

CS310 Operating Systems

Lecture 37 : File System System – 5 File Descriptor Manipulation

Ravi Mittal

IIT Goa

Acknowledgements !

- Contents of this class presentation has been taken from various sources. Thanks are due to the original content creators:
 - CS162, Operating System and Systems Programming, University of California, Berkeley

Reading

- Book: Linux System Programming: talking directly to the kernel and C library, by Robert Love
- Class presentation: University of California, Berkeley, CS162

Previous Classes

C library High level APIs vs Linux Syscalls

- A C library function is a function implemented by the C library implementation
 - C interface the library provides programmers to access **kernel related functions**
- If a C programmers directly uses file syscalls, he/she needs to read the documentation
- Each syscall has a **number of arguments** – makes programming complicated
 - A user most of the time may not use all arguments
 - User space programs are expected to find out arguments for each syscall by for example inspecting the documentation
- A C library function is a function implemented by the C library implementation

Opening a file



```
#include <stdio.h>
FILE *fopen( const char *path, const char *mode );
```

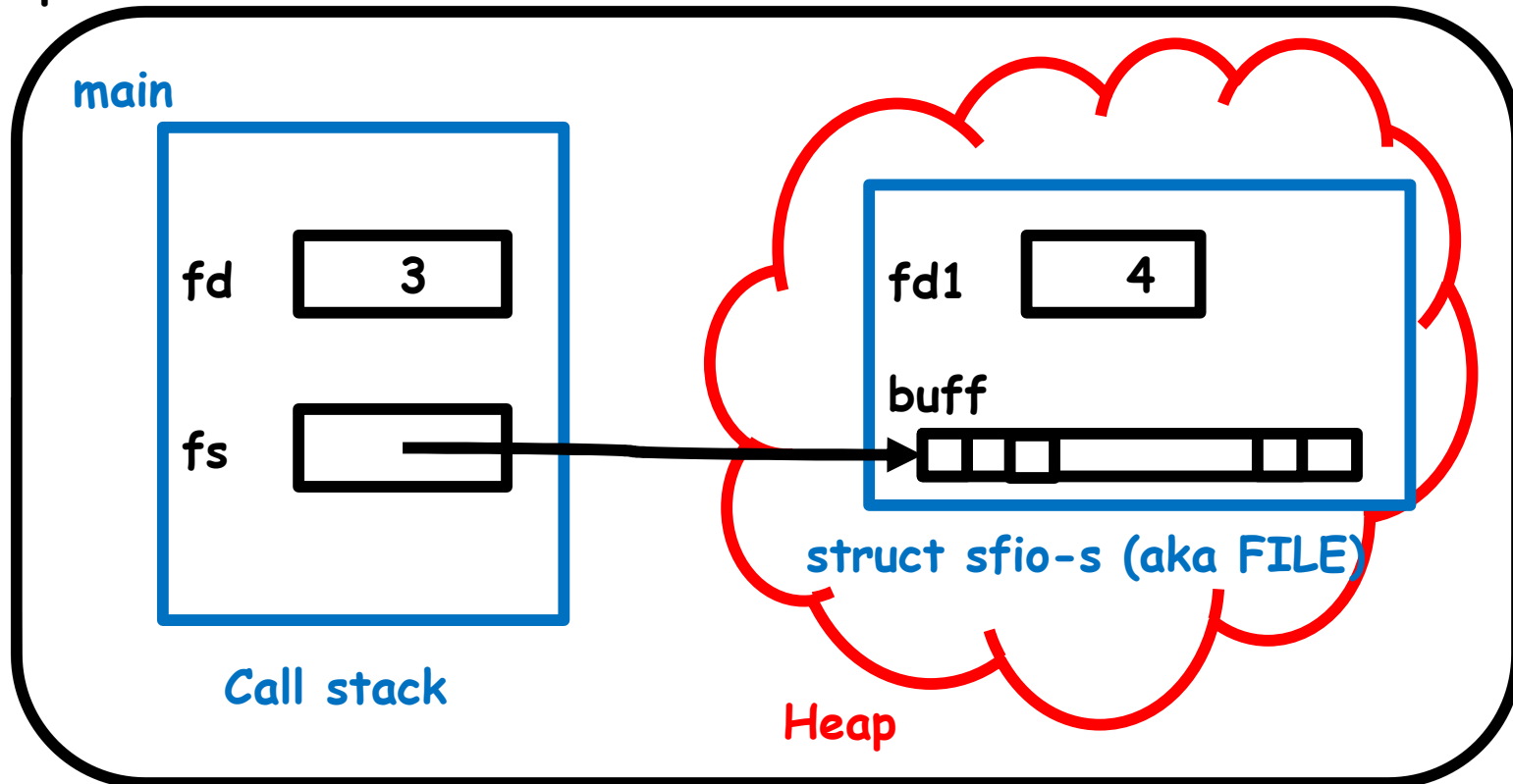
This function opens the file **path** with the behavior given by **mode** and associates a new stream with it.

read() syscall vs fread() API - User space

```
Int main(){  
    int fd = open("foo.txt", "O_RDONLY");        // for I/O syscall  
    FILE *fs = fopen("bar.txt", "w");            // for C lib I/O  
}
```

User space

Process 1101



Kernel Maintains State

```
char buffer1[100];  
char buffer2[100];  
int fd = open("foo.txt", O_RDONLY);  
read(fd, buffer1, 100);  
read(fd, buffer2, 100);
```

The kernel remembers that the fd corresponds to foo.txt



The kernel picks up where it left off in the file



State Maintained by the Kernel

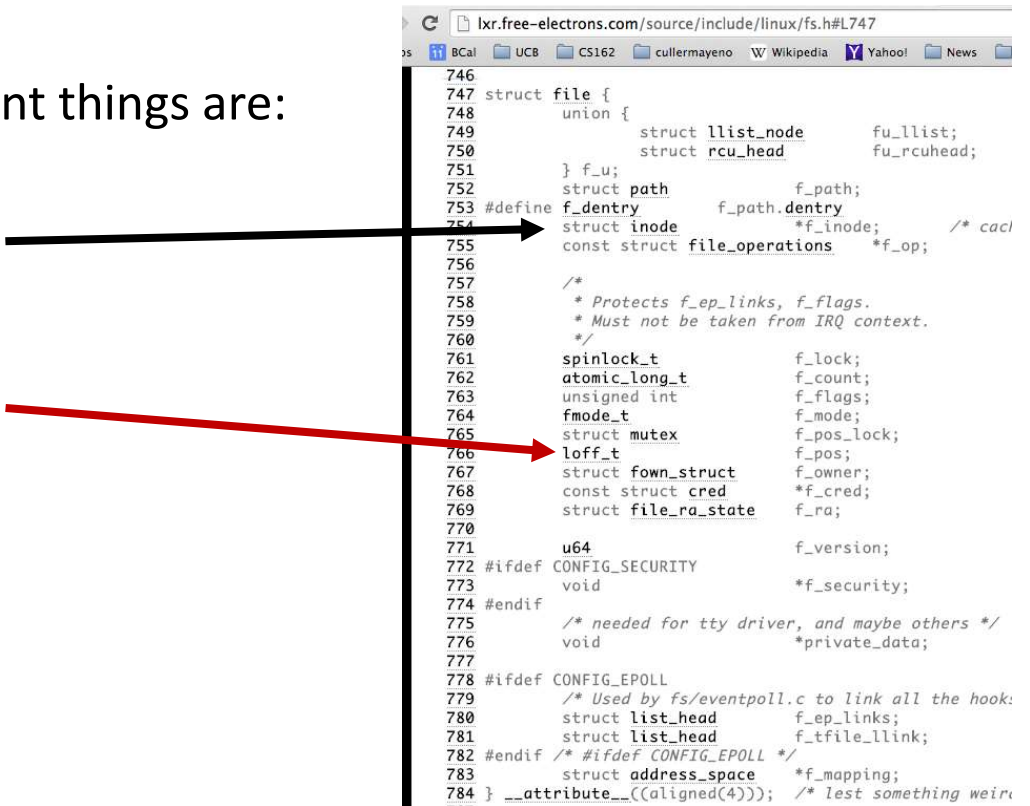
On a successful call to `open()`:

- A *file descriptor (int)* is returned to the user
- An *open file description* is created in the kernel
- For each process, the kernel maintains a mapping from *file descriptor* to *open file description*
- On future system calls (e.g., `read()`), the kernel looks up the open file description corresponding to the provided file descriptor and uses it to service the system call

What's in an Open File Description?

