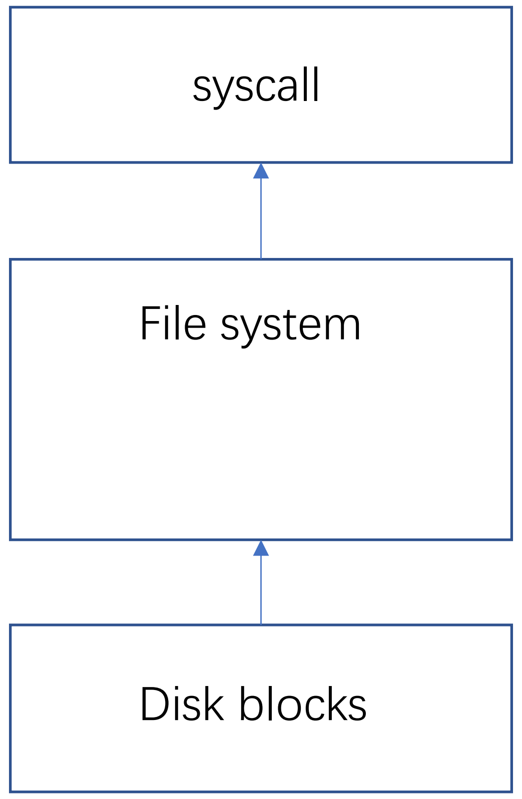
File System

Pintos lab4 session(by Ivory E.Si)

Filesystem: A layer of indirection over the disk

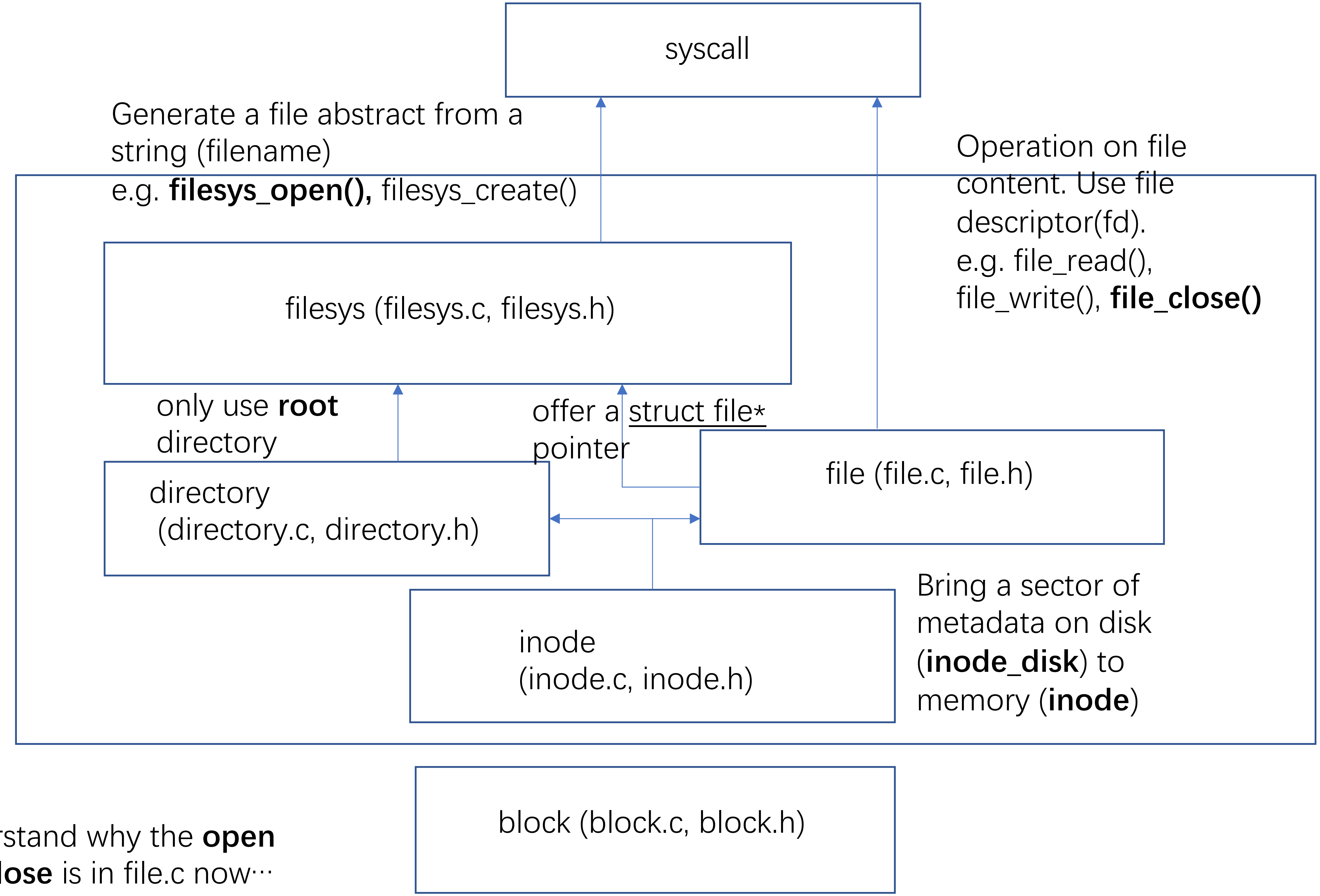


open(), read(), close(), remove(), mkdir(), chdir()…

filesys_open(), file_read(), file_close()…
filesys.c, file.c, directory.c,inode.c…

block_read(), block_write()

