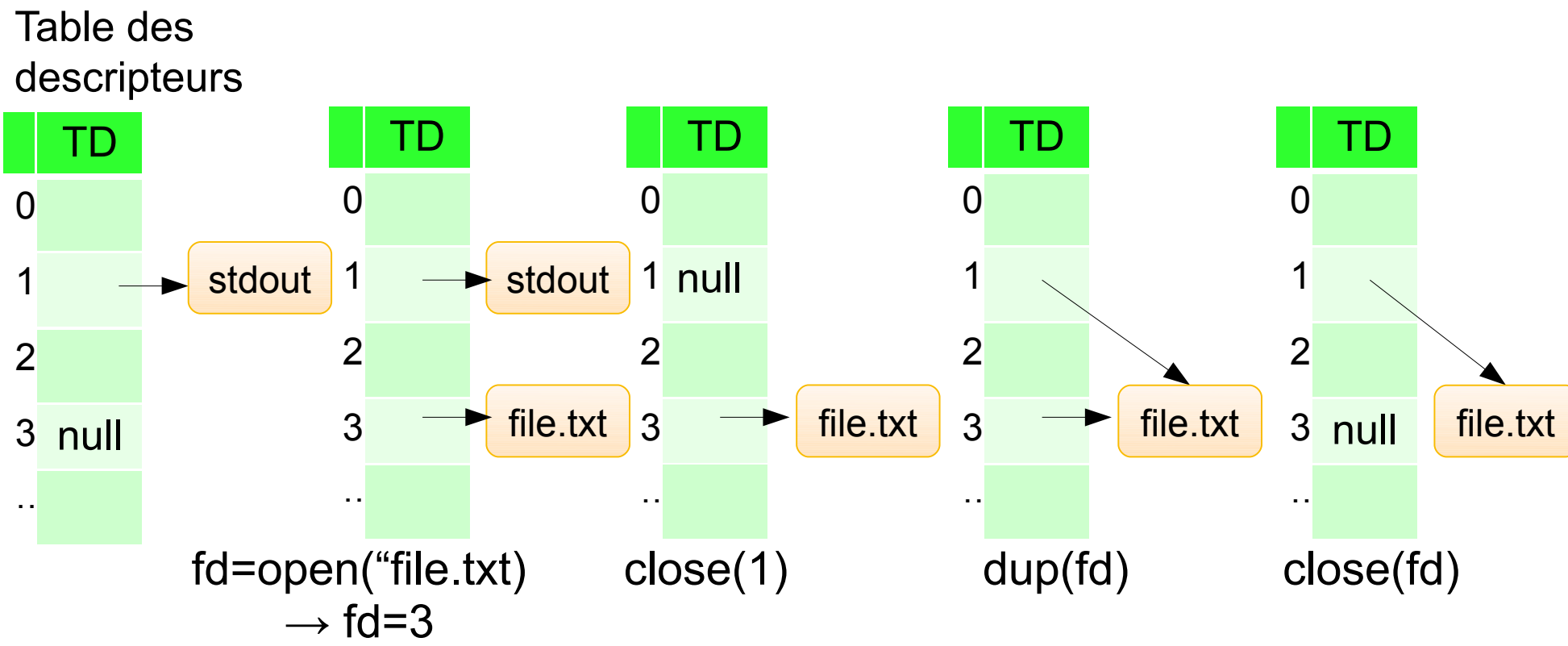
La duplication d'un descripteur

```
int newfd = dup(oldfd);
dup2(oldfd, newfd);
```

- La duplication d'un descripteur dans un autre descripteur
  $\rightarrow$ Les deux descripteurs travaillent sur le même fichier.
- Le système la première entré disponible dans la table des descripteurs : l'entré avec le plus petit indice.

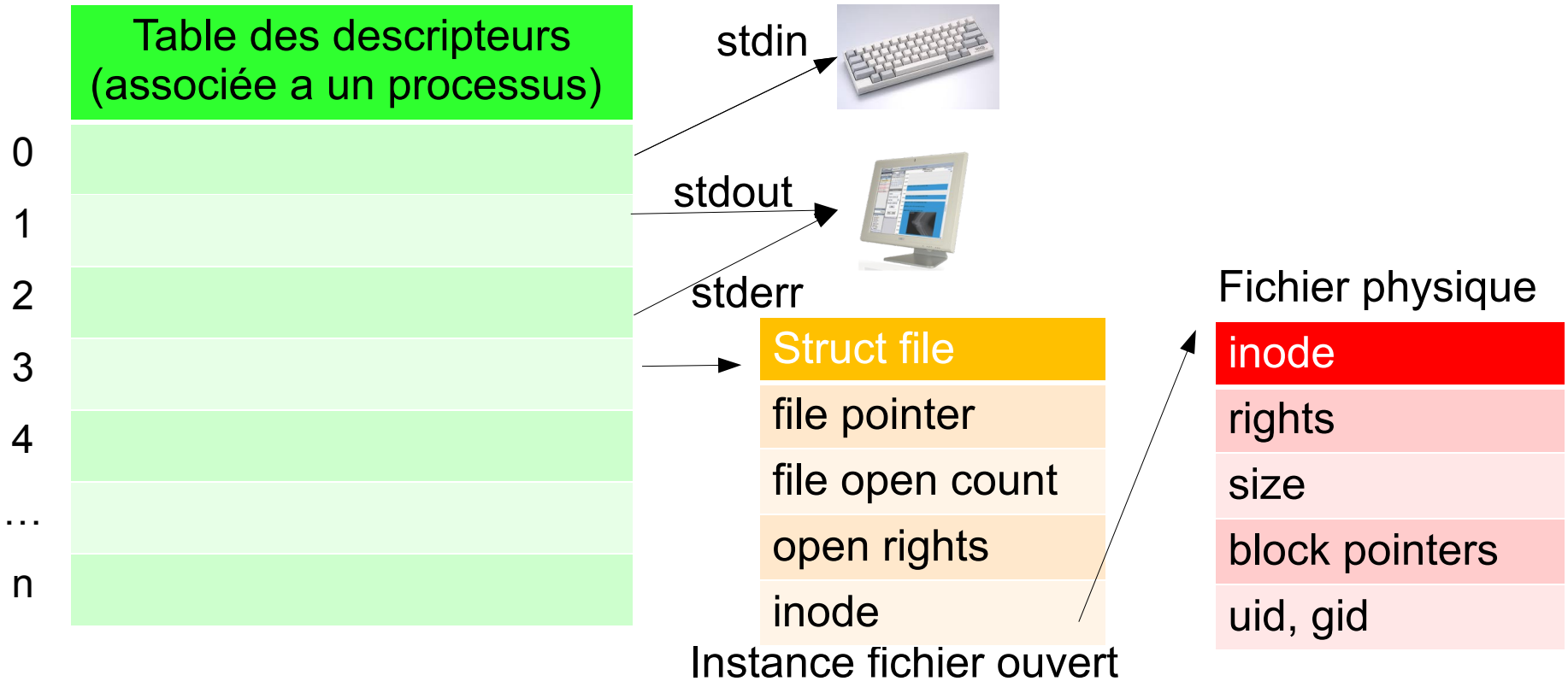
Les redirections (exemple duplication)