For our purposes, the two most important things are:

- Where to find the file data on disk
- The current position within the file



```
746 struct file {
747     union {
748         struct llist_node    fu_llist;
749         struct rcu_head      fu_rcuhead;
750     } f_u;
751     struct path              f_path;
752     struct f_dentry          f_path.dentry;
753 #define f_dentry             f_path.dentry
754     struct inode              *f_inode; /* caci
755     const struct file_operations *f_op;
756
757     /*
758      * Protects f_ep_links, f_flags.
759      * Must not be taken from IRQ context.
760      */
761     spinlock_t                f_lock;
762     atomic_long_t             f_count;
763     unsigned int              f_flags;
764     fmode_t                   f_mode;
765     struct mutex              f_pos_lock;
766     loff_t                    f_pos;
767     struct fown_struct        f_owner;
768     const struct cred          *f_cred;
769     struct file_ra_state      f_ra;
770
771     u64                       f_version;
772 #ifdef CONFIG_SECURITY
773     void                      *f_security;
774 #endif
775     /* needed for tty driver, and maybe others */
776     void                      *private_data;
777
778 #ifdef CONFIG_EPOLL
779     /* Used by fs/eventpoll.c to link all the hook:
780     struct list_head          f_ep_links;
781     struct list_head          f_tfile_llink;
782 #endif /* #ifdef CONFIG_EPOLL */
783     struct address_space      *f_mapping;
784 } __attribute__((aligned(4))); /* test something weird
```

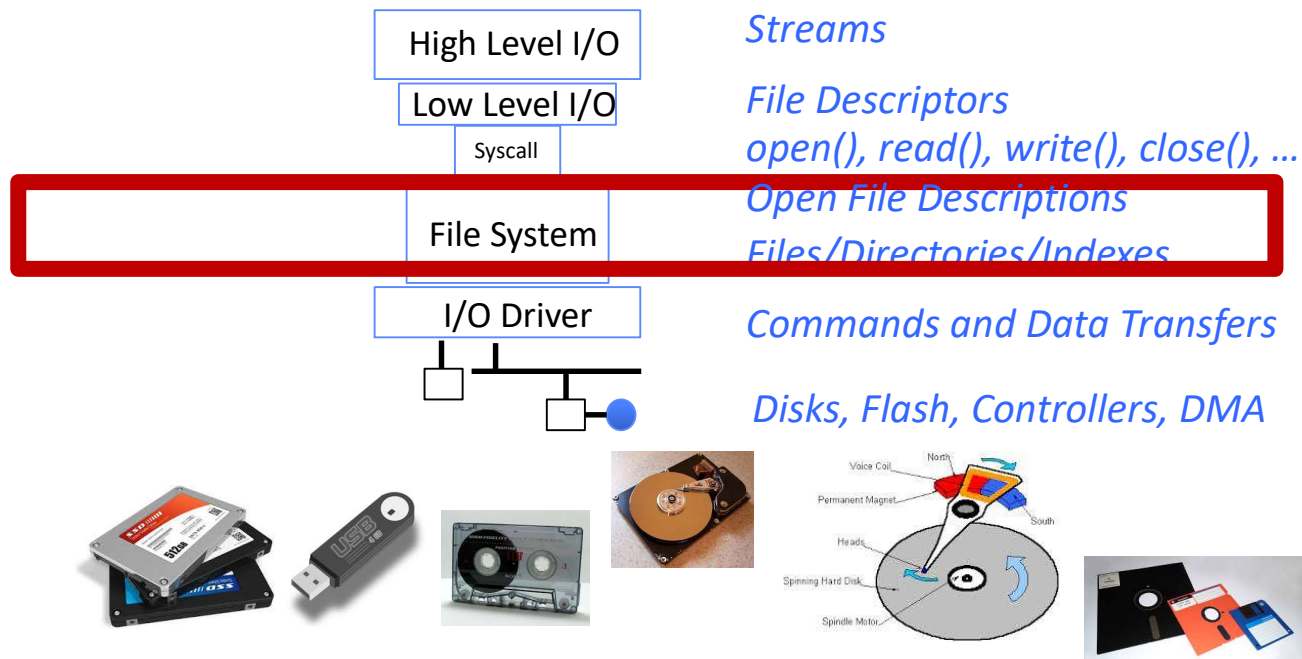
The image shows a snippet of the Linux kernel source code for the `struct file` definition in `fs.h`. Two arrows point from the text in the previous block to specific fields in the struct: a black arrow points to `f_dentry` (line 753) and a red arrow points to `f_pos` (line 766).

Today, we will study

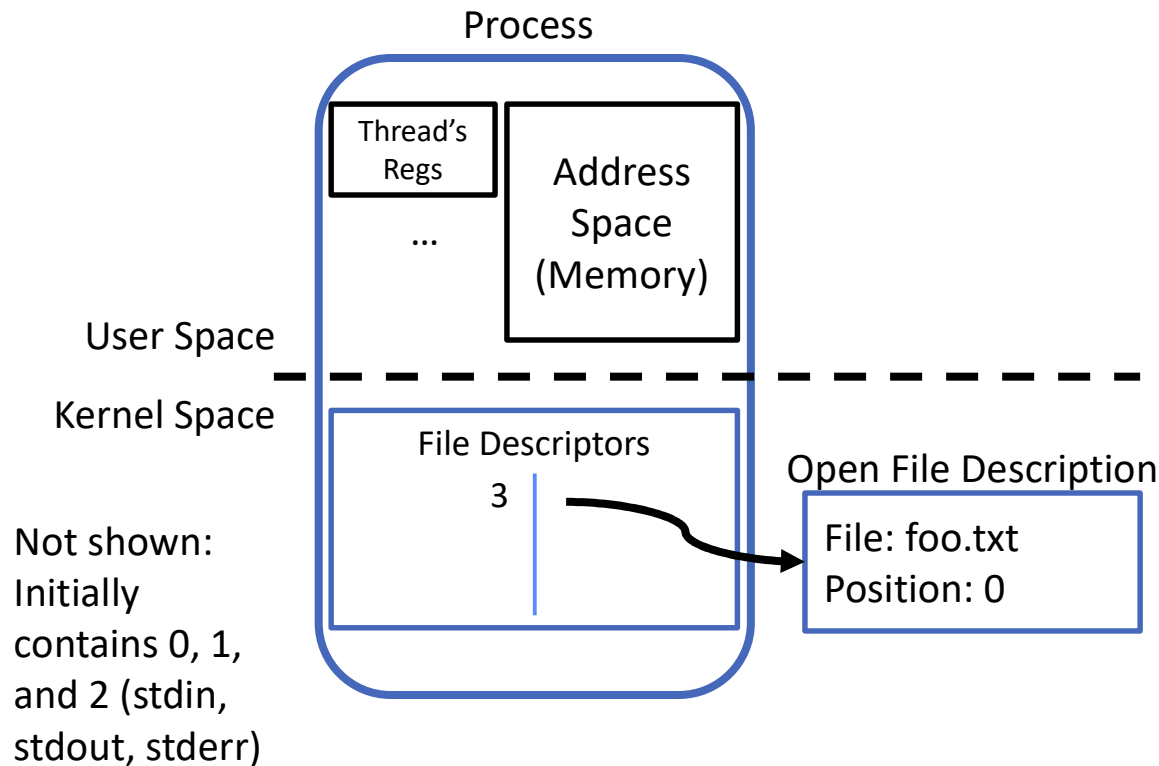
- Use of file descriptor within a process or across multiple processes

Today, we will study..