Some subtle design philosophy in pintos: interfaces are in .h files implementations are in .c files



Generate a file abstract from a string (filename)
e.g. **filesys_open(),** filesys_create()

Operation on file content. Use file descriptor(fd).
e.g. file_read(), file_write(), **file_close()**

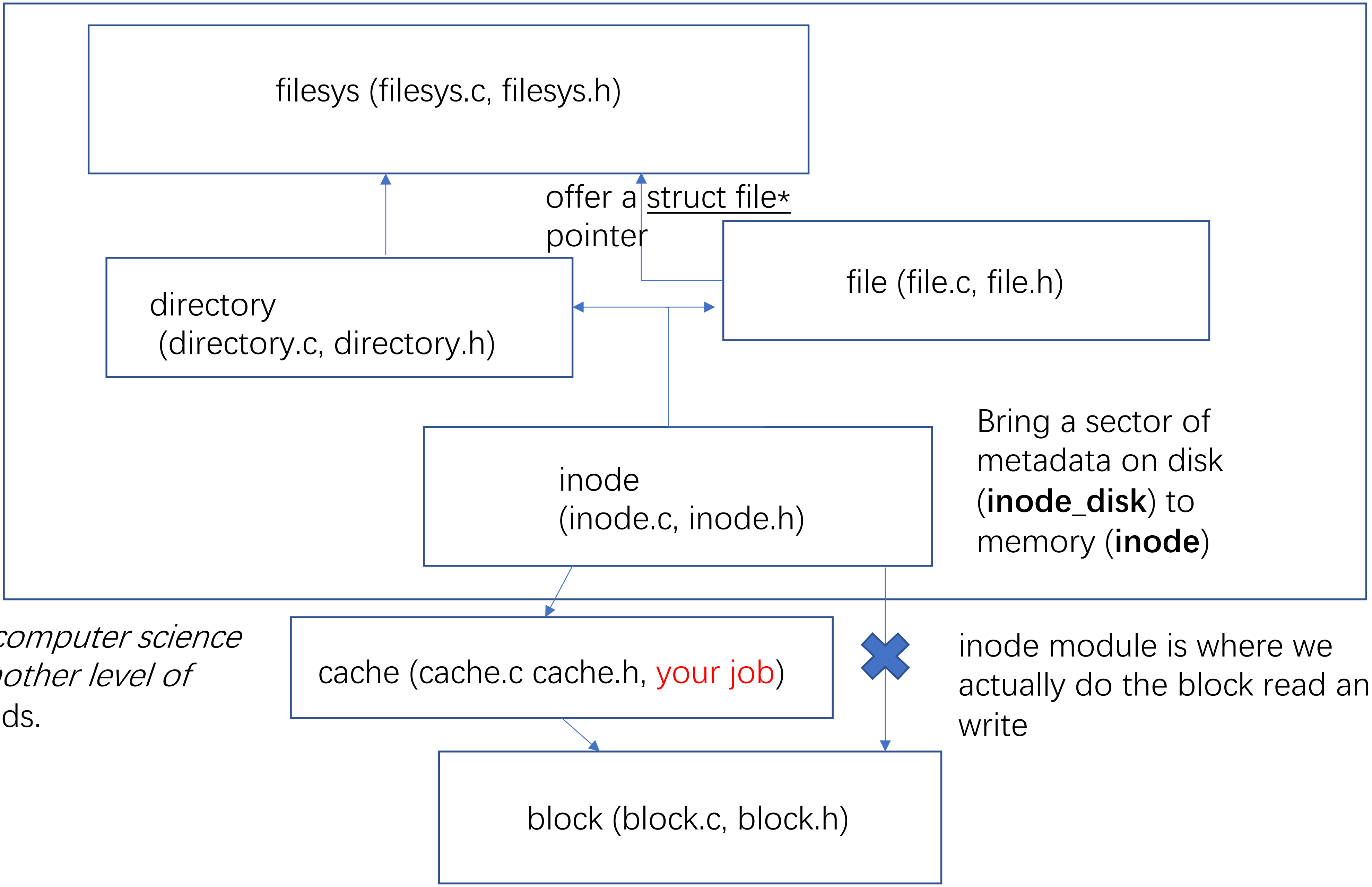
only use **root** directory

offer a <u>struct file*</u> pointer

Bring a sector of metadata on disk (**inode_disk**) to memory (**inode**)

So, you may understand why the **open** is in filesys.c and **close** is in file.c now…

Why cache is the **simplest** to implement?



filesys (filesys.c, filesys.h)
offer a struct file* pointer
directory (directory.c, directory.h)
file (file.c, file.h)
inode (inode.c, inode.h)
cache (cache.c cache.h, your job)
block (block.c, block.h)

Bring a sector of metadata on disk (**inode_disk**) to memory (**inode**)

inode module is where we actually do the block read and write

So, *all problems in computer science can be solved by another level of indirection,* my friends.

