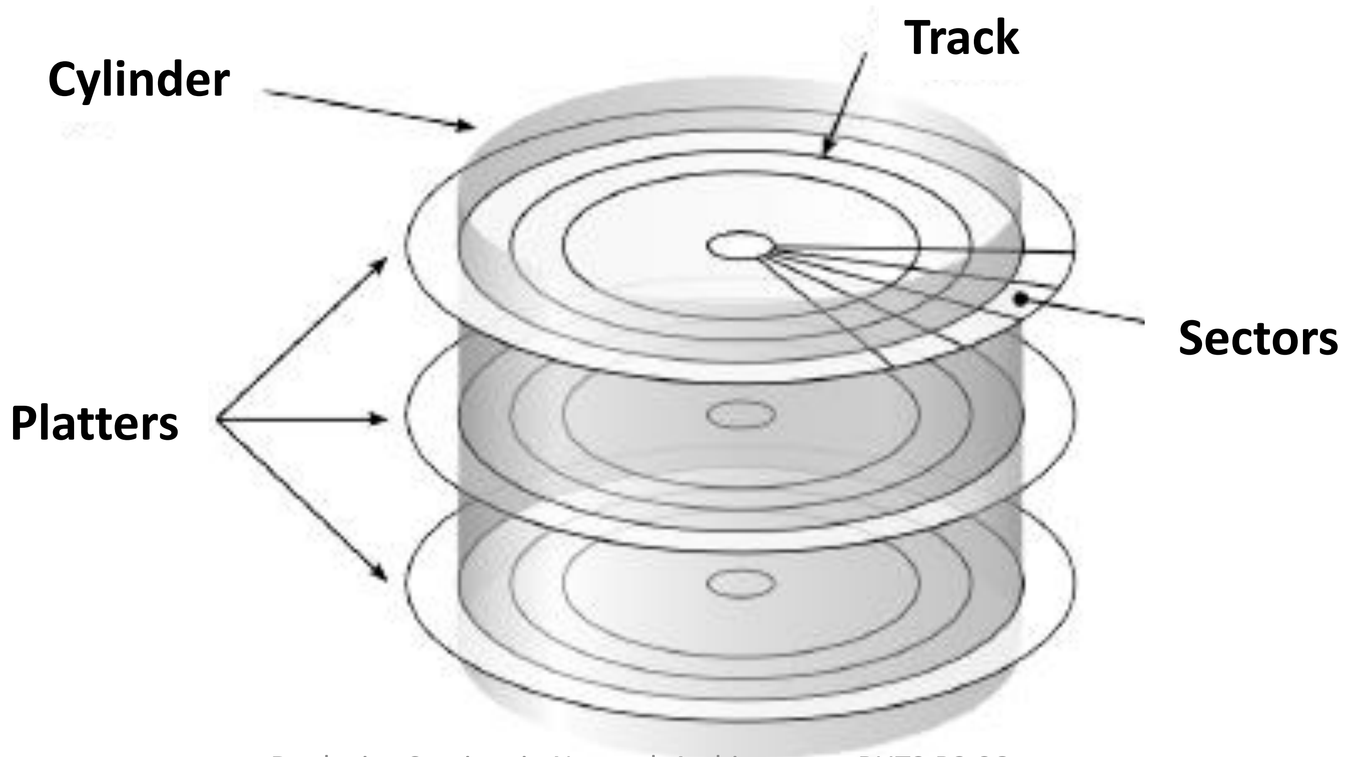
Chapter 3
File Management System

Class Syllabus

Hard Drive Geometry (1/2)

