Recitation 1

Welcome back, everybody!

Today's Topics

- Signals
- File Descriptors
- Redirection and Pipes

Signals

Signals

- Software interrupt - a notification sent to a process
- An example of inter-process communication
  - What is another example of inter-process communication?
- From the OS: some sort of exception has occurred
- From another process: used to coordinate activity or as notification
- Interrupts - i.e. Ctrl-C, Ctrl-Z pressed
- Examples: SIGINT (interrupt), SIGTSTP (terminal stop), SIGTERM (graceful termination request), SIGKILL (force kill), SIGCHLD (child terminated/stopped), ...

Default dispositions + modifying that behavior

- Signal dispositions = what they do when the signal is received.
  - Some, i.e. SIGTSTP: stops the process
  - Some, i.e. SIGCHLD: ignored by default
  - Some, i.e. SIGSEGV: segfault → core dump
- What if we don't like the default behavior?
- Sigaction allows us to override most signals (not SIGKILL, SIGSTOP) with i.e. SIG_IGN, SIG_DFL or self defined signal handlers.

What if we don’t want a signal?

- Could ignore …
- What if we want it later?
  - critical section (pause signal for now so not malformed!)?
  - Reaping without waiting (penn-shell extra challenge)?
- Block and set signal to pending (sigprocmask)
  - How do we reset? Sigprocmask again in i.e. parent, with original mask
- Idling in pennos … sigsuspend doesn’t return until new signal outside of the suspended set returns.

Activity: Responses to Signals

In each scenario, determine whether the signal was blocked, ignored, used default handling, or used a handler function (non-default)

1. In penn-shredder, I hit Ctrl+Z. In response, penn-shredder stopped, and now I'm back in the host shell.

Activity: Responses to Signals

In each scenario, determine whether the signal was blocked, ignored, used default handling, or used a handler function (non-default)

1. In penn-shredder, I hit Ctrl+Z. In response, penn-shredder stopped, and now I'm back in the host shell. **Default Action (Ctrl-Z = SIGTSTP)**
2. In penn-shell, I hit Ctrl+Z. In response, penn-shell reprompted.

Activity: Responses to Signals

In each scenario, determine whether the signal was blocked, ignored, used default handling, or used a handler function (non-default)

1. In penn-shredder, I hit Ctrl+Z. In response, penn-shredder stopped, and now I'm back in the host shell. **Default Action (Ctrl-Z = SIGTSTP)**
2. In penn-shell, I hit Ctrl+Z. In response, penn-shell reprompted. **Signal Handler Function**
3. The parent process takes terminal control from its child and gives it to itself. The parent process was not stopped by SIGTTIN or SIGTTOU, and continues as normal.

Activity: Responses to Signals

In each scenario, determine whether the signal was blocked, ignored, used default handling, or used a handler function (non-default)

1. In penn-shredder, I hit Ctrl+Z. In response, penn-shredder stopped, and now I'm back in the host shell. **Default Action**
2. In penn-shell, I hit Ctrl+Z. In response, penn-shell reprompted. **Signal Handler Function**
3. The parent process takes terminal control from its child and gives it to itself. The parent process was not stopped by SIGTTIN or SIGTTOU, and continues as normal. **Signal Ignored**
4. A process received a signal, and its signal handler called vec_push. While in the middle of executing vec_push, it received the same signal. Once the first call to vec_push returned, the signal handler ran vec_push again.

Activity: Responses to Signals

In each scenario, determine whether the signal was blocked, ignored, used default handling, or used a handler function (non-default)

1. In penn-shredder, I hit Ctrl+Z. In response, penn-shredder stopped, and now I'm back in the host shell. **Default Action**
2. In penn-shell, I hit Ctrl+Z. In response, penn-shell reprompted. **Signal Handler Function**
3. The parent process takes terminal control from its child and gives it to itself. The parent process was not stopped by SIGTTIN or SIGTTOU, and continues as normal. **Signal Ignored**
4. A process received a signal, and its signal handler called vec_push. While in the middle of executing vec_push, it received the same signal. Once the first call to vec_push returned, the signal handler ran vec_push again. **Signal Blocked, then Unblocked**

File Descriptors/File Descriptor Table

Process-Level File Descriptor Tables (FDT)

- Each process has its own file descriptor table managed by the OS
- A file descriptor (type int) is an index into a processes FD table
- Children made via fork inherit an identical FD table from their parent
    - Those files are not closed nor are they modified in any way. They are the exact same files referred to by the parent.
- Files opened exclusively in a process after a fork do not modify the FD table of another process, child or parent.

open(), close(), read(), write()

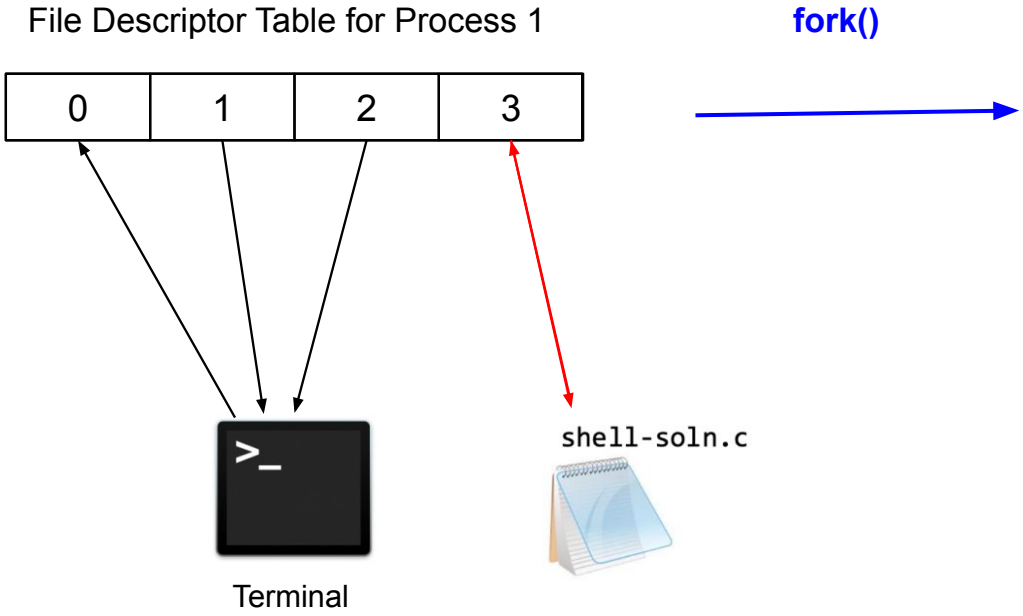
```
int open(const char* pathname, int flags, /* mode_t perm */ )
```

