CS 241: Systems Programming
Lecture 28. Signals

Spring 2020

Prof. Stephen Checkoway

Which of the following is the standard procedure to run a new program cmd with argument arg (error checking has been omitted below)?

```
A. pid_t pid = fork();
   if (pid == 0) execl(path, path, arg, (char *)0);

B. pid_t pid = fork();
   if (pid != 0) execl(path, path, arg, (char *)0);

C. int ret = execl(path, path, arg, (char *)0);
   if (ret == 0) fork();

D. int ret = execl(path, path, arg, (char *)0);
   if (ret != 0) fork();
```

Redirection

The dup2 system call creates a new file descriptor that refers to the same file as the old descriptor

```
int dup2(int oldfd, int newfd);
```

We can use this to perform redirection of `stdin`/`stdout`/`stderr`

1. Open the file we want to redirect input from/output to
2. `dup2` the returned file descriptor to **`STDIN_FILENO`**/**`STDOUT_FILENO`**/**`STDERR_FILENO`**
3. Close the original file descriptor

Strace example

When running

`$ /bin/echo hello > output.txt`

we can see the sequence of system calls Bash makes using `strace -f`

- On Linux, `fork()` uses the `clone` system call

```
8038  clone(...) = 8039
8039  openat(AT_FDCWD, "output.txt", O_WRONLY|O_CREAT|O_TRUNC, 0666) = 3
8039  dup2(3, 1)                                = 1
8039  close(3)                                  = 0
8039  execve("/bin/echo", ["/bin/echo", "hello"], ...) = 0
```

What about pipes?

```
$ /bin/echo hi | /usr/bin/head
```

We can `strace` this!

```
12449 execve("/bin/bash", ["bash", "-c", "/bin/echo hi | /usr/bin/head"], ...) = 0
12449 pipe([3, 4])                              = 0
12449 clone(...)                                = 12450
12449 close(4)                                  = 0
12449 clone(...)                                = 12451
12449 close(3)                                  = 0
12450 close(3)                                  = 0
12449 wait4(-1,  <unfinished ...>
12450 dup2(4, 1)                                = 1
12451 dup2(3, 0)                                = 0
12451 close(3)                                  = 0
12450 close(4)                                  = 0
12451 execve("/usr/bin/head", ["/usr/bin/head"], ...) = 0
12450 execve("/bin/echo", ["/bin/echo", "hi"], ...) = 0
12449 <... wait4 resumed> [{WIFEXITED(s) && WEXITSTATUS(s) == 0}], 0, NULL) = 12450
12449 wait4(-1,  <unfinished ...>
12449 <... wait4 resumed> [{WIFEXITED(s) && WEXITSTATUS(s) == 0}], 0, NULL) = 12451
```

12449 – bash
12450 – echo
12451 – head

Pipes

Oldest form of UNIX System interprocess communication (IPC)

Have some limitations:

- Historically have been half-duplex (data only flows one direction)
    - Data only flows one direction
    - Some systems have full-duplex, but this isn't standard
- Can only be used between processes with a common ancestor

