Unix Special Files & IPC

SOA, Deemed to be University
ITER, Bhubanewar

Book(s)

Text Book(s)

Kay A. Robbins, & Steve Robbins

Unix^{TM} Systems Programming

Communications, concurrency, and Treads

Pearson Education

Reference Book(s)

Brain W. Kernighan, & Rob Pike

The Unix Programming Environment

PHI

Introduction

- ☞ Two other important examples of special files are pipes and FIFOs
- ☞ Pipes and FIFOS are the interprocess communication mechanisms that allow processes running on the same system to share information and hence cooperate.

Pipes

```
#include <unistd.h>

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

```
Returns:

(1) If successful, pipe returns 0.

(2) If unsuccessful, pipe returns -1 and sets errno.
```

☞ The simplest UNIX interprocess communication mechanism is the pipe, which is represented by a special file.

☞ The `pipe` function creates a communication buffer that the caller can access through the file descriptors `fd[0]` and `fd[1]`.

☞ The data written to `fd[1]` can be read from `fd[0]` on a first-in-first-out basis.

About `pipe()`

- ☞ A pipe has no external or permanent name, so a program can access it only through its two descriptors.
- ☞ For this reason, a pipe can be used only by the process that created it and by descendants that inherit the descriptors on **`fork`**.
- ☞ The pipe function described here creates a unidirectional communication buffer.

The following code segment creates a pipe.

```
   int fd[2];

   if (pipe(fd) == -1)
      perror("Failed to create the pipe");
```

If the **`pipe`** call executes successfully, the process can read from **`fd[0]`** and write to **`fd[1]`**.

`pipe()` Returns

- When a process calls **`read`** on a pipe, the **`read`** returns immediately if the pipe is not **empty**.

- If the pipe is empty, the **`read`** blocks until something is written to the pipe, as long as some process has the pipe open for writing.

- On the other hand, if no process has the pipe open for writing, a **`read`** from an empty pipe returns 0, indicating an end-of-file condition

Pipelines

- ☞ Pipeline describes how to use redirection with pipes to connect processes together.
- ☞ A process can redirect standard input or output to a file. The following commands use the sort filter in conjunction with `ls` to output a directory listing **sorted by size**.

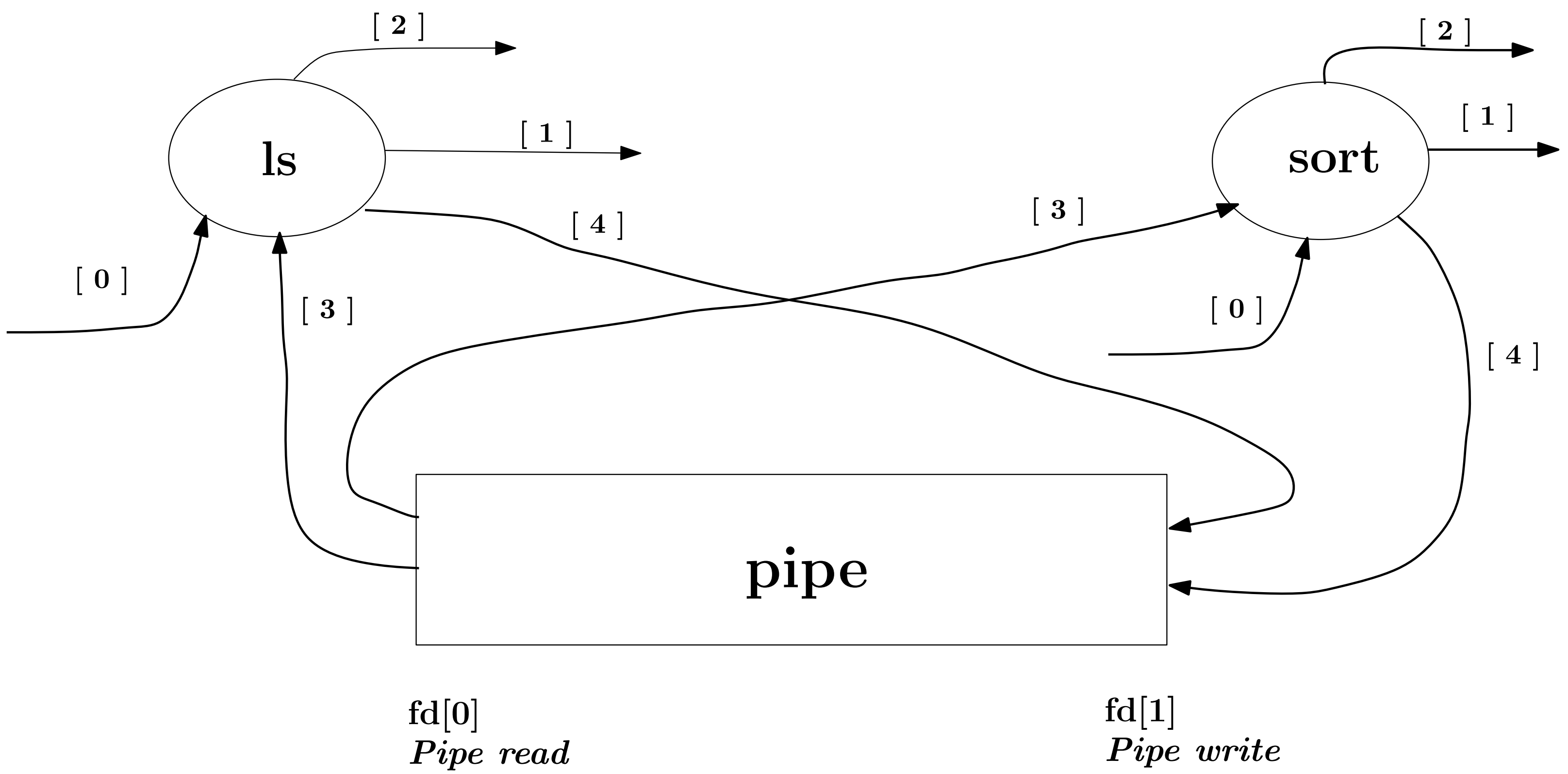
```
   ls -l > temp
sort -n  < temp
```

- ☞ An alternative approach for outputting a sorted directory listing is to use an interprocess communication (IPC) mechanism such as a `pipe` to send information directly from the `ls` process to the sort process.