Exemple : `> file.txt`



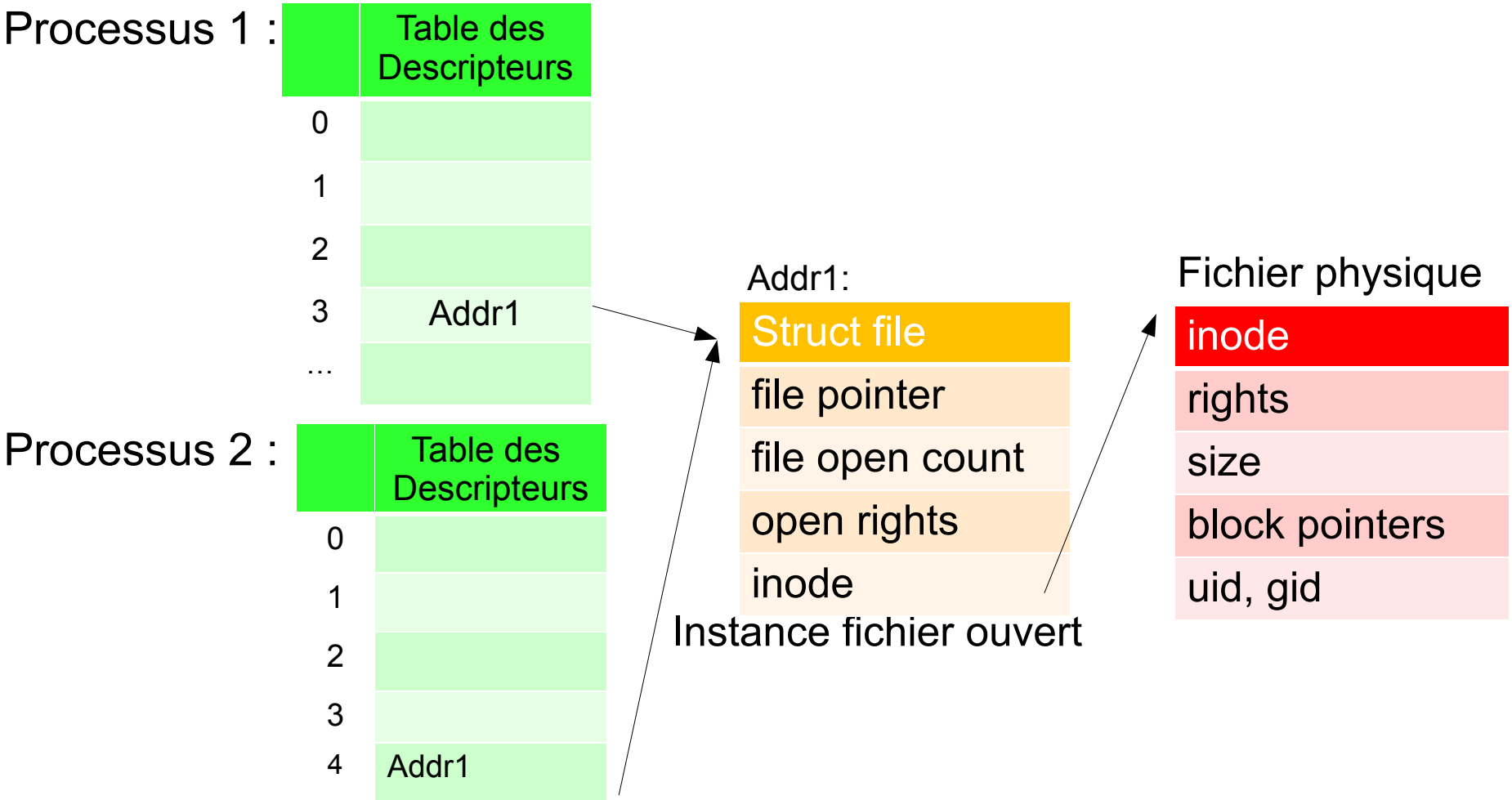
Instance du fichier ouvert

- *Le descripteur de fichier* indique un élément dans la table de descripteurs.
- L'élément est un pointeur vers *l'instance du fichier ouvert*.
- Cette structure garde une référence vers *l'inode du fichier*.



