OS161

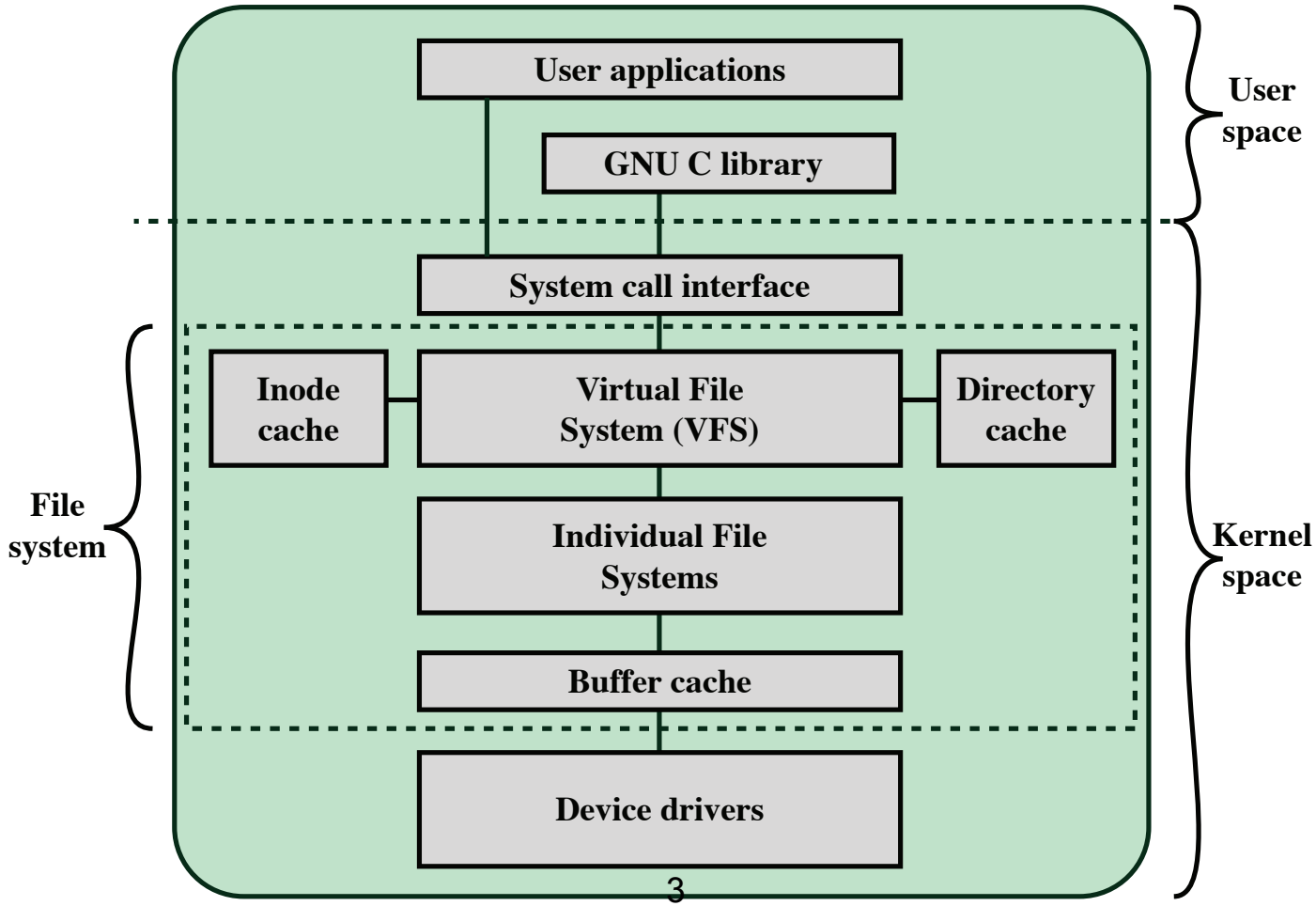
File open/close

System Calls Related to File Systems

unistd.h

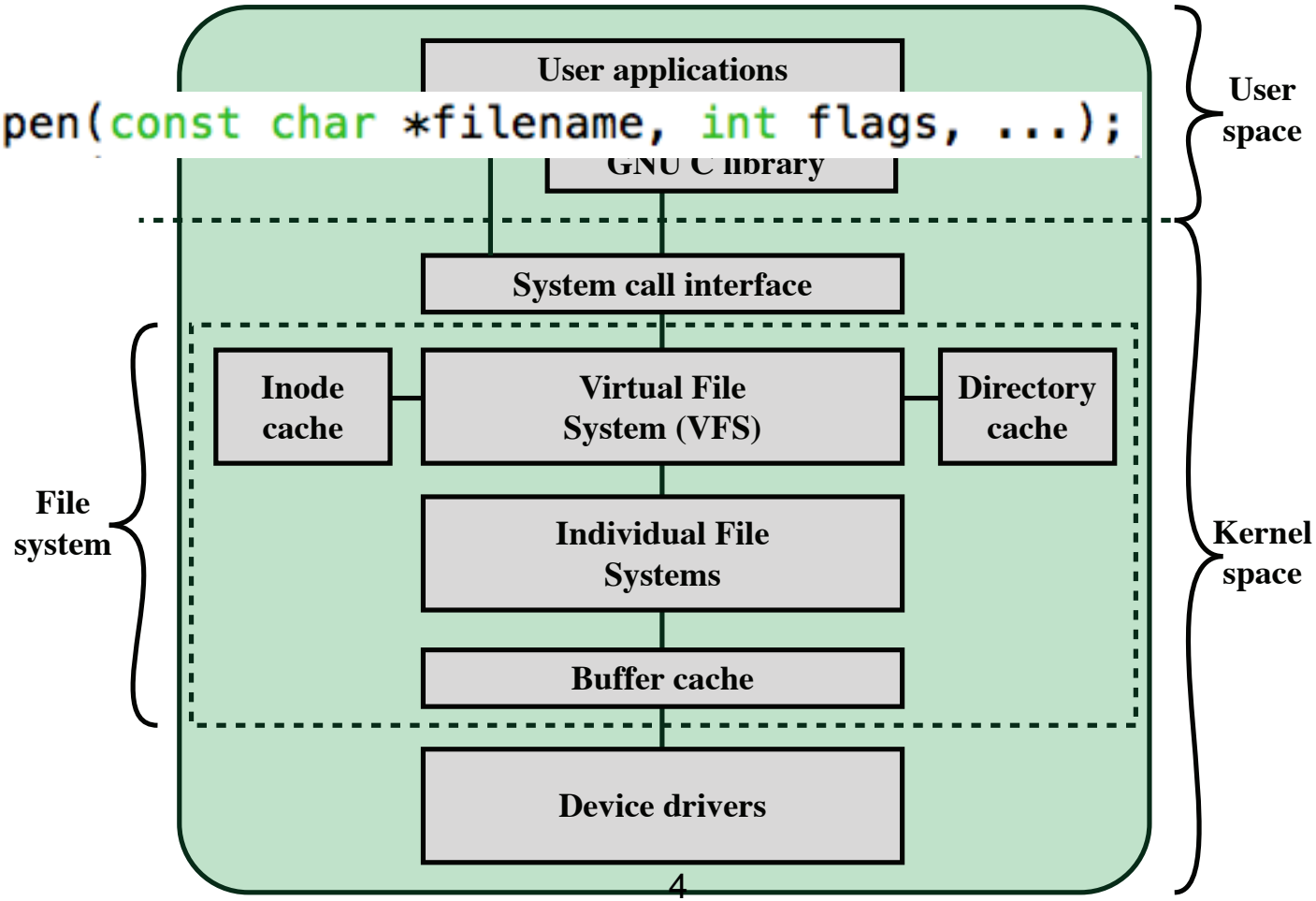
```
/*
 * Open actually takes either two or three args:
 * the optional third arg is the file mode used for creation.
 * Unless you're implementing security and permissions,
 * you can ignore it.
 */
int open(const char *filename, int flags, ...);
ssize_t read(int filehandle, void *buf, size_t size);
ssize_t write(int filehandle, const void *buf, size_t size);
int close(int filehandle);
off_t lseek(int filehandle, off_t pos, int code);
int dup2(int filehandle, int newhandle);
```