pipe(2)

```
#include <unistd.h>
```

```
int pipe(int fd[2])
```

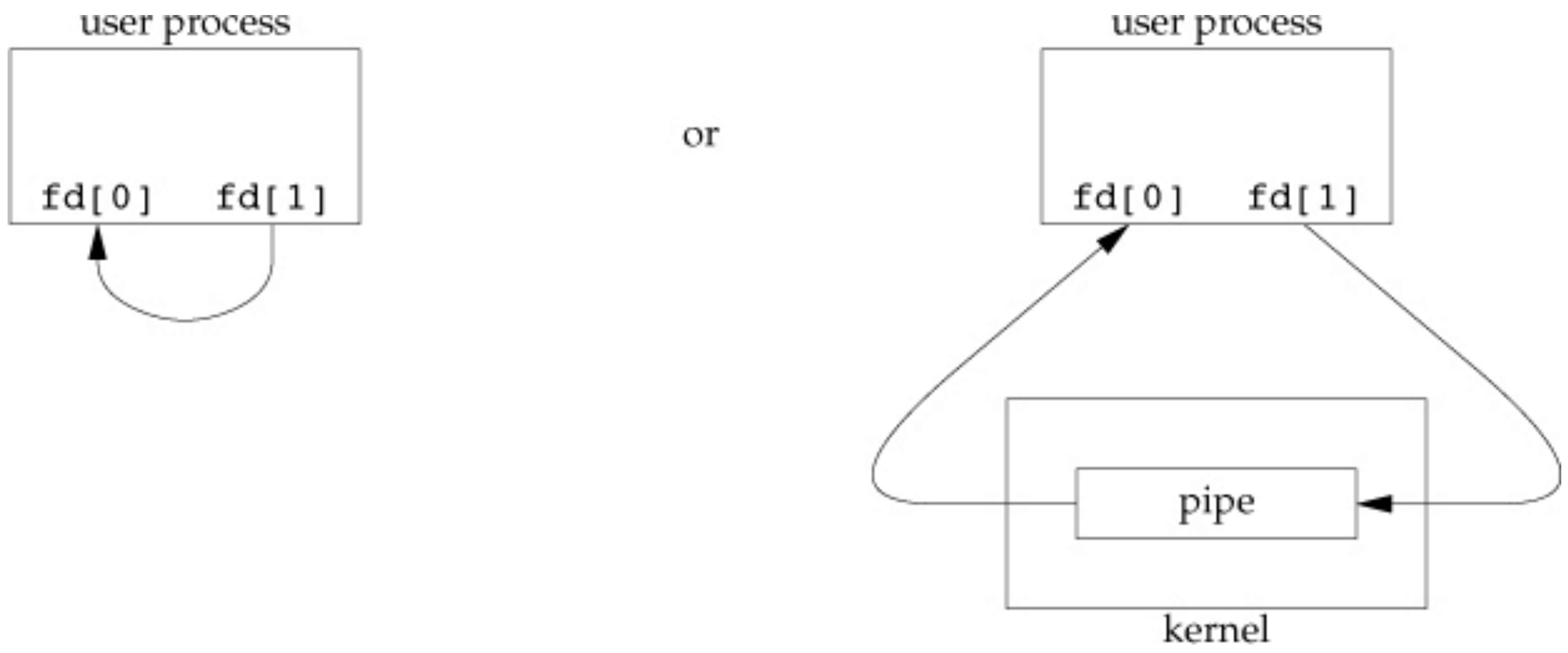
- Returns 0 on success, -1 on error

Returns values in array fd

- `fd[0]` is opened for reading
- `fd[1]` is opened for writing

File descriptors are connected to each other!

After call to pipe()



Ok…

A pipe in a single process is usually unnecessary

- We can already talk to ourselves!

Normally, you create a pipe and then fork()

This creates a channel from parent to child (or vice versa)

After call to fork()