Application / Service



Abstract Representation of a Process

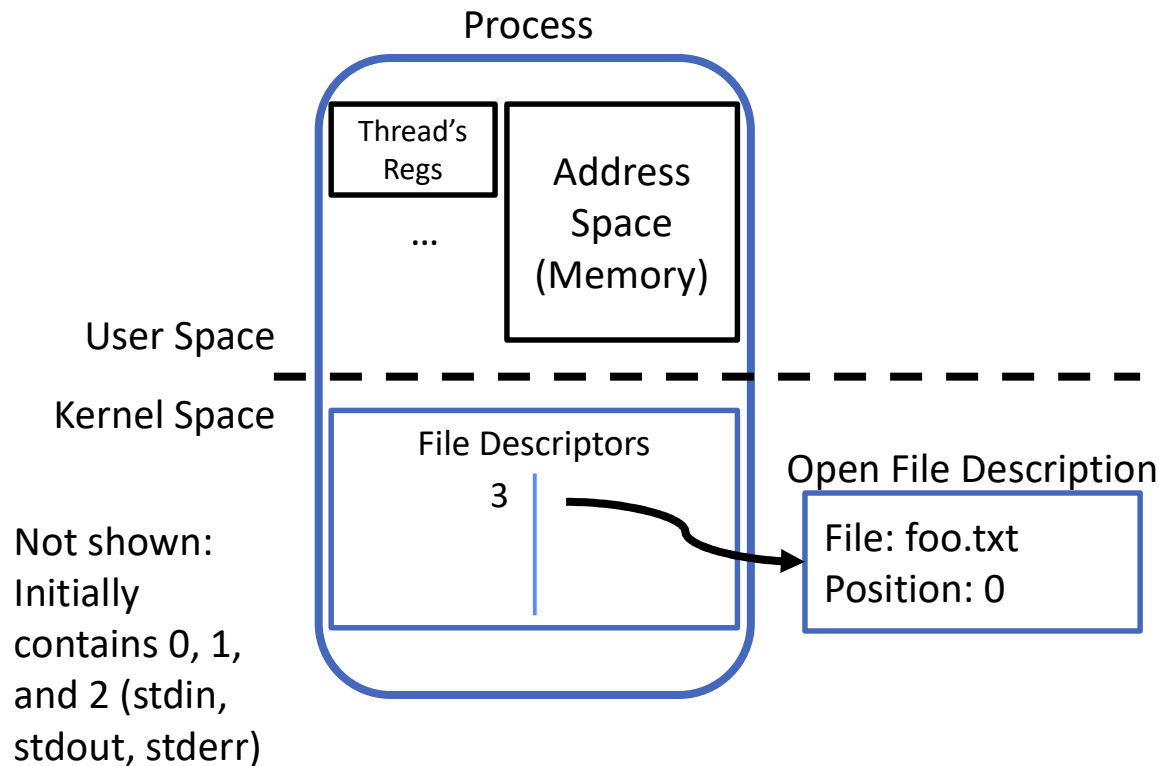


Suppose that we execute

`open("foo.txt")`

and that the result is 3

Abstract Representation of a Process



Suppose that we execute

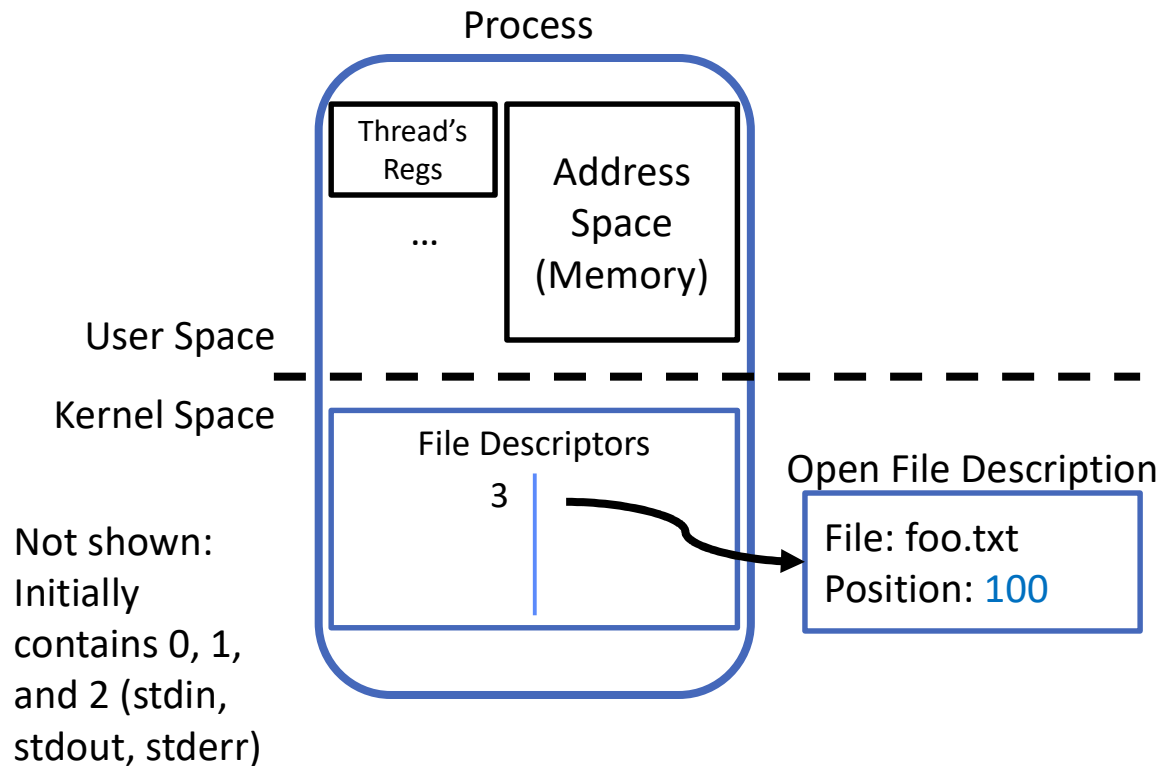
```
open("foo.txt")
```

and that the result is 3

Next, suppose that we execute

```
read(3, buf, 100)
```

Abstract Representation of a Process



Suppose that we execute

```
open("foo.txt")
```

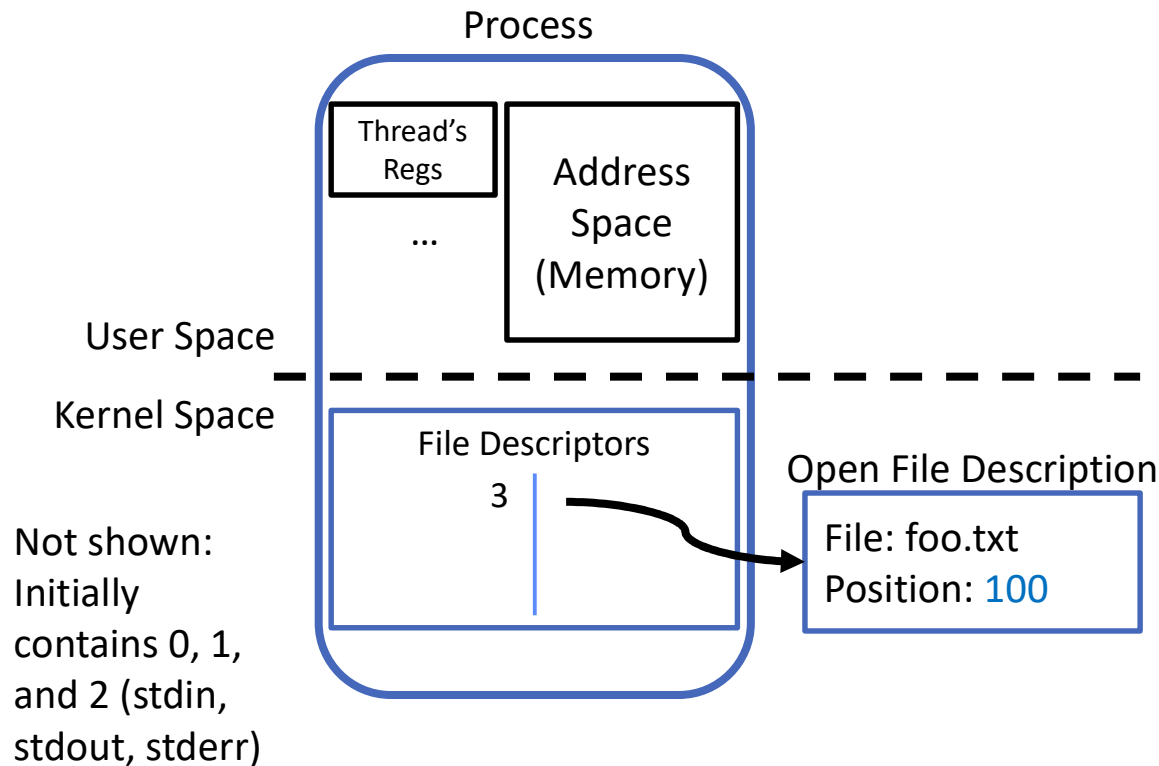
and that the result is 3

Next, suppose that we execute

```
read(3, buf, 100)
```

The file position is 100

Abstract Representation of a Process



Suppose that we execute

```
open("foo.txt")
```

and that the result is 3

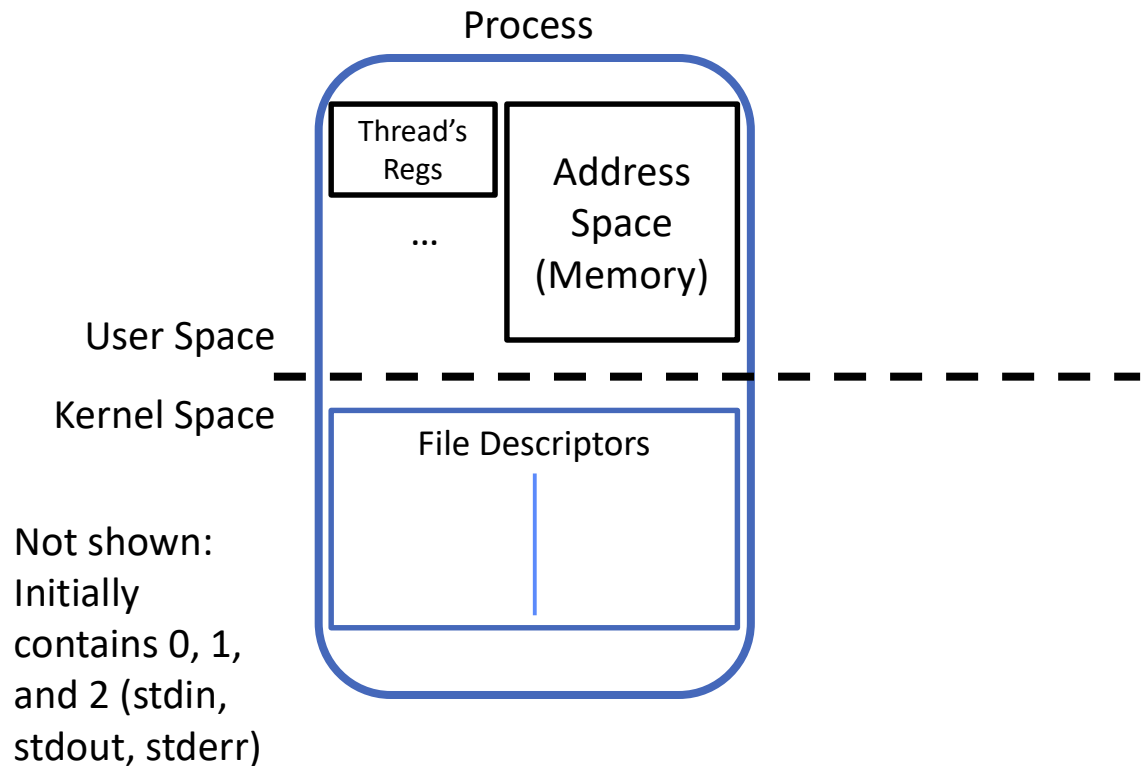
Next, suppose that we execute

```
read(3, buf, 100)
```

