Interprocess Communication: Pipes and FIFOs

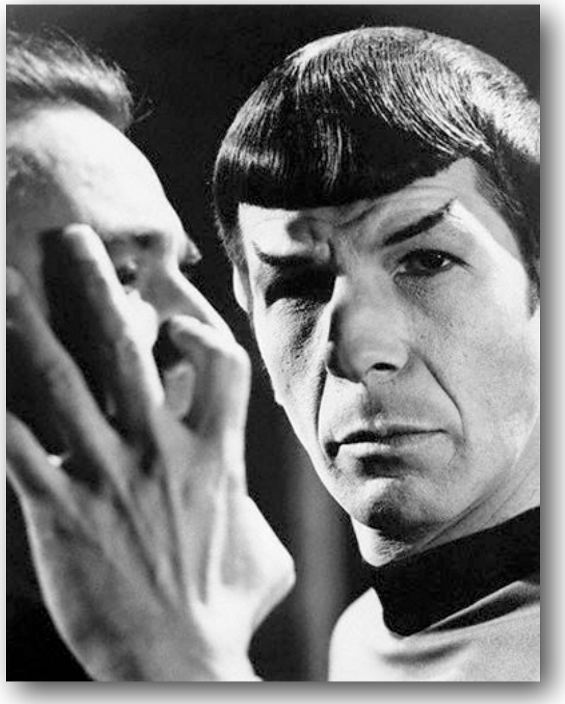
CS 241

April 7, 2014

University of Illinois

Two kinds of IPC

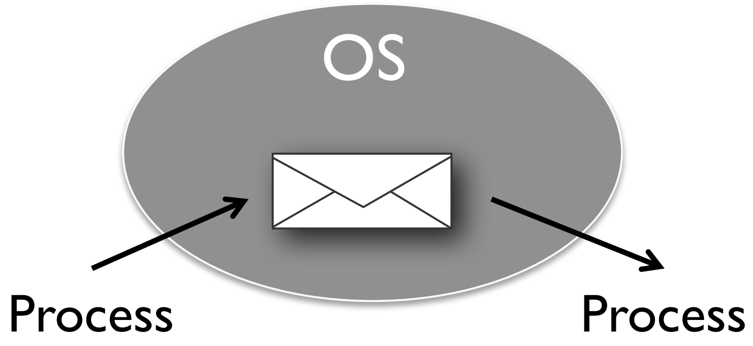
"Mind meld"