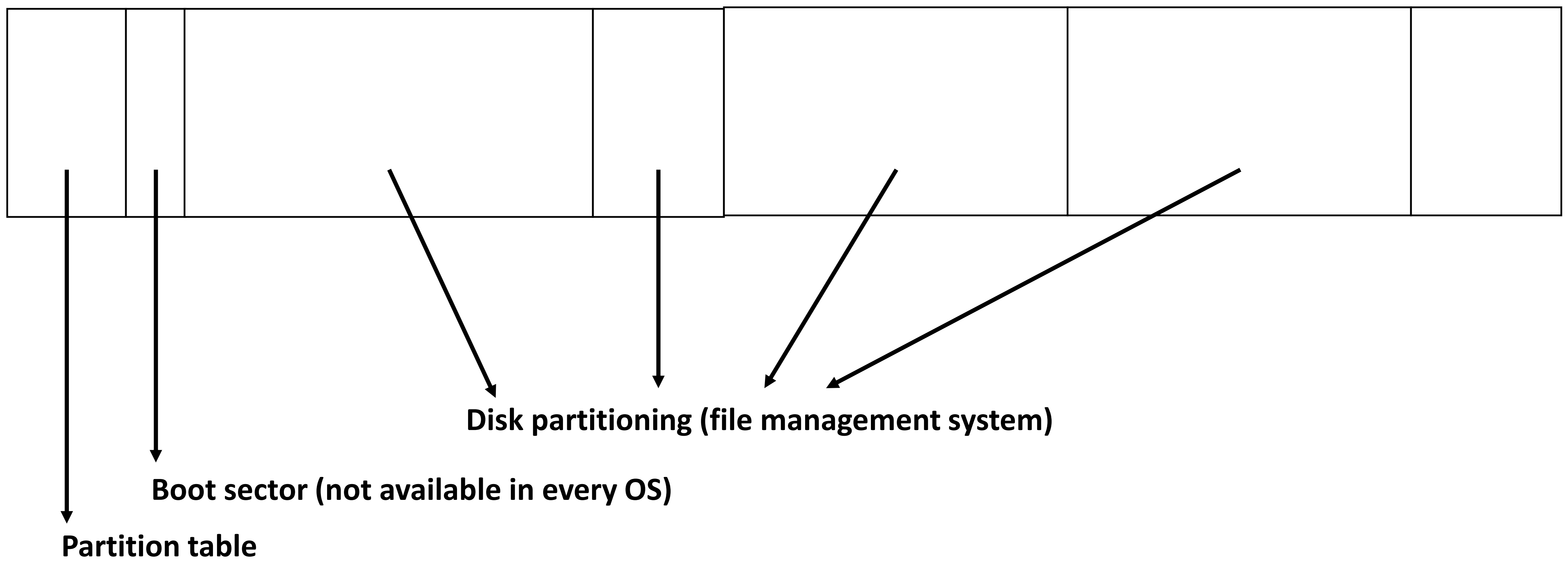
- Physical Disk
    - Several disk platters, 2 faces per platter
    - A face made up of tracks and sectors



Hard Drive Geometry (2/2)

- Disk linearization $(\text{CHS} \rightarrow \text{LBA } (n_l))$
  - Total space: *(#faces * # cylinders * # sectors/# tracks)* **sectors**
  - $n_l = n_s + n_c$ ** # sectors/p +* n_f ** # cylinders * # sectors/p*
  - How to retrieve the physical coordinates:
    - $n_s = n_l \%$ *# sectors/p*
    - $n_c = (n_l \div$ *# sectors/p*$) \%$ *# cylinders*
    - $n_f = (n_l \div ($*# sectors/p * # cylinders*$)$

Hard Drive Organization


Disk partitioning (file management system)
Boot sector (not available in every OS)
Partition table

File Management System

- Goal: to mask the "sector" aspect and instead only work with files
- Files are fragmented on the hard drive
- Easiest: **FAT Dos**
  - **FAT:** File Allocation Table
  - Huge table showing for each block successor or his state (free/defunct/EOF)
  - Limits
    - Too much FAT access (nightmare in sequential access)
    - FAT memory (huge if huge disk)