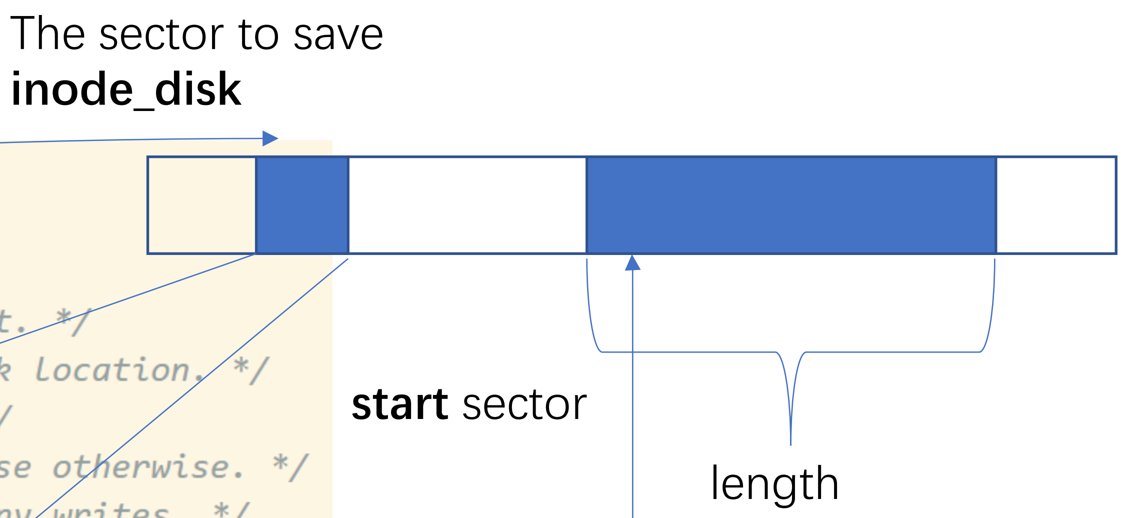
What is the entity of file?

Structure in inode.c

```c
/** In-memory inode. */
struct inode
  {
    struct list_elem elem;              /**< Element in inode list. */
    block_sector_t sector;              /**< Sector number of disk location. */
    int open_cnt;                       /**< Number of openers. */
    bool removed;                       /**< True if deleted, false otherwise. */
    int deny_write_cnt;                 /**< 0: writes ok, >0: deny writes. */
    struct inode_disk data;             /**< Inode content. */
  };
```

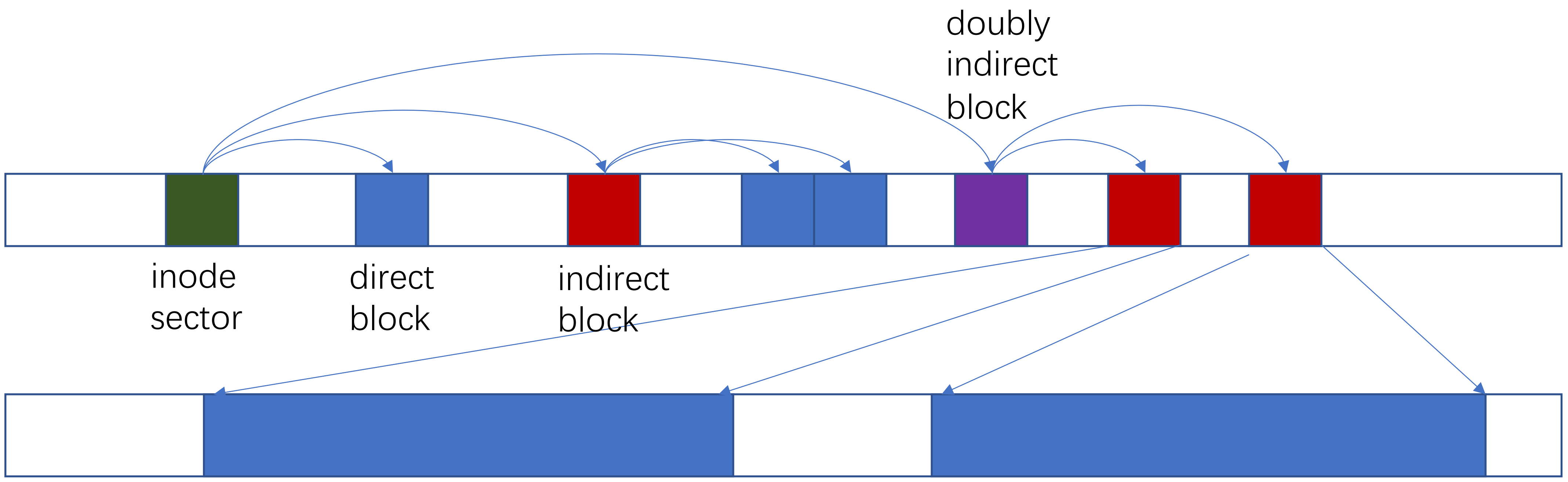
```c
/** On-disk inode.
   Must be exactly BLOCK_SECTOR_SIZE bytes long. */
struct inode_disk
  {
    block_sector_t start;               /**< First data sector. */
    off_t length;                       /**< File size in bytes. */
    unsigned magic;                     /**< Magic number. */
    uint32_t unused[125];               /**< Not used. */
  };
```

Hint: use these fields.