```
   ls -l |  sort -n
```

- ☞ A programmer can build complicated transformations from simple filters by feeding the standard output of one filter into the standard input of the other filter through an intermediate pipe. The pipe acts as a buffer between the processes,

Redirection Usages: `ls -l | sort -n`



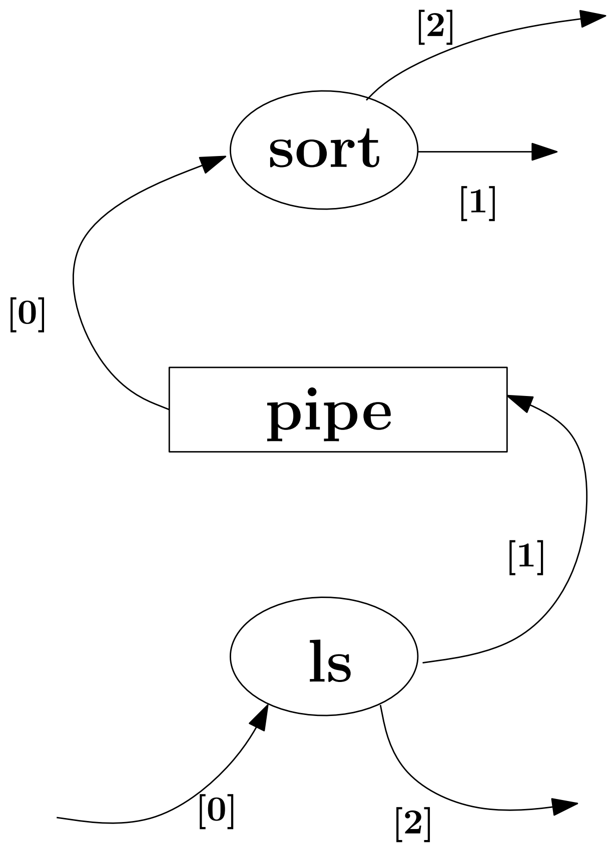
ls

	file descriptor table
[0]	standard *input*
[1]	standard *output*
[2]	standard *error*
[3]	pipe *read*
[4]	pipe *write*

sort

	file descriptor table
[0]	standard *input*
[1]	standard *output*
[2]	standard *error*
[3]	pipe *read*
[4]	pipe *write*

Execution: `ls -l | sort -n`



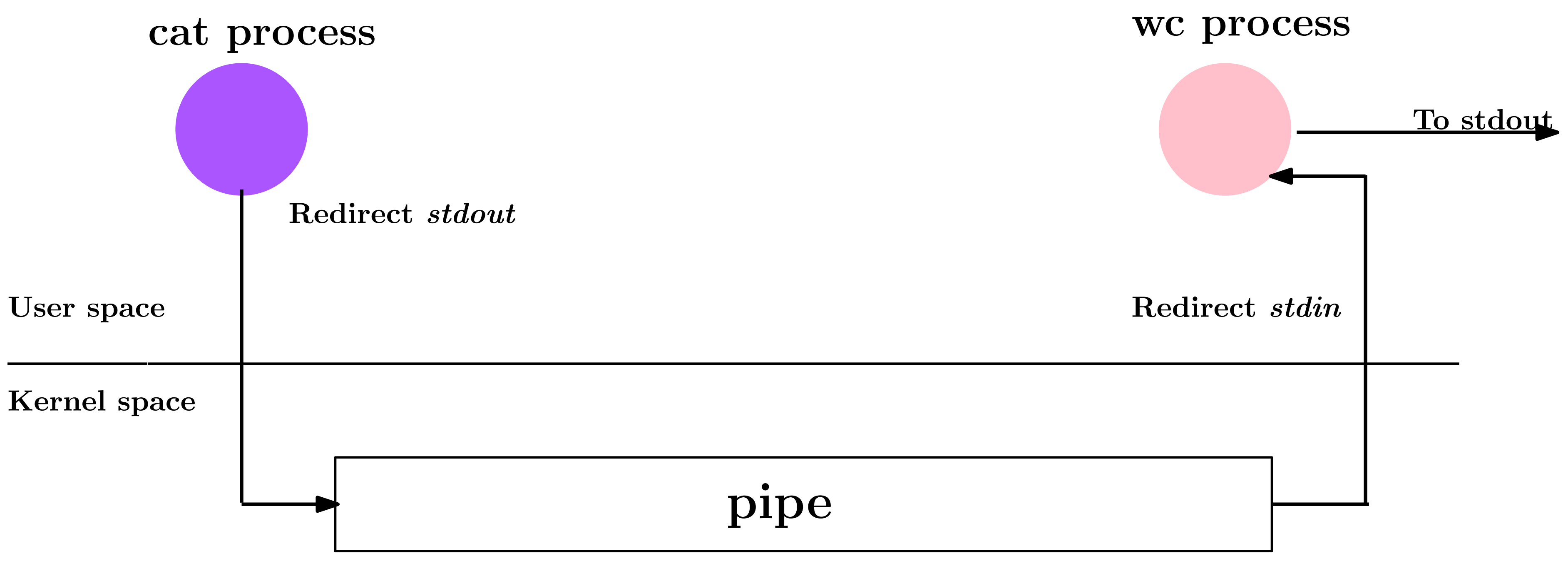
sort

	file descriptor table
[0]	`pipe` *read*
[1]	`standard` *output*