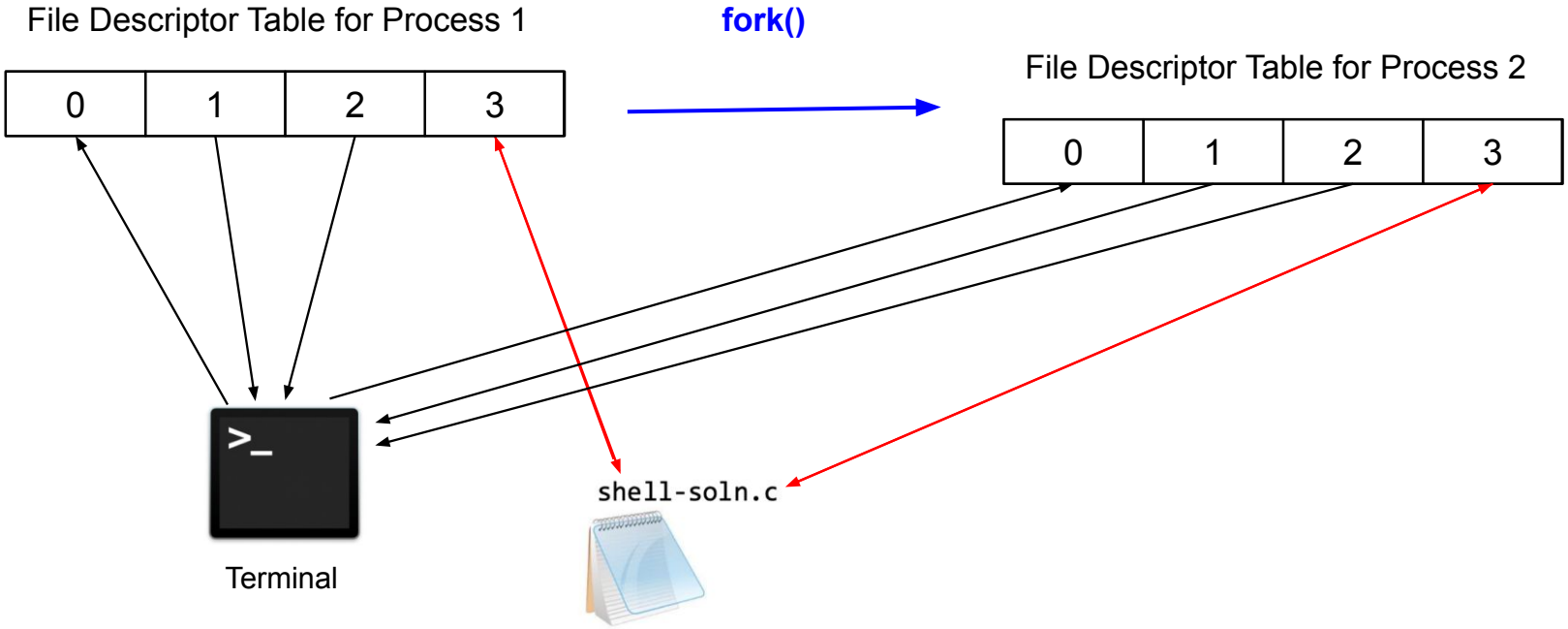
```
int close(int fd)
```

```
ssize_t read(int fd, void *buf, size_t count);
```