Les attributs d'un fichier ouvert

- file pointer : curseur dans le fichier
- file-open count : combien de processus l'ont ouvert
- open rights les droits d'ouverture



Récupération des informations du SF

Vérification que le processus peut accéder à un fichier

```
int access(const char *pathname, int mode);
```

l'argument `mode` peut prendre les valeurs :

- `R_OK` : accessible en lecture
- `W_OK` : accessible en écriture
- `X_OK` : accessible en exécution
- `F_OK` : le fichier existe

Méthodes d'accès

- Accès séquentiel
  - Lecture ou écriture dans l'ordre avec possibilité de revenir au début.
  - Adapté au supports de stockage séquentiels : bandes magnétiques
- Accès direct (aléatoire : random access)
  - Lecture ou écriture à n'importe quel endroit
  - Adapté aux supports de stockage à accès direct (disques)

Lecture d'un fichier

- Lecture des données changées en préalable dans un buffer
- Le curseur (file pointer) est avancé dans le fichier

ISO C

```
n_recs = fread(buf, RECSZ, N_RECS, fin);
```

Attention : toujours vérifier la fin avec feof(), ferror() !

Appel système Unix

```
n_read = read(fd, buf, count);
```

Attention : Utiliser toujours dans un while !

Windows

```
ReadFile(fHandle,buf,bytesToRead,bytesRead,NULL);
```

Ecriture d'un fichier

- Ecriture des données dans un buffer
- Le curseur est avancé dans le fichier

ISO C

```
n_recs = fwrite(buf, RECSZ, N_RECS, fin);
```

Attention : Ecriture garantie, ou résultat erreur

Unix

```
n_written = write(fd, buf, count);
```

Attention : Toujours dans un while !

Windows

```
WriteFile(fHandle,buf,bytesToWrite,&bytesWritten,NULL)
```

Appels systèmes Unix : read & write

```
ssize_t read(int fd, void *buf, size_t count);
ssize_t write(int fd, const void *buf, size_t count);
```

fd	Descripteur de fichier
buf	Adresse de la structure à traiter
count	Taille de la structure (en octets, utiliser `sizeof()`)
valeur de retour	Nombre d'octets effectivement traité

Exemple

```
#define BLKSIZE 1024
int main (int argc, char **argv) {
   int fd_in, fd_out, n;
   char buffer [BLKSIZE];
   if (argc != 3) {
      fprintf(stderr," ERREUR : %s infile outfile\n, argv[0]");
      return 1;
   }
   fd_in=open(argv[1],O_RDONLY);
   if (fd == -1) {
      perror ("Ouverture du fichier d'entrée");
      return 2;}
   fd_out=open(argv[2],O_WRONLY|O_CREAT|O_TRUNC,0644);
   if (fd_out == -1) {
      perror ("Ouverture du fichier de sortie");
      return 2;}
   while ((n=read(fd_in, buffer,BLKSIZE))>0)
      write(fd_out, buffer, n);
   close (fd_in); close (fd_out);
}
```

Le curseur d'un fichier ouvert

- Il est initialisé à l'ouverture
- Il est modifié à la lecture et à l'écriture
- Il peut être modifié avec l' instruction suivante :

```
off_t lseek(int fd, off_t offset, int whence);
```

offset Décalage souhaité (en octets)

whence Référence : `SEEK_SET`, `SEEK_CUR` ou `SEEK_END`

$\Rightarrow$ Pour les périphérique en mode caractère, `lseek` n'a pas d'effet

Tronquer un fichier

- Tronquer à l'ouverture :

```
open(const char *path, O_RDWR | O_TRUNC);
```

  - Elimine le contenu d'un fichier.
  - Le curseur est positionné sur la position 0 à la fin.

```
int truncate(const char *path, off_t length);
```

- Tronquer âpres l'ouverture ;

```
int ftruncate(int fd, off_t length);
```

length

- Si length est plus grand que la taille du fichier, ce dernier sera étendu par des octets nuls.
- Si length est plus petit que la taille du fichier, le reste des données sera perdu.

Fermeture d'un fichier

- L'accès au fichier est perdu
- L'entré dans la table des descripteurs est perdue

```
close(fd);
```

Plan

1. Le fichier
   - Types de fichiers
   - Organisation générale
   - Manipulation de fichiers
   - Le dossier
2. Gestionnaire du cache
3. La structure du système de fichiers

Les dossiers sur le disque

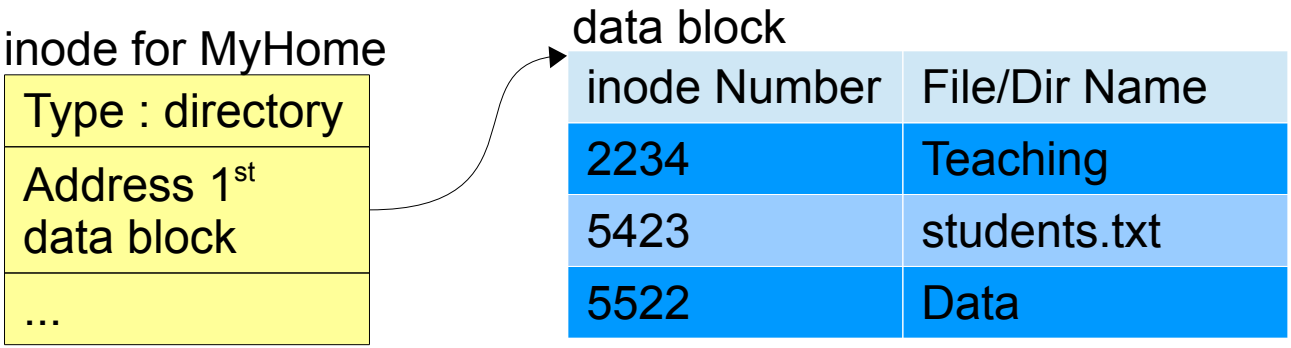
Un dossier est constitué :

