

CSC552 – Advanced UNIX Programming

Pipes

Dr. L. Frye
Kutztown University



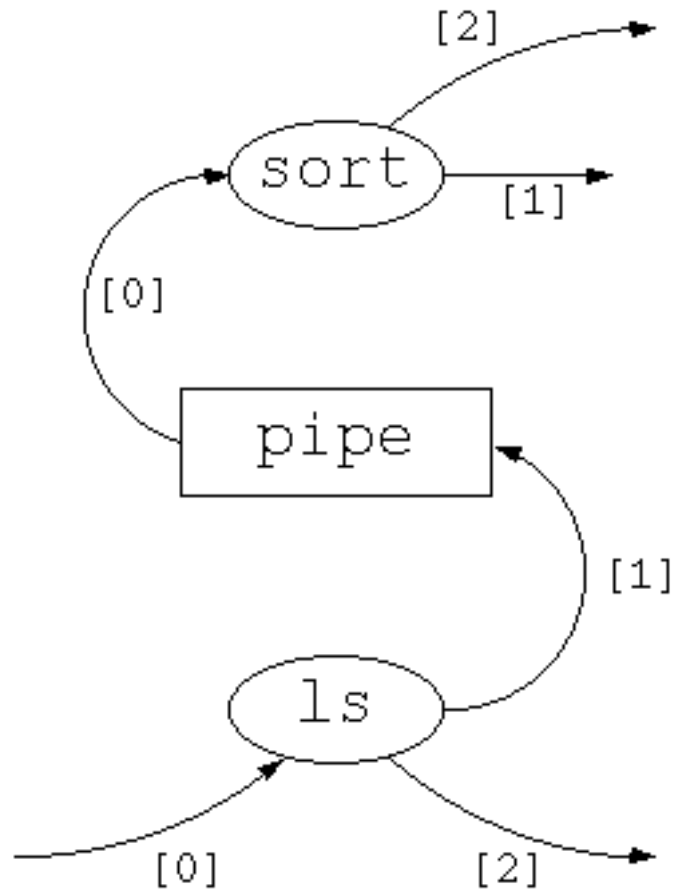
Pipes in the Shell

- ▶ `ps -ef | grep frye`
- ▶ Parent – fork
 - ▶ Child – exec
- ▶ Pipe commands
 - ▶ Need a pipe
 - ▶ Need a process (fork) for each command
 - ▶ Redirect standard out for first command to write end of pipe
 - ▶ Redirect standard in for second command to read end of pipe

Pipes

- ▶ Characteristics
 - ▶ Half-duplex
 - ▶ Common ancestor
- ▶ Types
 - ▶ Unnamed
 - ▶ Named