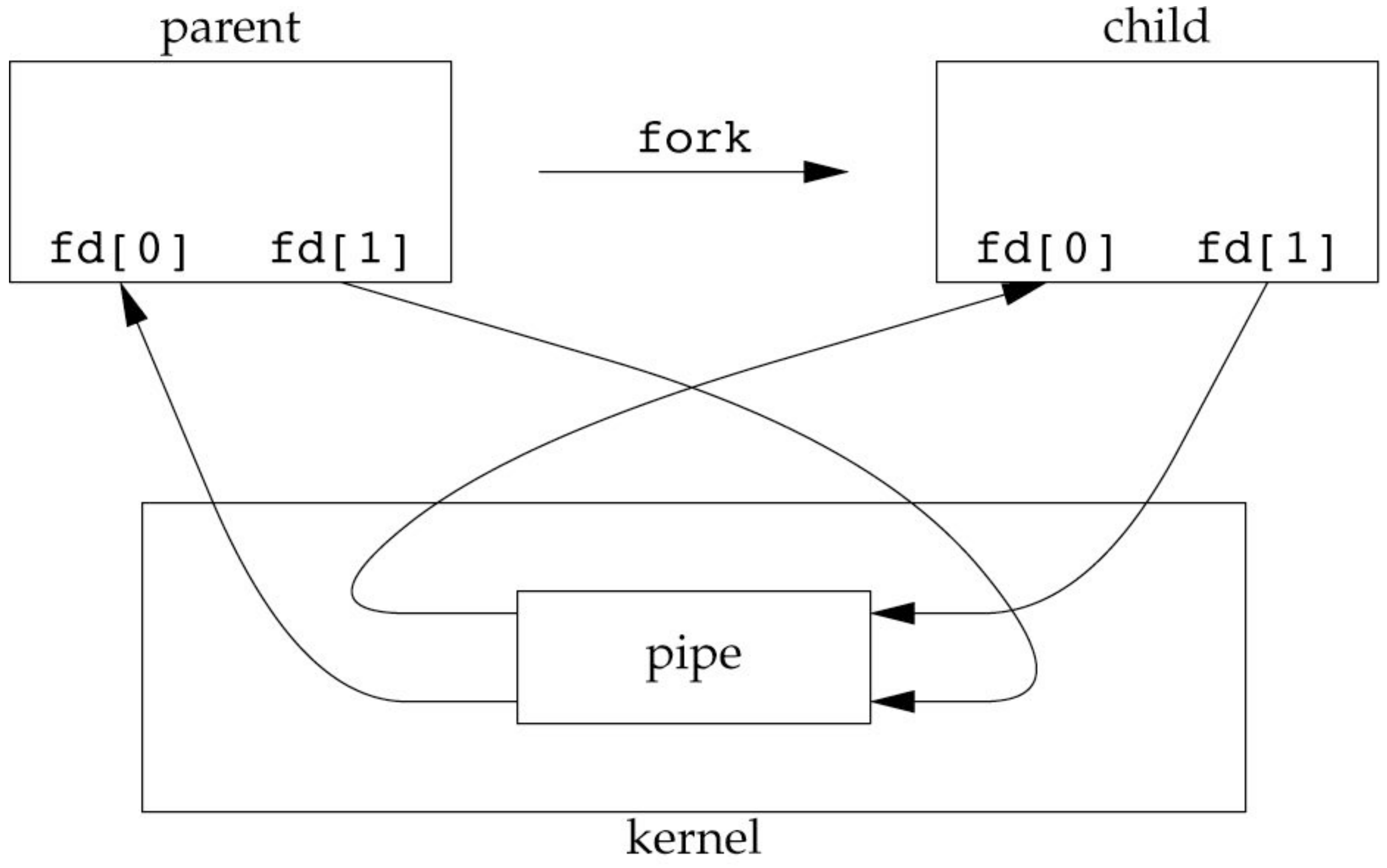


Figure 15.3 Half-duplex pipe after a `fork`

Close unneeded descriptors

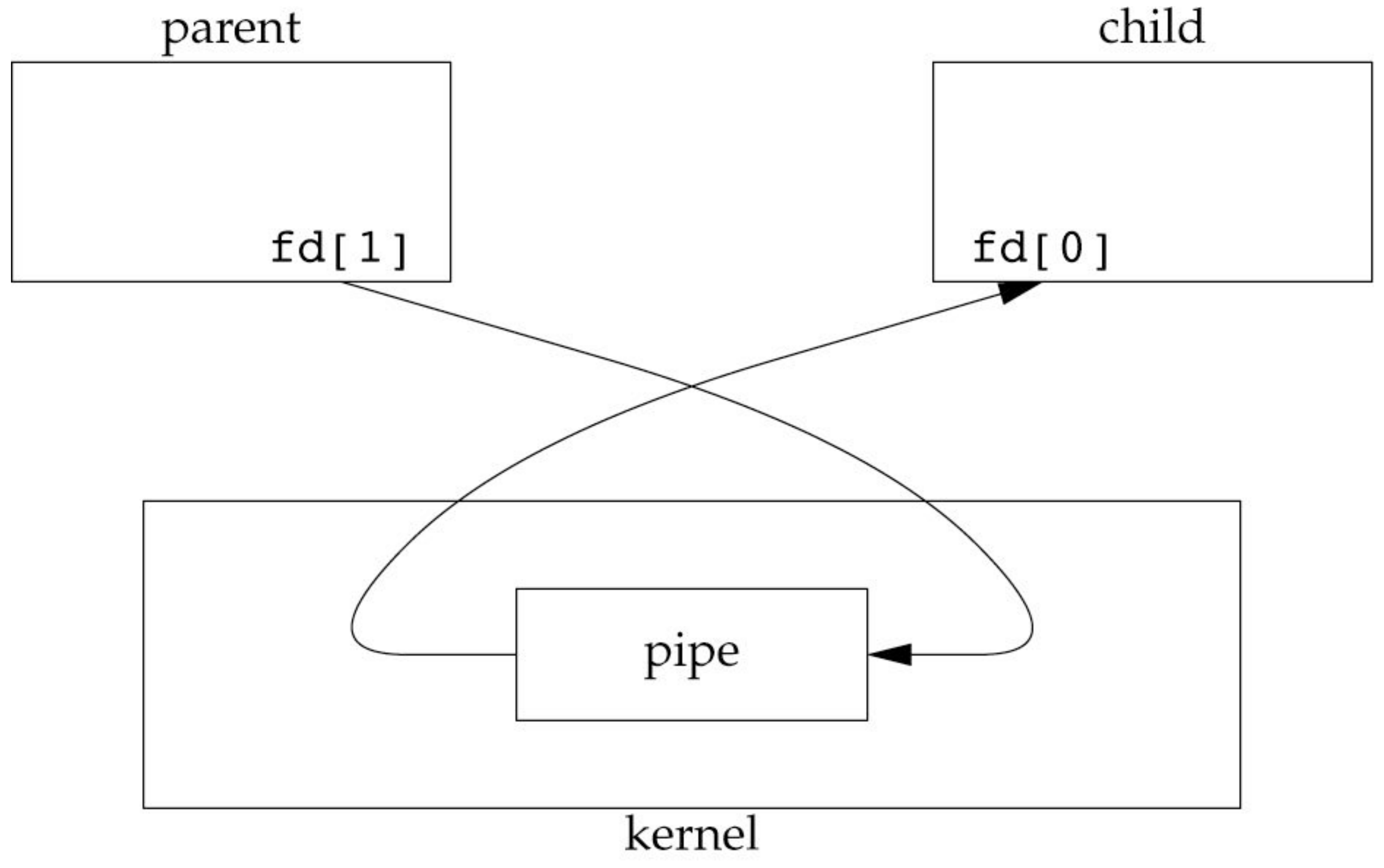


Figure 15.4 Pipe from parent to child

```
12449 execve("/bin/bash", ["bash", "-c", "/bin/echo hi | /usr/bin/head"], ...) = 0
12449 pipe([3, 4])                           = 0
12449 clone(...)                             = 12450
12449 close(4)                               = 0
12449 clone(...)                             = 12451
12449 close(3)                               = 0
12450 close(3)                               = 0
12449 wait4(-1,  <unfinished ...>
12450 dup2(4, 1)                             = 1
12451 dup2(3, 0)                             = 0
12451 close(3)                               = 0
12450 close(4)                               = 0
12451 execve("/usr/bin/head", ["/usr/bin/head"], ...) = 0
12450 execve("/bin/echo", ["/bin/echo", "hi"], ...) = 0
12449 <... wait4 resumed> [{WIFEXITED(s) && WEXITSTATUS(s) == 0}], 0, NULL) = 12450
12449 wait4(-1,  <unfinished ...>
12449 <... wait4 resumed> [{WIFEXITED(s) && WEXITSTATUS(s) == 0}], 0, NULL) = 12451
```

fd	strace'
0	terminal
1	terminal
2	terminal
3	
4	

12449	bash
12450	echo
12451	head

The primes (' or '') denote forked children that haven't execed

```
12449 execve("/bin/bash", ["bash", "-c", "/bin/echo hi | /usr/bin/head"], ...) = 0
12449 pipe([3, 4])                       = 0
12449 clone(...)                         = 12450
12449 close(4)                           = 0
12449 clone(...)                         = 12451
12449 close(3)                           = 0
12450 close(3)                           = 0
12449 wait4(-1,  <unfinished ...>
12450 dup2(4, 1)                         = 1
12451 dup2(3, 0)                         = 0
12451 close(3)                           = 0
12450 close(4)                           = 0
12451 execve("/usr/bin/head", ["/usr/bin/head"], ...) = 0
12450 execve("/bin/echo", ["/bin/echo", "hi"], ...) = 0
12449 <... wait4 resumed> [{WIFEXITED(s) && WEXITSTATUS(s) == 0}], 0, NULL) = 12450
12449 wait4(-1,  <unfinished ...>
12449 <... wait4 resumed> [{WIFEXITED(s) && WEXITSTATUS(s) == 0}], 0, NULL) = 12451
```

fd	bash
0	terminal
1	terminal
2	terminal
3	
4	

12449	bash
12450	echo
12451	head