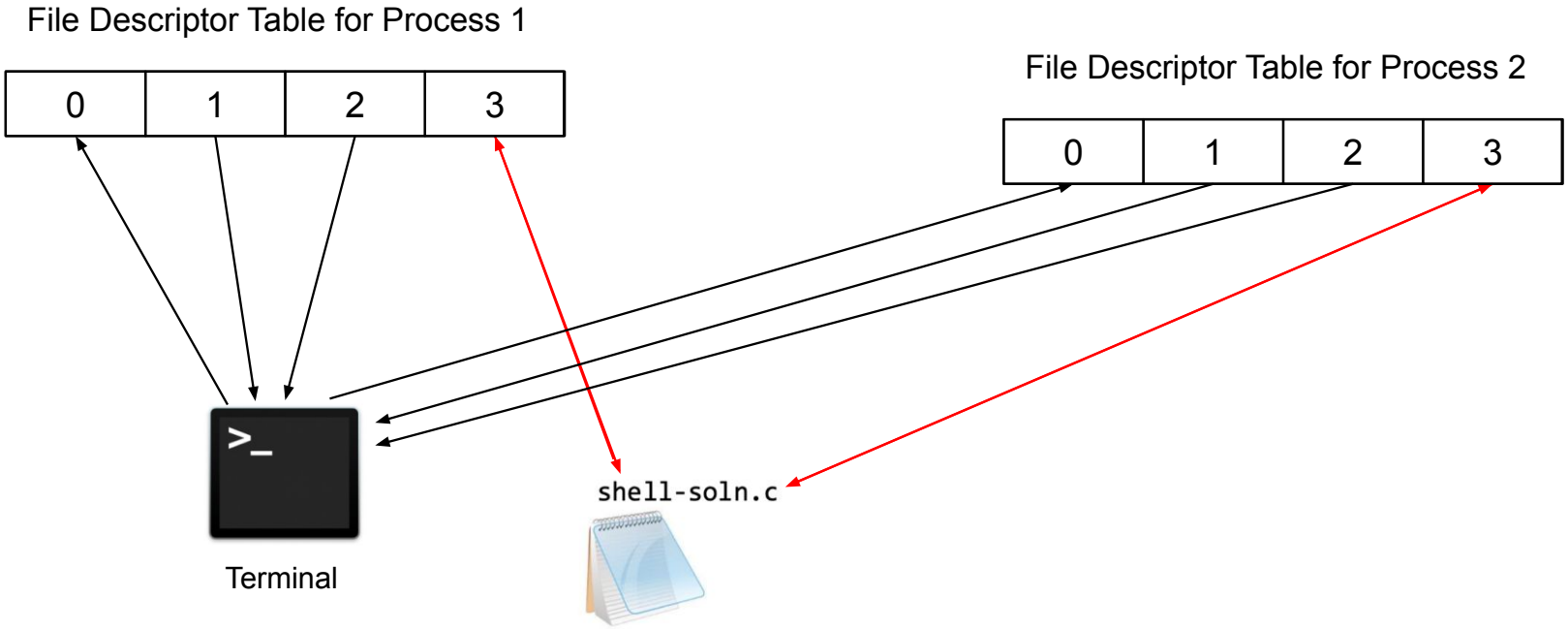
```
ssize_t write(int fd, void *buf, size_t count);
```

EX: Process-Level File Descriptor Tables (FDT)

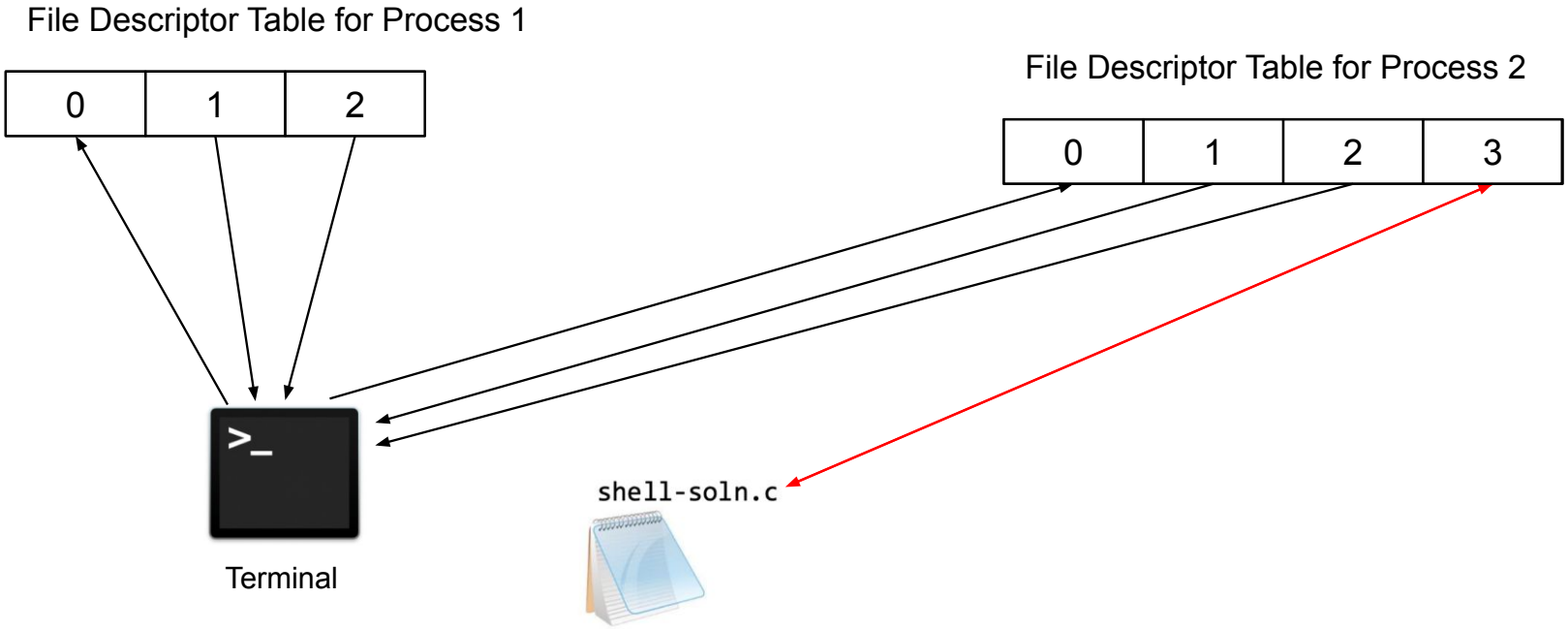



File Descriptor Table for Process 1
0
1
2
3
fork()
File Descriptor Table for Process 2
0
1
2
3
>_
Terminal
shell-soln.c

close(4);