Pipe Shell Example



`sort`

file descriptor table

[0]	<i>pipe read</i>
[1]	<i>standard output</i>
[2]	<i>standard error</i>

`ls`

file descriptor table

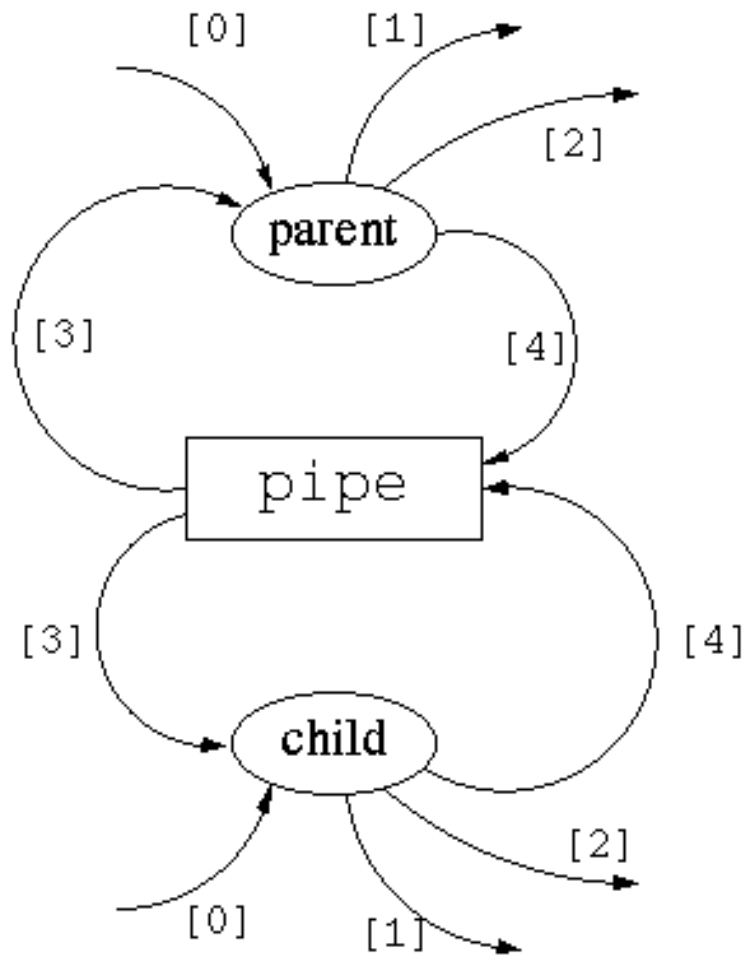
[0]	<i>standard input</i>
[1]	<i>pipe write</i>
[2]	<i>standard error</i>

Pipe Creation

- ▶ pipe()
 - Two file descriptors
 - Read
 - Write
- ▶ File descriptors after fork()
- ▶ pipes / pipeEx.c

Pipe Example

- ▶ `dup2()` function call
- ▶ `pipes/simpleredirect.c`



parent

file descriptor table

[0]	<i>standard input</i>
[1]	<i>standard output</i>
[2]	<i>standard error</i>
[3]	<i>pipe read</i>
[4]	<i>pipe write</i>

child

file descriptor table

[0]	<i>standard input</i>
[1]	<i>standard output</i>
[2]	<i>standard error</i>
[3]	<i>pipe read</i>
[4]	<i>pipe write</i>