The primes (' or '') denote forked children that haven't execed

```
12449 execve("/bin/bash", ["bash", "-c", "/bin/echo hi | /usr/bin/head"], ...) = 0
12449 pipe([3, 4])                          = 0
12449 clone(...)                            = 12450
12449 close(4)                              = 0
12449 clone(...)                            = 12451
12449 close(3)                              = 0
12450 close(3)                              = 0
12449 wait4(-1,  <unfinished ...>
12450 dup2(4, 1)                            = 1
12451 dup2(3, 0)                            = 0
12451 close(3)                              = 0
12450 close(4)                              = 0
12451 execve("/usr/bin/head", ["/usr/bin/head"], ...) = 0
12450 execve("/bin/echo", ["/bin/echo", "hi"], ...) = 0
12449 <... wait4 resumed> [{WIFEXITED(s) && WEXITSTATUS(s) == 0}], 0, NULL) = 12450
12449 wait4(-1,  <unfinished ...>
12449 <... wait4 resumed> [{WIFEXITED(s) && WEXITSTATUS(s) == 0}], 0, NULL) = 12451
```

fd	bash
0	terminal
1	terminal
2	terminal
3	pipe_read
4	pipe_write

12449	bash
12450	echo
12451	head

The primes (' or '') denote forked children that haven't execed

```
12449 execve("/bin/bash", ["bash", "-c", "/bin/echo hi | /usr/bin/head"], ...) = 0
12449 pipe([3, 4])                          = 0
12449 clone(...)                            = 12450
12449 close(4)                              = 0
12449 clone(...)                            = 12451
12449 close(3)                              = 0
12450 close(3)                              = 0
12449 wait4(-1,  <unfinished ...>
12450 dup2(4, 1)                            = 1
12451 dup2(3, 0)                            = 0
12451 close(3)                              = 0
12450 close(4)                              = 0
12451 execve("/usr/bin/head", ["/usr/bin/head"], ...) = 0
12450 execve("/bin/echo", ["/bin/echo", "hi"], ...) = 0
12449 <... wait4 resumed> [{WIFEXITED(s) && WEXITSTATUS(s) == 0}], 0, NULL) = 12450
12449 wait4(-1,  <unfinished ...>
12449 <... wait4 resumed> [{WIFEXITED(s) && WEXITSTATUS(s) == 0}], 0, NULL) = 12451
```

fd	bash	bash'
0	terminal	terminal
1	terminal	terminal
2	terminal	terminal
3	pipe_read	pipe_read
4	pipe_write	pipe_write

```
12449 — bash
12450 — echo
12451 — head
```