But you should implement a file entity with discontinuous blocks assignment

Indexed Files

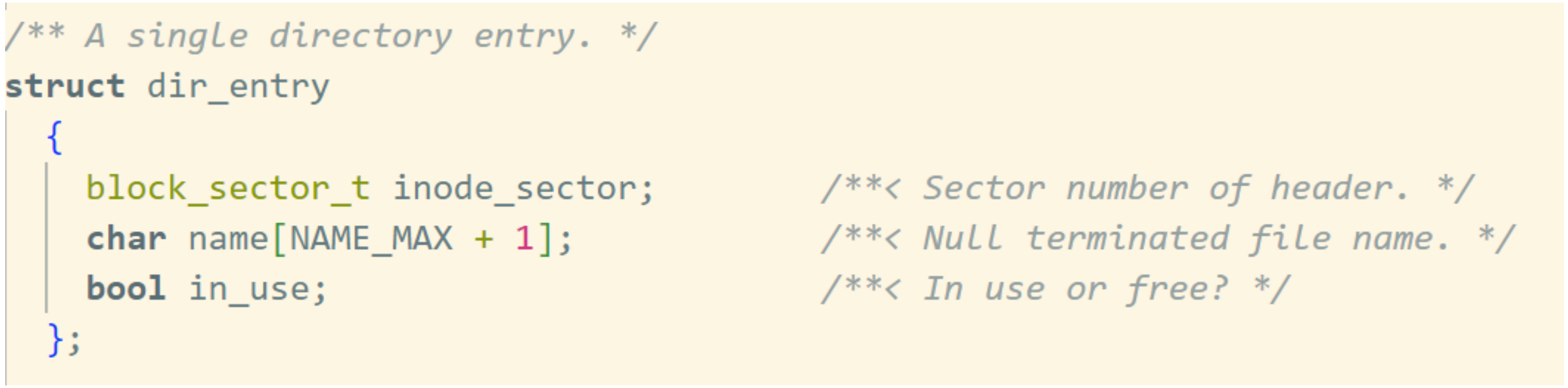


As you may have guessed, use place in the sector to save index entries.

Subdirectories

Actually, directory is a **FILE** which saves entries of files or subdirectories.

But we still need to identify **type of an inode.**(So, what mark will you use to indicate it and where to save this information?) Is this inode a common file or a directory?



```c
/** A single directory entry. */
struct dir_entry
  {
    block_sector_t inode_sector;        /**< Sector number of header. */
    char name[NAME_MAX + 1];            /**< Null terminated file name. */
    bool in_use;                        /**< In use or free? */
  };
```

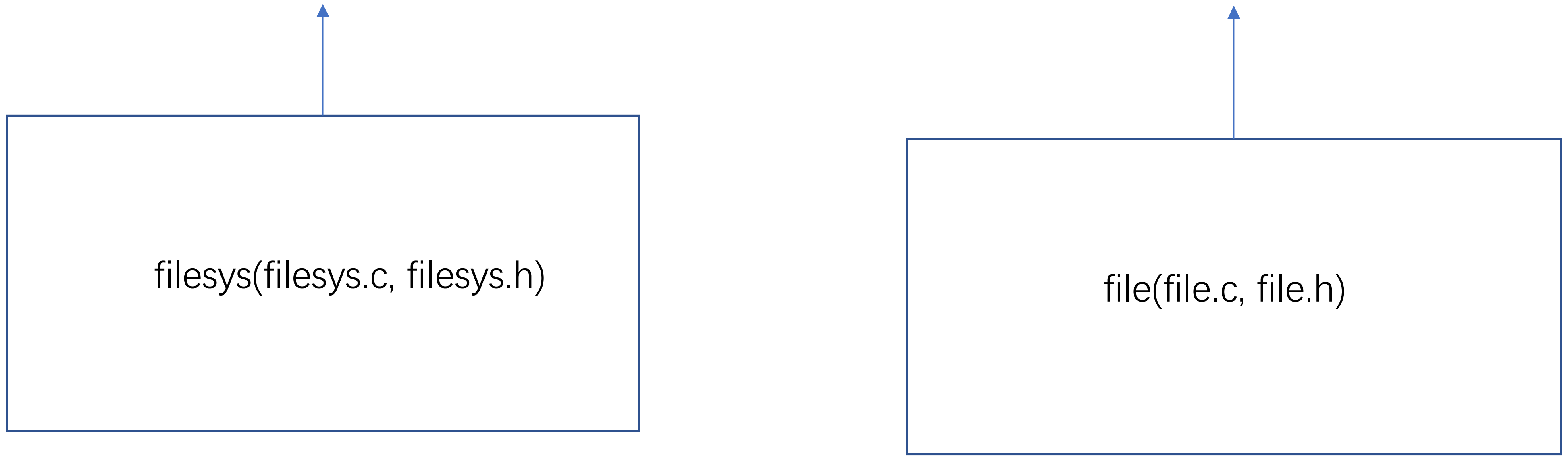
directory entry in directory.c

About adding some new system calls

If you treat your directory as “file” , there is a design parttern. (It is a bit tricky in readdir)

From directory name to a file abstraction.
bool chdir (const char *dir)
bool mkdir (const char *dir)

From “file” (directory?) descriptor to operate the directory
bool readdir (int fd, char *name)
bool isdir (int fd)
int inumber (int fd)