Shared address space

- Shared memory
- Memory mapped files

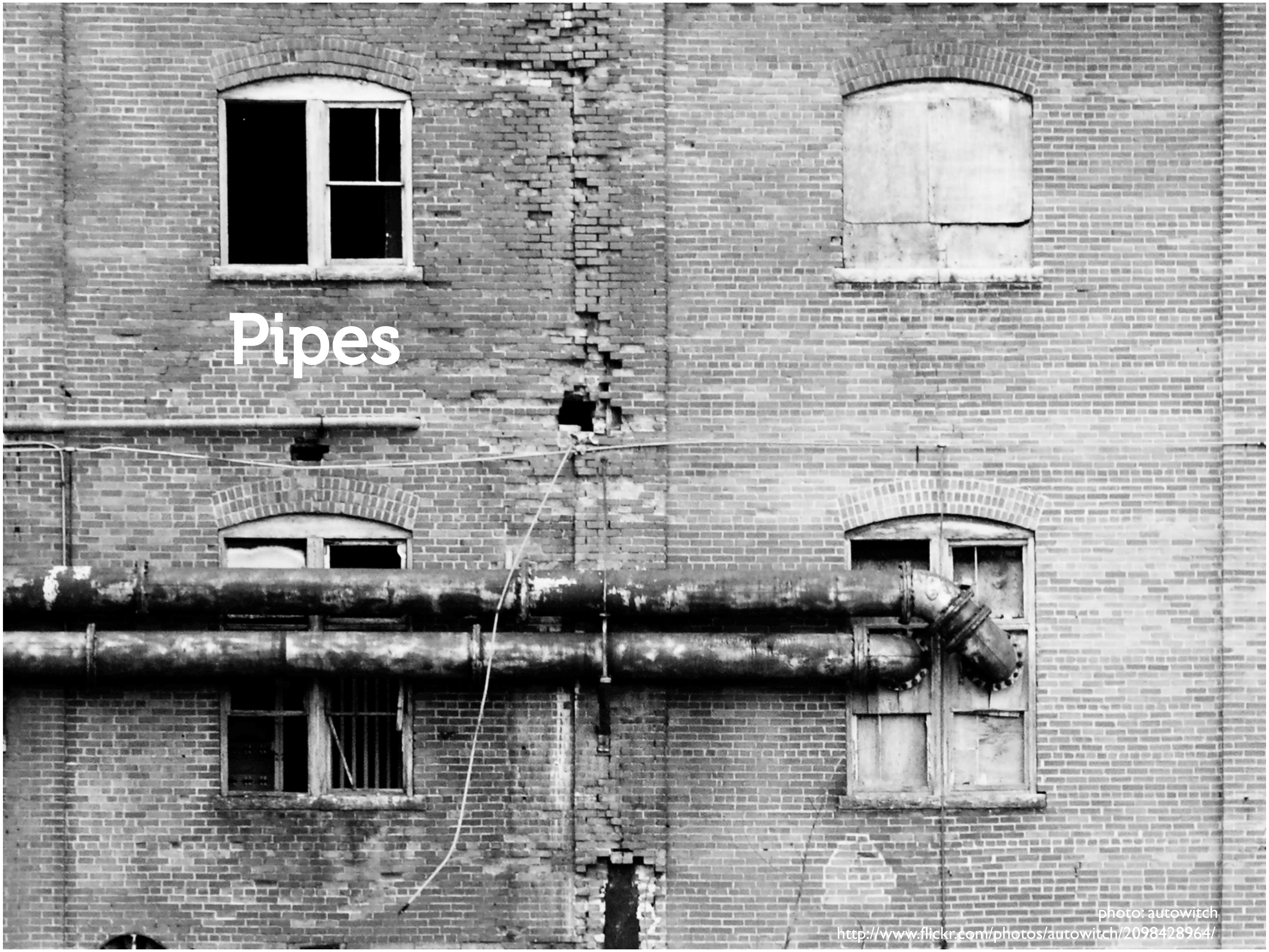
"Intermediary"