Finally, suppose that we execute

```
close(3)
```


Abstract Representation of a Process



Suppose that we execute

```
open("foo.txt")
```

and that the result is 3

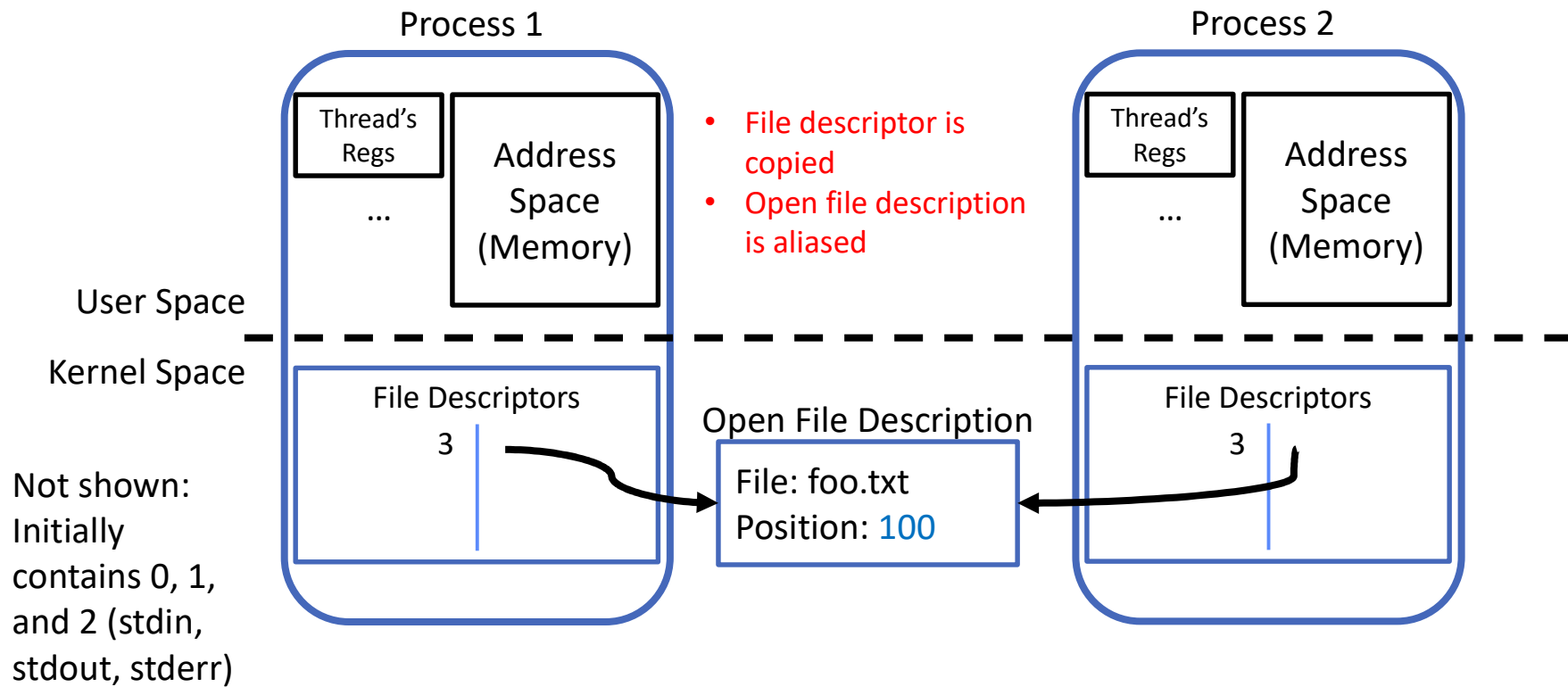
Next, suppose that we execute

```
read(3, buf, 100)
```

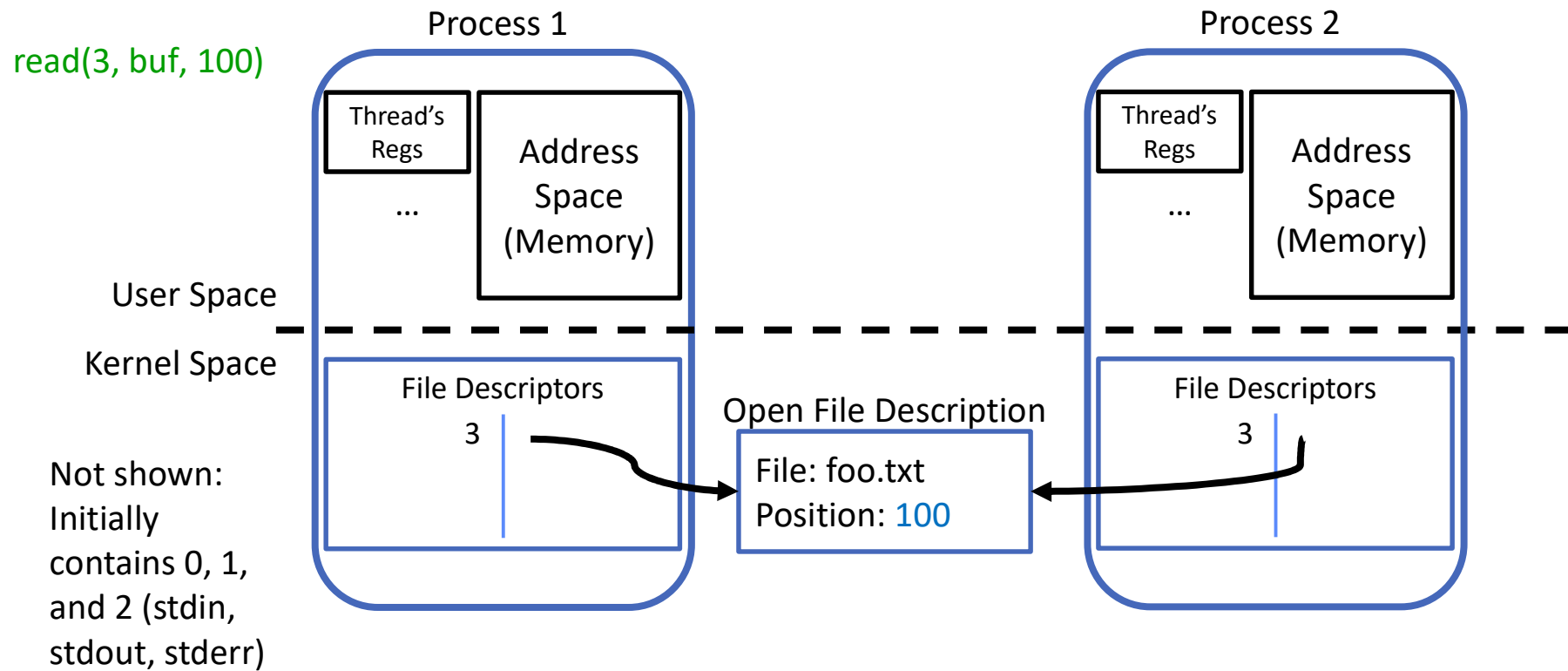
Finally, suppose that we execute

```
close(3)
```

