CS 170 Week 8

Winter 2026

https://sites.cs.ucsb.edu/~rich/class/cs170/

Agenda

Lab 3: Pipe Dream

1. Overview of file descriptors and pipes
2. dup, dup2, and pipe syscalls
3. Hints to get started
4. Subtle lab 3 behaviors

What are files?

“Everything is a file” on UNIX-like systems.

What are files?

“Everything is a file” on UNIX-like systems.

- File descriptors (fd or fildes) are a generic way to represent different “objects” such as the console (stdio) and pipes.

What are files?

“Everything is a file” on UNIX-like systems.

- File descriptors (fd or fildes) are a generic way to represent different “objects” such as the console (stdio) and pipes.
  - fds are numbers that represent a object that can be read from/written to (depending on the RW permissions)

What are files?

“Everything is a file” on UNIX-like systems.

- File descriptors (fd or fildes) are a generic way to represent different “objects” such as the console (stdio) and pipes.
  - fds are numbers that represent a object that can be read from/written to (depending on the RW permissions)
- Each process has a fd table that keeps track of what objects the process can interact with.

File Descriptor Table

Process 1

FD	“Reference” to the open file object
0	stdin
1	stdout
2	stderr
3	
4	

File Descriptor Table

Process 1

FD	“Reference” to the open file object
0	stdin
1	stdout
2	stderr
3	pipe (read end)
4	pipe (write end)

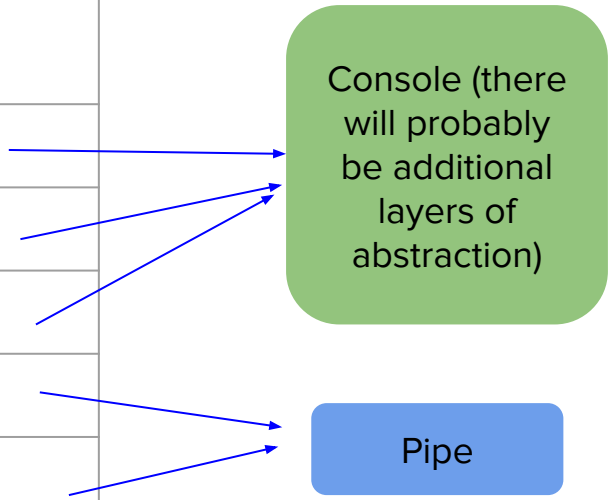
```
int pipefd[2];
int ret = pipe(pipefd);
```

pipefd[0] is read end
pipefd[1] is write end

File Descriptor Table

Process 1

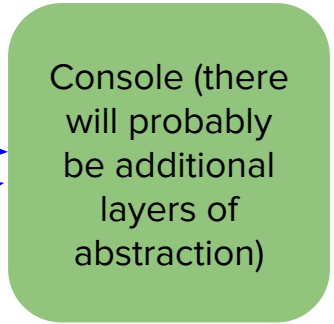
FD	“Reference” to the open file object
0	stdin
1	stdout
2	stderr
3	pipe (read end)
4	pipe (write end)



File Descriptor Table – close()

Process 1

FD	“Reference” to the open file object
0	stdin
1	stdout
2	stderr
3	~~pipe (read end)~~
4	pipe (write end)



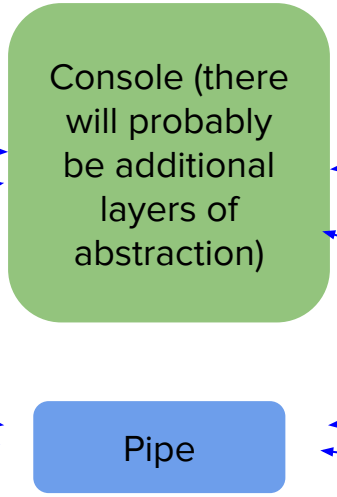
Calling close() simply closes the FD and removes it from the table.

close(3);

File Descriptor Table With Fork

Process 1

FD	“Reference” to the open file object
0	stdin
1	stdout
2	stderr
3	pipe (read end)
4	pipe (write end)



Process 2 (fork of process 1)

FD	“Reference” to the open file object
0	stdin
1	stdout
2	stderr
3	pipe (read end)
4	pipe (write end)

Pipe has now 2 readers and 2 writers.