Message transported by OS from one address space to another

- Files
- Pipes
- FIFOs

Today



Pipes

photo: autowitch
http://www.flickr.com/photos/autowitch/2098428964/

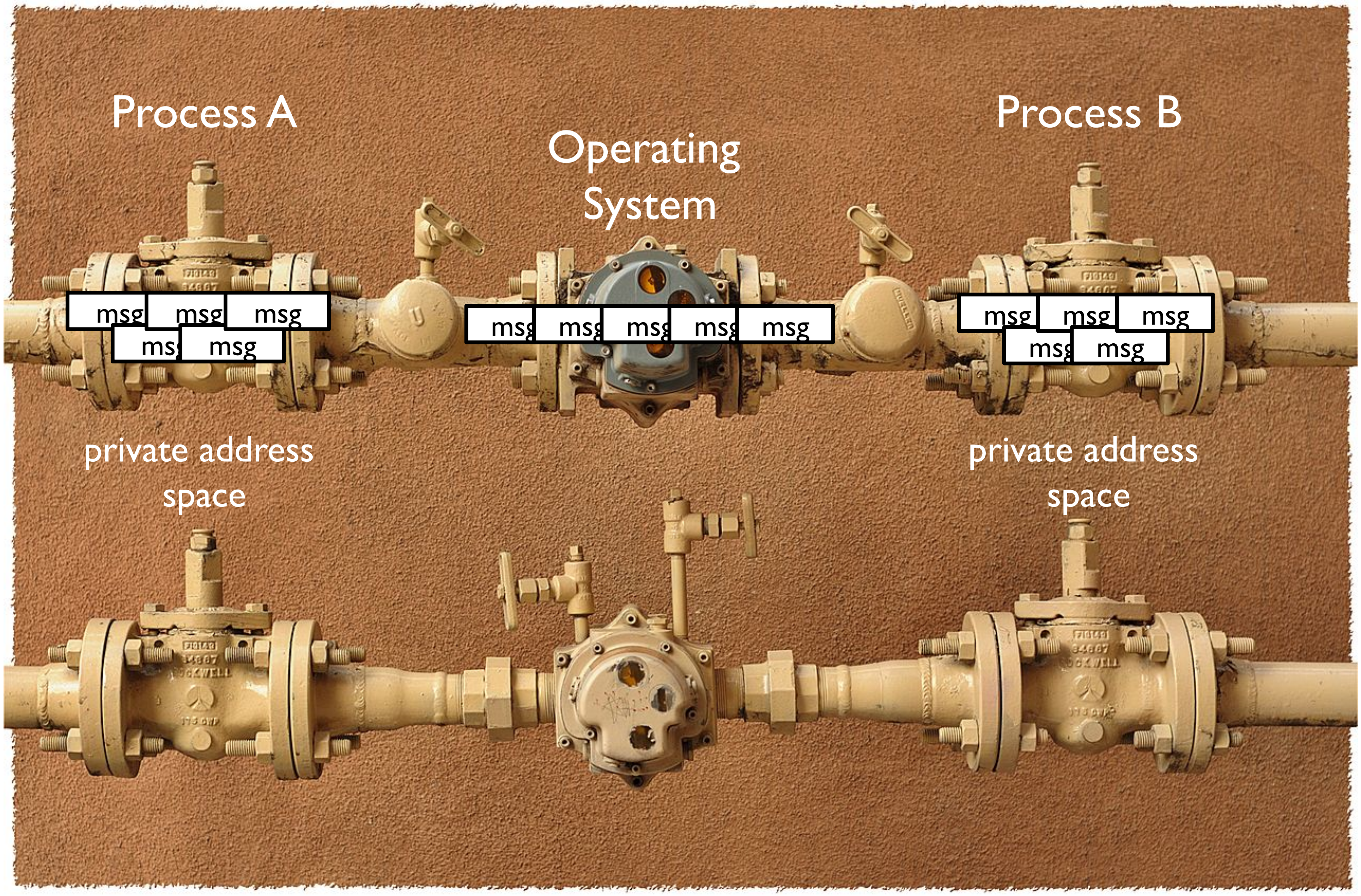


photo: theilr
http://www.flickr.com/photos/theilr/4283377543/

Pipe example

```
int main(void) {
    int pfds[2];
    char buf[30];

    pipe(pfds);

    if (!fork()) {
        printf(" CHILD: writing to pipe\n");
        write(pfds[1], "test", 5);
        printf(" CHILD: exiting\n");

    } else {
        printf("PARENT: reading from pipe\n");
        read(pfds[0], buf, 5);
        printf("PARENT: read \"%s\"\n", buf);
        wait(NULL);
    }

    return 0;
}
```

`pfds[0]`: read end of pipe
`pfds[1]`: write end of pipe

A pipe dream

```
ls | wc -l
```

Can we implement a command-line pipe with `pipe()`?

How do we attach the stdout of `ls` to the stdin of `wc`?

Duplicating a file descriptor

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

```
int dup(int oldfd);
```

Create a copy of an open file descriptor

Put new copy in first unused file descriptor

Returns:

- Return value ≥ 0 : Success. Returns new file descriptor
- Return value = -1: Error. Check value of `errno`

Parameters:

- `oldfd`: the open file descriptor to be duplicated

Duplicating a file descriptor

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

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

Create a copy of an open file descriptor

Put new copy in specified location

- ...after closing **`newfd`**, if it was open

Returns:

- Return value ≥ 0 : Success. Returns new file descriptor
- Return value = -1: Error. Check value of **`errno`**

Parameters:

- **`oldfd`**: the open file descriptor to be duplicated

A pipe dream

```
ls | wc -l
```

Can we implement a command-line pipe with `pipe()`?

How do we attach the stdout of `ls` to the stdin of `wc`?

Wait, what does this even mean?

A pipe dream

$$\begin{tikzcd}
\texttt{stdin} \arrow[r] & \texttt{ls} \arrow[r] & \texttt{wc -l} \arrow[r] & \texttt{stdout}
\end{tikzcd}$$

Can we implement a command-line pipe with `pipe()`?

How do we attach the stdout of `ls` to the stdin of `wc`?