parent

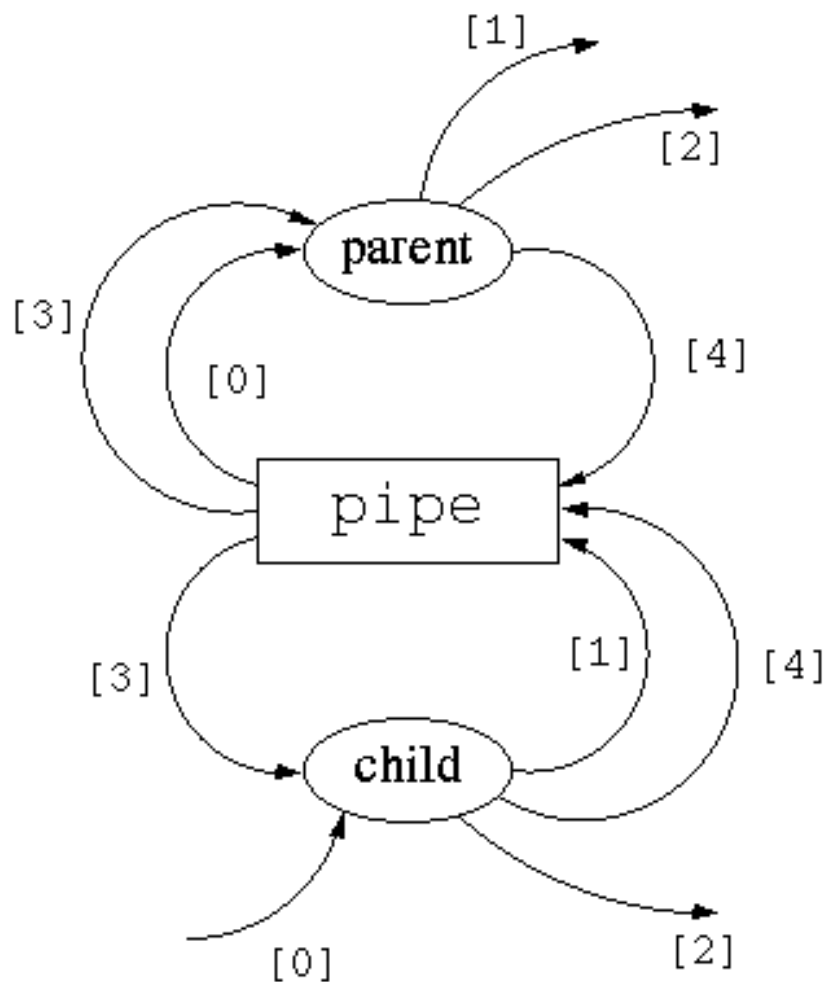
file descriptor table

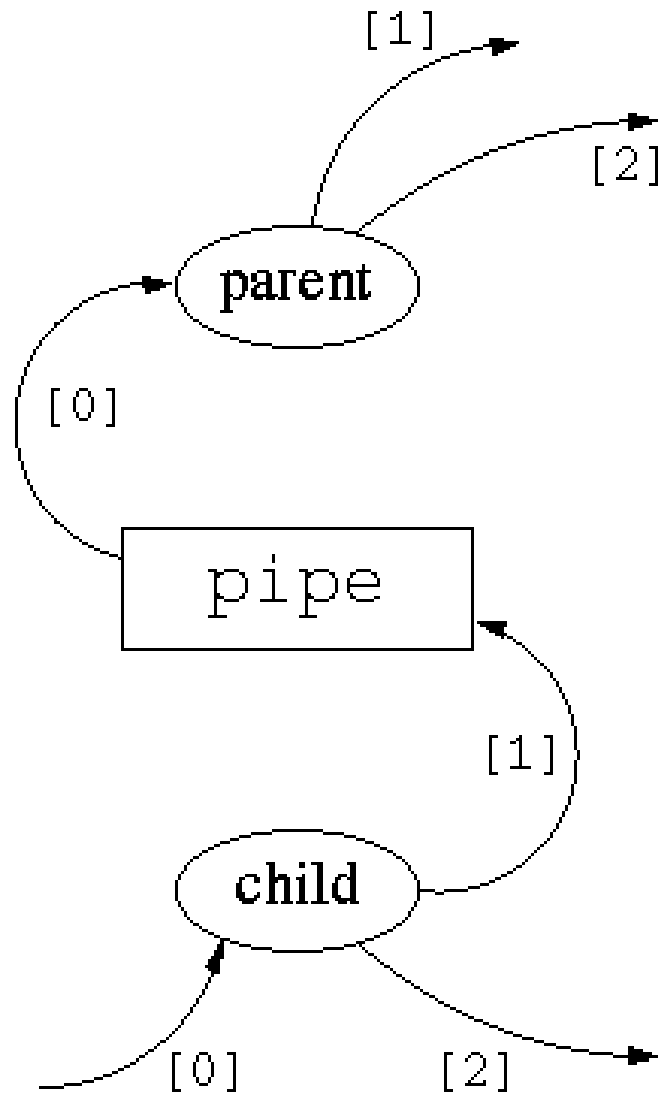
[0]	<i>pipe read</i>
[1]	<i>standard output</i>
[2]	<i>standard error</i>
[3]	<i>pipe read</i>
[4]	<i>pipe write</i>

child

file descriptor table

[0]	<i>standard input</i>
[1]	<i>pipe write</i>
[2]	<i>standard error</i>
[3]	<i>pipe read</i>
[4]	<i>pipe write</i>