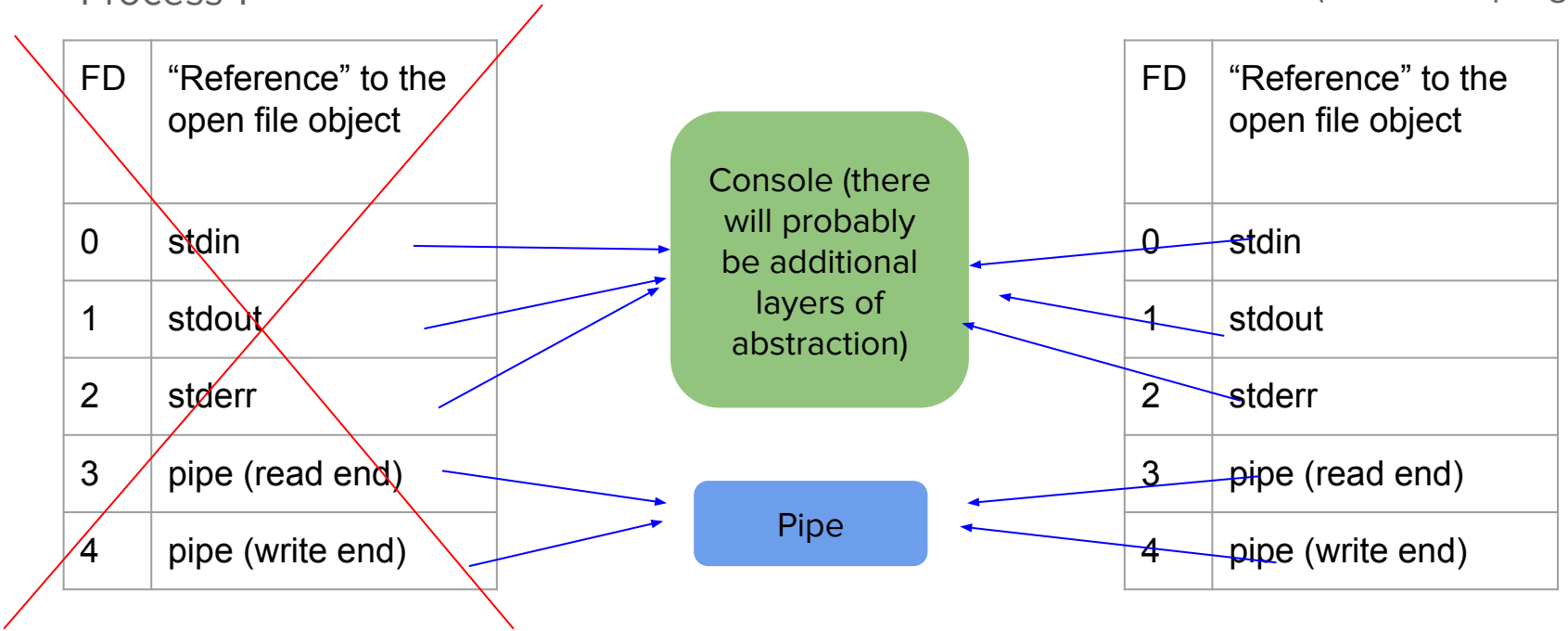
File Descriptor Table With Exec

~~Process 1~~

FD	“Reference” to the open file object
0	stdin
1	stdout
2	stderr
3	pipe (read end)
4	pipe (write end)

Process 2 (Exec new program)

FD	“Reference” to the open file object
0	stdin
1	stdout
2	stderr
3	pipe (read end)
4	pipe (write end)



Pipe has only 1 reader and 1 writer. (No copying!)