Pipe dream: ls | wc –l

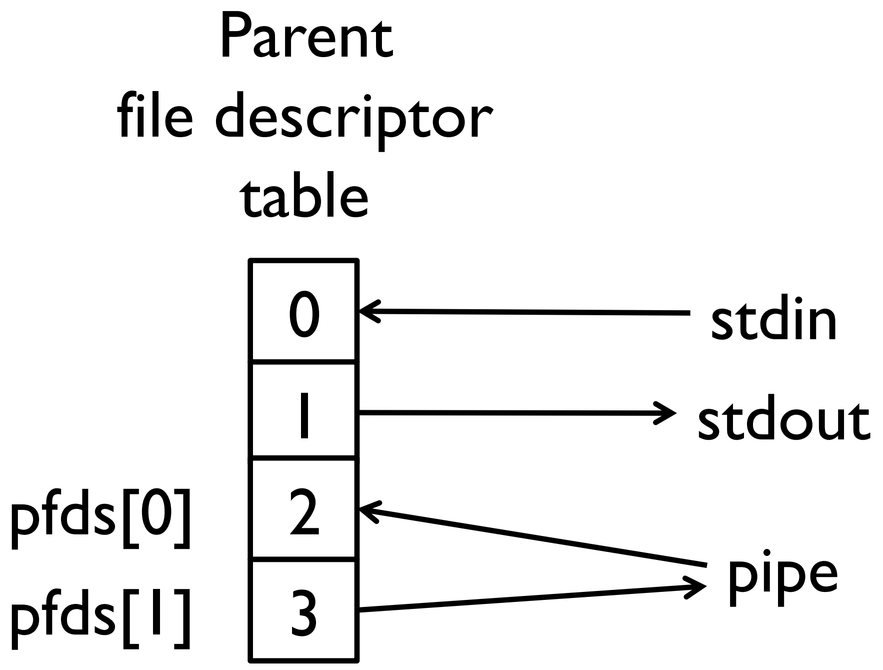
```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main(void) {
    int pfds[2];

    pipe(pfds);

    if (!fork()) {

        ???

    } else {

        ???

    }
    return 0;
}
```

Pipe dream come true: ls | wc –l

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main(void) {
    int pfds[2];

    pipe(pfds);