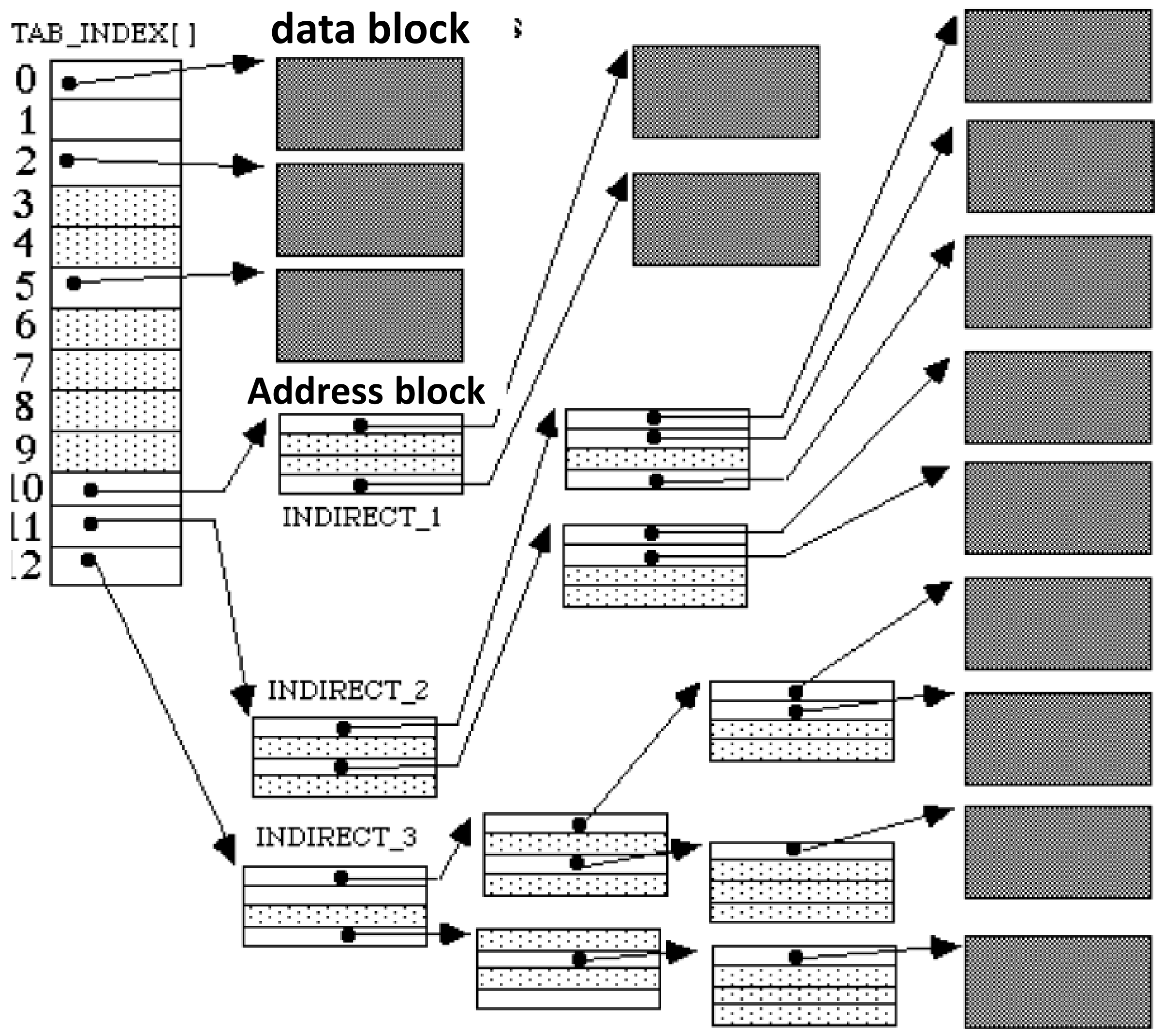
Unix File Management System

- Data blocks

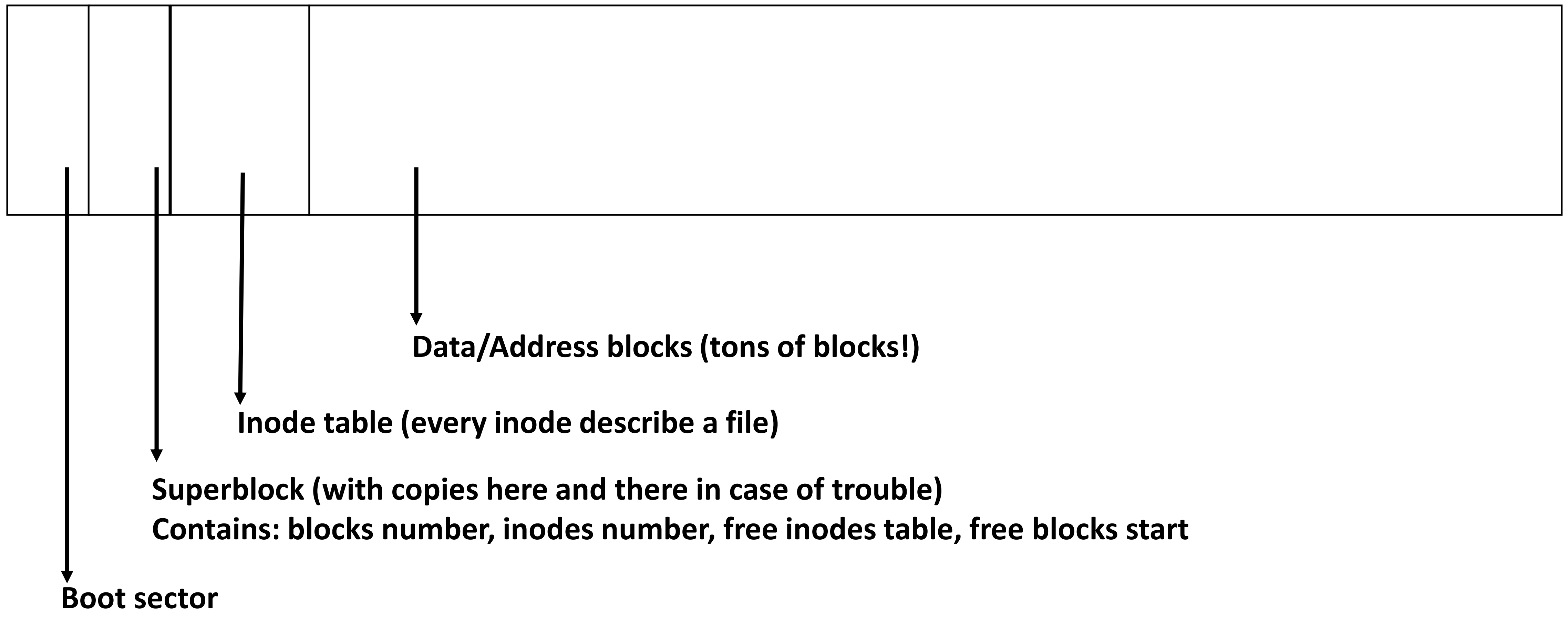
/Address block

- Allow for huge file storage

- No FAT in memory



Partition structure


Data/Address blocks (tons of blocks!)
Inode table (every inode describe a file)
Superblock (with copies here and there in case of trouble)
Contains: blocks number, inodes number, free inodes table, free blocks start
Boot sector

Linking partition/directory

- Mounting action
- Broader sense than a simple partition
  - Distant Systems (NFS, sshFS, etc.)
  - Logical volume (partitions union)
  - Mounting relating to a driver (kernel)
    - */proc* (proc)
    - */proc/bus/usb* (usbdevfs)