parent

file descriptor table

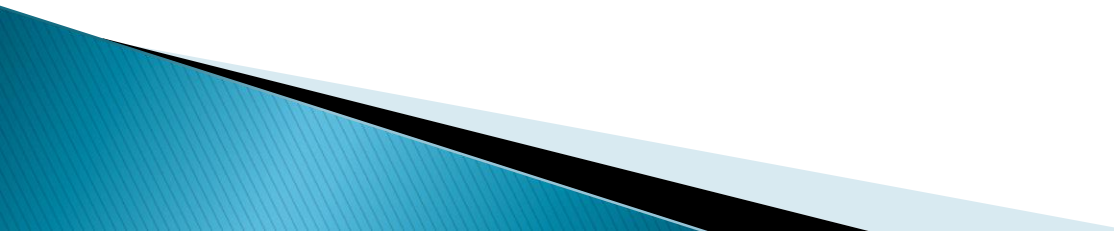
[0]	<i>pipe read</i>
[1]	<i>standard output</i>
[2]	<i>standard error</i>

child

file descriptor table

[0]	<i>standard input</i>
[1]	<i>pipe write</i>
[2]	<i>standard error</i>

Pipe Usage

- ▶ read
 - ▶ write
 - ▶ Protocol for reading and writing
 - ▶ close()
- 

Reading and Writing

- ▶ Finite size
- ▶ Read
 - Blocks on empty pipe
 - Otherwise, returns immediately
 - Returns 0 on EOF
- ▶ Write
 - Blocks on full pipe
 - Fails if read end not open (SIGPIPE)

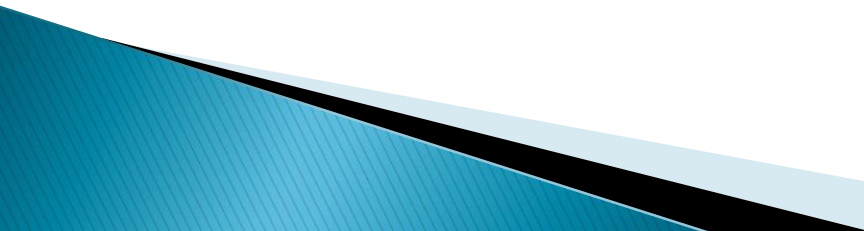
Pipe Synchronization

- ▶ What must be done if a pipe is used for two-way communication?
- ▶ `pipes/synchronizefan.c`

Named Pipes

- ▶ FIFO
- ▶ Advantages
 - Exist in filesystem
 - Unrelated processes
 - Explicitly delete
- ▶ Create
 - mknod command
 - mknod() function
 - mkfifo() function
- ▶ pipes / fifoEx.c

Reading and Writing

- ▶ File functions
 - ▶ Must open for read and write
 - ▶ `O_NONBLOCK`
 - Non-blocking open
 - Read returns immediately
 - Write returns error
 - ▶ `fstat()` or `fcntl()` functions
- 

Blocking vs Non-Blocking

Current Operation	Existing opens of pipe of FIFO	Return	
		Blocking (default)	O_NONBLOCK set
open FIFO read-only	FIFO open for writing	returns OK	returns OK
	FIFO not open for writing	blocks until FIFO is opened for writing	returns OK
open FIFO write-only	FIFO open for reading	returns OK	returns OK
	FIFO not open for reading	blocks until FIFO is opened for reading	returns an error of ENXIO
read empty pipe or empty FIFO	pipe of FIFO not open for writing	blocks until data is in the pipe of FIFO, or until the pipe or FIFO is no longer open for writing	returns an error of EAGAIN
	pipe of FIFO not open for writing	read returns 0 (end-of-file)	read returns 0 (end-of-file)
write to pipe or FIFO	pipe of FIFO open for reading	depends on number of bytes in pipe and room available	depends on number of bytes in pipe and room available
	pipe or FIFO not open for reading	SIGPIPE signal generated	SIGPIPE signal generate

Pipe Examples

- ▶ Reader Writer Example
 - pipes/reader.c
 - pipes/writer.c
- ▶ Client–Server Communication
 - Simple–request
 - Request–reply
 - Example
 - pipes/pipeserver.c (page 197)
 - pipes/pipeclient.c (page 198)

Shell Example Flow

- ▶ `ps -ef | grep frye`
- ▶ Draw a flowchart for this