    if (!fork()) {
        close(1);       /* close stdout */
        dup(pfds[1]);   /* make stdout pfds[1] */
        close(pfds[0]); /* don't need this */
        execlp("ls", "ls", NULL);
    } else {
        close(0);       /* close stdin */
        dup(pfds[0]);   /* make stdin pfds[0] */
        close(pfds[1]); /* don't need this */
        execlp("wc", "wc", "-l", NULL);
    }
    return 0;
}
```

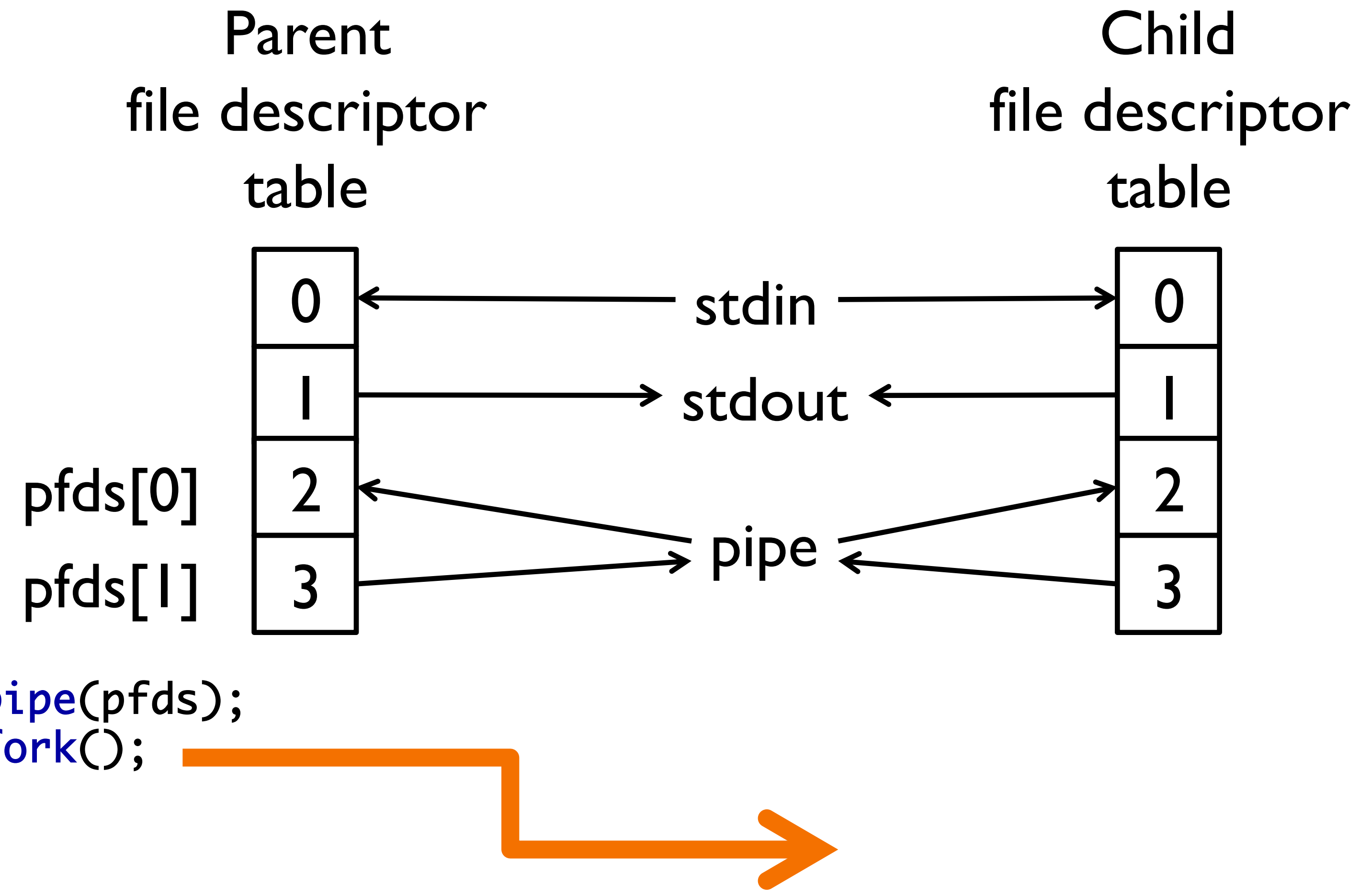
Run demo

Pipe dream come true: ls | wc –l



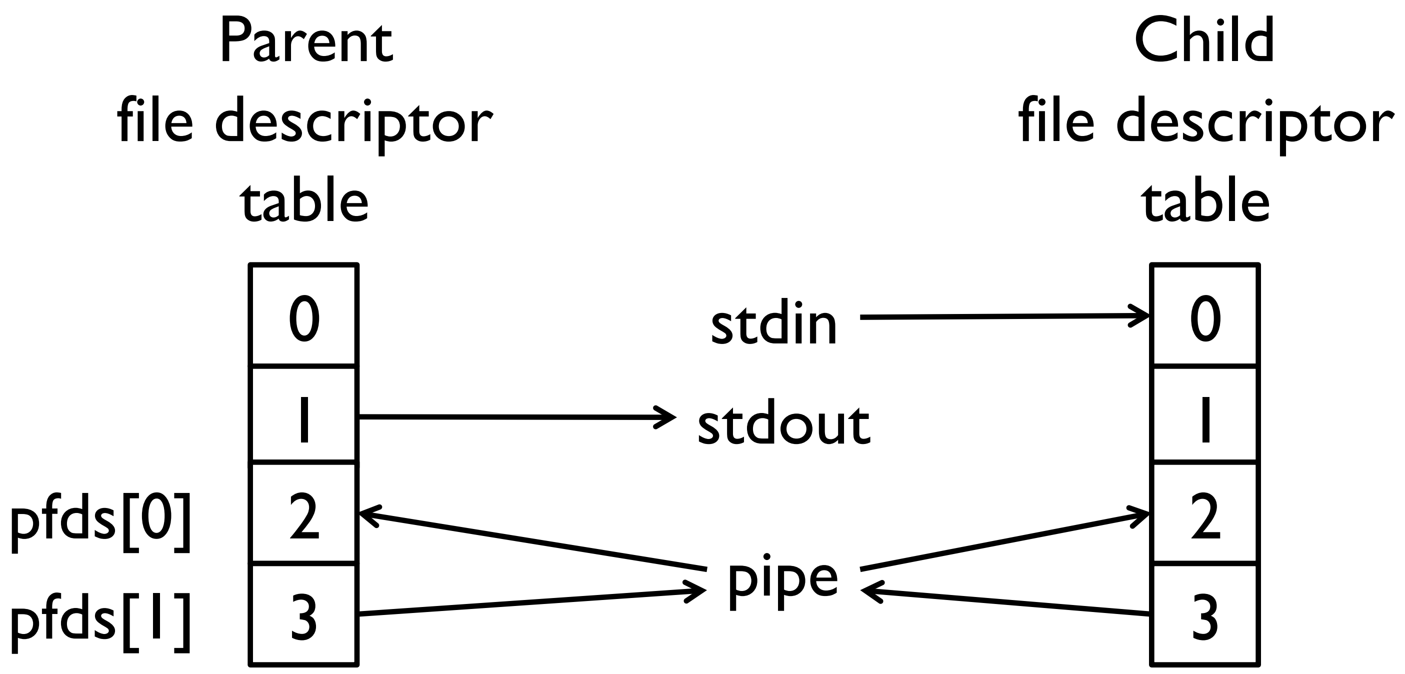
```
pipe(pfds);
```

Pipe dream come true: ls | wc –l