Inode (Index node)

- Describe and identify a file (not working with filename anymore)
  - gid/uid, types, access rights, dates (amc), hard links number, size
  - Pointer on a data block

- Association name/inode
  - For every recording
    - Inode number
    - Relative name
  - Different Name size (< 256 characters on Linux)

Hard links

- Hard link
  - Two different entries in the name/inode table with the same inode **BUT** different names
  - Source and destination must be on the same partition

- Create a hard link using Shell
  - ln **without** option –s
  - ln *existingSource hardLinkName*

- Remove a file => decrease the hard link number
  - Effectively removed if and only if hard link number = 0

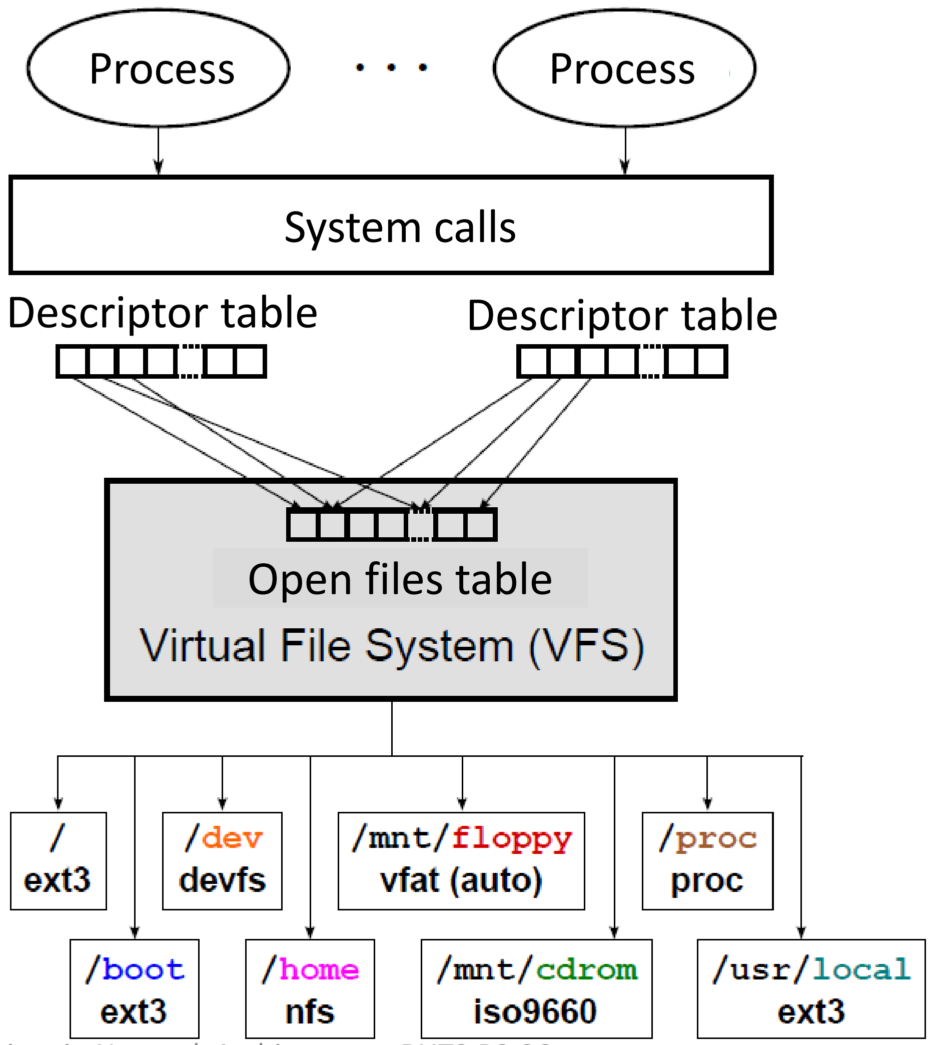
File types

```
$ ls -l
total 4
brw-rw----   1 root  floppy   2,   0 mar  3  2002 fd0
lrwxrwxrwx   1 root  root          3 nov 17 08:43 lien -> fd0
-rw-r--r--   1 root  root          0 nov 17 08:44 regulier
drwxr-xr-x   2 root  root       4096 nov 17 08:44 répertoire
srwx------   1 root  root          0 nov 16 22:01 socket
crw-rw----   1 root  dialout  4,  64 mar  3  2002 ttyS0
prw-r--r--   1 root  root          0 nov 17 08:40 tube
```