The primes (' or '') denote forked children that haven't execed

```
12449 execve("/bin/bash", ["bash", "-c", "/bin/echo hi | /usr/bin/head"], ...) = 0
12449 pipe([3, 4])                       = 0
12449 clone(...)                         = 12450
12449 close(4)                           = 0
12449 clone(...)                         = 12451
12449 close(3)                           = 0
12450 close(3)                           = 0
12449 wait4(-1,  <unfinished ...>
12450 dup2(4, 1)                         = 1
12451 dup2(3, 0)                         = 0
12451 close(3)                           = 0
12450 close(4)                           = 0
12451 execve("/usr/bin/head", ["/usr/bin/head"], ...) = 0
12450 execve("/bin/echo", ["/bin/echo", "hi"], ...) = 0
12449 <... wait4 resumed> [{WIFEXITED(s) && WEXITSTATUS(s) == 0}], 0, NULL) = 12450
12449 wait4(-1,  <unfinished ...>
12449 <... wait4 resumed> [{WIFEXITED(s) && WEXITSTATUS(s) == 0}], 0, NULL) = 12451
```

fd	bash	bash'
0	terminal	terminal
1	terminal	terminal
2	terminal	terminal
3	pipe_read	pipe_read
4	~~pipe_write~~	pipe_write

12449 — bash
12450 — echo
12451 — head

The primes (' or '') denote forked children that haven't execed

```
12449 execve("/bin/bash", ["bash", "-c", "/bin/echo hi | /usr/bin/head"], ...) = 0
12449 pipe([3, 4])                        = 0
12449 clone(...)                          = 12450
12449 close(4)                            = 0
12449 clone(...)                          = 12451
12449 close(3)                            = 0
12450 close(3)                            = 0
12449 wait4(-1,  <unfinished ...>
12450 dup2(4, 1)                          = 1
12451 dup2(3, 0)                          = 0
12451 close(3)                            = 0
12450 close(4)                            = 0
12451 execve("/usr/bin/head", ["/usr/bin/head"], ...) = 0
12450 execve("/bin/echo", ["/bin/echo", "hi"], ...) = 0
12449 <... wait4 resumed> [{WIFEXITED(s) && WEXITSTATUS(s) == 0}], 0, NULL) = 12450
12449 wait4(-1,  <unfinished ...>
12449 <... wait4 resumed> [{WIFEXITED(s) && WEXITSTATUS(s) == 0}], 0, NULL) = 12451
```

fd	bash	bash'	bash''
0	terminal	terminal	terminal
1	terminal	terminal	terminal
2	terminal	terminal	terminal
3	pipe_read	pipe_read	pipe_read
4		pipe_write	

12449 — bash
12450 — echo
12451 — head

The primes (' or '') denote forked children that haven't execed

```
12449 execve("/bin/bash", ["bash", "-c", "/bin/echo hi | /usr/bin/head"], ...) = 0
12449 pipe([3, 4])                              = 0
12449 clone(...)                                = 12450
12449 close(4)                                  = 0
12449 clone(...)                                = 12451
12449 close(3)                                  = 0
12450 close(3)                                  = 0
12449 wait4(-1,  <unfinished ...>
12450 dup2(4, 1)                                = 1
12451 dup2(3, 0)                                = 0
12451 close(3)                                  = 0
12450 close(4)                                  = 0
12451 execve("/usr/bin/head", ["/usr/bin/head"], ...) = 0
12450 execve("/bin/echo", ["/bin/echo", "hi"], ...) = 0
12449 <... wait4 resumed> [{WIFEXITED(s) && WEXITSTATUS(s) == 0}], 0, NULL) = 12450
12449 wait4(-1,  <unfinished ...>
12449 <... wait4 resumed> [{WIFEXITED(s) && WEXITSTATUS(s) == 0}], 0, NULL) = 12451
```

fd	bash	bash'	bash''
0	terminal	terminal	terminal
1	terminal	terminal	terminal
2	terminal	terminal	terminal
3	~~pipe_read~~	pipe_read	pipe_read
4		pipe_write	

12449	bash
12450	echo
12451	head

The primes (' or '') denote forked children that haven't execed

```
12449 execve("/bin/bash", ["bash", "-c", "/bin/echo hi | /usr/bin/head"], ...) = 0
12449 pipe([3, 4])                             = 0
12449 clone(...)                               = 12450
12449 close(4)                                 = 0
12449 clone(...)                               = 12451
12449 close(3)                                 = 0
12450 close(3)                                 = 0
12449 wait4(-1,  <unfinished ...>
12450 dup2(4, 1)                               = 1
12451 dup2(3, 0)                               = 0
12451 close(3)                                 = 0
12450 close(4)                                 = 0
12451 execve("/usr/bin/head", ["/usr/bin/head"], ...) = 0
12450 execve("/bin/echo", ["/bin/echo", "hi"], ...) = 0
12449 <... wait4 resumed> [{WIFEXITED(s) && WEXITSTATUS(s) == 0}], 0, NULL) = 12450
12449 wait4(-1,  <unfinished ...>
12449 <... wait4 resumed> [{WIFEXITED(s) && WEXITSTATUS(s) == 0}], 0, NULL) = 12451
```

fd	bash	bash'	bash''
0	terminal	terminal	terminal
1	terminal	terminal	terminal
2	terminal	terminal	terminal
3		~~pipe_read~~	pipe_read
4		pipe_write	