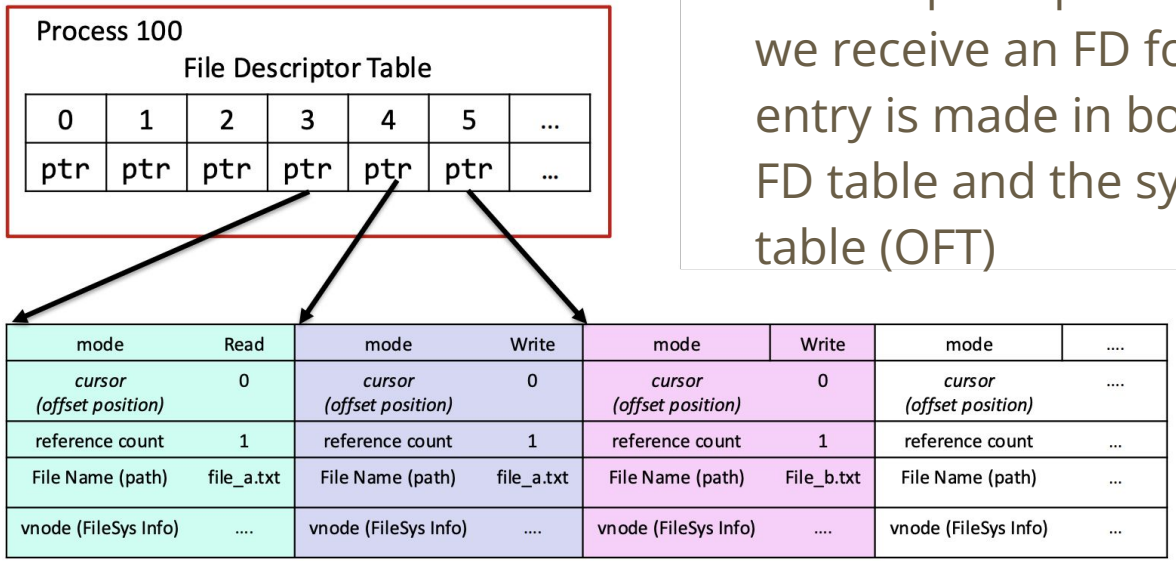

File Descriptor Table for Process 1
0
1
2
3
File Descriptor Table for Process 2
0
1
2
3
>_
Terminal
shell-soln.c

close(4);


File Descriptor Table for Process 1
0
1
2
File Descriptor Table for Process 2
0
1
2
3
>_
Terminal
shell-soln.c

System-Wide Open File Table (OFT)

- Each entry in the process-level FD Table has a pointer to an entry in the system-wide open file table mainted by the kernel!



- As we open up more files, using open(), we receive an FD for that file AND an entry is made in both the process-level FD table and the system-wide open file table (OFT)

File Descriptor Table for Process 1

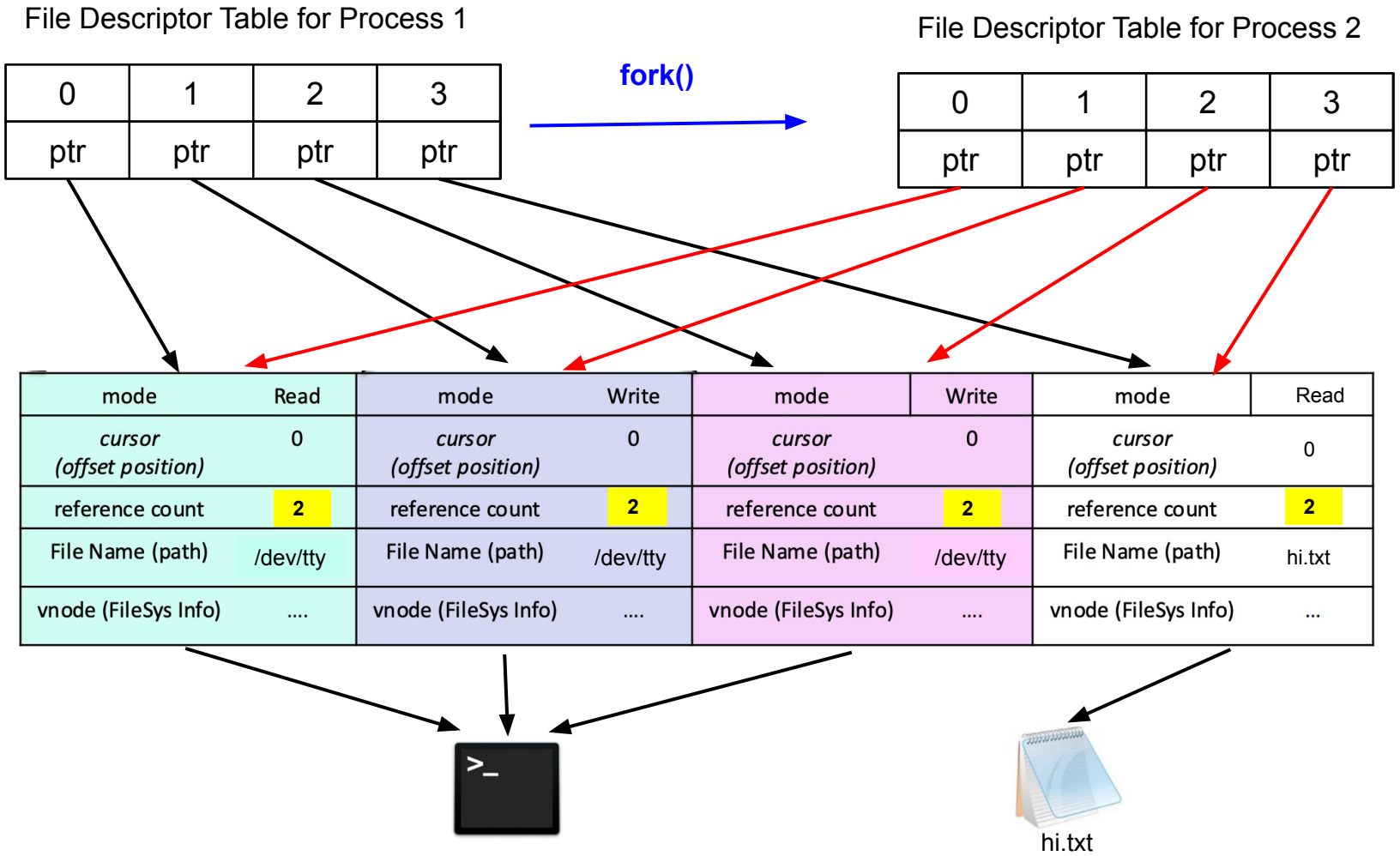
EX 1

0	1	2	3
ptr	ptr	ptr	ptr

fork()

