



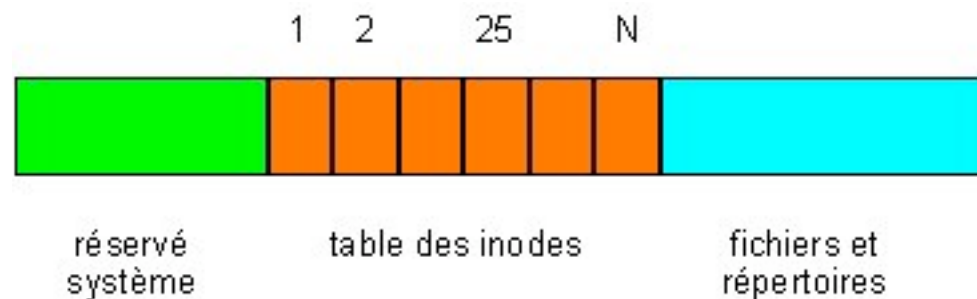
# **Fichiers Unix, interface avec C**

`christophe.cerin@iutv.univ-paris13.fr`

# SGF = Système de Gestion de fichiers

Structure générale d'un SGF sous Linux :

- une partie système (bloc 0 : démarrage ; bloc 1 : Superblock = information de structures)
- une partie table[1..N] des inodes, chaque inode contenant des informations sur un fichier ou répertoire du SGF.
- une partie fichiers et répertoires : bloc de données





# Les inodes

- Chaque fichier Unix est représenté par un inode de 64 bytes
- Cette information n'est pas visible par une commande ls
- Chaque inode contient de quoi retrouver tous les blocs de disque contenant les données du fichier : les blocs ne sont pas forcément contigus ;
- Ce qui vient d'être dit joue à la fois sur le niveau matériel (archi du disque) et le niveau logiciel (structure noyau Linux)



# Les inodes

- Un inode contient le fichier (les informations de localisation sur le disque), les permissions, le type du fichier...
- Un **répertoire** est simplement une liste de noms et d'inodes;
- Quand on crée une « entrée » on ajoute une paire (name-number) à un répertoire ; Quand on détruit un fichier, on supprime une « paire » d'un répertoire : ce n'est pas la peine de parcourir les liens pour faire des remise à zéro !

# Organisation des répertoires

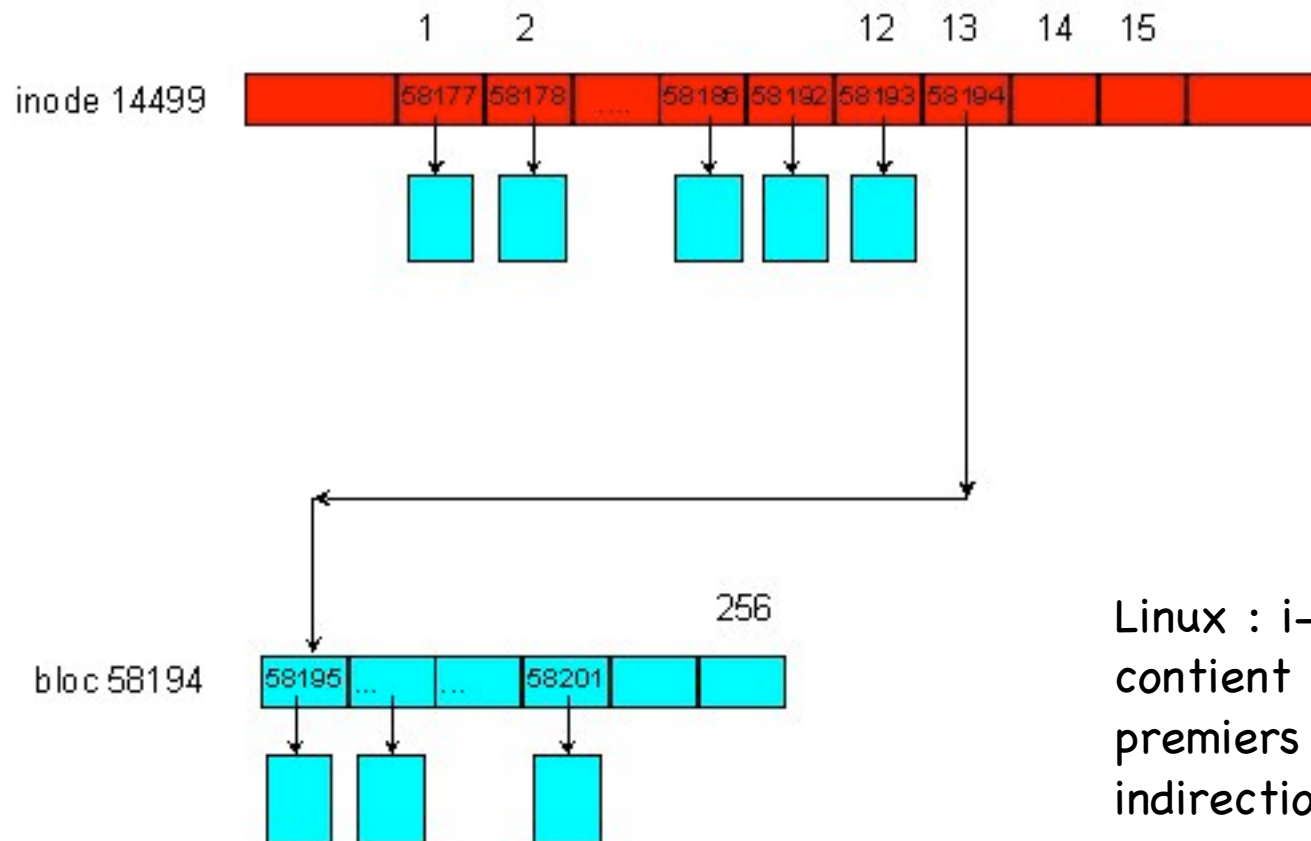
**Structure d'un répertoire** (partie fichiers et répertoires) :