```
pipe(pfds);
fork();
```

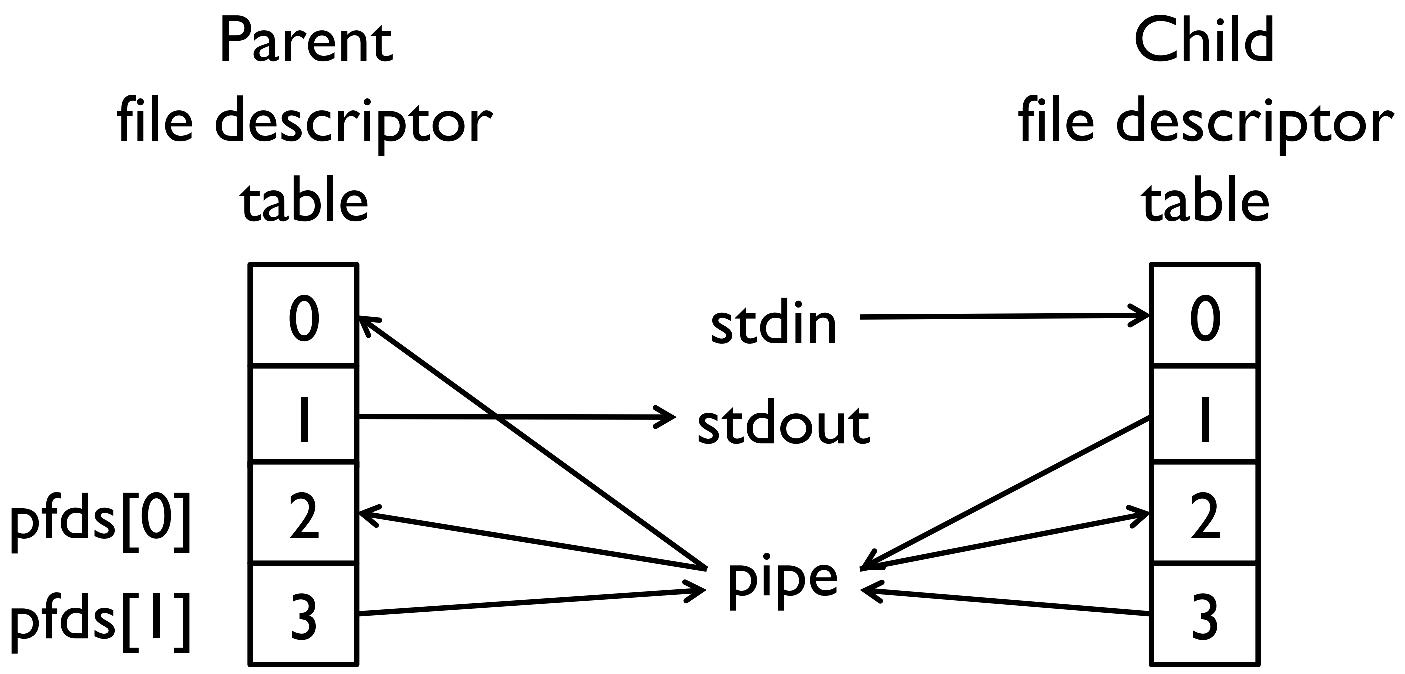
Pipe dream come true: ls | wc –l