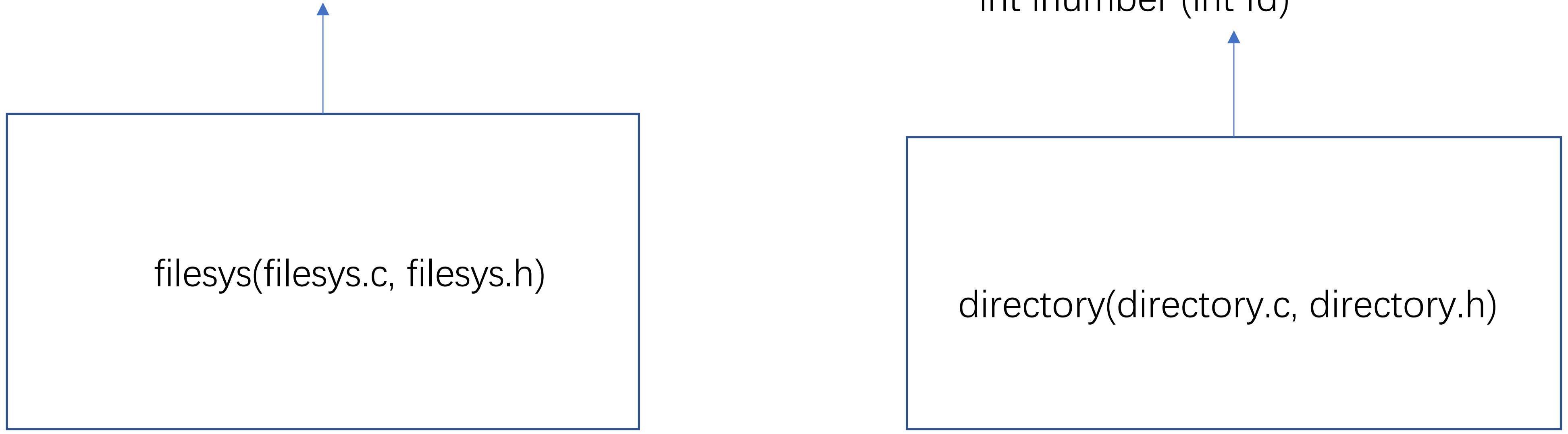


About adding some new system calls

Another way is to use directory module to serve these system calls.
It may need some changes on your file descriptor structure.

From directory name to a file abstraction.
bool chdir (const char *dir)
bool mkdir (const char *dir)

From "file" (directory?) descriptor to operate the directory
bool readdir (int fd, char *name)
bool isdir (int fd)
int inumber (int fd)


filesys(filesys.c, filesys.h)
directory(directory.c, directory.h)