12449 — bash	🟨
12450 — echo	🟦
12451 — head	🟥

The primes (' or '') denote forked children that haven't execed

```
12449 execve("/bin/bash", ["bash", "-c", "/bin/echo hi | /usr/bin/head"], ...) = 0
12449 pipe([3, 4])                                = 0
12449 clone(...)                                  = 12450
12449 close(4)                                    = 0
12449 clone(...)                                  = 12451
12449 close(3)                                    = 0
12450 close(3)                                    = 0
12449 wait4(-1,  <unfinished ...>
12450 dup2(4, 1)                                  = 1
12451 dup2(3, 0)                                  = 0
12451 close(3)                                    = 0
12450 close(4)                                    = 0
12451 execve("/usr/bin/head", ["/usr/bin/head"], ...) = 0
12450 execve("/bin/echo", ["/bin/echo", "hi"], ...) = 0
12449 <... wait4 resumed> [{WIFEXITED(s) && WEXITSTATUS(s) == 0}], 0, NULL) = 12450
12449 wait4(-1,  <unfinished ...>
12449 <... wait4 resumed> [{WIFEXITED(s) && WEXITSTATUS(s) == 0}], 0, NULL) = 12451
```

fd	bash	bash'	bash''
0	terminal	terminal	terminal
1	terminal	terminal	terminal
2	terminal	terminal	terminal
3			pipe_read
4		pipe_write	

```
12449 — bash
12450 — echo
12451 — head
```

The primes (' or '') denote forked children that haven't execed

```
12449 execve("/bin/bash", ["bash", "-c", "/bin/echo hi | /usr/bin/head"], ...) = 0
12449 pipe([3, 4])                            = 0
12449 clone(...)                              = 12450
12449 close(4)                                = 0
12449 clone(...)                              = 12451
12449 close(3)                                = 0
12450 close(3)                                = 0
12449 wait4(-1,  <unfinished ...>
12450 dup2(4, 1)                              = 1
12451 dup2(3, 0)                              = 0
12451 close(3)                                = 0
12450 close(4)                                = 0
12451 execve("/usr/bin/head", ["/usr/bin/head"], ...) = 0
12450 execve("/bin/echo", ["/bin/echo", "hi"], ...) = 0
12449 <... wait4 resumed> [{WIFEXITED(s) && WEXITSTATUS(s) == 0}], 0, NULL) = 12450
12449 wait4(-1,  <unfinished ...>
12449 <... wait4 resumed> [{WIFEXITED(s) && WEXITSTATUS(s) == 0}], 0, NULL) = 12451
```

fd	bash	bash'	bash''
0	terminal	terminal	terminal
1	terminal	pipe_write	terminal
2	terminal	terminal	terminal
3			pipe_read
4		pipe_write	

12449 — bash
12450 — echo
12451 — head

The primes (' or '') denote forked children that haven't execed

```
12449 execve("/bin/bash", ["bash", "-c", "/bin/echo hi | /usr/bin/head"], ...) = 0
12449 pipe([3, 4])                                = 0
12449 clone(...)                                  = 12450
12449 close(4)                                    = 0
12449 clone(...)                                  = 12451
12449 close(3)                                    = 0
12450 close(3)                                    = 0
12449 wait4(-1,  <unfinished ...>
12450 dup2(4, 1)                                  = 1
12451 dup2(3, 0)                                  = 0
12451 close(3)                                    = 0
12450 close(4)                                    = 0
12451 execve("/usr/bin/head", ["/usr/bin/head"], ...) = 0
12450 execve("/bin/echo", ["/bin/echo", "hi"], ...) = 0
12449 <... wait4 resumed> [{WIFEXITED(s) && WEXITSTATUS(s) == 0}], 0, NULL) = 12450
12449 wait4(-1,  <unfinished ...>
12449 <... wait4 resumed> [{WIFEXITED(s) && WEXITSTATUS(s) == 0}], 0, NULL) = 12451
```

fd	bash	bash'	bash''
0	terminal	terminal	pipe_read
1	terminal	pipe_write	terminal
2	terminal	terminal	terminal
3			pipe_read
4		pipe_write	

12449 — bash
12450 — echo
12451 — head

The primes (' or '') denote forked children that haven't execed

```
12449 execve("/bin/bash", ["bash", "-c", "/bin/echo hi | /usr/bin/head"], ...) = 0
12449 pipe([3, 4])                                = 0
12449 clone(...)                                  = 12450
12449 close(4)                                    = 0
12449 clone(...)                                  = 12451
12449 close(3)                                    = 0
12450 close(3)                                    = 0
12449 wait4(-1,  <unfinished ...>
12450 dup2(4, 1)                                  = 1
12451 dup2(3, 0)                                  = 0
12451 close(3)                                    = 0
12450 close(4)                                    = 0
12451 execve("/usr/bin/head", ["/usr/bin/head"], ...) = 0
12450 execve("/bin/echo", ["/bin/echo", "hi"], ...) = 0
12449 <... wait4 resumed> [{WIFEXITED(s) && WEXITSTATUS(s) == 0}], 0, NULL) = 12450
12449 wait4(-1,  <unfinished ...>
12449 <... wait4 resumed> [{WIFEXITED(s) && WEXITSTATUS(s) == 0}], 0, NULL) = 12451
```

fd	bash	bash'	bash''
0	terminal	terminal	pipe_read
1	terminal	pipe_write	terminal
2	terminal	terminal	terminal
3			~~pipe_read~~
4		pipe_write	