- Une liste de numéro d'inodes
- Une liste de noms de fichiers associés

Exemple : Le contenu du répertoire MyHome

inode Number	File/Dir Name
2234	Teaching
5423	students.txt
5522	Data

Le répertoire MyHome sur le disque :

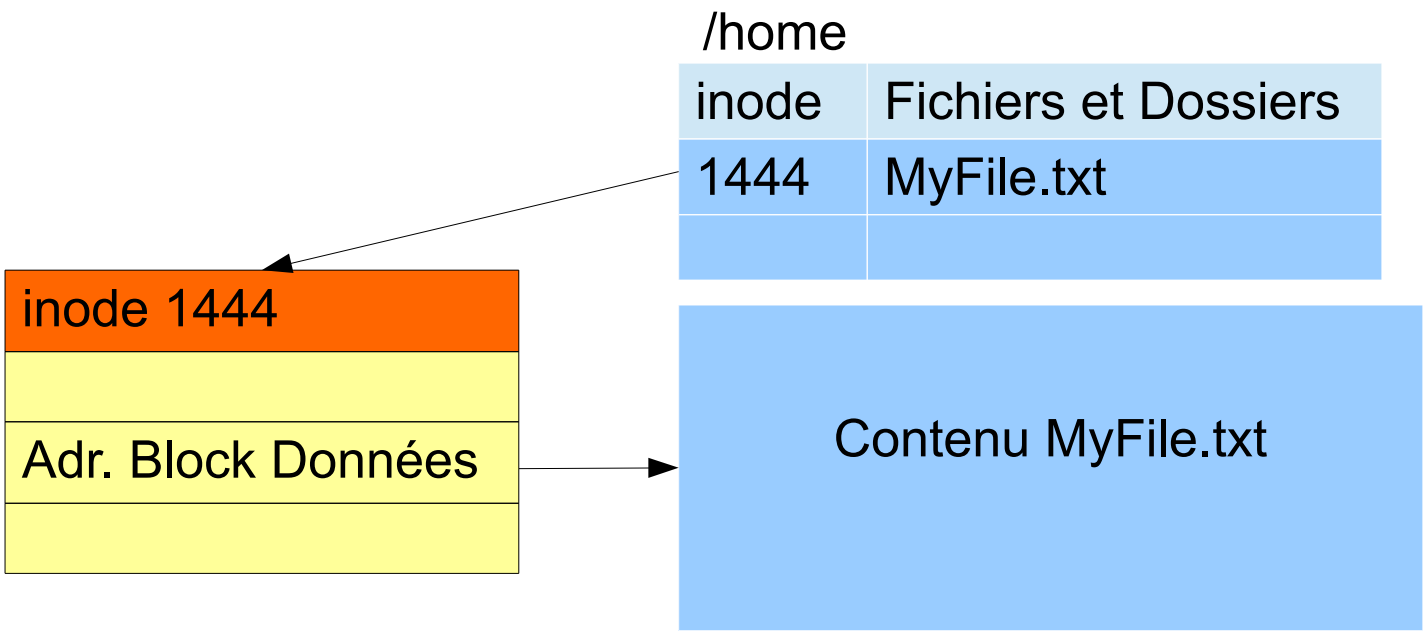


Le contenu des répertoires dans le système Unix V

- nom fichier/répertoire= 14 caractères maximum
- numéro d'inode codé sur 2 octets