```
pipe(pfds);
fork();

close(0);
```

```
close(1);
```

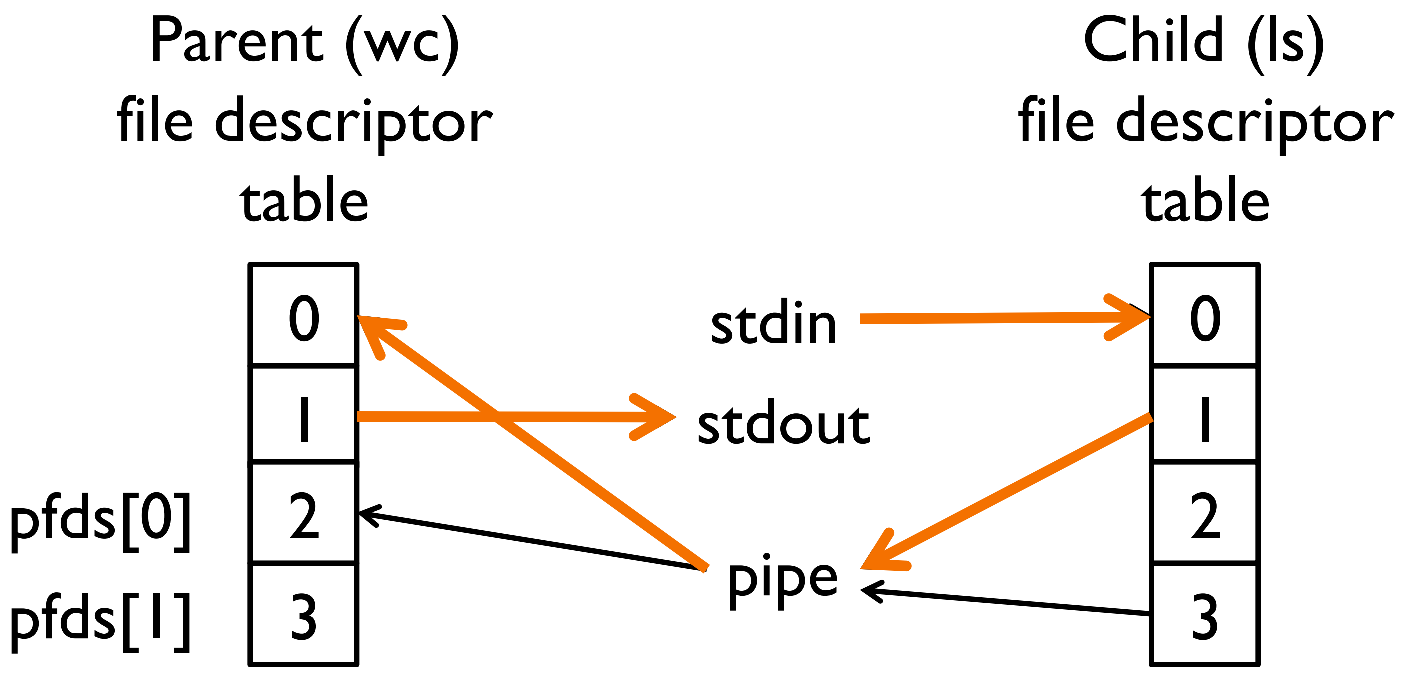
Pipe dream come true: ls | wc –l



```
pipe(pfds);
fork();

close(0);
dup(pfds[0]);
```

```
close(1);
dup(pfds[1]);
```

Pipe dream come true: ls | wc –l



```
pipe(pfds);
fork();