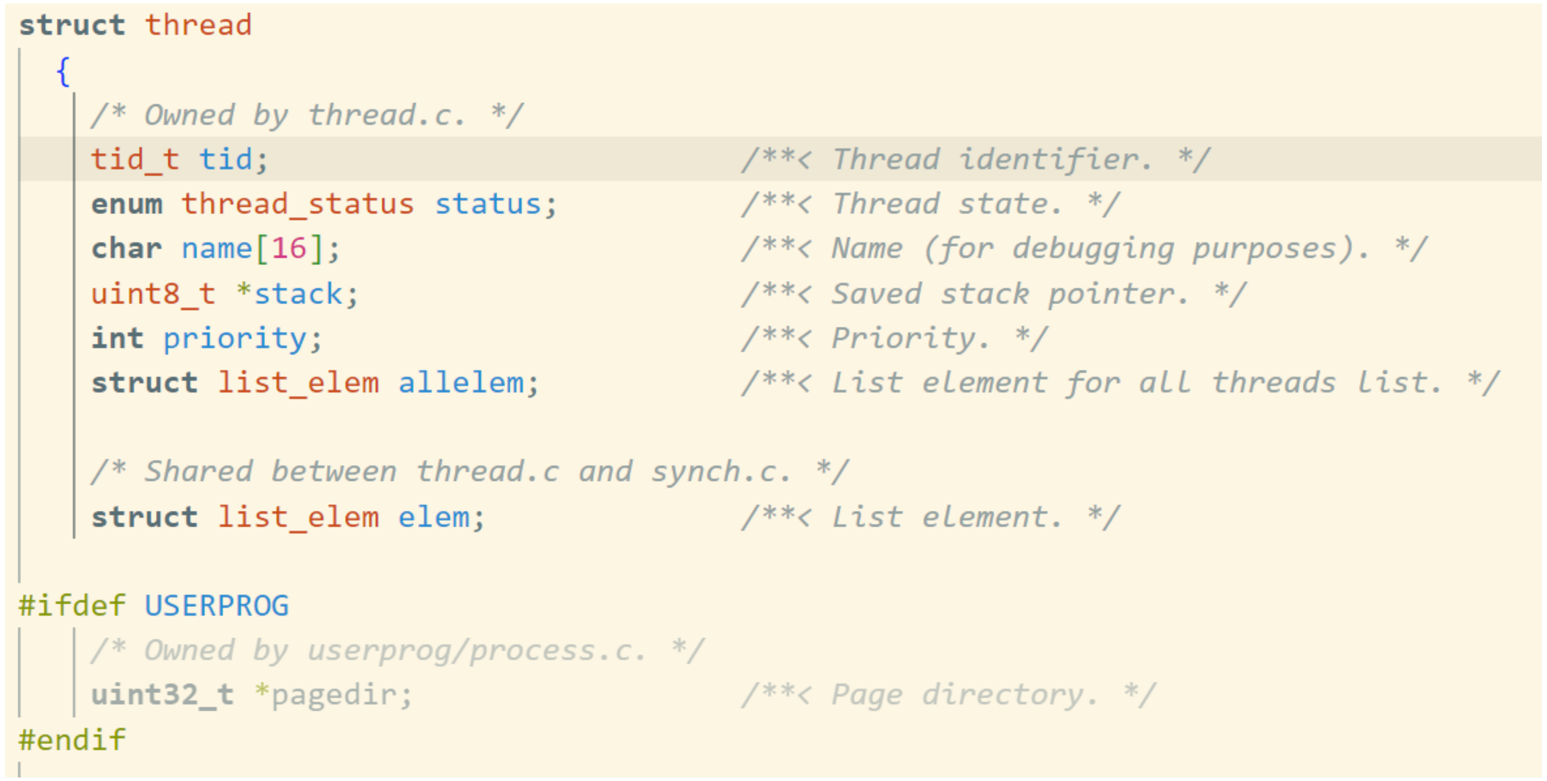
About adding some new system calls

You can design your file system freely!

I’ d like to know about your elegant design pattern!

Object-Oriented in C language(?)

- In the thread lab, the details of the thread structure are exposed to all other source code files.

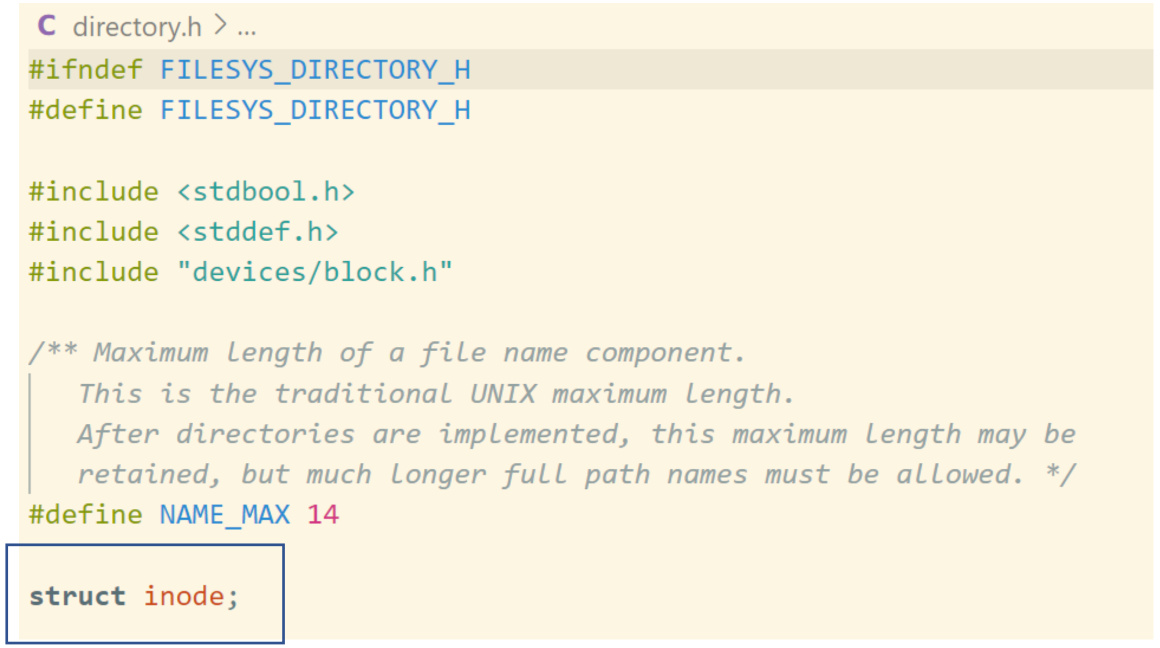


```c
struct thread
  {
    /* Owned by thread.c. */
    tid_t tid;                          /**< Thread identifier. */
    enum thread_status status;          /**< Thread state. */
    char name[16];                      /**< Name (for debugging purposes). */
    uint8_t *stack;                     /**< Saved stack pointer. */
    int priority;                       /**< Priority. */
    struct list_elem allelem;           /**< List element for all threads list. */

    /* Shared between thread.c and synch.c. */
    struct list_elem elem;              /**< List element. */

#ifdef USERPROG
    /* Owned by userprog/process.c. */
    uint32_t *pagedir;                  /**< Page directory. */
#endif
```

Then, you may have noticed that your code in the previous three labs are highly-coupled.

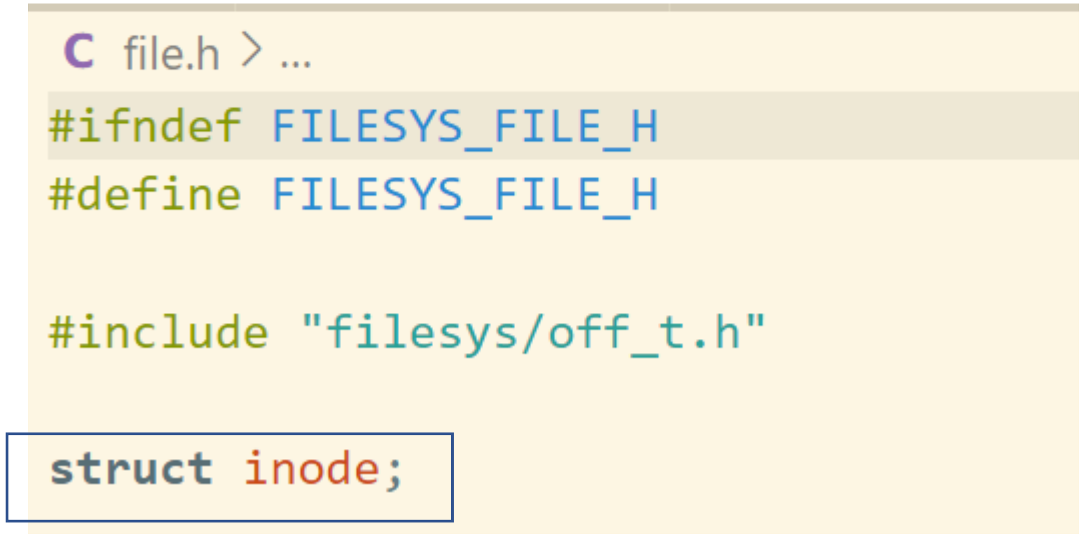
Object-Oriented in C language(?)