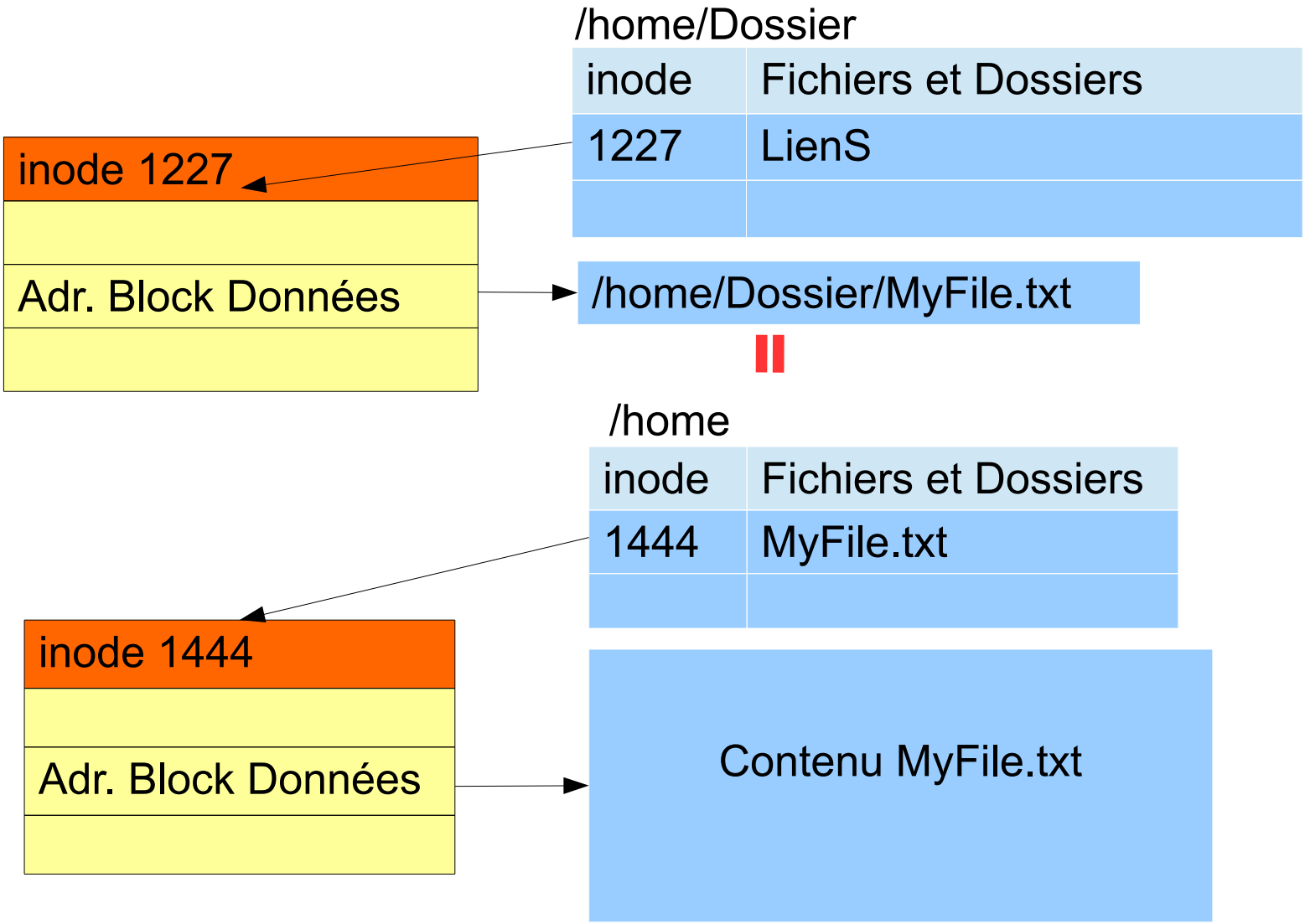
Exemple lien symbolique

```
ln -s /home/MyFile.txt /home/Dossier/lienS
```



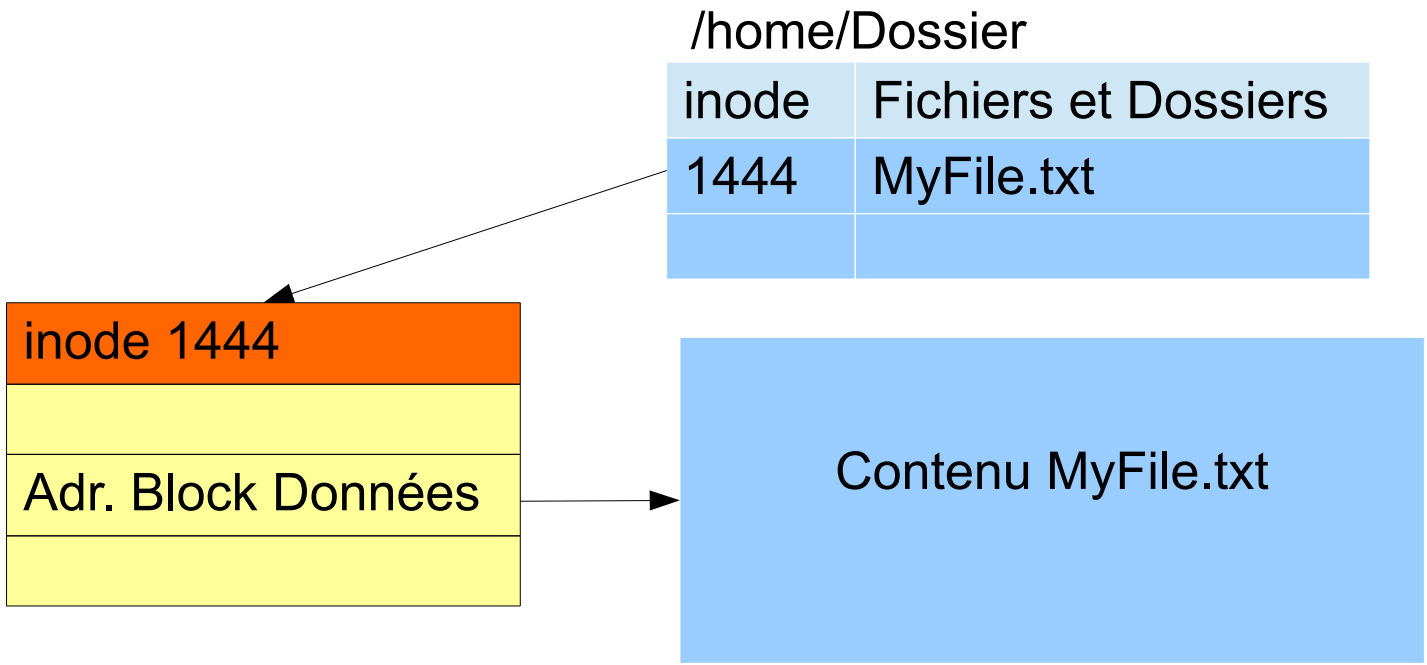
Exemple lien symbolique

```
ln -s /home/MyFile.txt /home/Dossier/lienS
```