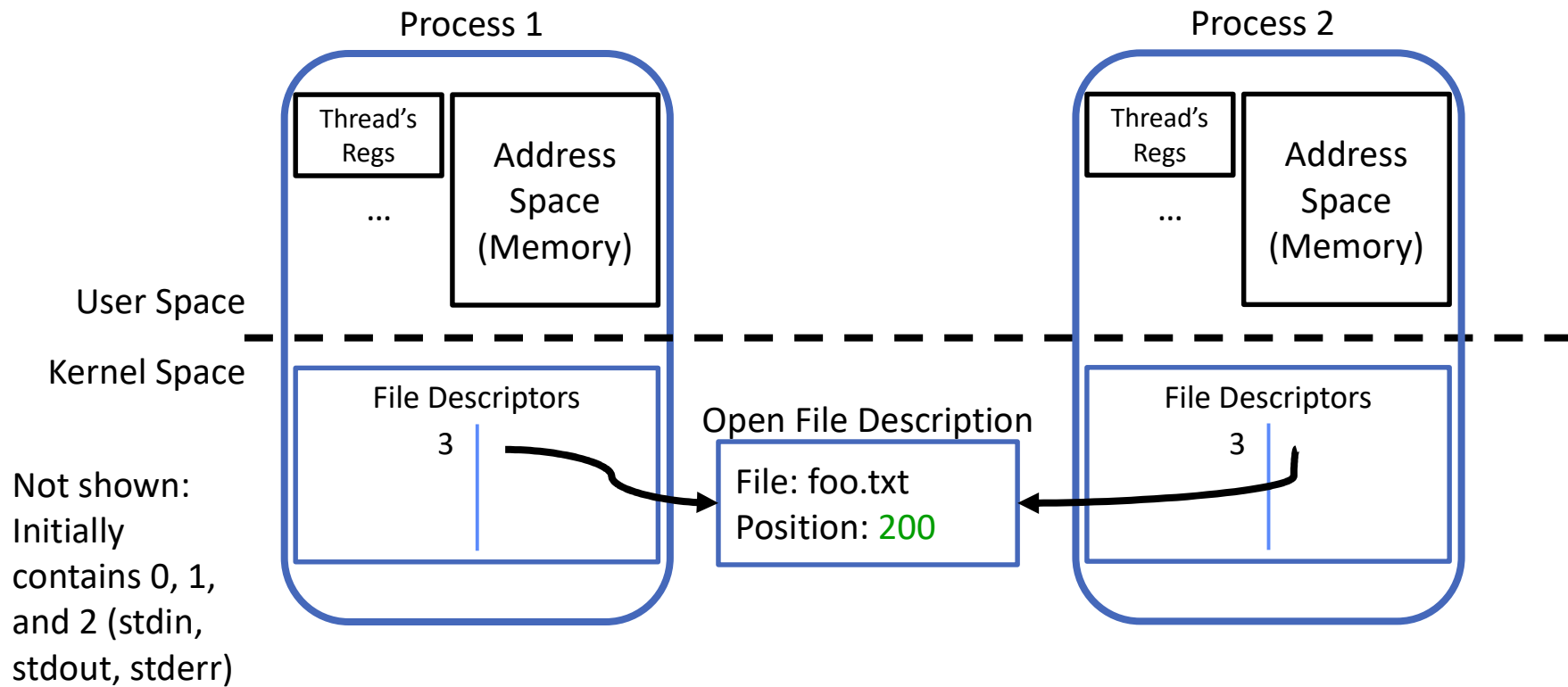
Now, let's `fork()`!