System Calls Interface

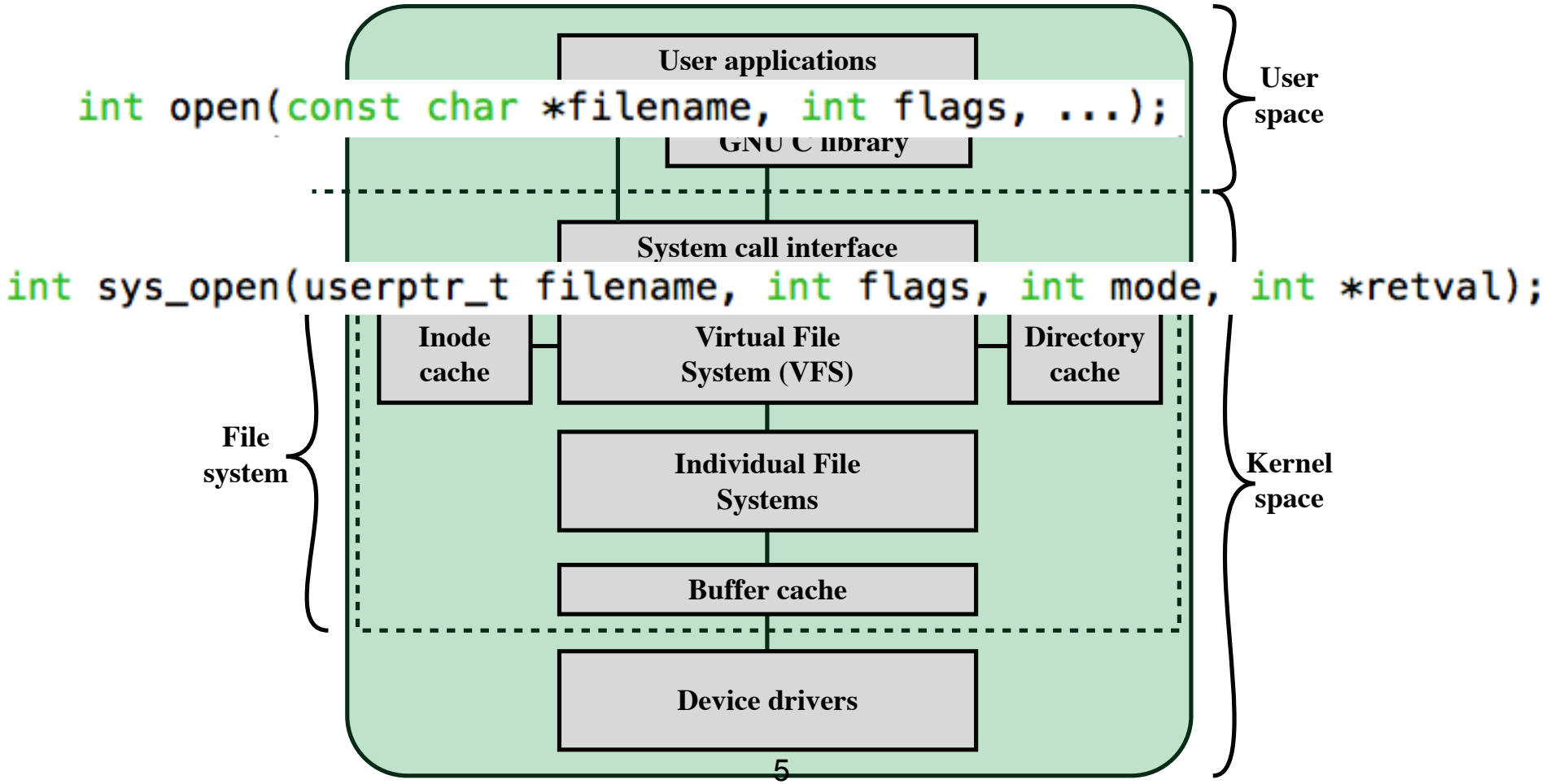


System Calls Interface

```
int open(const char *filename, int flags, ...);
```



System Calls Interface



Data Structure Questions

1. How to represent opened files for each process?
2. Where should you keep the above information?

Do not consider data structure for storing and organizing data in files (inode/vnode)

Focus on meta data of files

Note: Forked process (child) shares files with parent (same offset into open file)

Data Structure Question 1:

How to represent opened files for each process?

Answer:

- The `openfile` structure
- An array (or list) of `openfile*` items: `fileTable (per-process)`
- The array is referred by a `File Descriptor` (FD)

Each opened file has an `openfile` struct

`Read()/write()/lseek()` use file descriptor to operate each opened file

How will you design the `openfile` structure?

File pointer:

- To locate data
- A pointer to `vnode` (How to obtain it?)
- `vnode` is defined in From `vfs_open()`

  `kern/include/vnode.h`

Mode: Read-only, write-only, read-write, etc.