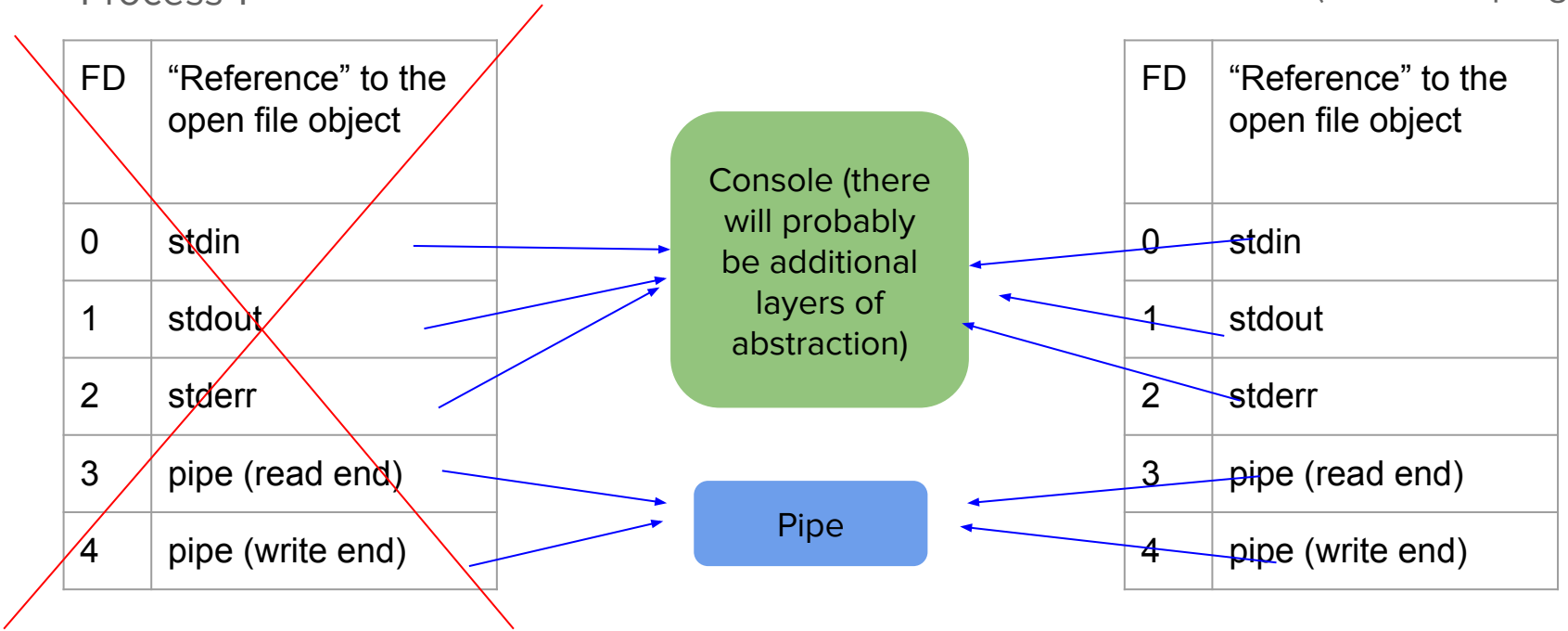
File Descriptor Table With Exec

~~Process 1~~

FD	“Reference” to the open file object
0	stdin
1	stdout
2	stderr
3	pipe (read end)
4	pipe (write end)

Process 2 (Exec new program)

FD	“Reference” to the open file object
0	stdin
1	stdout
2	stderr
3	pipe (read end)
4	pipe (write end)

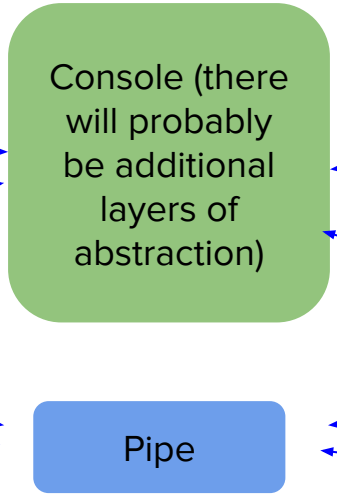


Important: the FDs do not change after exec! (they are preserved)

File Descriptor Table With Exec

Process 1

FD	“Reference” to the open file object
0	stdin
1	stdout
2	stderr
3	pipe (read end)
4	pipe (write end)