- But in this lab there is some tricks about modular design.



directory.h > ...

```
#ifndef FILESYS_DIRECTORY_H
#define FILESYS_DIRECTORY_H

#include <stdbool.h>
#include <stddef.h>
#include "devices/block.h"

/** Maximum length of a file name component.
   This is the traditional UNIX maximum length.
   After directories are implemented, this maximum length may be
   retained, but much longer full path names must be allowed. */
#define NAME_MAX 14

struct inode;
```



file.h > ...

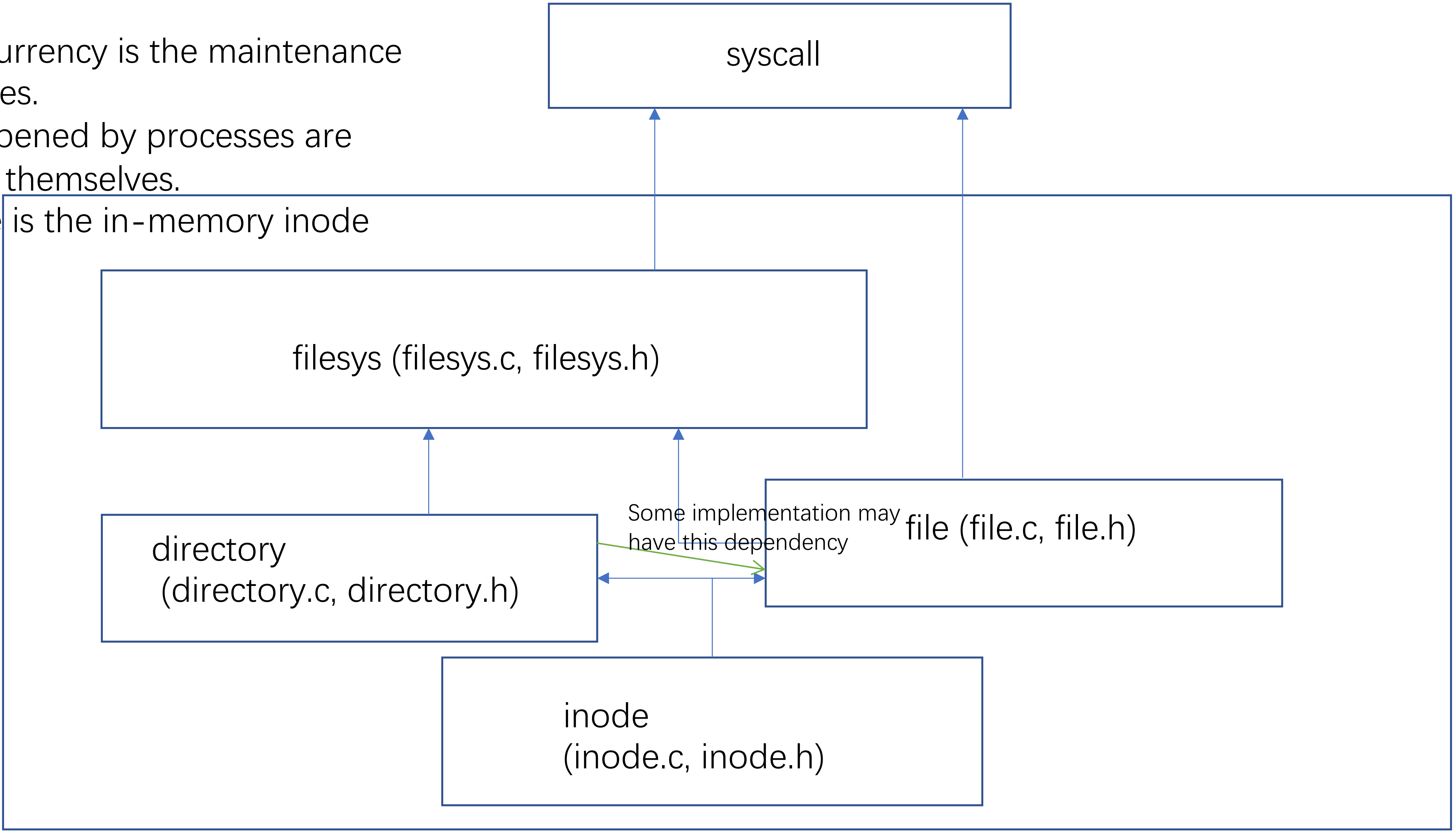
```
#ifndef FILESYS_FILE_H
#define FILESYS_FILE_H

#include "filesys/off_t.h"

struct inode;
```

The directory and file modules only know that "there is a inode structure" .
But they do not know the specific definition of the inode structure.