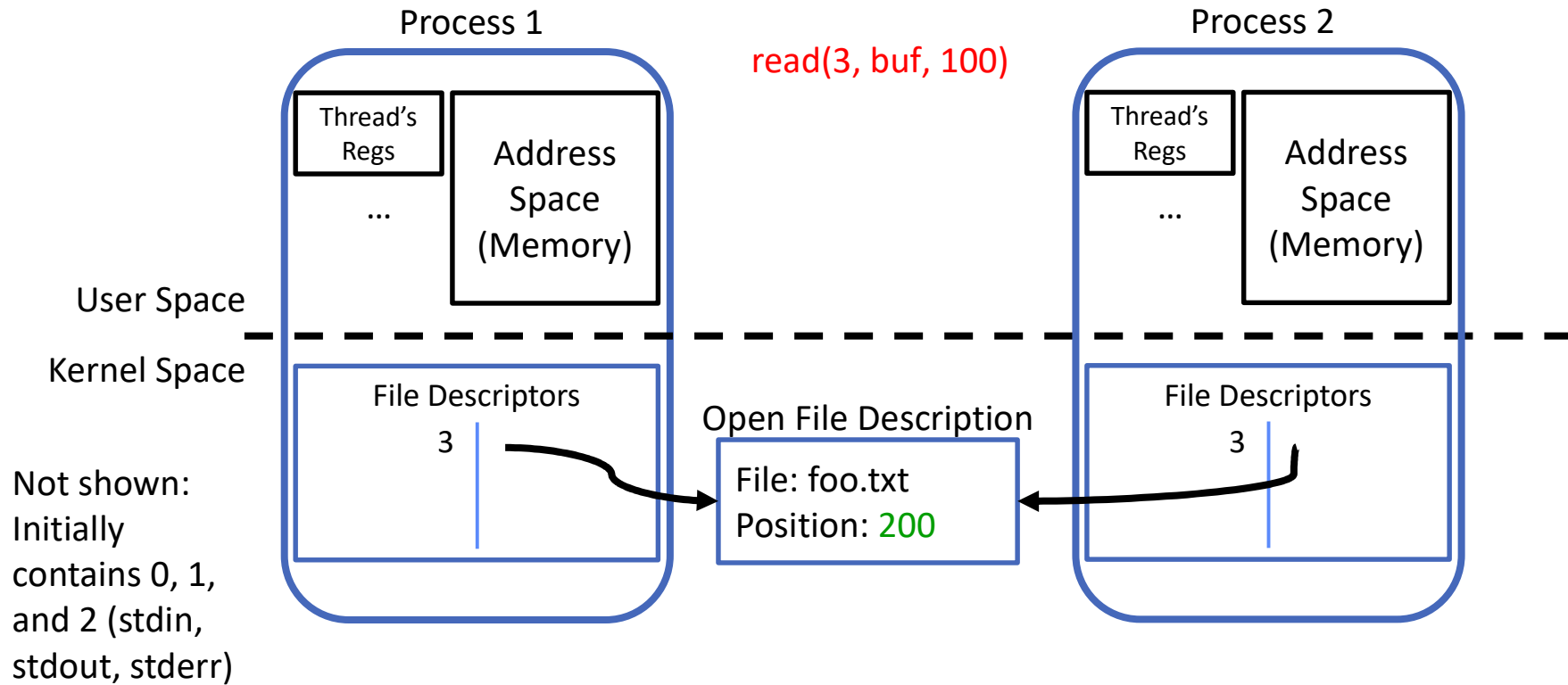
Open File Description is *Aliased*



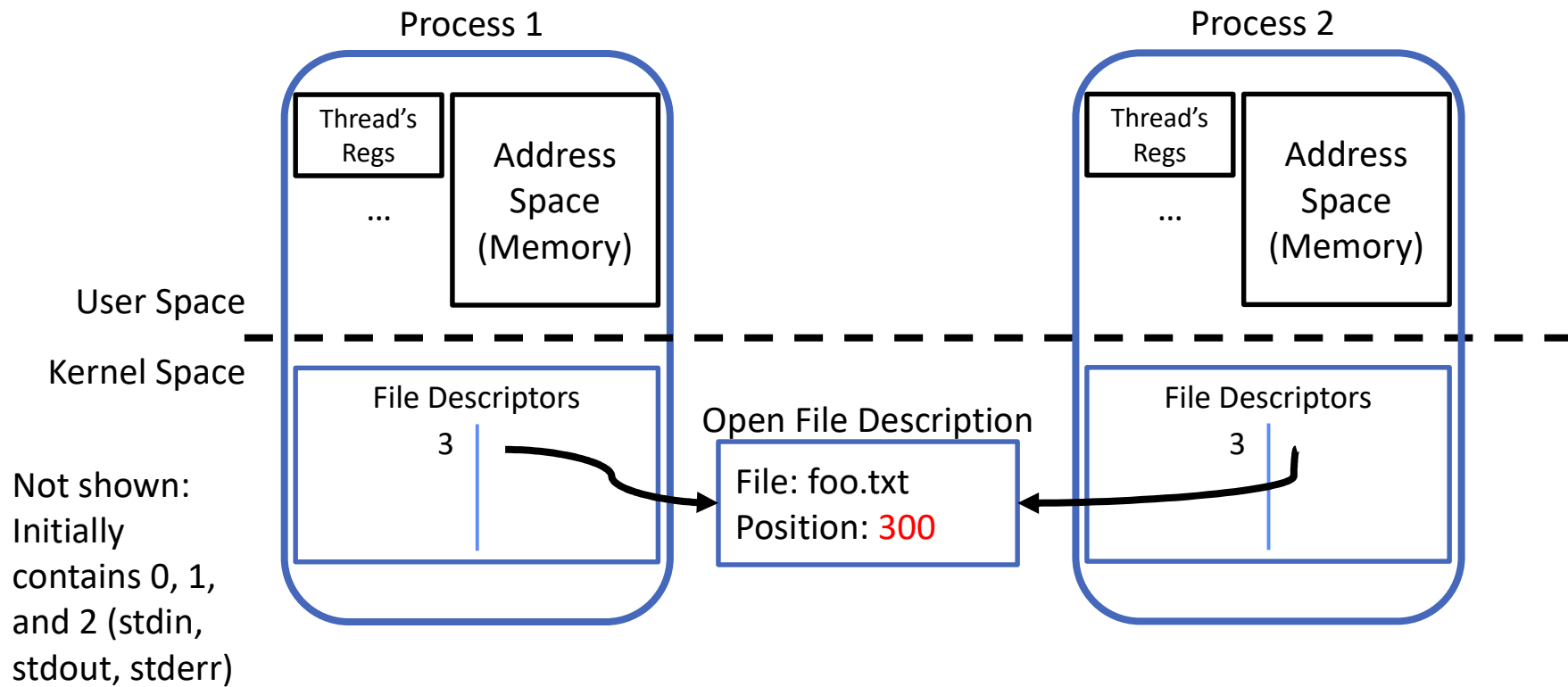
Open File Description is *Aliased*



Open File Description is *Aliased*

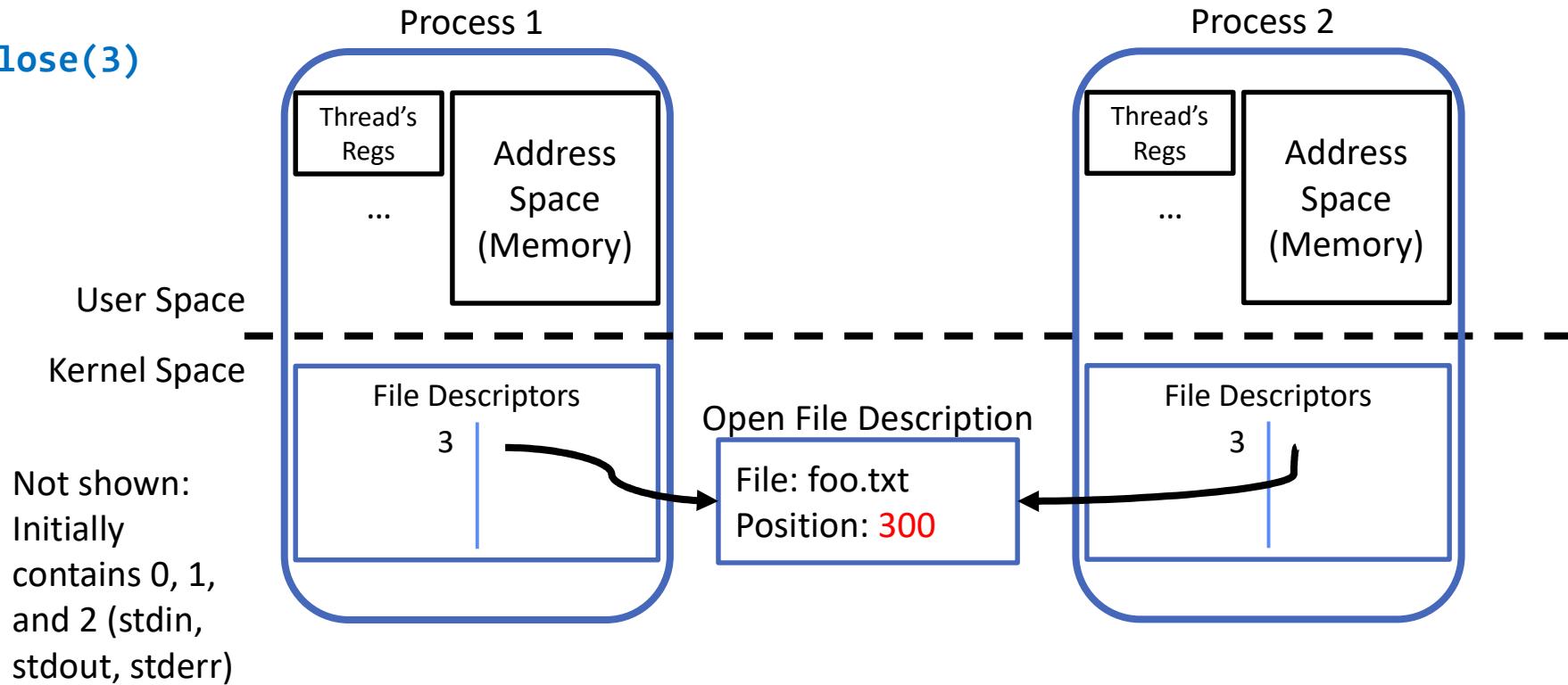


Open File Description is *Aliased*

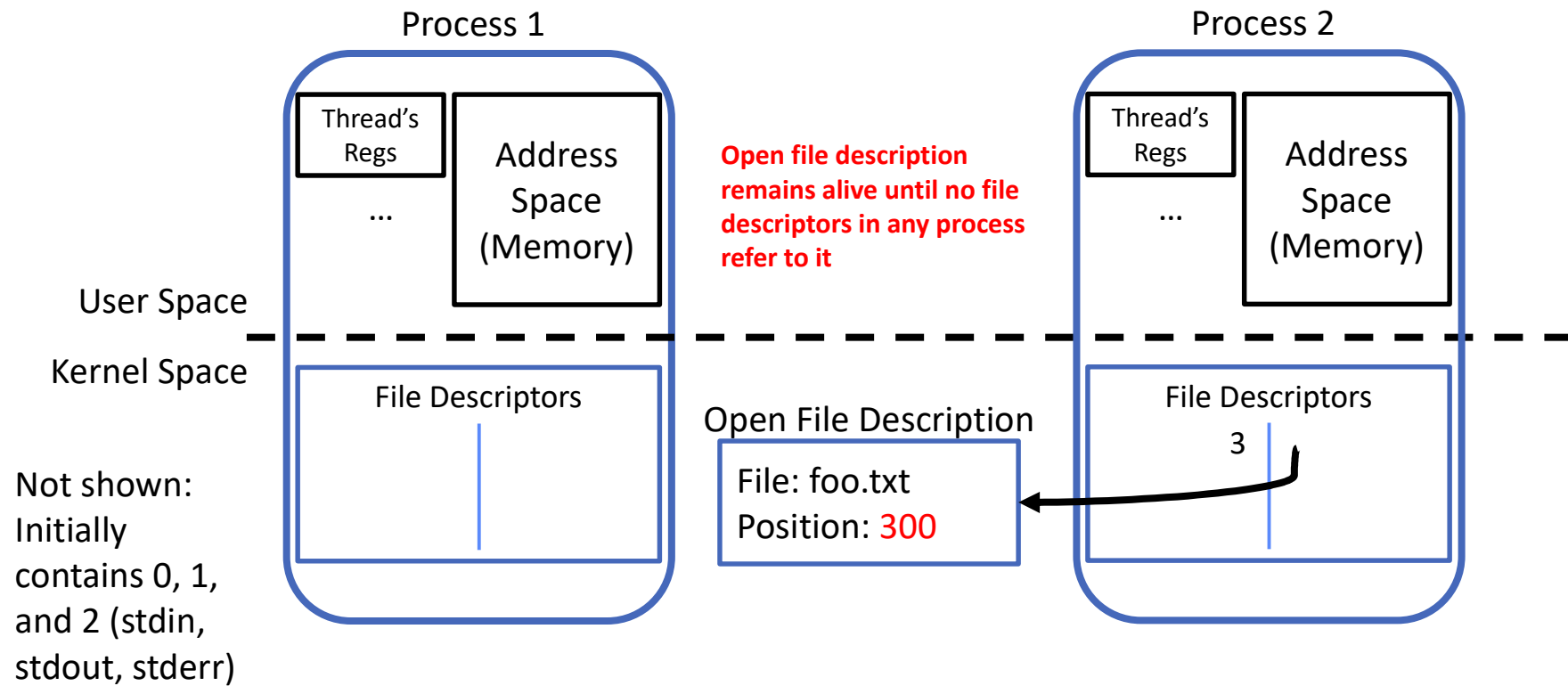


File Descriptor is *Copied*

`close(3)`



File Descriptor is Copied



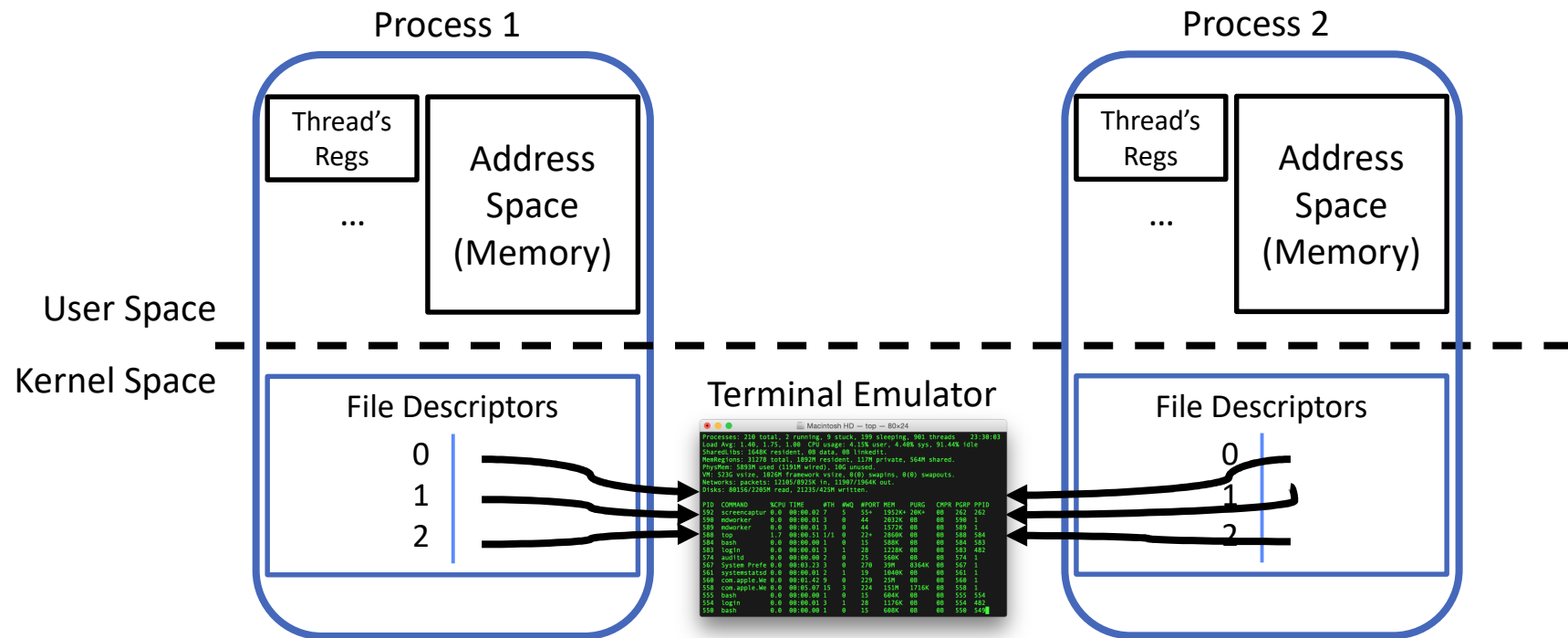