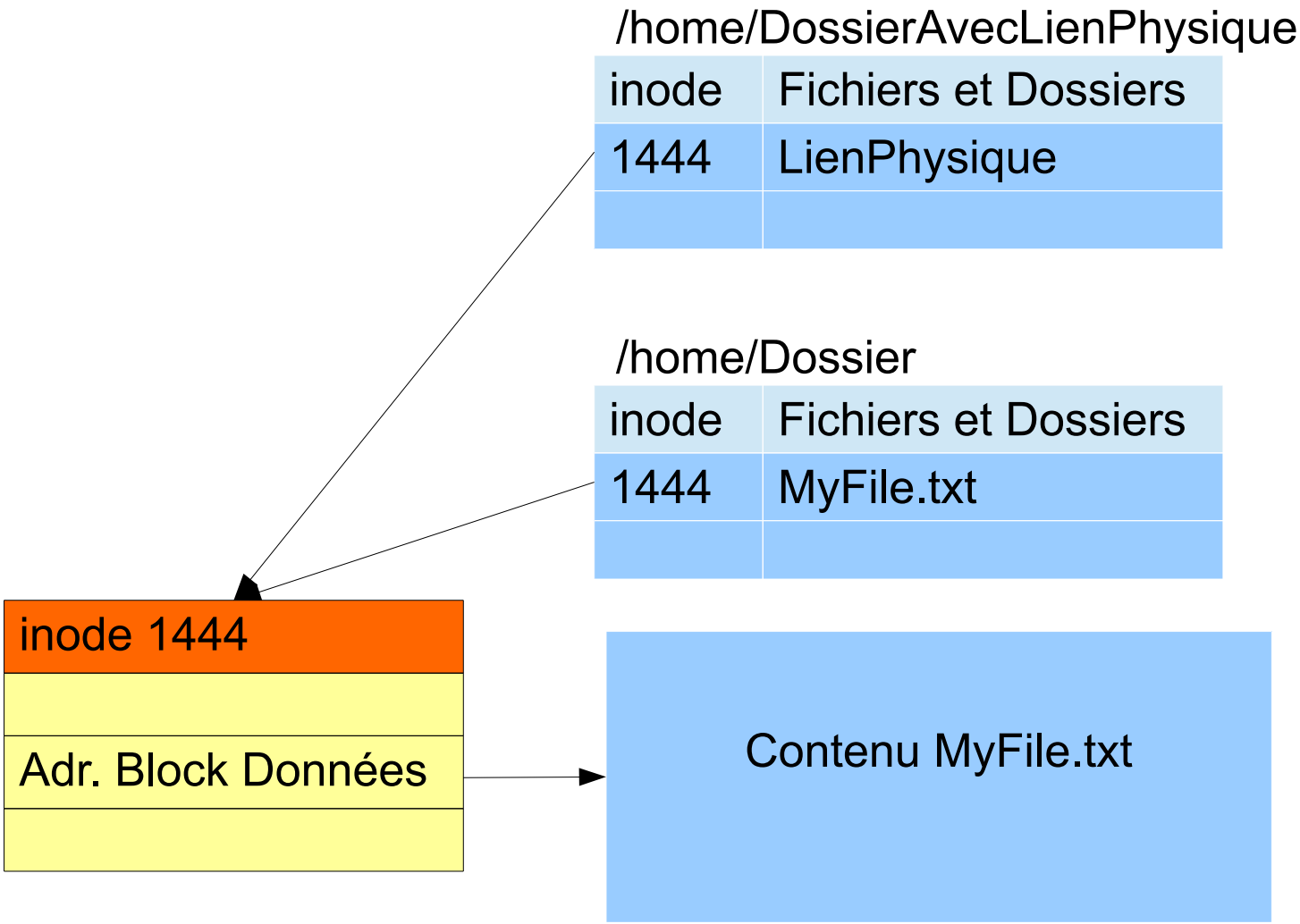
Exemple lien physique

```
ln /home/MyFile.txt /home/Dossier/lienP
```



Exemple lien physique

```
ln /home/MyFile.txt /home/Dossier/lienP
```



inodes : la clé pour comprendre "les liens"

Les liens physiques « hard »

- Un partage d'un inode dans un SF
- Impossible de définir un lien hard depuis un autre SF
- Si le lien disparait, les fichier peut être retrouvé par d'autres liens

Les liens symboliques « soft »

- Création d'un nouveau inode
- Comme un pointeur sur l'objet, donc comme un raccourci
- Si l'objet n'existe pas on perd l'accès à l'objet
- Peut référencer une entrée dans un autre SF

Les répertoires

Définition de `struct dirent` dans `dirent.h`

```
ino_t d_ino
char d_name []
```

Les fonctions associées :

```
#include <sys/types.h>
#include <dirent.h>
int             closedir(DIR *);
DIR            *opendir(const char *);
struct dirent  *readdir(DIR *);
```

Il n'existe pas de fonction `writedir`.