Process 2 (Exec new program)

FD	“Reference” to the open file object
0	stdin
1	stdout
2	stderr
3	pipe (read end)
4	pipe (write end)

Pipes implements one form of inter-process communication (IPC)

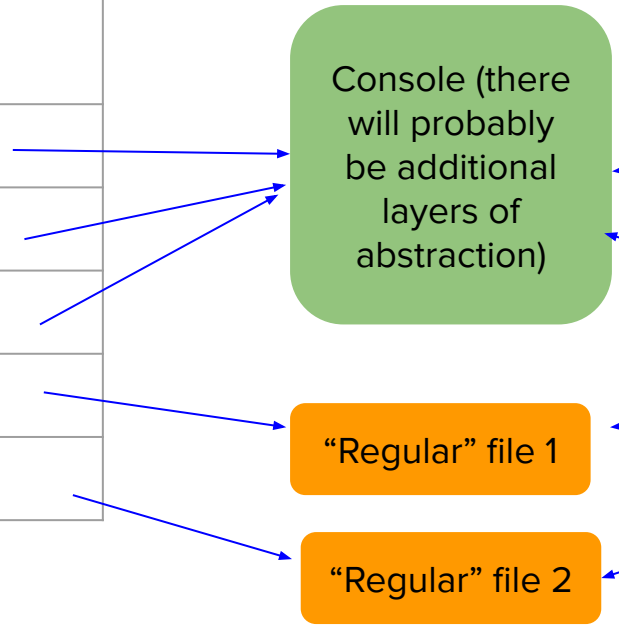
File Descriptor Table With “Regular” Files (not part of lab 3)

Process 1

FD	“Reference” to the open file object
0	stdin
1	stdout
2	stderr
3	my_file.txt
4	my_file2.txt

Process 2

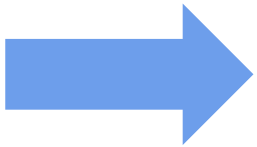
FD	“Reference” to the open file object
0	stdin
1	stdout
2	stderr
3	my_file.txt
4	my_file2.txt



Dup/Dup2 System Call

Before dup(1)

FD	“Reference” to the open file object
0	stdin
1	stdout
2	stderr
3	



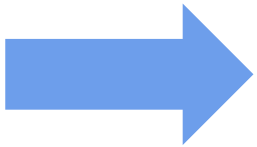
After dup(1)

FD	“Reference” to the open file object
0	stdin
1	stdout
2	stderr
3	stdout

Dup/Dup2 System Call

Before dup(1)

FD	“Reference” to the open file object
0	stdin
1	stdout
2	stderr
3	



Duplicates a file descriptor

After dup(1)

FD	“Reference” to the open file object
0	stdin
1	stdout
2	stderr
3	stdout

dup() places new fd in lowest index.

Dup/Dup2 System Call

Before dup(1)

FD	“Reference” to the open file object
0	stdin
1	stdout
2	stderr
3	

After dup(1)

FD	“Reference” to the open file object
0	stdin
1	stdout
2	stderr
3	stdout

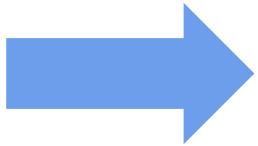
Why is this useful? We’ll see after learning about pipes.

Pipe System Call

Create a pipe with a read end and a write end.

FD	“Reference” to the open file object
0	stdin
1	stdout
2	stderr
3	
4	

pipe syscall



FD	“Reference” to the open file object
0	stdin
1	stdout
2	stderr
3	pipe (read end)
4	pipe (write end)

How a pipe works (one process)

FD	“Reference” to the open file object
0	stdin
1	stdout
2	stderr
3	pipe (read end)
4	pipe (write end)

Unidirectional data channel! Writing to the write end will make the data available to the read end.