mode	Read	mode	Write	mode	Write	mode	Read
cursor (offset position)	0	*cursor (offset position)*	0	*cursor (offset position)*	0	*cursor (offset position)*	0
reference count	1	reference count	1	reference count	1	reference count	1
File Name (path)	/dev/tty	File Name (path)	/dev/tty	File Name (path)	/dev/tty	File Name (path)	hi.txt
vnode (FileSys Info)	....	vnode (FileSys Info)	....	vnode (FileSys Info)	....	vnode (FileSys Info)	...

hi.txt


File Descriptor Table for Process 1
0
1
2
3
ptr
ptr
ptr
ptr
fork()
File Descriptor Table for Process 2
0
1
2
3
ptr
ptr
ptr
ptr
mode
Read
cursor
(offset position)
0
reference count
2
File Name (path)
/dev/tty
vnode (FileSys Info)
....
mode
Write
cursor
(offset position)
0
reference count
2
File Name (path)
/dev/tty
vnode (FileSys Info)
....
mode
Write
cursor
(offset position)
0
reference count
2
File Name (path)
/dev/tty
vnode (FileSys Info)
....
mode
Read
cursor
(offset position)
0
reference count
2
File Name (path)
hi.txt
vnode (FileSys Info)
...
>_
hi.txt

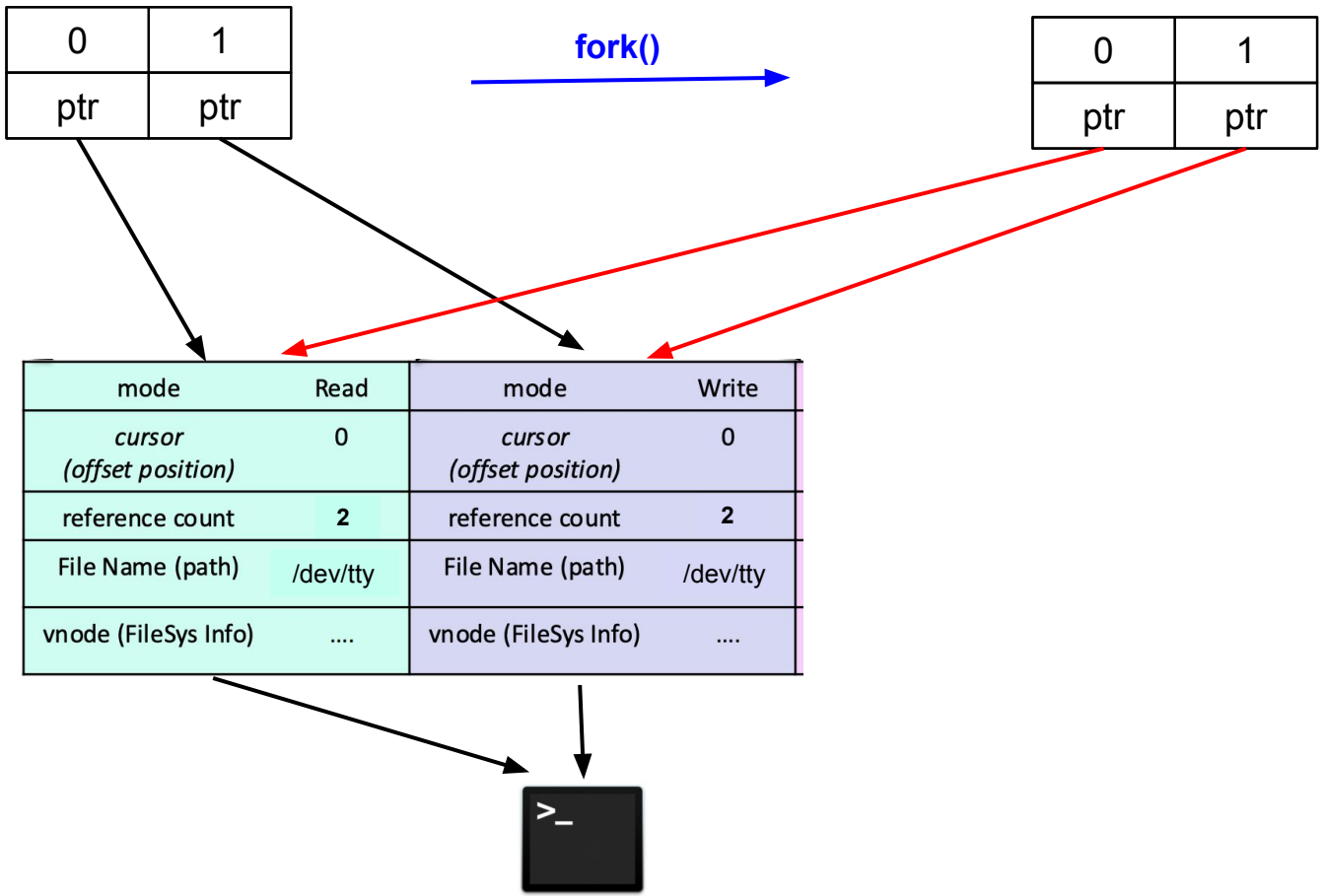
Note:

In this class, the 2 main ways multiple entries from process-level FDTs point to the same entry in the system-wide Open File Table (OFT) are

1. Fork()
2. Dup2()

File Descriptor Table for Process 1

File Descriptor Table for Process 2



EX 2

File Descriptor Table for Process 1

0	1
ptr	ptr

open("hi.txt", O_WRONLY);

File Descriptor Table for Process 2

0	1
ptr	ptr

open("hi.txt", O_WRONLY);

mode	Read	mode	Write
cursor (offset position)	0	*cursor (offset position)*	0
reference count	2	reference count	2
File Name (path)	/dev/tty	File Name (path)	/dev/tty
vnode (FileSys Info)	....	vnode (FileSys Info)	....