-	**Regular file**
d	**Directory**
l	**Symbolic link**
p	**Named pipe**
b	**Device file (block mode)**
c	**Device file (character mode)**
s	**Named socket**

File tree

```
.
├─ bin
├─ boot
├─ dev
├─ etc
├─ home
│   ├─ titi
│   └─ toto
├─ lib
├─ lost+found
├─ mnt
│   ├─ cdrom
│   └─ floppy
├─ proc
├─ root
├─ sbin
├─ tmp
├─ usr
│   └─ local
└─ var
```



File manipulation system calls (1/6)

chmod, fchmod – Change file access rights

```
int chmod(const char *pathname, mode_t mode);
int fchmod(int fildes, mode_t mode);
```

chown, fchown, lchown – Change file owner and group

```
int chown(const char *path, uid_t owner, gid_t group);
int fchown(int fd, uid_t owner, gid_t group);
int lchown(cost char *path, uid_t owner, gid_t group);
```

(Non POSIX)

close – Close a file descriptor

```
int close(int fd);
```

dup, dup2 – Duplicate a file descriptor

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

File manipulation system calls (2/6)

fcntl – **Manipulate file descriptor**

```
int fcntl(int fd, int cmd);
int fcntl(int fd, int cmd, long arg);
int fcntl(int fd, int cmd, struct flock *lock);
```

fsync – **Synchonize changes to file**

```
int fsync (int fd);
```

ioctl – **Control device**

```
int ioctl(int d, int requete, ...);
```

(Non POSIX)

link – **Create a new link to a file**

```
int link (const char *oldpath, const char *newpath);
```

lseek – **Move the read/write offset**

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

mknod – **Create a node in the file system** (POSIX if FIFO)

```
int mknod(const char *pathname, mode_t mode, dev_t dev);
```

File manipulation system calls (3/6)

mount, umount – Mount/unmount file systems

```
int mount(const char *specialfile, ...
int umount(const char *dir);
```

(Non POSIX)

open – Open/possibly create a file

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

pipe – Create a pipe

```
int pipe(int filedes[2]);
```

poll – Wait for some event on the file descriptor

```
int poll(struct pollfd *ufds, unsigned int nfds, int delai);
```

(Non POSIX)

read – Read from a file descriptor

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

File manipulation system calls (4/6)

readlink – **Read value of symbolic link**

```
ssize_t readlink(const char *path, char *buf, size_t bufsiz);
```

rename – **Change name/location of file**

```
int rename(const char *oldpath, const char *newpath);
```

select, pselect – **Synchronous I/O multiplexing**

(Non POSIX)

```
int select(int n, fd_set *readfds,
           fd_set *writefds, fd_set *exceptfds,
           struct timeval *timeout);
int pselect(int n, fd_set *readfds, fd_set *writefds,
            fd_set *exceptfds, const struct timespec *timeout,
            sigset_t * sigmask);
```

socket – **Create endpoint for communication**

```
int socket(int domain, int type, int protocol);
```

File manipulation system calls (5/6)

stat, fstat, lstat – **File status**

```
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);
```

(Non POSIX)

