File Systems

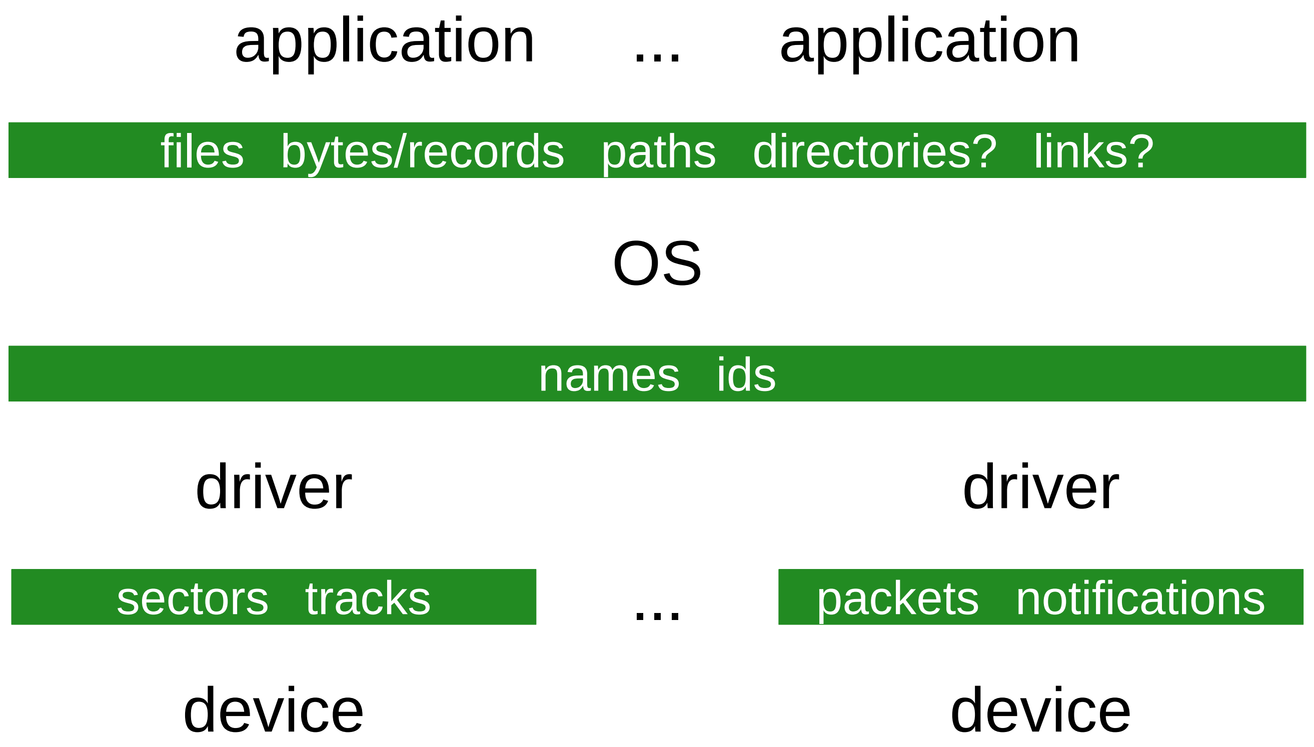
File system = most common abstraction for ***persistence***

Also provides

- Large-data storage with random access
- Data organization
- Mobility (e.g., CD ROM, NFS)
- Sharing & protection
- Communication

“File system” sometimes refers to the abstraction and sometimes refers to a particular disk format. We mean the former.