close(0);
dup(pfds[0]);
close(pfds[1]);
execlp("wc", "wc", "-l", NULL);
```

```
close(1);
dup(pfds[1]);
close(pfds[0]);
execlp("ls", "ls", NULL);
```

FIFOs

FIFOs

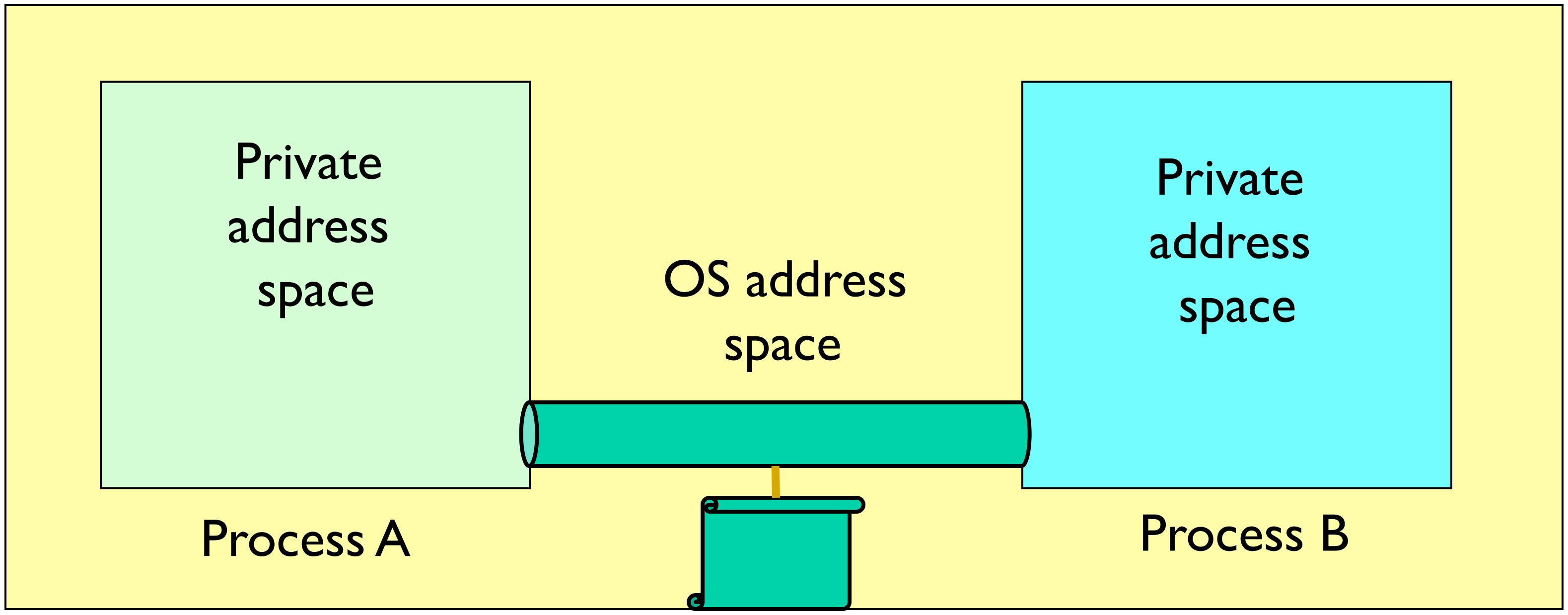
A pipe disappears when no process has it open

FIFOs = named pipes

- Special pipes that persist even after all the processes have closed them
- Actually implemented as a file

```
#include <sys/types.h>
#include <sys/stat.h>

int status;
...
status = mkfifo("/home/cnd/mod_done", /* mode=0644: */
              S_IWUSR | S_IRUSR | S_IRGRP | S_IROTH);
```

Communication Over a FIFO



First **open** blocks until second process opens the FIFO

Can use **O_NONBLOCK** flag to make operations non-blocking

FIFO is persistent : can be used multiple times

Like pipes, OS ensures atomicity of writes and reads

FIFO Example: Producer-Consumer

Producer

- Writes to FIFO

Consumer

- Reads from FIFO
- Outputs data to file

FIFO ensures atomicity of write

FIFO Example

```c
#include <errno.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/stat.h>
#include "restart.h"

int main (int argc, char *argv[]) {
  int requestfd;

  if (argc != 2) { /* name of consumer fifo on the command line */
    fprintf(stderr, "Usage: %s fifoname > logfile\n", argv[0]);
    return 1;
  }
```

FIFO Example

```
/* create a named pipe to handle incoming requests */
 if ((mkfifo(argv[1], S_IRWXU | S_IWGRP| S_IWOTH) == -1)
     && (errno != EEXIST))
 {
   perror("Server failed to create a FIFO");
   return 1;
 }
```

```
  /* open a read/write communication endpoint to the pipe */
 if ((requestfd = open(argv[1], O_RDWR)) == -1) {
   perror("Server failed to open its FIFO");
   return 1;
 }
 /* Write to pipe like you would to a file */
 ...
}
```

What if there are multiple producers?

Examples

- Multiple children to compute in parallel; wait for output from any
- Network server connected to many clients; take action as soon as any one of them sends data

Problem

- Can use read / write scanf, but ..... problem?
- Blocks waiting for that one file, even if another has data ready & waiting!

Solution

- Need a way to wait for any one of a set of events to happen
- Something similar to wait() to wait for any child to finish, but for events on file descriptors

Key points to remember

Pipes and FIFOs enable IPC through messaging

- “unnamed” (Pipes) or “named” (FIFOs)

OS takes care of synchronization for you!

- Assuming one process writes, and one process reads
- No need to worry about when you read or write, even though behind the scenes there’s a shared data structure

FIFOs use the filesystem interface for a nontraditional file

Next: need a way to receive notifications of events on pipes/ FIFOs