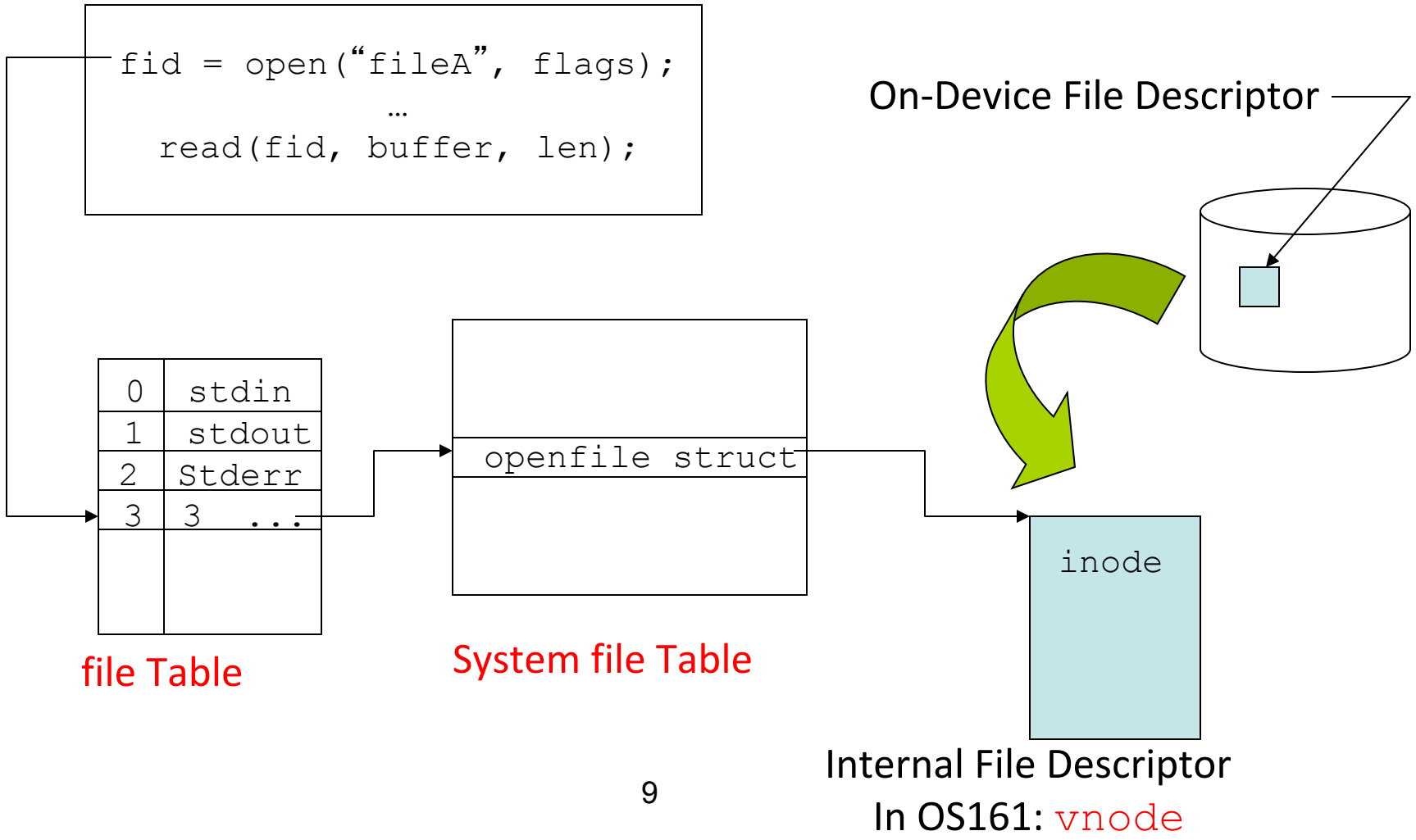
Offset

Lock

Reference count

User-Level Interface for System Calls

`include/unistd.h`



Can a file be opened multiple times?

Have two separate `openfile` struct

Indicated by two file descriptors

Will `openfile` be shared by concurrent threads?

No. You do not need to deal with the critical section issue.