File Descriptor Table for Process 1

0	1	2
ptr	ptr	ptr

File Descriptor Table for Process 2

0	1	2
ptr	ptr	ptr

mode	Read	mode	Write	mode	Write	mode	Write
cursor (offset position)	0	*cursor (offset position)*	0	*cursor (offset position)*	0	*cursor (offset position)*	0
reference count	2	reference count	2	reference count	1	reference count	1
File Name (path)	/dev/tty	File Name (path)	/dev/tty	File Name (path)	hi.txt	File Name (path)	hi.txt
vnode (FileSys Info)	....	vnode (FileSys Info)	....	vnode (FileSys Info)	....	vnode (FileSys Info)	...

hi.txt

File Descriptor Table for Process 1 **close(3);**

0	1	2	3
ptr	ptr	ptr	ptr

File Descriptor Table for Process 2

0	1	2	3
ptr	ptr	ptr	ptr

mode	Read	mode	Write	mode	Write	mode	Read
cursor (offset position)	0	*cursor (offset position)*	0	*cursor (offset position)*	0	*cursor (offset position)*	0
reference count	2	reference count	2	reference count	2	reference count	2
File Name (path)	/dev/tty	File Name (path)	/dev/tty	File Name (path)	/dev/tty	File Name (path)	hi.txt
vnode (FileSys Info)	....	vnode (FileSys Info)	....	vnode (FileSys Info)	....	vnode (FileSys Info)	...

hi.txt

EX 3

File Descriptor Table for Process 1 **close(3);**

0	1	2
ptr	ptr	ptr

1. Remove pointer
2. Decrement reference count

File Descriptor Table for Process 2

0	1	2	3
ptr	ptr	ptr	ptr

mode	Read	mode	Write	mode	Write	mode	Read
cursor (offset position)	0	*cursor (offset position)*	0	*cursor (offset position)*	0	*cursor (offset position)*	0
reference count	2	reference count	2	reference count	2	reference count	1
File Name (path)	/dev/tty	File Name (path)	/dev/tty	File Name (path)	/dev/tty	File Name (path)	hi.txt
vnode (FileSys Info)	....	vnode (FileSys Info)	....	vnode (FileSys Info)	....	vnode (FileSys Info)	...

hi.txt

File Descriptor Table for Process 1 **read(3, buf, 10);**

0	1	2	3
ptr	ptr	ptr	ptr

File Descriptor Table for Process 2

0	1	2	3
ptr	ptr	ptr	ptr

mode	Read	mode	Write	mode	Write	mode	Read
cursor (offset position)	0	*cursor (offset position)*	0	*cursor (offset position)*	0	*cursor (offset position)*	0
reference count	2	reference count	2	reference count	2	reference count	2
File Name (path)	/dev/tty	File Name (path)	/dev/tty	File Name (path)	/dev/tty	File Name (path)	hi.txt
vnode (FileSys Info)	....	vnode (FileSys Info)	....	vnode (FileSys Info)	....	vnode (FileSys Info)	...

hi.txt

EX 4

File Descriptor Table for Process 1 **read(3, buf, 10);**

0	1	2	3
ptr	ptr	ptr	ptr

File Descriptor Table for Process 2

0	1	2	3
ptr	ptr	ptr	ptr

mode	Read	mode	Write	mode	Write	mode	Read
cursor (offset position)	0	*cursor (offset position)*	0	*cursor (offset position)*	0	*cursor (offset position)*	**10**
reference count	**2**	reference count	**2**	reference count	**2**	reference count	2
File Name (path)	/dev/tty	File Name (path)	/dev/tty	File Name (path)	/dev/tty	File Name (path)	hi.txt
vnode (FileSys Info)	....	vnode (FileSys Info)	....	vnode (FileSys Info)	....	vnode (FileSys Info)	...

hi.txt

File Descriptor Table for Process 1

0	1	2	3
ptr	ptr	ptr	ptr

File Descriptor Table for Process 2

0	1	2	3
ptr	ptr	ptr	ptr

read(3, buf, 10);

mode	Read	mode	Write	mode	Write	mode	Read
cursor (offset position)	0	*cursor (offset position)*	0	*cursor (offset position)*	0	*cursor (offset position)*	10
reference count	2	reference count	2	reference count	2	reference count	2
File Name (path)	/dev/tty	File Name (path)	/dev/tty	File Name (path)	/dev/tty	File Name (path)	hi.txt
vnode (FileSys Info)	....	vnode (FileSys Info)	....	vnode (FileSys Info)	....	vnode (FileSys Info)	...

hi.txt

File Descriptor Table for Process 1

0	1	2	3
ptr	ptr	ptr	ptr

File Descriptor Table for Process 2

0	1	2	3
ptr	ptr	ptr	ptr

read(3, buf, 10);

mode	Read	mode	Write	mode	Write	mode	Read
cursor (offset position)	0	*cursor (offset position)*	0	*cursor (offset position)*	0	*cursor (offset position)*	**20**
reference count	**2**	reference count	**2**	reference count	**2**	reference count	2
File Name (path)	/dev/tty	File Name (path)	/dev/tty	File Name (path)	/dev/tty	File Name (path)	hi.txt
vnode (FileSys Info)	....	vnode (FileSys Info)	....	vnode (FileSys Info)	....	vnode (FileSys Info)	...