File system Layers



Files

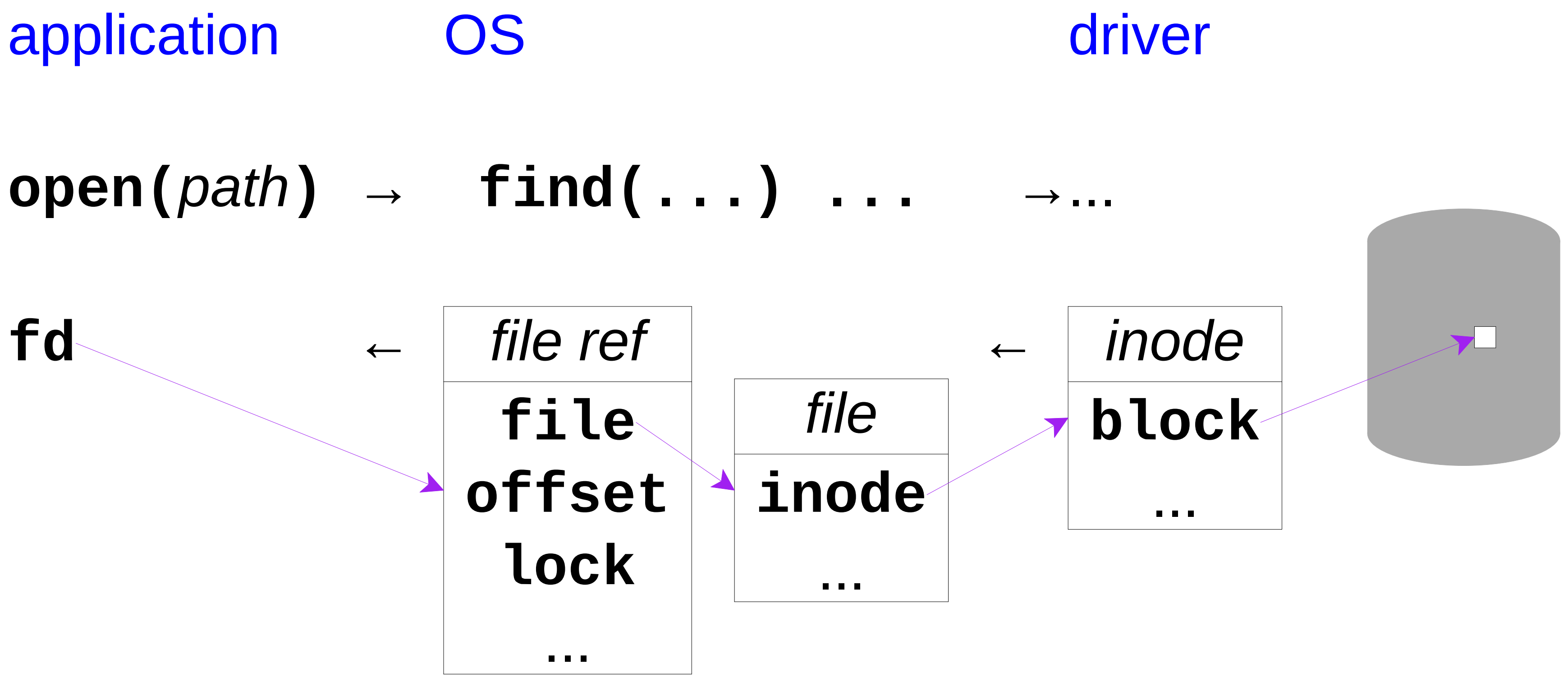
Typically, a ***file*** is

- A sequence of bytes
- Metadata, including modification time, permissions, and type

Typically, a file is accessed through a ***path***

- Access results in a ***file descriptor*** or ***file handle***
- Descriptor or handle sticks with a file, while the path can change

Opening a File



Common File System Operations

Data

- **`Create()`**
- **`Delete()`**
- **`Open()`**
- **`Close()`**
- **`Read()`**
- **`Write()`**
- **`Seek()`**

Naming

- **`Rename()`**
- **`HardLink()`**
- **`SoftLink()`**

Metadata

- **`GetAttribute()`**
- **`SetAttribute()`**

`Create()`

Unix:

`int open(const char *path, int oflag, mode_t mode);`
with **`O_CREAT`**

also opens

`int mkdir(const char *path, mode_t mode);`

Windows:

`HANDLE CreateFile(LPCTSTR lpFileName, ....);`
with **`CREATE_ALWAYS`**

also opens

`HANDLE CreateDirectory(LPCTSTR lpPathName, ....);`

see **`create.c`**

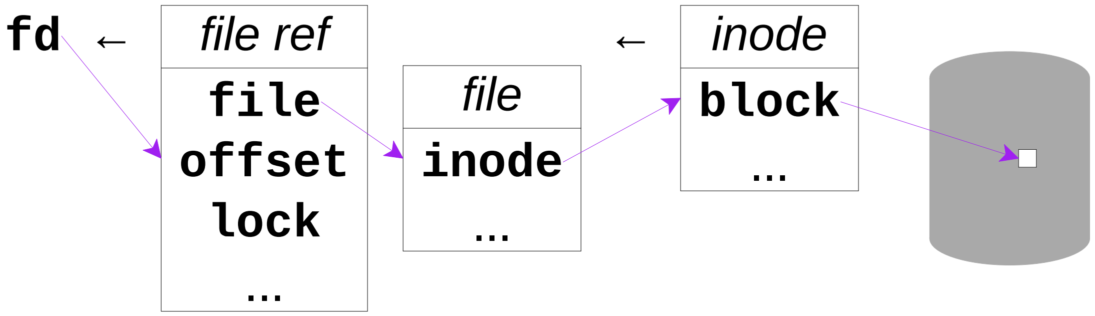
`Delete()`

Unix:

```
int unlink(const char *path);
int rmdir(const char *path);
```

Windows:

```
BOOL DeleteFile(LPCTSTR lpPathName);
BOOL RemoveDirectory(LPCTSTR lpPathName);
```



Removes the path mapping, but doesn't actually delete until all references are closed (see **`create.c`**)

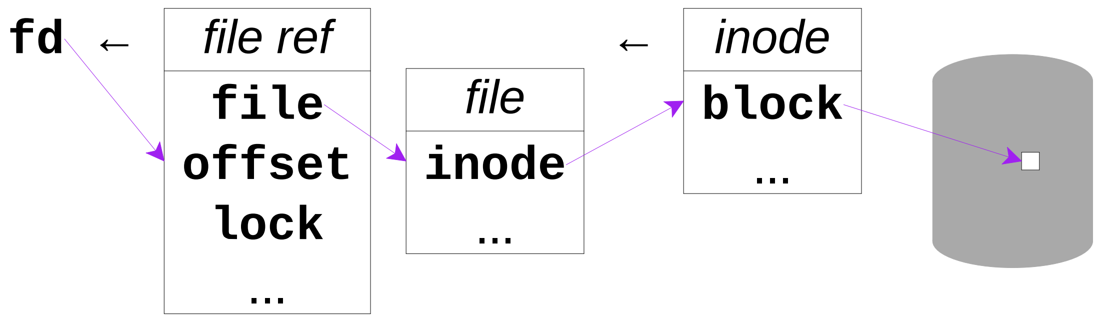
Open()

