

# Pipes

CPSC 328 - Network Programming

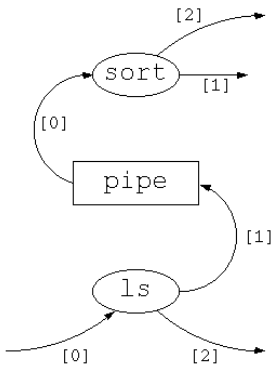
# Pipes

- Interprocess Communication (IPC)
- Characteristics
  - Half-duplex (data flows in one direction)
  - Common ancestor
- Types
  - Unnamed
  - Named

# Pipes in the shell

- Vertical bar (|): connect the standard out of one process to the standard in of another process
- Example: `ls | sort`
- 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

# Pipe 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

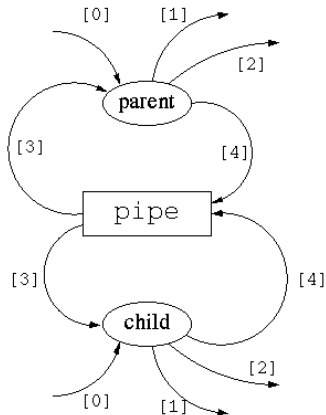
- `pipe()` takes two file descriptors: read and write
- File descriptors after `fork()`
- Example:

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

# Pipe Example

```
int fd[2];
pipe(fd);
pid_t child_pid = fork()
switch (child_pid) {
    case 0:
        dup2(fd[1], STDOUT_FILENO);
        close(fd[0]); // close read end
    default:
        dup2(fd[0], STDIN_FILENO);
        close(fd[1]); // close write end
}
```

# Pipe Example



parent

file descriptor table

[0] *standard input*

[1] *standard output*

[2] *standard error*

[3] pipe read

[4] pipe write

child

file descriptor table

[0] *standard input*

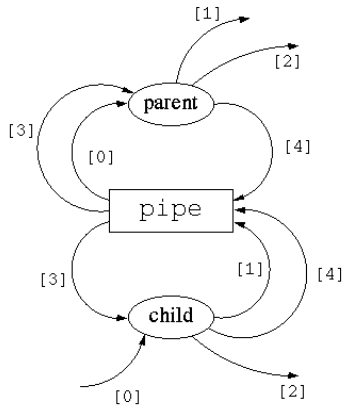
[1] *standard output*

[2] *standard error*

[3] pipe read

[4] pipe write

# Pipe Example



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>

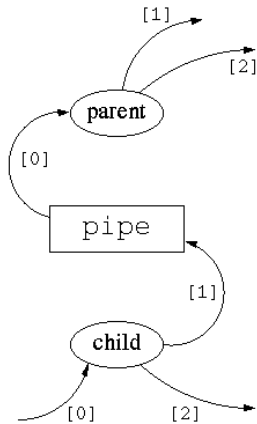
child

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>



# Pipe Example



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
- close
- Need a protocol for reading and writing

# 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

# Pipe Synchronization

- For bidirectional communication we need two pipes
- We need to create a synchronization point or barrier