So you can take the advantage of this segregation to.............
············implement synchronization

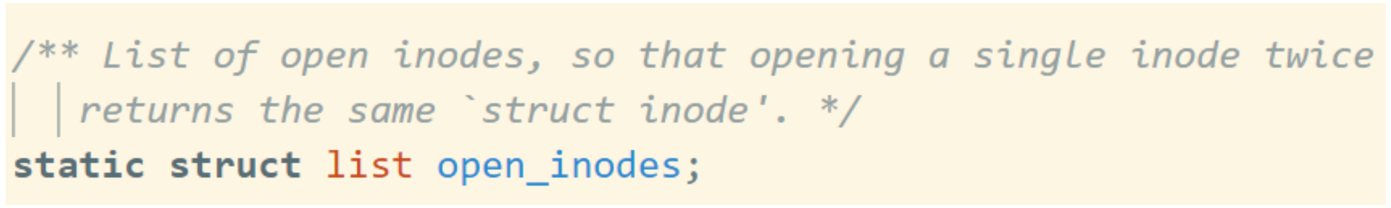
Synchronization

The essence of concurrency is the maintenance of shared global states.
Files or directories opened by processes are owned by processes themselves.
What we really share is the in-memory inode


syscall
filesys (filesys.c, filesys.h)
directory (directory.c, directory.h)
Some implementation may have this dependency
file (file.c, file.h)
inode (inode.c, inode.h)

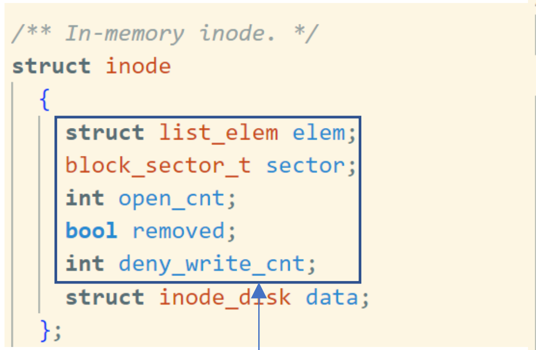
Well, you may have noticed that if we make inode a race-free black box, our task is finished.

What the inode module have

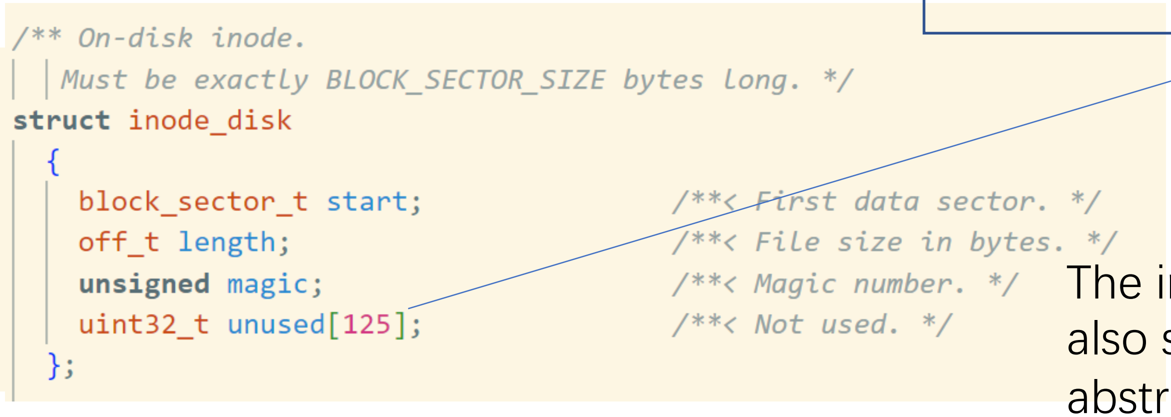


```c
/** List of open inodes, so that opening a single inode twice
    returns the same `struct inode'. */
static struct list open_inodes;
```

A inodes list. (It is global! Watch out!)



```c
/** In-memory inode. */
struct inode
  {
    struct list_elem elem;
    block_sector_t sector;
    int open_cnt;
    bool removed;
    int deny_write_cnt;
    struct inode_disk data;
  };
```



```c
/** On-disk inode.
    Must be exactly BLOCK_SECTOR_SIZE bytes long. */
struct inode_disk
  {
    block_sector_t start;               /**< First data sector. */
    off_t length;                       /**< File size in bytes. */
    unsigned magic;                     /**< Magic number. */
    uint32_t unused[125];               /**< Not used. */
  };
```

The index sectors or content sectors are also shared. (As a file or a directory abstraction offered to users)

Some in-memory information are shared.

All the variables shared with two or more threads need to be protected.

You may encounter a similar setting like the reader-writer problem.

Don’ t forget your cache module

- If you treat cache module as a indenpent part like me, try to find some ways to make it a race-free black box with the knowledge in the OS class.

At last

- If you don’ t have enough confidence on your code in previous three labs, I don’ t suggest you do this lab for PF…
  - I once deeply believed my code in previous labs, but after finishing this lab, I think that my previous implementation sucks.
- Well, I do not think my code writing ability is outstanding. You can have a try anyway.

Q&A

- If you have any question, you can add me from the wx group or send email to sigongzi@stu.pku.edu.cn (?)

- Well…I also know that I am notorious and you may do not want to talk with me. Anyway, have fun with your system design.