Why is Aliasing the Open File Description a Good Idea?

- It allows for *shared resources* between processes

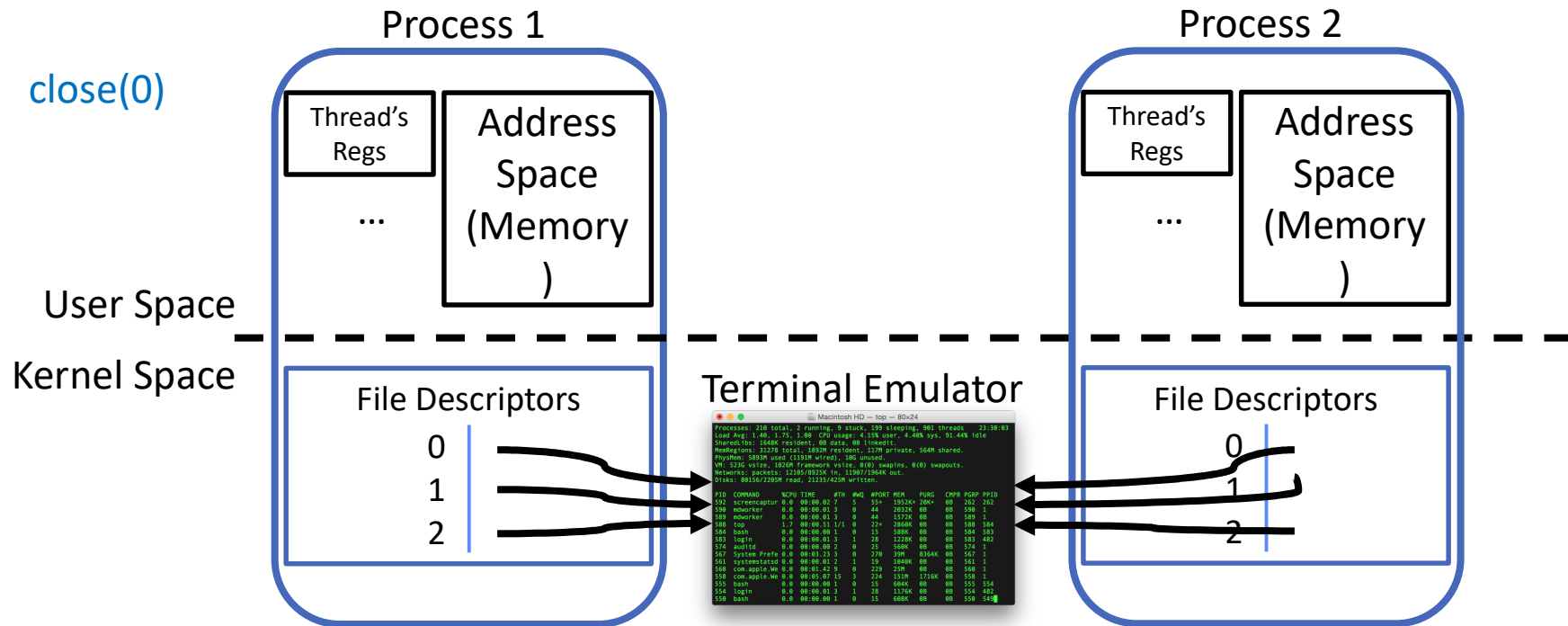
In Linux, Everything is a File

- Identical interface for:
 - Files on disk
 - Devices (terminals, printers, etc.)
 - Regular files on disk
 - Networking (sockets)
 - Local inter-process communication (pipes, sockets)
- Based on the system calls `open()`, `read()`, `write()`, and `close()`