[2]	`standard` *error*

ls

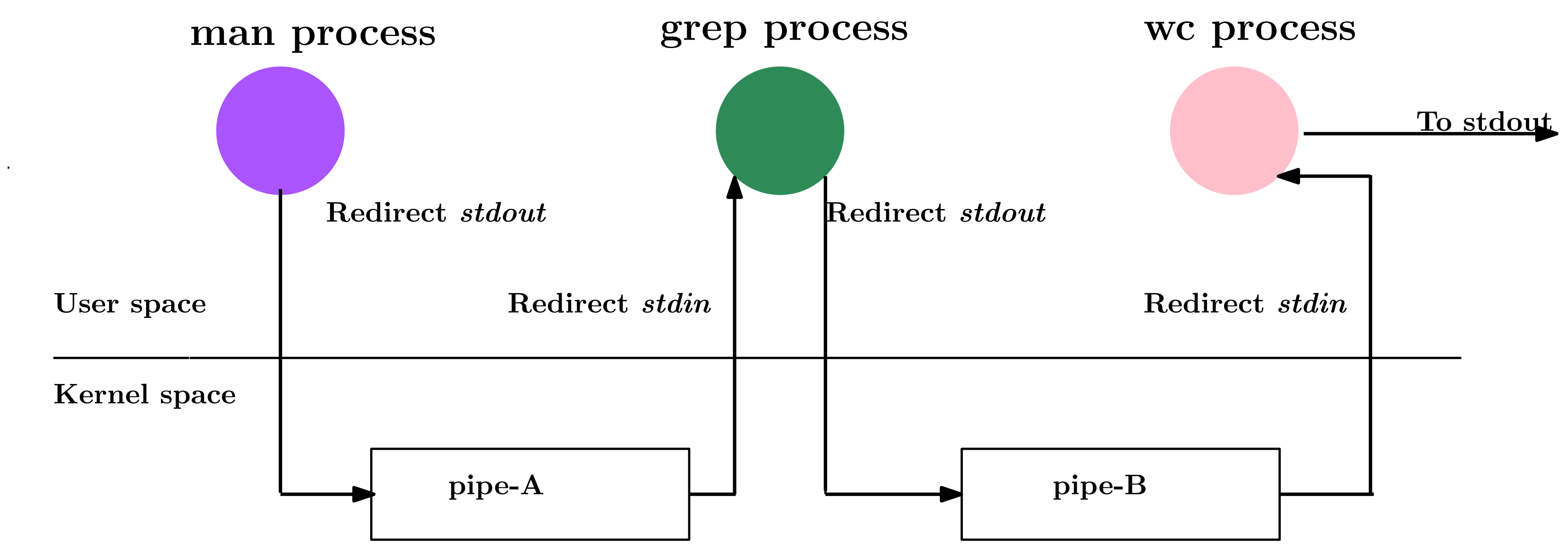
	file descriptor table
[0]	`standard` *input*
[1]	`pipe` *write*
[2]	`standard` *error*

Pipeline: `cat test.txt | wc -l`



```
cat test.txt | wc -l
```

Pipeline: `man ls | grep ls | wc -l`



```
man ls | grep ls | wc -l
```

Pipe Limitations

Pipes are the oldest form of UNIX System IPC and are provided by all UNIX systems. Pipes have limitations.

- ✍ Half duplex (i.e., data flows in only one direction).
- ✍ Pipe can be used only between processes that have a common ancestor.
- ✍ Normally, a pipe is created by a process, that process calls **`fork`**, and the pipe is used between the parent and the child.

FIFOs get around the related and unrelated processes.