hi.txt

File Descriptor Table for Process 1

0	1	2
ptr	ptr	ptr

read(2, buf, 10);

File Descriptor Table for Process 2

0	1	2
ptr	ptr	ptr

read(2, buf, 10);

mode	Read	mode	Write	mode	Write	mode	Write
cursor (offset position)	0	*cursor (offset position)*	0	*cursor (offset position)*	0	*cursor (offset position)*	0
reference count	2	reference count	2	reference count	1	reference count	1
File Name (path)	/dev/tty	File Name (path)	/dev/tty	File Name (path)	hi.txt	File Name (path)	hi.txt
vnode (FileSys Info)	....	vnode (FileSys Info)	....	vnode (FileSys Info)	....	vnode (FileSys Info)	...

hi.txt

EX 5

File Descriptor Table for Process 1

0	1	2
ptr	ptr	ptr

read(2, buf, 10);

File Descriptor Table for Process 2

0	1	2
ptr	ptr	ptr

read(2, buf, 10);

mode	Read	mode	Write	mode	Write	mode	Write
cursor (offset position)	0	*cursor (offset position)*	0	*cursor (offset position)*	10	*cursor (offset position)*	10
reference count	2	reference count	2	reference count	1	reference count	1
File Name (path)	/dev/tty	File Name (path)	/dev/tty	File Name (path)	hi.txt	File Name (path)	hi.txt
vnode (FileSys Info)	....	vnode (FileSys Info)	....	vnode (FileSys Info)	....	vnode (FileSys Info)	...

hi.txt

Redirection

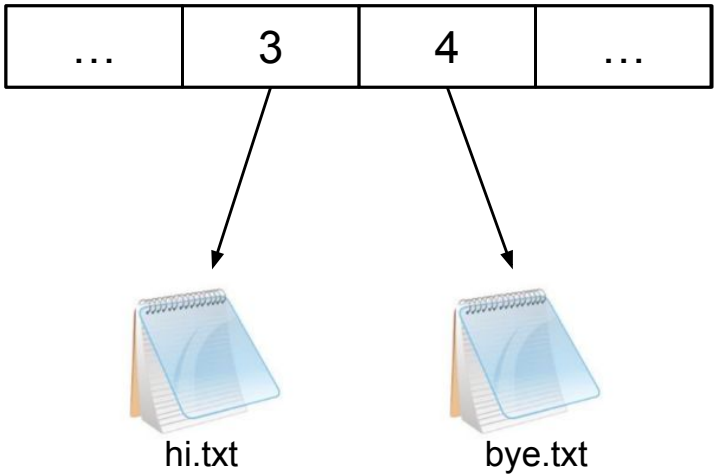
Dup2

- We can manipulate the File Table so that a FD Table entry is associated with another file.
- **int dup2(int oldfd, int newfd);**
  - The file descriptor newfd is adjusted so that it now refers to the same open file pointed to by oldfd.
  - (newfd is closed silently)

EX: Dup2

dup2(int oldfd, int newfd)

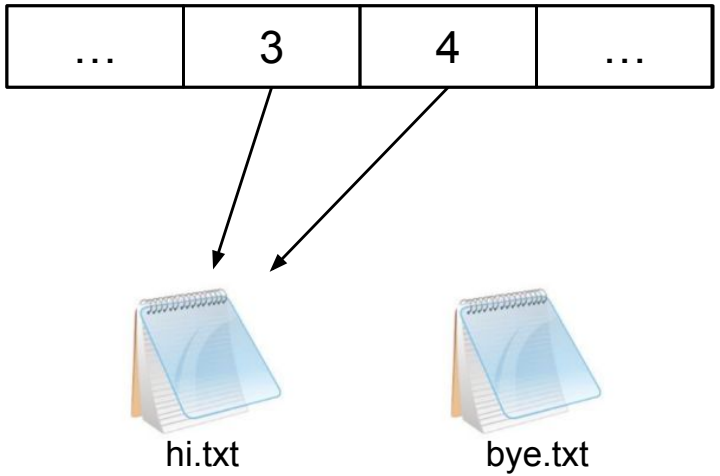
dup2(3, 4);



EX: Dup2

dup2(int oldfd, int newfd)

dup2(3, 4);

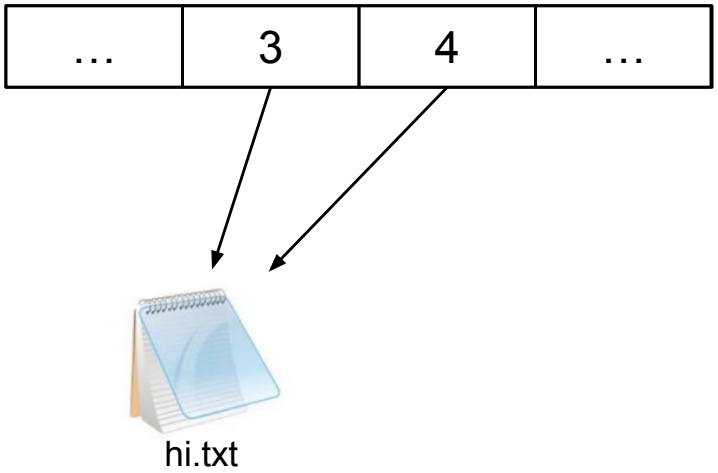


file descriptor newfd is adjusted so that it now refers to the same open file as oldfd

EX: Dup2

dup2(int oldfd, int newfd)

dup2(3, 4);