How a pipe works (inter-process communication)

Process 1

FD	“Reference” to the open file object
0	stdin
1	stdout
2	stderr
3	pipe (read end)
4	pipe (write end)

Process 2

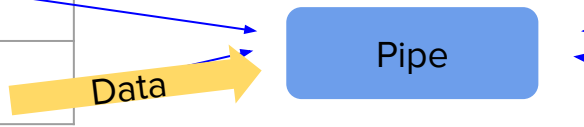
FD	“Reference” to the open file object
0	stdin
1	stdout
2	stderr
3	pipe (read end)
4	pipe (write end)

How a pipe works (inter-process communication)

Process 1

FD	“Reference” to the open file object
0	stdin
1	stdout
2	stderr
3	pipe (read end)
4	pipe (write end)

Step 1: Process 1 writes some data to the pipe.



Process 2

FD	“Reference” to the open file object
0	stdin
1	stdout
2	stderr
3	pipe (read end)
4	pipe (write end)

How a pipe works (inter-process communication)

Process 1

FD	“Reference” to the open file object
0	stdin
1	stdout
2	stderr
3	pipe (read end)
4	pipe (write end)

Step 1: Process 1 writes some data to the pipe.

Step 2: Process 2 reads from the pipe.

Process 2

FD	“Reference” to the open file object
0	stdin
1	stdout
2	stderr
3	pipe (read end)
4	pipe (write end)

Why is dup useful?

Suppose I want to pass some information from one program to another.

Example: run the command “hw | cat” in ksh

The “|” operator, or pipe operator (related but not the pipe syscall) is a *shell* operator that configures the **stdout** of the first program to the **stdin** of the second program.

This allows message passing between processes.

Question: What system calls are used by the shell to configure this io redirection? 🤔

Some suggestions on how to get started

There is no cook book for lab 3! Here are some general suggestions, but you can implement it however you like.

1. Make sure you understand how everything works from lab 1 and 2.
2. Understand and implement file descriptors
    a. How to implement/refactor the console (aka read/write for stdin, stdout, stderr)?
    b. How to implement pipes?
    c. How to differentiate between console and pipes. Do you want to design a “general” solution or a “hacky” implementation (that only support a console and pipes)?

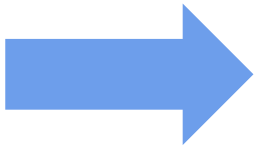
2.5. Refactor how IO is handled with fd changes (read, write, etc syscalls). Make sure everything still works before adding new features.

3. Add support for dup/dup2 and pipe syscalls
4. Implement close syscall

Dup/Dup2 System Call

Before dup(1)

FD	“Reference” to the open file object
0	stdin
1	stdout
2	stderr
3	



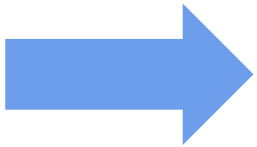
After dup(1)

FD	“Reference” to the open file object
0	stdin
1	stdout
2	stderr
3	stdout

Dup/Dup2 System Call

Before dup2(4, 1)

FD	“Reference” to the open file object
0	stdin
1	stdout
2	stderr
3	pipe_read_end
4	pipe_write_end



After dup2(4, 1)

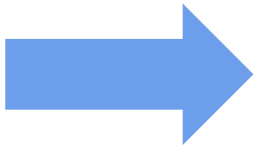
FD	“Reference” to the open file object
0	stdin
1	~~Stdout~~ pipe_write_end
2	stderr
3	pipe_read_end
4	pipe_write_end

Dup/Dup2 System Call

dup2(4, 1) closes FD 1 and duplicates FD 4 into 1.

Before dup2(4, 1)

FD	“Reference” to the open file object
0	stdin
1	stdout
2	stderr
3	pipe_read_end
4	pipe_write_end



Duplicates a file descriptor

After dup2(4, 1)

FD	“Reference” to the open file object
0	stdin
1	~~Stdout~~ pipe_write_end
2	stderr
3	pipe_read_end
4	pipe_write_end

Subtle Lab 3 Behaviors (for your convenience)

Pipe block and unblock behavior:

1. Start reading from a pipe when there is no data ➔ blocks
2. Reading from a pipe when some data is there, but is less than the attempted read size ➔ unblocks
3. When the read/write end closes, the other end should unblock and be handled properly (read returns 0, write returns error)
4. Recommend pipe buffer size to be 8192 bytes for **unblock.c** to behave as expected

Questions