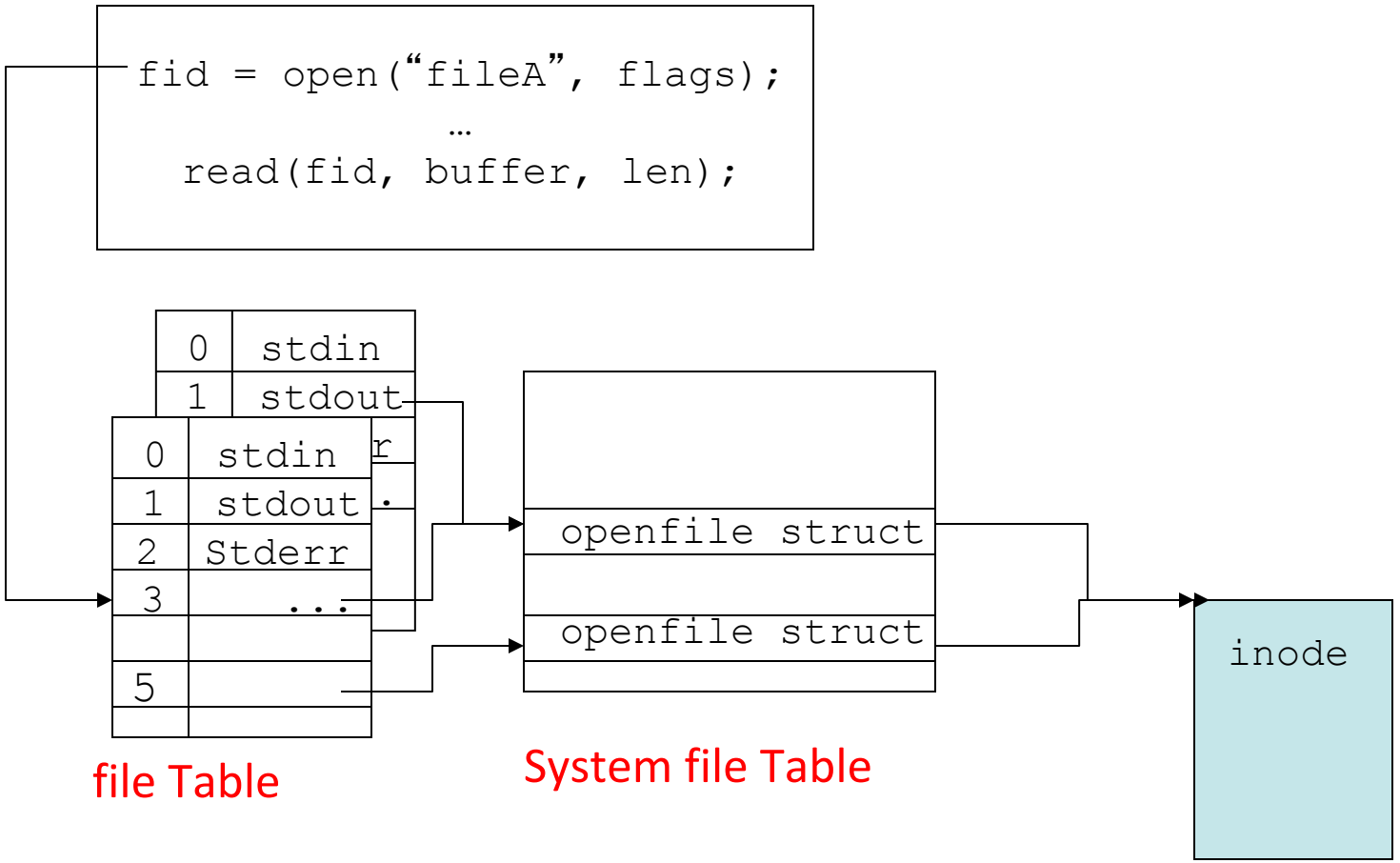
- Consider this first

Yes. You need use lock to protect this shared `openfile`

- If you have time, implement this feature
- Question: Where to add a lock?

in the `openfile` struct

Shared vnode vs. shared openfile



Data Structure Question 2:

Where should you keep `fileTable`?

Answer:

- Place `fileTable` in the `proc` struct
- The `proc` struct is defined in `kern/include/proc.h`

Each opened file has an `openfile` struct

`Read()/write()/lseek()` use file descriptor to operate each opened file

Algorithm Questions: How to design

`sys_open(filename, flag, retfd)`?

Opens a file: create an `openfile` item

Obtain `vnode` from `vfs_open()`

Initialize `offset` in `openfile`

File descriptor `fd` = Place `openfile` in `systemFiletable` (Where is the table?)

Return the file descriptor of the `openfile` item.

Algorithm Questions:

How does `sys_open(filename, flag, retfd)` **call** `vfs_open()` **?**

`vfs_open()`: prototype in

- `kern/include/vfs.h`

Check sample code here:

- `kern/test/fstest.c`

```
struct vnode *vn;
...
err = vfs_open(name, flags, &vn);
```

`err` from `vfs_open()`

Algorithm Questions: How to design `int sys_close(fd)`?

Use `fd` to locate the `openfile` item from `fileTable`

Are you supporting multiple opens?

- Delete `openfile` if this is the last open

Delete `openfile` from System fileTable

- Array: How to delete an item from an array?
- Singly-linked list: How to delete from a list?

Algorithm Questions: How to design

```
int sys_read(int fd, userptr_t buf, size_t size,
             int *retval)?
```

- To translate the file descriptor number to a file handle object
- To make a `uio` record: userspace I/O
  - See `kern/include/uio.h`
- To call `VOP_READ`
  - update the current seek position.
  - Prototype: `kern/include/vnode.h`
  - Sample Code: `kern/userprog/loadelf.c`
- The file is locked while this occurs.

Algorithm Questions: How to design

```
int sys_read(int fd, userptr_t buf, size_t size,
             int *retval)?
```

- Use `fd` to locate the `openfile` item from `fileTable`
- Access `offset` from `openfile`
- `userio` = setup a uio record
- Call `VOP_READ(openfile->vnode, userio)`
- `Openfile->offset = userio.offset;`
- Set `*retval` to the amount read