- colonne 1 : le numéro d'inode du fichier ou répertoire
- colonne 2 : la longueur occupée par cette entrée dans la table
- colonne 3 : la longueur du nom du fichier ou Répertoire
- colonne 4 : le nom du fichier ou répertoire

Les caractéristiques du fichier sont contenues dans l'inode.

2	12	1	.
2	12	2	..
11	20	10	lost+found
12	16	8	COURRIER
13	12	4	JEUX
14	952	12	APPLICATIONS

# Organisation des inodes





# Les types de fichiers

- Information retournée par `ls -l` :
  - d (directory)
  - l (symbolic link)
  - b (block device)
  - c (character device)
  - p (named pipe)
  - s (socket)



# Les types : device files

- Il en existe deux types : ceux dits « par caractères » et ceux dits « par blocs »
- *Character devices* can be accessed one character at a time
- *Block devices* must be accessed in larger units called blocks, which contain a number of characters. Votre disque dur est un block device.
- Les device files sont créés par la commande `/dev/MAKEDEV` (on spécifie le type : joystick, usb, parallel port...)

## Le pseudo SF /proc

- Pseudo car il ne correspond a aucun device
- Permet de collecter des informations sur votre système : /proc/cpuinfo, /proc/meminfo, /proc/diskstats, /proc/net, /proc/ide/ide0/model => SAMSUNG CDRW/DVD SM-352F
- Faire un `man 5 proc` pour de plus amples détails



## **stdio.h & errno.h**

- La fonction de la bibliothèque perror() retourne une description de la dernière erreur rencontrée dans un appel système (voir cours précédent)
- Elle est utilisée en conjonction avec `extern int errno;` pour donner le numéro, donc le type de l'erreur



# Les fichiers

- Opérations de base : création, ouverture, fermeture, adjonction

```
FILE *stream, *fopen();
/* declare a stream and prototype fopen */
stream = fopen("`myfile.dat'", "`r'");
/*it is good practice to to check file */
/*is opened correctly: */
if ((stream = fopen("`myfile.dat'", "`r'")) == NULL)
    { printf("`Can't open %sn'", "`myfile.dat'");
      exit(1);
    }
    .....
```



# Lire et écrire dans un fichier

```
int fprintf(FILE *stream, char *format,  
args..)
```

```
int fscanf(FILE *stream, char *format,  
args..)
```

```
int getc(FILE *stream), int fgetc(FILE  
*stream)
```

```
int putc(char ch, FILE *s)
```

```
int fputc(char ch, FILE *s)
```



# Vérifications

**feof()** -- returns true if the stream is currently at the end of the file. So to read a stream, `fp`, line by line you could do:

```
while ( !feof(fp) ) fscanf(fp, "%s", line);
```

**ferror()** -- reports on the error state of the stream and returns true if an error has occurred.

**clearerr()** -- resets the error indication for a given stream.

**fileno()** -- returns the integer file descriptor associated with the named stream.



## Low level I/O

- Les commandes qui suivent ne sont **pas** buffurisées : on accède directement au device sans passer par un tampon intermédiaire :
- **open, close, write, creat, sizeof**
- Note : nous avons déjà vu un exemple (open, write, close) dans le chapitre des appels système

```

/* program to read a list of floats from a binary file */
/* first byte of file is an integer saying how many */
/* floats in file. Floats follow after it, File name got from */
/* command line */
#include<stdio.h>
#include<fcntl.h>
float bigbuff[1000];
main(int argc, char **argv) {
    int fd;
    int bytes_read;
    int file_length;
    if ( (fd = open(argv[1],O_RDONLY)) = -1)
    { /* error file not open */
        perror("Datafile");
        exit(1);
    }
    if ( (bytes_read = read(fd,&file_length, sizeof(int))) == -1)
    { /* error reading file */
        exit(1);}
    if ( file_length > 999 )
    {/* file too big */ .....}
    if ((bytes_read = read(fd,bigbuff,file_length*sizeof(float)))== -1)
    { /* error reading open */ exit(1);}
}

```

# Consultation d'inode

- L'appel système `stat` permet d'obtenir certaines informations de l'inode ;
- Résultat d'un appel à `stat()` → `struct stat`;

STAT(2) BSD System Calls Manual STAT(2)

## NAME

`stat`, `lstat`, `fstat` - get file status

## SYNOPSIS

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

```
int
stat(const char *path, struct stat *sb);
```

```
int
lstat(const char *path, struct stat *sb);
```

```
int
fstat(int fd, struct stat *sb);
```

## DESCRIPTION

The `stat()` function obtains information about the file pointed to by `path`. Read, write or execute permission of the named file is not required, but all directories listed in the path name leading to the file must be searchable.