12449 — bash	🟨
12450 — echo	🟦
12451 — head	🟥

The primes (' or '') denote forked children that haven't execed

```
12449 execve("/bin/bash", ["bash", "-c", "/bin/echo hi | /usr/bin/head"], ...) = 0
12449 pipe([3, 4])                             = 0
12449 clone(...)                               = 12450
12449 close(4)                                 = 0
12449 clone(...)                               = 12451
12449 close(3)                                 = 0
12450 close(3)                                 = 0
12449 wait4(-1,  <unfinished ...>
12450 dup2(4, 1)                               = 1
12451 dup2(3, 0)                               = 0
12451 close(3)                                 = 0
12450 close(4)                                 = 0
12451 execve("/usr/bin/head", ["/usr/bin/head"], ...) = 0
12450 execve("/bin/echo", ["/bin/echo", "hi"], ...) = 0
12449 <... wait4 resumed> [{WIFEXITED(s) && WEXITSTATUS(s) == 0}], 0, NULL) = 12450
12449 wait4(-1,  <unfinished ...>
12449 <... wait4 resumed> [{WIFEXITED(s) && WEXITSTATUS(s) == 0}], 0, NULL) = 12451
```

fd	bash	bash'	bash''
0	terminal	terminal	pipe_read
1	terminal	pipe_write	terminal
2	terminal	terminal	terminal
3			
4		~~pipe_write~~	

12449 — bash
12450 — echo
12451 — head

The primes (' or '') denote forked children that haven't execed

```
12449 execve("/bin/bash", ["bash", "-c", "/bin/echo hi | /usr/bin/head"], ...) = 0
12449 pipe([3, 4])                                   = 0
12449 clone(...)                                     = 12450
12449 close(4)                                       = 0
12449 clone(...)                                     = 12451
12449 close(3)                                       = 0
12450 close(3)                                       = 0
12449 wait4(-1,  <unfinished ...>
12450 dup2(4, 1)                                     = 1
12451 dup2(3, 0)                                     = 0
12451 close(3)                                       = 0
12450 close(4)                                       = 0
12451 execve("/usr/bin/head", ["/usr/bin/head"], ...) = 0
12450 execve("/bin/echo", ["/bin/echo", "hi"], ...) = 0
12449 <... wait4 resumed> [{WIFEXITED(s) && WEXITSTATUS(s) == 0}], 0, NULL) = 12450
12449 wait4(-1,  <unfinished ...>
12449 <... wait4 resumed> [{WIFEXITED(s) && WEXITSTATUS(s) == 0}], 0, NULL) = 12451
```

fd	bash	bash'	head
0	terminal	terminal	pipe_read
1	terminal	pipe_write	terminal
2	terminal	terminal	terminal
3			
4			

12449 – bash
12450 – echo
12451 – head

The primes (' or '') denote forked children that haven't execed

```
12449 execve("/bin/bash", ["bash", "-c", "/bin/echo hi | /usr/bin/head"], ...) = 0
12449 pipe([3, 4])                                   = 0
12449 clone(...)                                     = 12450
12449 close(4)                                       = 0
12449 clone(...)                                     = 12451
12449 close(3)                                       = 0
12450 close(3)                                       = 0
12449 wait4(-1,  <unfinished ...>
12450 dup2(4, 1)                                     = 1
12451 dup2(3, 0)                                     = 0
12451 close(3)                                       = 0
12450 close(4)                                       = 0
12451 execve("/usr/bin/head", ["/usr/bin/head"], ...) = 0
12450 execve("/bin/echo", ["/bin/echo", "hi"], ...) = 0
12449 <... wait4 resumed> [{WIFEXITED(s) && WEXITSTATUS(s) == 0}], 0, NULL) = 12450
12449 wait4(-1,  <unfinished ...>
12449 <... wait4 resumed> [{WIFEXITED(s) && WEXITSTATUS(s) == 0}], 0, NULL) = 12451
```

fd	bash	echo	head
0	terminal	terminal	pipe_read
1	terminal	pipe_write	terminal
2	terminal	terminal	terminal
3			
4			

12449	— bash
12450	— echo
12451	— head

The primes (' or '') denote forked children that haven't execed

When creating two children with a pipe (e.g., `$ cmd1 | cmd2`), the pipe is created by the parent process before the first fork and ultimately closes both ends of the pipe. Why doesn't one the children create the pipe?