Unix:

```
int open(const char *path, int oflag);
```

Windows:

```
HANDLE CreateFile(LPCTSTR lpFileName, ....);
```



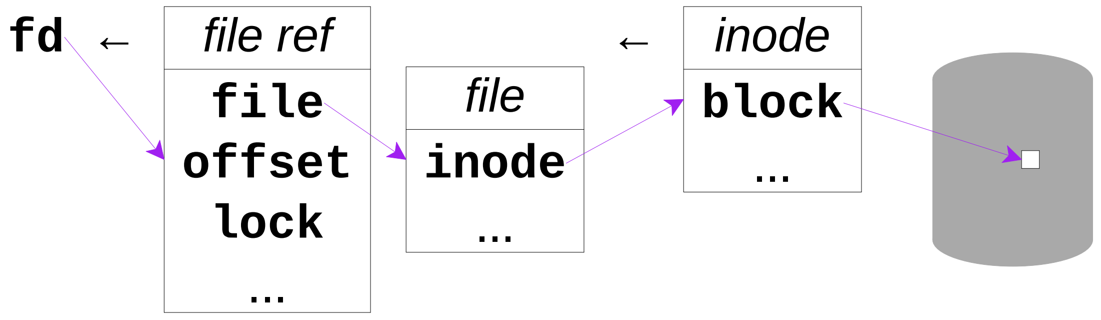
`Close()`

Unix:

```
int close(int filedes);
```

Windows:

```
BOOL CloseHandle(HANDLE hFile);
```



Last copy of decriptor/handle ⇒ free decriptor/handle

Last decriptor/handle ⇒ close file

see **`opens.c`**

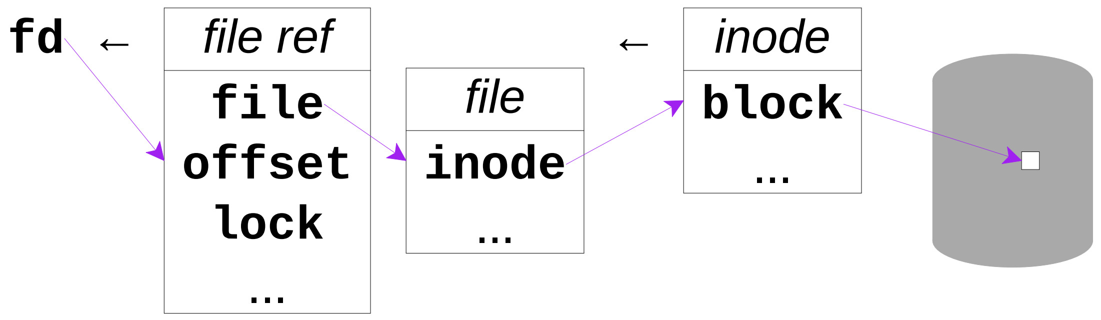
Read()

Unix:

```
ssize_t read(int fildes, void *buf, size_t nbyte);
```

Windows:

```
BOOL ReadFile(HANDLE hFile, LPVOID lpBuf, ....);
```



Updates descriptor/handle offset

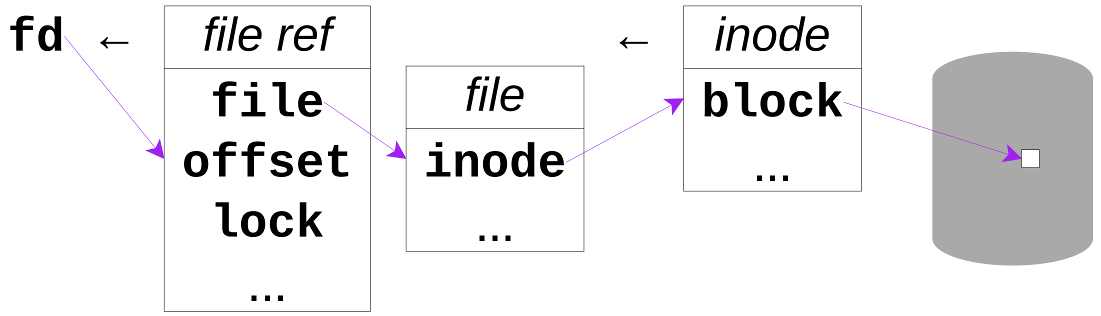
`Write()`

Unix:

```
ssize_t write(int fildes, void *buf, size_t nbyte);
```

Windows:

```
BOOL WriteFile(HANDLE hFile, LPVOID lpBuf, ....);
```



Updates descriptor/handle offset

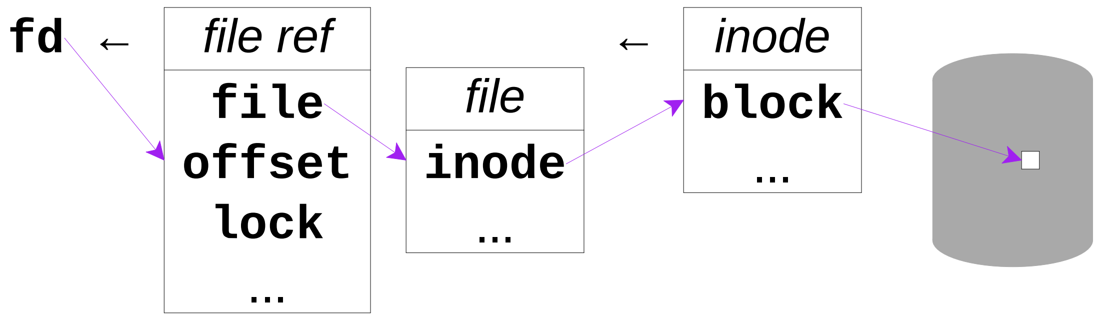
`Seek()`

Unix:

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

Windows:

```
DWORD SetFilePointer(HANDLE hFile, LONG lOff, ...);
```



Updates descriptor/handle offset

see **`share.c`**

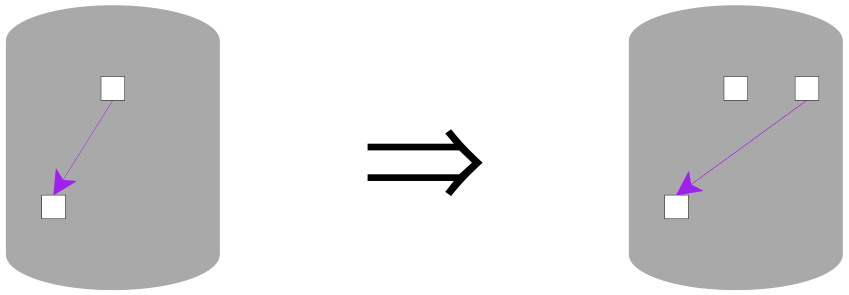
Rename()

Unix:

```
int rename(const char *old, const char *new);
```

Windows:

```
BOOL MoveFile(LPCTSTR lpOld, LPCTSTR lpNew);
```



No effect on open descriptors/handles

Atomic update when on the same device

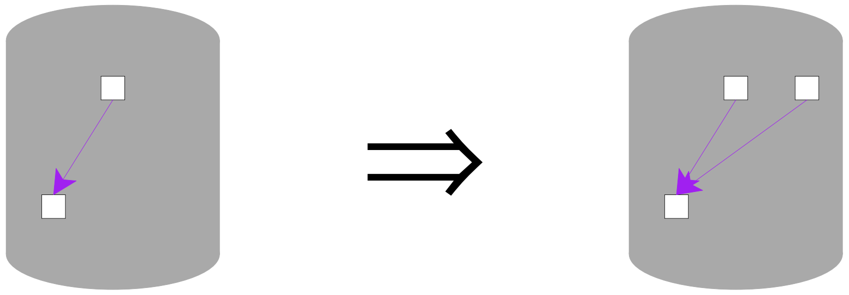
`HardLink()`

Unix:

```
int link(const char *old, const char *new);
```

Windows:

```
BOOL CreateHardLink(LPCTSTR lpNew, LPCTSTR lpOld,
                    ...);
```



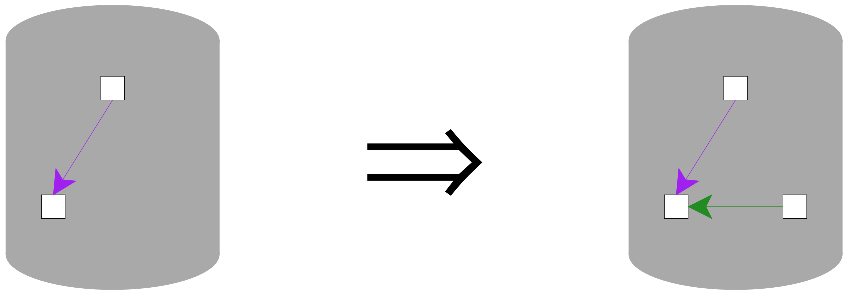
No effect on open descriptors/handles

see **`share2.c`**

SoftLink()

Unix:

```
int symlink(const char *path, const char *new);
```



No effect on open descriptors/handles

GetAttribute()

Unix:

```
int fstat(int filedes, struct stat *buf);
```

Windows:

```
BOOL GetFileInformationByHandle(HANDLE hFile, ...);
```

File type, size, maybe permissions

`SetAttribute()`

Unix:

```
int fchmod(int fildes, mode_t mode);

int futimes(int fildes, struct timeval times[2]);
```

Windows:

```
BOOL SetFileInformationByHandle(HANDLE hFile, ...);
```

File type, size, maybe permissions

Unix Paths

- A ***path*** is a sequence of byte-strings elements, where **`/`** is disallowed in an element

```
usr  local  bin  pdf2ps
```

- A path is normally written as a single byte string using **`/`** as a separator
  - Path starts with **`/`** ⇒ absolute

```
/usr/local/bin/pdf2ps
```

  - Path does not start with **`/`** ⇒ relative

```
bin/pdf2ps
```

- Each process has a ***working directory*** that prefixes relative paths

Unix Paths

- A device is ***mounted*** at one or more path prefixes

```
$ /usr/bin/mount
/dev/sda2 on / type ext3 (rw)
proc on /proc type proc (rw)
sysfs on /sys type sysfs (rw)
devpts on /dev/pts type devpts (rw,gid=5,mode=620)
/dev/sda5 on /usr/vice type ext3 (rw)
/dev/sda1 on /boot type ext3 (rw)
zfs:/server/home2/mflatt on /home/mflatt type nfs ...
```