(Non POSIX)

symlink – **Make a symbolic link to a file**

```
int symlink(const char *cible, const char *nom);
```

umask – **Set file mode creation mask**

```
int umask(int mask);
```

unlink – **Delete a name and possibly the file it refers to**

```
int unlink(const char *pathname);
```

write – **Write to a file descriptor**

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

opendir – **Open a directory**

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

File manipulation system calls (6/6)

closedir – **Close a directory**

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

readdir – **Read a directory**

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

struct dirent – **Read a directory**

```
struct dirent {
  long           d_ino;      /* inode number */
  off_t          d_off;      /* offset to the next dirent */
  unsigned short d_reclen;   /* length of this record */
  unsigned char  d_type;     /* type of file */
  char           d_name[?];  /* filename */
};
```

rewinddir – **Reset directory stream**

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

open()

- **int** open(**const char** *pathname, **int** flags[, mode_t mode]);

- **Flags**(combined with "|" character):
  - O_RDONLY, O_WRONLY, O_RDWR
  - O_CREAT [| O_EXCL]
  - O_TRUNC, O_APPEND
  - O_NONBLOCK, O_SYNC, O_NOFOLLOW

- **Mode**: specify the mode(chmod-like) in octal or macro format

read()

- **ssize_t** read(**int** fd, **void** *buf, **size_t** count)

- **Read** receives:
  - The file descriptor of the current read file
  - The address of the zone where the system puts the read octets
  - The number of octets we wish to read
- **Read** returns:
  - Real number of read octets
  - 0 if end of file
  - -1 if an error occurs(see **errno**)

- **Write** follow the same rules as **read,** but for writing

stat()

```
int stat(const char *file_name, struct stat *buf);

struct stat {
        dev_t           st_dev;         /* Device                               */
        ino_t           st_ino;         /* Inode number                         */
        mode_t          st_mode;        /* Protection                           */
        nlink_t         st_nlink;       /* Physical link number                 */
        uid_t           st_uid;         /* Owner UID                            */
        gid_t           st_gid;         /* Owner GID                            */
        dev_t           st_rdev;        /* Device type                          */
        off_t           st_size;        /* Total size (octets)                  */
        unsigned long   st_blksize;     /* Block size for I/O                   */
        unsigned long   st_blocks;      /* Allowed blocks number                */
        time_t          st_atime;       /* Last access time(hour)               */
        time_t          st_atime;       /*  Last modification time(hour)        */
        time_t          st_atime;       /*  Last system modification time(hour) */
}
```

C library file manipulation (1/2)

FILE **fdopen***(int** *fd*, **const char** **mode***);**

- Change a file description to **FILE***

FILE **freopen***(const char** **path*, **const char** **mode*, **FILE** **stream***);**

- Open the *path* file and link it to the *stream* FILE(flux)
- The older file opened by *stream* is closed
- Example:
  - freopen("/tmp/errlog", "w+", *stderr*);
  - Redirect the *stderr* flux to the file "/tmp/errlog"

C library file manipulation (2/2)

- **char** **strstr*(**const char** **haystack*, **const char** **needle*);
- **char** **strncpy*(**char** **dest*, **const char** *src, **size_t** *n*);
- **char** **strncat*(**char** **dest*, **const char** *src, **size_t** *n*);
- **int** *strncmp*(**const char** **s1*, **const char** *s2, **size_t** *n*);
- **void** **memcpy*(**void** *dest, **const void** *src, **size_t** *n*);
- **void** **memmove*(**void** **dest*, **const void** *src, **size_t** *n*);
  - Overlap allowed
- **void** **memset*(**void** **s*, **int** *c*, **size_t** *n*);
- **int** *sprintf*(**char** **str*, **const char** **format*, ...);
- *atof, atoi, atol*
- **and static variables**

Exercise

- Create the program **mycp** with makes a copy of a file
- **mycp** takes two arguments : name of the source file and name of the destination file
- **mycp** read the source file in blocks of 4kiB each, and copy each of those blocks in the destination file, which will be created or truncdated if necessary