## Exemple (from Michel Divay)

- Le code proposé permet de lister les propriétés d'un inode
- Illustration de l'appel système stat... mais aussi open, read, write...

[Code source](#)



# Résultat exécution (Mac OSX)

```
Fichier courant tub
Votre choix ? 3
```

```
Lister l'inode de tub
```

```
sizeof (struct stat) : 96
Dev      (en hexa)    : 0xe000009
Dev (major, minor)   : 14/9
Num?ro de l'inode    : 1384561
Mode      (en hexa)   : 0x11b6
Droits d'acc?s      : ?rw-rw-rw-
Link       : 1
Num?ro du propri?taire : 501
Num?ro du groupe     : 501
Taille           : 0
Dernier acc?s en lecture : 1110981260 Wed Mar 16 14:54:20 2005
Dernier acc?s en ?criture : 1110981260 Wed Mar 16 14:54:20 2005
Derni?re modification de l'inode : 1110981260 Wed Mar 16 14:54:20 2005
```



# Gestion des répertoires

- Dans le code précédent il y a aussi des fonctions de gestion des répertoires :  
opendir... (man 3 opendir)
- Répertoire = fichier contenant des noms de fichiers et/ou répertoires
- Interface : via un pointeur sur une structure de type DIR (<dirent.h>)



# Liste des fonctions

DIRECTORY(3)

BSD Library Functions Manual

DIRECTORY(3)

## NAME

opendir, readdir, readdir\_r, telldir, seekdir, rewinddir, closedir, dirfd  
- directory operations

## LIBRARY

Standard C Library (libc, -lc)

## SYNOPSIS

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

```
DIR *      /* ouverture du repertoire de nom filename */
opendir(const char *filename);
```

```
struct dirent *      /* lecture d'une entrée dans le répertoire */
readdir(DIR *dirp);
```



# Liste des fonctions

```
int /* readdir avec rangement du résultat dans result */  
readdir_r(DIR *dirp, struct dirent *entry, struct dirent **result);
```

```
long /* repère une entrée pour s'y repositionner plus tard */  
telldir(DIR *dirp);
```

```
void /* indique l'entrée loc du répertoire (obtenue avec */  
seekdir(DIR *dirp, long loc); /* telldir concernée par un prochain */  
/* appel à readdir */
```

```
void /* se positionne au début du répertoire dirp */  
rewinddir(DIR *dirp);
```

```
int /* dealloue la structure pointée par dirp */  
closedir(DIR *dirp);
```

```
int  
dirfd(DIR *dirp); /* returns the integer file descriptor associated with  
the named directory stream, see open(2).*/
```

# Exemples d'utilisation

```
/* structure describing an open directory. <dirent.h> */
typedef struct _dirdesc {
    int      dd_fd;          /* file descriptor associated with directory */
    long     dd_loc;        /* offset in current buffer */
    long     dd_size;       /* amount of data returned by getdirentries */
    char     *dd_buf;       /* data buffer */
    int      dd_len;        /* size of data buffer */
    long     dd_seek;       /* magic cookie returned by getdirentries */
    long     dd_rewind;     /* magic cookie for rewinding */
    int      dd_flags;      /* flags for readdir */
    pthread_mutex_t dd_lock; /* for thread locking */
    struct _telldir *dd_td; /* telldir position recording */
} DIR;
struct dirent { /* <sys/dirent.h> */
    u_int32_t d_fileno;     /* file number of entry */
    u_int16_t d_reclen;     /* length of this record */
    u_int8_t  d_type;      /* file type, see below */
    u_int8_t  d_namlen;    /* length of string in d_name */
#ifdef _POSIX_SOURCE
    char      d_name[255 + 1]; /* name must be no longer than this */
#else
#define MAXNAMLEN      255
    char      d_name[MAXNAMLEN + 1]; /* name must be no longer than this */
#endif
};
```



# Exemple d'utilisation

Sample code which searches a directory for entry  
`name' is:

```
len = strlen(name);
dirp = opendir(".");
while ((dp = readdir(dirp)) != NULL)
    if (dp->d_namlen == len && !strcmp(dp->d_name, name)) {
        (void)closedir(dirp);
        return FOUND;
    };
(void)closedir(dirp);
return NOT_FOUND;
```



# Autre exemple

Et si le répertoire  
contient des répertoires ?

```
/* Sample code which lists a directory */

void listerRep(char *nom){
    DIR* dp;
    struct dirent* dirp;

    if((dp = opendir(name))==NUL) {
        perror("listerRep"); exit(0);
    } else {
        while ((dirp = readdir(dp)) != NULL)
            printf( »%s\n »,dirp->d_name);
        (void)closedir(dirp);
    }
}
```