A. File descriptors are inherited by children so bash creates the pipe before either child so the children can communicate via the file descriptors

B. The pipes are reused so that running a second pipeline like `$ cmd3 | cmd4` doesn't require creating a new pipe. This wouldn't work if the children created the pipe

C. It doesn't matter which of the three processes (bash and the two children) creates the pipe because prior to the exec()s, all three are copies of the same program (bash) so creating the pipe in any one creates them in all three. Bash just happens to do it in the parent.

Signals

Signals are how the kernel communicates with user processes

When the kernel wants to signal the process, it checks the processes signal mask to see if the process is willing to accept receipt of the signal

If it is willing, then the action depends on the **disposition** of the signal

- If the process is ignoring the signal, it's dropped
- If the process has installed a signal handler, the handler is run
- If the process has done neither, then the default action is performed
  - either the signal is ignored by default; or
  - the process is terminated (or a small handful of other things)

If the signal is masked, then the signal remains pending until it is unmasked

Signal delivery

Signal delivery is deferred until the kernel next returns to the process

- At the completion of a system call
- The next time the process is scheduled to run

Some system calls can be interrupted, others cannot

- System calls like read(2) and write(2) can read/write less than requested when interrupted by a signal; return value reflects this
- Other calls may return `-1` and set **`errno`** to **`EINTR`** to indicate it was interrupted

Only one of each (standard) signal may be pending at a time

Common signals: signal(7)

SIGINT	— Interrupt from keyboard (ctrl-C on the terminal)
SIGQUIT	— Quit from keyboard (ctrl-\ on the terminal)
SIGILL	— Illegal instruction
SIGABRT	— Signal from abort() (or assert() which calls abort())
SIGFPE	— Floating point exception; integer divide by 0 on some systems
SIGKILL	— Kill signal, cannot be handled or ignored
SIGSEGV	— Segmentation fault
SIGPIPE	— Write to pipe with no readers
SIGTERM	— Termination signal
SIGCHLD	— Child stopped or terminated
SIGSTOP	— Suspend the process (ctrl-Z on the terminal)
SIGCONT	— Resume the process (fg or bg on terminal)
SIGWINCH	— Terminal window resized

Similar sounding signals

`SIGINT`	— Interrupt from keyboard (ctrl-C on the terminal)
`SIGQUIT`	— Quit from keyboard (ctrl-\ on the terminal)
`SIGKILL`	— Kill signal, cannot be handled or ignored
`SIGTERM`	— Termination signal
`SIGSTOP`	— Suspend the process (ctrl-Z on the terminal)

`SIGINT` and **`SIGQUIT`** should only come from the user typing at the terminal

If one process wants to stop another, it should (typically) request the process terminate via **`SIGTERM`** and, if after a few seconds it hasn't, use **`SIGKILL`**

`SIGSTOP` is about job control, not about terminating processes

Consider the following sequence of events

- The process installs a signal handler for **`SIGINT`**
- The process masks (blocks) **`SIGINT`**
- The user presses ctrl-c twice
- The process unmasks (unblocks) **`SIGINT`**

Which of the following is correct?

A. The handler never runs

B. The handler runs the first time ctrl-c is pressed

C. The handler runs both times ctrl-c is pressed

D. The handler runs once after the signal is unmasked

E. The handler runs twice after the signal is unmasked

Sending a signal

From the shell: `kill(1)` or `killall(1)`

- `$ kill -9 1234 # Send SIGKILL (signal 9) to PID 1234`
- `$ kill -l # List all of the signals`

```c
int kill(pid_t pid, int sig);
```

- Sends signal sig to process pid
- Different behavior depending on pid < 0, pid = 0, pid > 0, sig = 0, sig > 0

```c
int raise(int sig);
```

- Sends signal sig to the own process

Ignoring a signal/setting default

```
typedef void (*sighandler_t)(int);
sighandler_t signal(int signum, sighandler_t handler);
```

- Use **`SIG_IGN`** for `handler` to ignore the signal
- Use **`SIG_DFL`** for `handler` to use the default behavior
- You can also pass a function (pointer) of type `void handler(int)` but this isn't portable

Setting a handler portably

Use sigaction(2)

- Takes a `const` pointer to a struct that holds a new handler and flags
- Takes a pointer to a struct that holds the old handler and flags
- flags specify the behavior of interrupted system calls, what information is given to the signal handler, and whether the same signal can be received while its handler is running
- Read the man page!

Masking signals

Signal masks can be manipulated with `sigprocmask(2)`

Handling a signal

To handle a signal and then continue running,

- The signal handler should be installed with sigaction(2)

To handle a fatal signal and then exit as a result,

- The signal handler should be installed with sigaction(2)
- After performing any cleanup actions, the signal disposition should be reset to the default and the signal reraised

```
int handler(int sig) {
  // Clean up actions, beware of signal handler limitations
  signal(sig, SIG_DFL);
  raise(sig);
}
```

In-class exercise

https://checkoway.net/teaching/cs241/2020-spring/exercises/Lecture-28.html

Grab a laptop and a partner and try to get as much of that done as you can!