Unix Paths

- OS communicates to driver in terms of IDs, known as ***inodes*** and immediate names
    - A file is a kind of inode
    - A directory is a kind of inode
    - A ***hard link*** is when a directory points to a file's inode
    - A ***soft link*** is an inode that contains another path, automatically followed (usually) by the OS
- Case sensivity is managed by the driver
    - ext3 (Linux) is case-sensitive
    - HFS+ (Mac OS) is case-insensitive by default

Windows Paths

- A ***path*** combines a drive with a UTF-16 code unit sequence
- A path is normally written as a single string using a letter name for a drive and **`\`** as a separator, in which case **`<`**, **`>`**, **`:`**, **`"`**, **`/`**, **`\`**, and **`|`** are disallowed in an element

```
C:\Program Files\PLT\DrScheme.exe
```

- A drive can also be **`\\`***machine***`\`***volume*
- Except that special files names like **`aux`** refer to devices, independent of the drive, path, or extension
- At some layers of the Windows API, various automatic transformations are applied, such as converting **`/`** to **`\`** and dropping trailing spaces

```
C:/Program Files/PLT\DrScheme.exe
```

Windows Paths

- Path starts with drive and **`\`** ⇒ absolute

  `C:\Program Files\PLT\DrScheme.exe`

- Path does not start with drive or **`\`** ⇒ relative

  `PLT\DrScheme.exe`

- Path starts with drive but not **`\`** ⇒ drive-relative

  `C:PLT\DrScheme.exe`

- Path starts with **`\`** ⇒ drive-absolute

  `\Program Files\PLT\DrScheme.exe`

- Each process has a ***working drive*** and each drive per process has a ***working directory***

Windows Paths

- OS communicates to driver in terms of paths
  - Use the **`\\?\`** prefix to specify driver path directly

```
\\?\c:\wE|Rd\<path>
```

- Case sensivity is managed by the OS

Paths

- Generally cannot get a cannonical path for a file
    - The path can change
    - May have multiple mount points
    - May have multiple links
- File descriptor/handle provides cannonical references
    - e.g, get inode
    - Only works for open files

Locks

What if cooperating processes want to modify a file, and only one process should modify the file at a time?

- ***Advisory locks*** — provided by the OS to let cooperating programs declare exclusive access
    - Unix, typically
- ***Mandatory locks*** — provided by the OS to let programs (cooperative or not) gain exclusive access
    - Windows

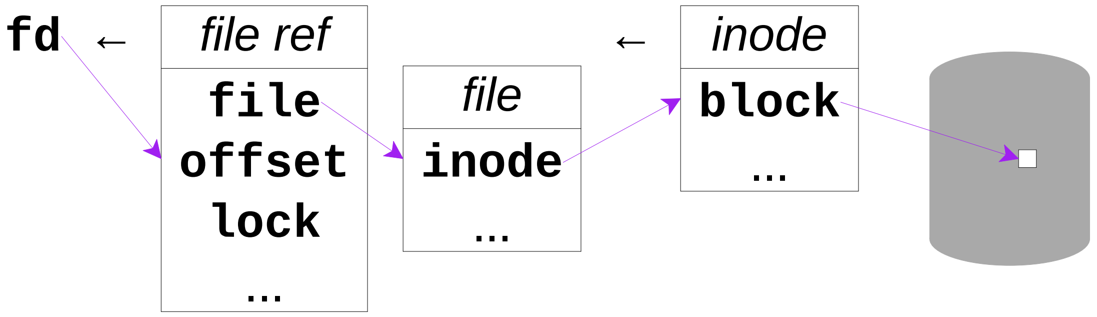
`Lock()` and `Unlock()`

Unix:

```
int flock(int fd, int operation);
```

Windows:

```
HANDLE CreateFile(LPCTSTR lpFileName, ....,
                  DWORD dwShareMode, ....);
```



see **`locks.c`**, **`locks2.c`**, **`locks2.c`**

Permissions

- ***Access-control list*** (ACL) determines for each file which userid can perform which of a handful of operations
    - Typical operations: read, write, execute, append, delete, list
- Unix-style simplified mapping:
    - owner vs. group vs. everyone
    - read, write, execute