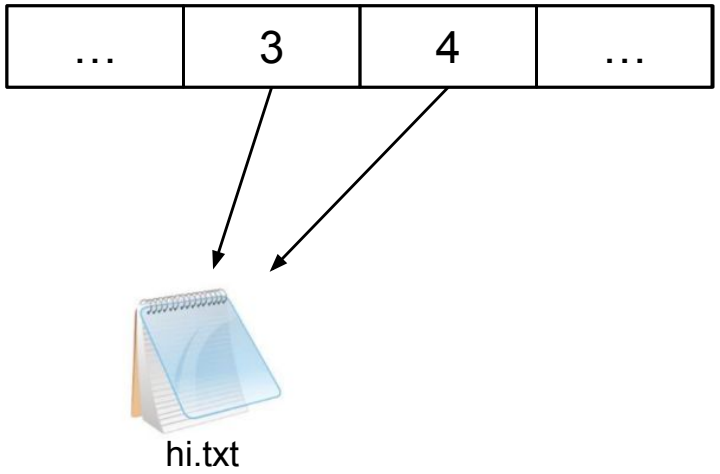
file descriptor newfd is adjusted so that it now refers to the same open file as oldfd

EX: Dup2

dup2(int oldfd, int newfd)

dup2(int oldfd, int newfd)

dup2(3, 4);



dup2(3, 4);

file descriptor newfd is adjusted so that it now refers to the same open file as oldfd

Pipes

Pipelines

What is a pipe?

- Kernel buffer with two file descriptors - one for each end
- Push from/pull from buffer from write and read end respectively

Pipelined functions: i.e. cat log | grep brains | wc -l

- What does this do? Output log → grep for lines that start with “brains” → count number of such matching lines

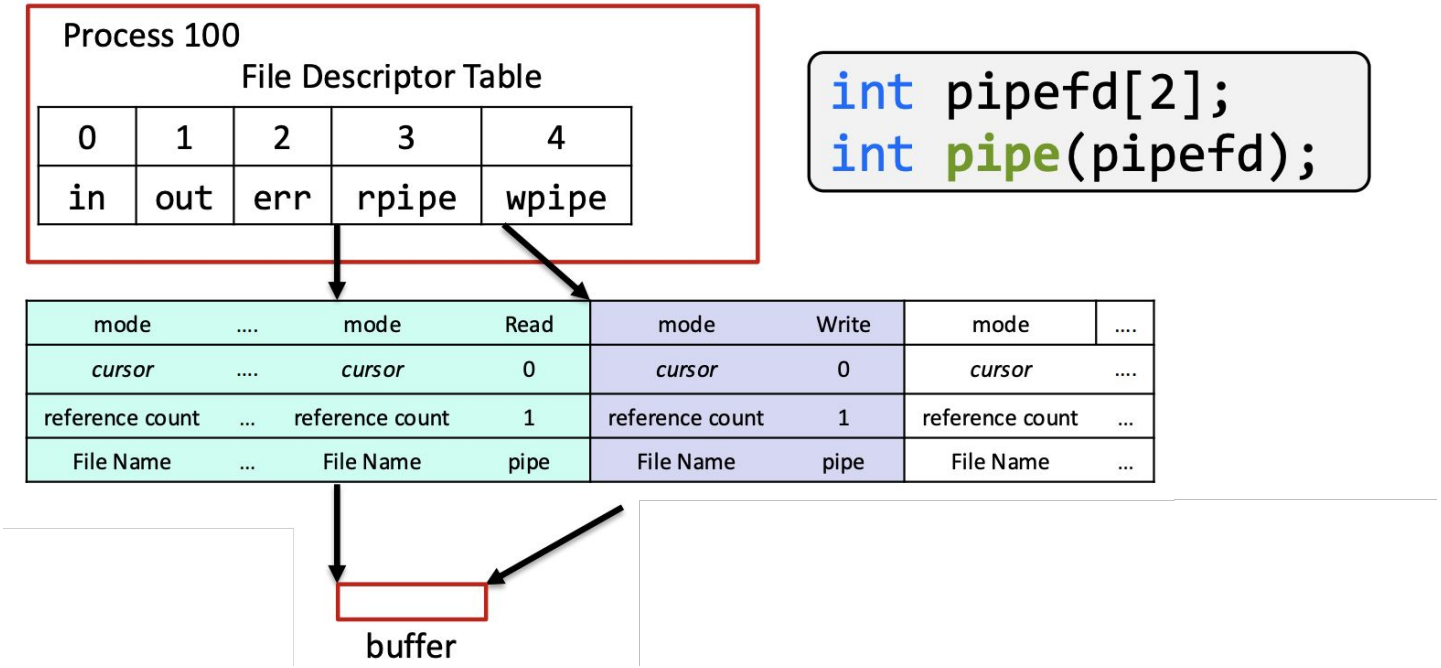
Pipelines

But how does this happen?

For an N-stage pipeline:

1) Fork N children, create N-1 pipes
2) In each child i:
    a) Dup2 pipes from input/output to STDIN_FILENO and STDOUT_FILENO
    b) Close pipes
    c) Execvp command
3) Cleanup in parent

Pipe Visualization



Pipelines Check-In

- Consider the following pipelined command:

```
# sleep 1 | sleep 20 | sleep 100
```

  How long does it take to finish?

Pipelines Check-In

- Consider the following pipelined command:
  ```
  # sleep 1 | sleep 20 | sleep 100
  ```

  How long does it take to finish?
  100 seconds

  Why?

Pipelines Check-In

- Consider the following pipelined command:
  ```
  # sleep 1 | sleep 20 | sleep 100
  ```

  How long does it take to finish?
  100 seconds

  Why?
  Pipelined processes run in parallel