→ utiliser `creat`, `mkdir` ou `mknod`

Plan

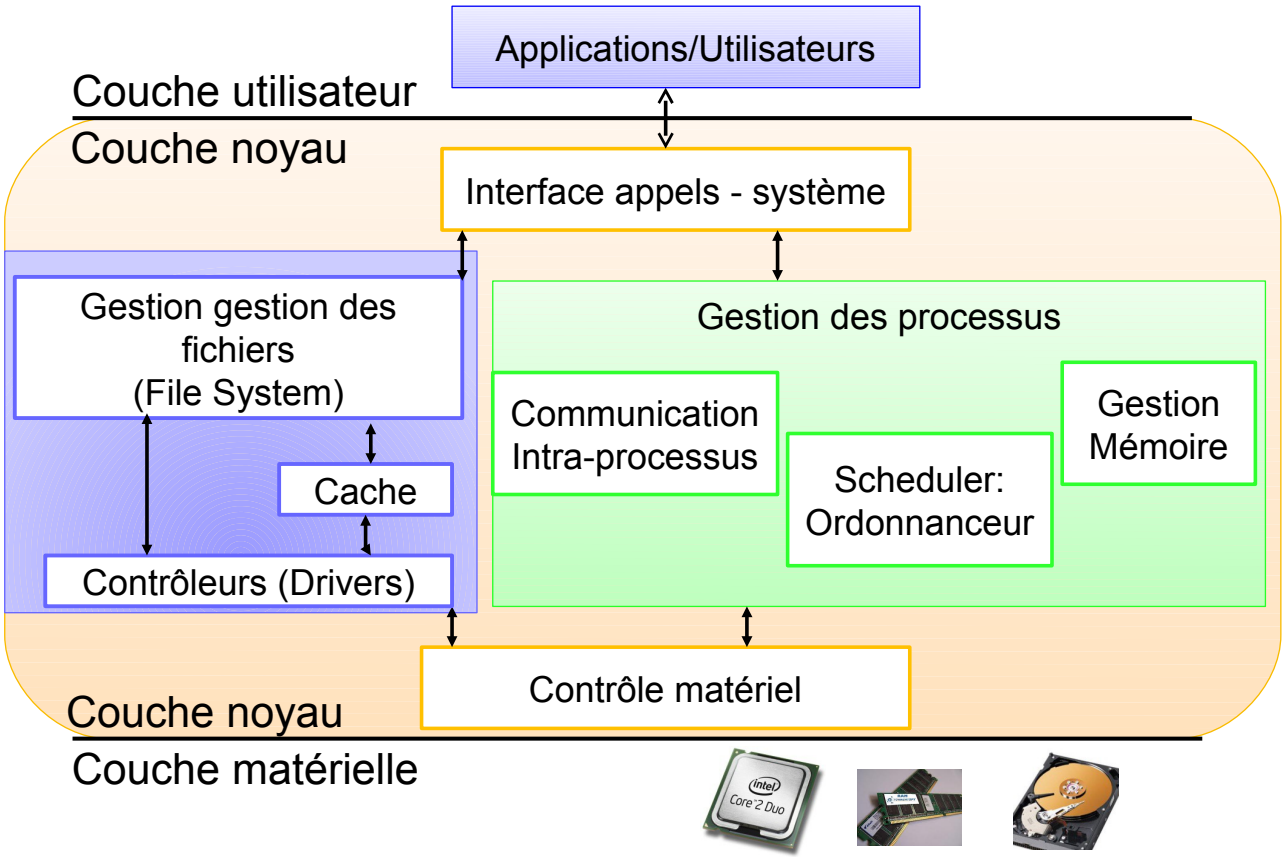
1. Le fichier
   - Types de fichiers
   - Organisation générale
   - Manipulation de fichiers
   - Le dossier
2. Gestionnaire du cache
3. La structure du système de fichiers

Motivation : le coûts d'accès au disque

$$t_{\textit{accès au bloc disc}} \gg t_{\textit{lecture/écriture bloc disc}} \gg t_{\textit{lecture/écriture bloc mémoire}}$$

Solution : Charger les fichiers en mémoire $\rightarrow$ dans le cache

Positionnement du gestionnaire du cache



Principe buffer cache

Pour interagir avec un fichier :

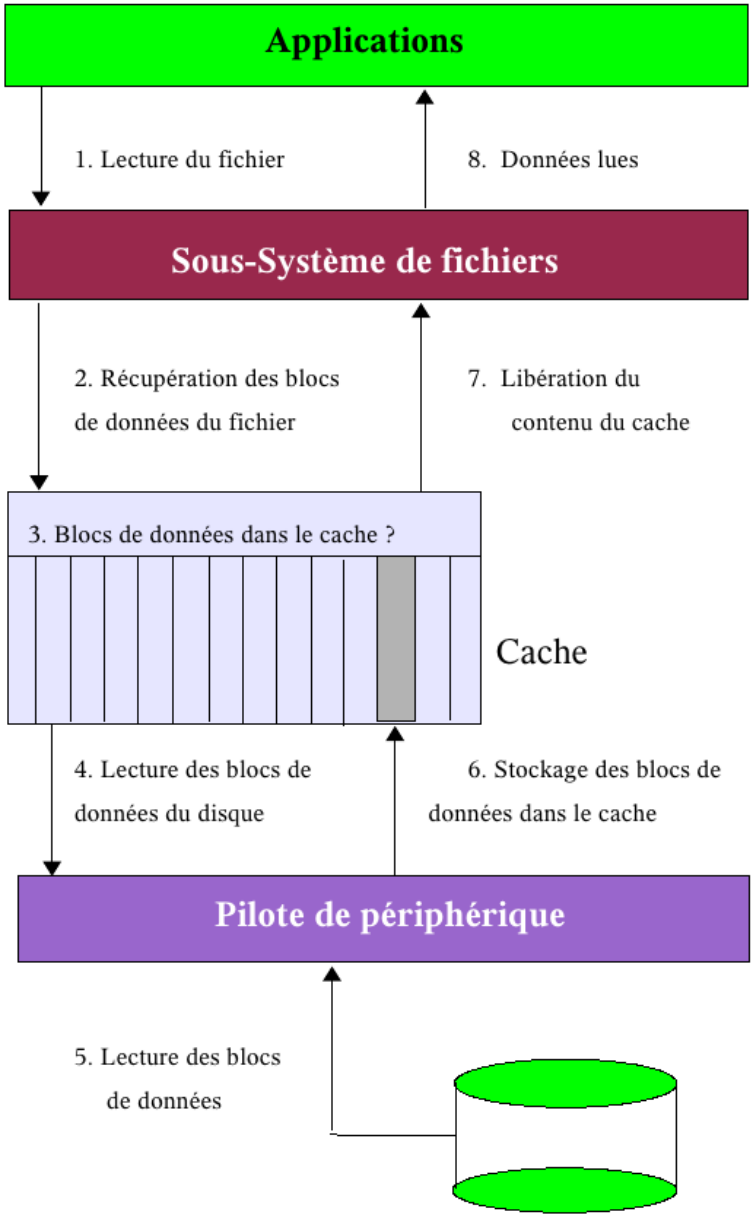
1. Chargement en mémoire principale
2. Modifications
3. Sauvegarde sur le SF