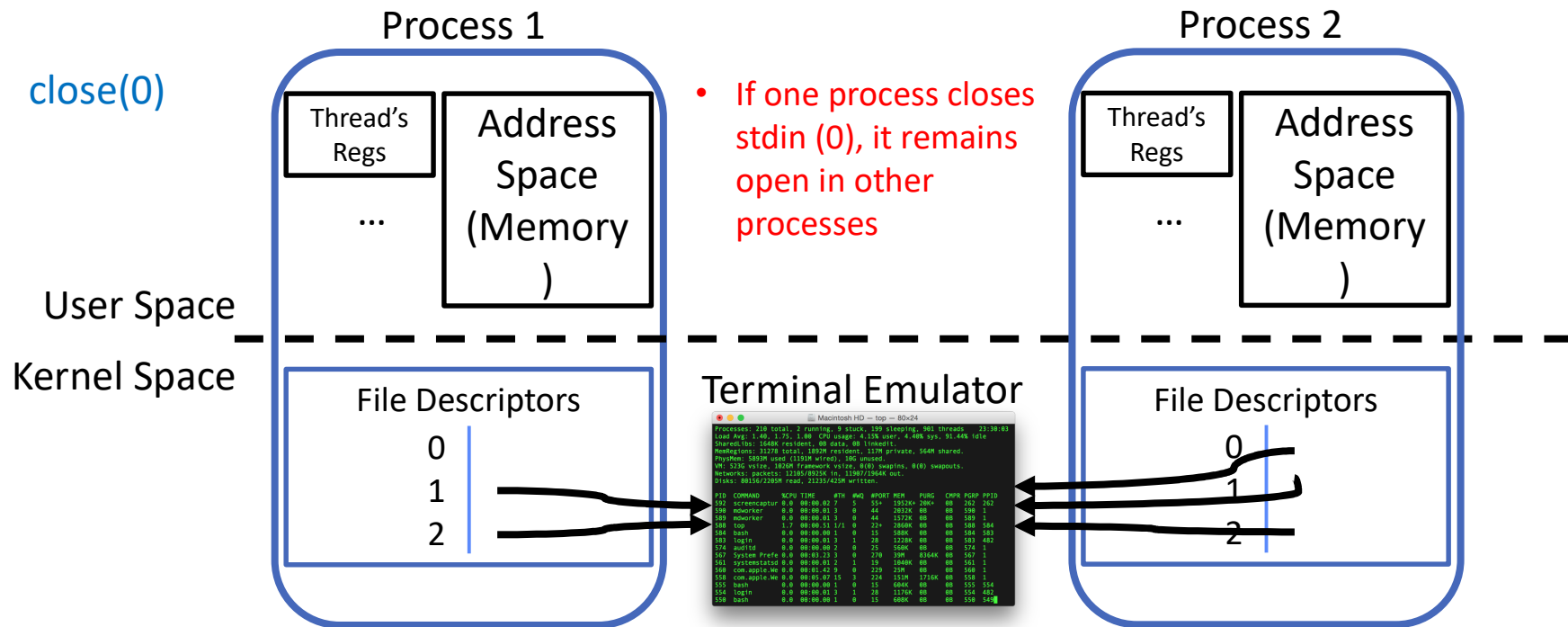
Example: Shared Terminal Emulator



Example: Shared Terminal Emulator



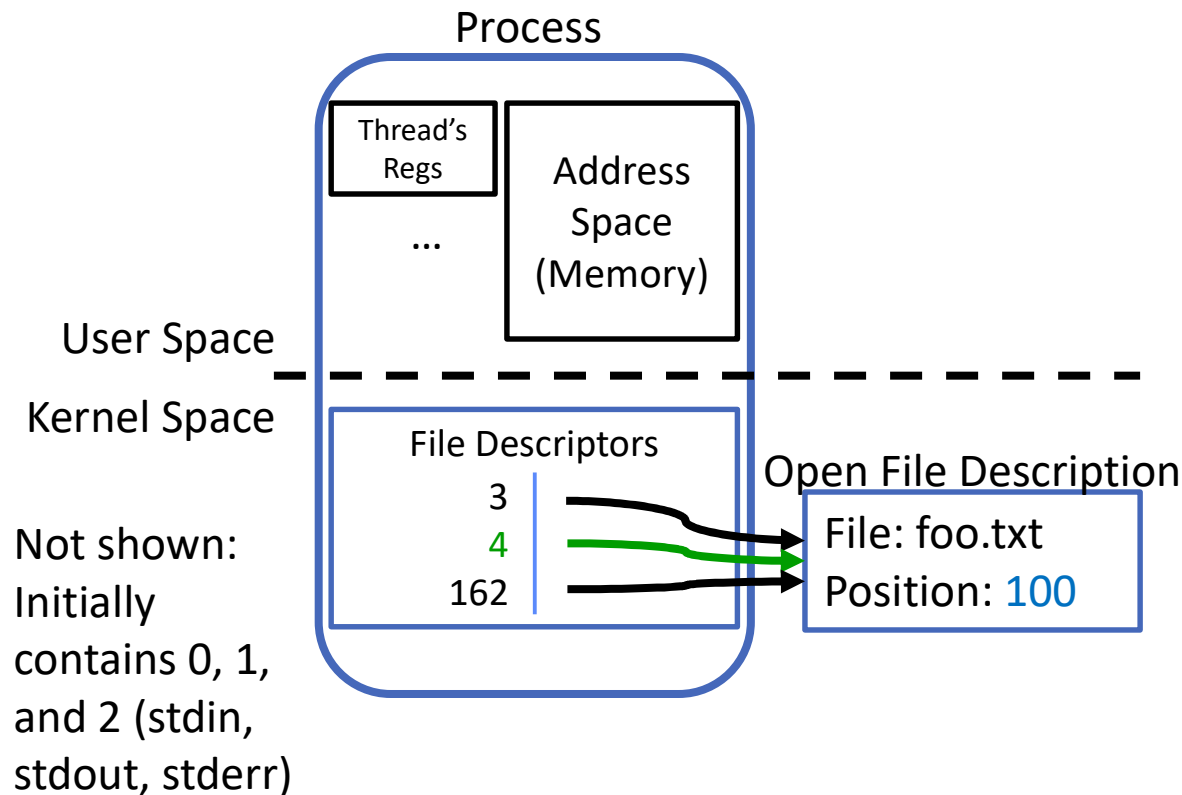
Example: Shared Terminal Emulator



Other Syscalls: dup and dup2

- They allow you to duplicate the file descriptor
- But the open file description remains aliased

Other Syscalls: dup and dup2



Suppose that we execute
`open("foo.txt")`
and that the result is 3

Next, suppose that we execute
`read(3, buf, 100)`
and that the position is 100

Next, suppose that we execute
`dup(3)`
And that the result is 4

Finally, suppose that we execute
`dup2(3, 162)`

Pitfalls of OS abstraction

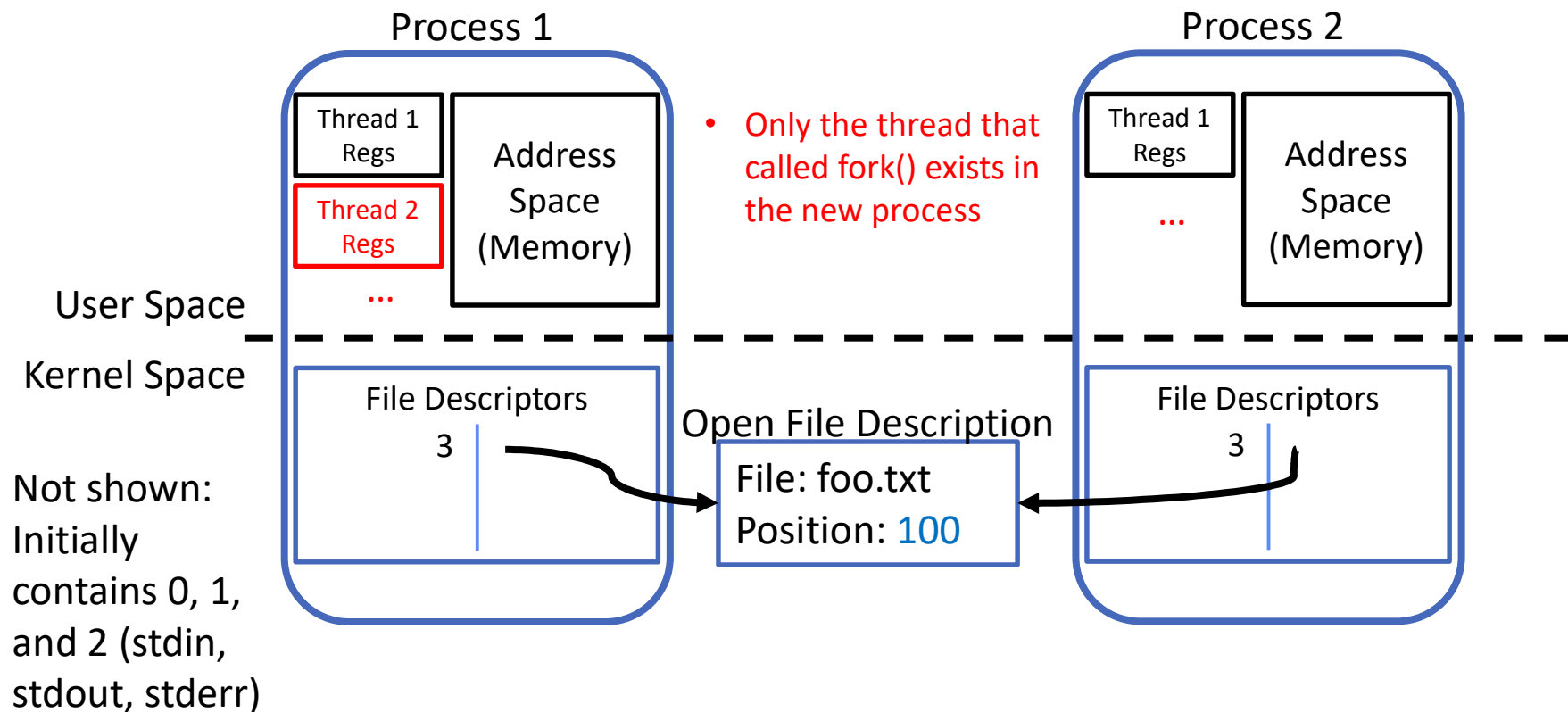
Don't fork() in a process that already has multiple threads

- Unless you plan to call `exec()` in the child process

fork() in Multithreaded Processes (in POSIX)

- The child process always has just a single thread
 - The thread in which fork() was called
- The other threads of the child process just vanish without any notice

fork() in a Multithreaded Processes



pfile2.c

```
2 // Creating a sequential file
3 #include <stdio.h>
4
5 int main(void){
6     FILE *cfPtr = NULL; // cfPtr = clients.txt file pointer
7
8     // fopen opens the file. Exit the program if unable to create the file
9     if ((cfPtr = fopen("clients.txt", "w")) == NULL) {
10         puts("File could not be opened");
11     }
12     else {
13         puts("Enter the account, name, and balance.");
14         puts("Enter EOF to end input.");
15         printf("%s", "? ");
16
17         int account = 0; // account number
18         char name[30] = ""; // account name
19         double balance = 0.0; // account balance
20
21         scanf("%d%29s%lf", &account, name, &balance);
22
23         // write account, name and balance into file with fprintf
24         while (!feof(stdin)) {
25             fprintf(cfPtr, "%d %s %.2f\n", account, name, balance);
26             printf("%s", "? ");
27             scanf("%d%29s%lf", &account, name, &balance);
28         }
29
30         fclose(cfPtr); // fclose closes file
31     }
32 }
```

```
(base) Ravis-MacBook-Pro-2:cp ravimittal$ ./pf
Enter the account, name, and balance.
Enter EOF to end input.
? 10 ravi 10.0
? 200 ram 50.50
? 300 sam 20.0
```

\$ cat clients.txt

```
10 ravi 10.00
200 ram 50.50
300 sam 20.00
```

EOF character

Linux/MAC OS : <Ctrl> d

Windows: <Ctrl> z enter

Lecture Summary

- File descriptors can be manipulated within processes or across processes