- Le noyau minimise la fréquence d'accès au disque
- Maintient un *buffer cache* des fichiers récemment utilisés

Principes :

- Le noyau ne charge un fichier que s'il n'est pas dans le cache
- Le cache conserve les fichiers modifiés pour leur utilisation future
- Évite les écritures en analysant si les données sont transitoires

Buffer cache



Plan

1. Le fichier
   - Types de fichiers
   - Organisation générale
   - Manipulation de fichiers
   - Le dossier
2. Gestionnaire du cache
3. La structure du système de fichiers

Plusieurs système(s) de fichiers sur la même machine

- Chaque disque est partagé en partitions (ou volumes)
- A chaque partition est associée un système de fichiers : e.g. MSDOS, ext2, ext3, HFS, NTFS, FAT32, ISO9660, UDF)
- Le système de fichiers peut être disponible (monté dans l'arborescence) ou indisponible

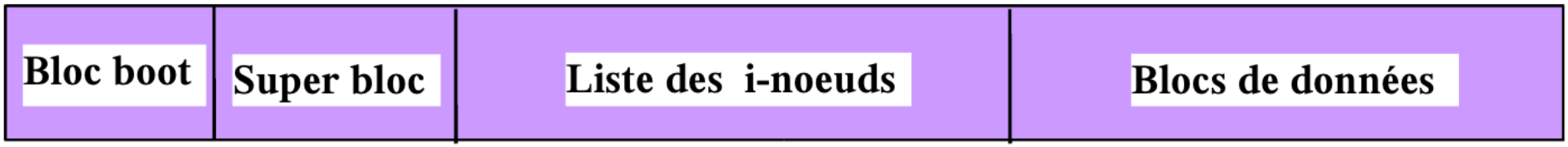
Nota bene :

Il y a une seule arborescence (une racine) par machine UNIX !

Structure du SF

Un SF est structuré en :

- Bloc boot : utilisé pour démarrer la machine
- Super bloc : méta-données du SF
- Liste des inodes : méta-information des fichiers
- Bloc de données : blocs libres et utilisés



Super bloc

- la taille du système de fichiers,
- le nombre de blocs libres,
- une liste des blocs libres,
- l'indice du premier bloc libre dans la liste des blocs libres,
- la taille de la liste des inodes,
- le nombre de inodes libres,
- une liste des inodes libres,
- l'indice du premier inodes libre dans la liste des inodes libres,
- des champs de verrous pour l'accès aux listes d'inodes et de blocs libres,
- un drapeau indiquant si le super bloc a été modifié.

Structure importante : dupliquée, verrouillée et chargée en mémoire.

Vérification de la cohérence d'un SF

Le noyau tente de minimiser la corruption du SF en cas de panne

La commande `fsck` vérifie les incohérences et les répare :

1. test sur les inodes
   → contrôler la taille des fichiers et marquer les blocs parcourus
2. test par les répertoires
   → vérifier la cohérence des en-têtes et des noms
3. test de connexion inter-répertoires
   → vérifier que chaque inode est référencé
4. test du nombre de références
   → Vérification du bon nombre de lien
5. test des blocs libres et des ensembles de blocs
   → somme des blocs libres et occupés = taille du disque

Montage et démontage d'un SF

- Création du SF avec `mkfs`
- Attacher un SF à un répertoire de `\` avec `mount`
- Détacher un SF avec `umount`

Remarques

- Seul l'administrateur peut monter des périphériques
- Il est possible de faire des montage réseau
- Automatisation possible avec `/etc/fstab`

```
$ cat /etc/fstab
# /etc/fstab: static file system information.
# <fs>  <mountpoint>  <type>  <opts>  <dump/pass>
/dev/sda3  /           reiser noatime          0 1
/dev/sda1  /boot       ext2   noatime          1 2
/dev/sda2  none        swap   sw               0 0
/dev/cdrom /mnt/cdrom auto   noauto,ro,users 0 0
```

Organisation de l'arborescence Unix

`/etc`	les fichiers de configuration
`/bin`	les binaires nécessaires en mode mono-utilisateur
`/sbin`	les binaires pour l'utilisateur root
`/tmp`	le répertoire temporaire en mode mono-utilisateur
`/dev`	les périphériques
`/var`	le répertoire des fichiers "variables"
`/var/log`	les fichiers de "log"
`/var/crash`	les image mémoire en cas de crash du système
`/usr/bin`	tous les autres fichiers binaires
`/usr/lib`	les bibliothèques système
`/usr/sbin`	les autres binaires pour l'utilisateur root
`/usr/share`	les fichiers indépendants de l'architecture comme les manuels, les sources, les fichiers de définitions
`/home`	le nom standard pour le répertoire des utilisateurs