CS 361: Computer Systems **Lab Session 6 Worksheet**
Fall 2019

1 Unnamed Piping in Unix

1.1 Definition and Usage

1. A pipe is a form of redirection - transfer of standard output to some other destination, that is used in Linux and other Unix-like operating systems.

2. A pipe is used to combine two or more commands, the output of one command acts as input to another command, and this command's output may act as input to the next command and so on. It can also be visualized as a temporary connection between two or more commands.

1.2 Illustration

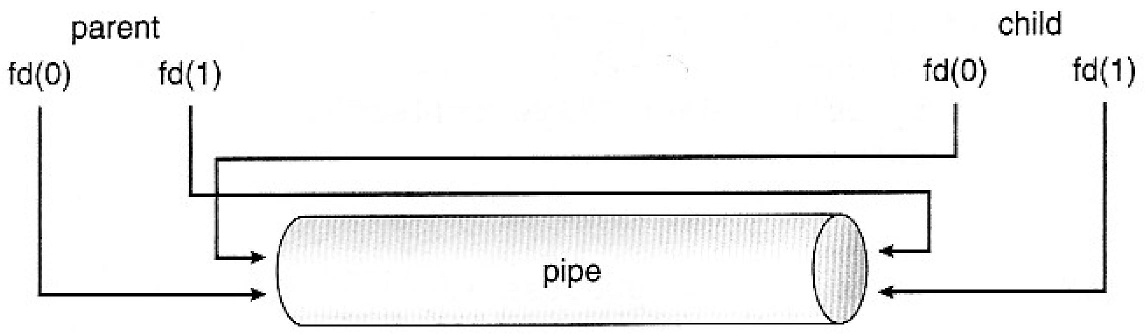


Figure 1: Pipe.

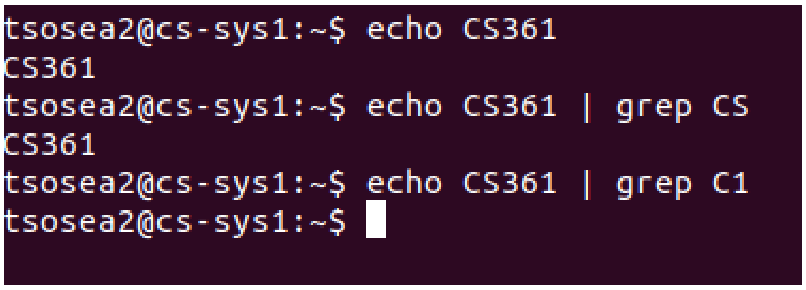
1.3 Command Line Syntax

A pipe can be used in command line the following way:

```
C1 | C2 | ... | CN
```

This means the output of C1 will be the input of C2, the output of C2 will be the input of C3 and so on.

Example:



```
tsosea2@cs-sys1:~$ echo CS361
CS361
tsosea2@cs-sys1:~$ echo CS361 | grep CS
CS361
tsosea2@cs-sys1:~$ echo CS361 | grep C1
tsosea2@cs-sys1:~$ 
```

Figure 2: Example 1.

Please answer the first two question on Gradescope.

1.4 Using pipes in C

What are pipes exactly?

Every program you run automatically has 3 data streams:

- STDIN (0)
- STDOUT (1)
- STDERR (2)

Piping and redirection is the means by which we may connect these streams between programs and files to direct data in interesting and useful ways. *pipe()* is a system call that takes a single argument, an array of 2 integers, and if the call succeeds, this array will contain **two new file descriptors** to be used for piping. As you know, the forking child inherits a copy of a parent's file descriptors. Now, that both the child and the parent have a '*file*' in common, they can communicate with eachother through that '*file*'. How will they do this? One process will write to the pipe, the other will read from it.

From the manual:

```
#include <unistd.h>
int pipe(int pipefd[2]);
```

- pipe() creates a pipe, a unidirectional data channel that can be used for interprocess communication
- the array pipefd is used to return two file descriptors referring to the ends of the pipe
- pipefd[0] refers to the read end of the pipe
- pipefd[1] refers to the write end of the pipe
- data written to the write end of the pipe is buffered by the kernel until it is read from the read end of the pipe.

Very Important

If the parent wants to receive data from the child, it **should close fd1**, and the child **should close fd0**. If the parent wants to send data to the child, it **should close fd0**, and the child **should close fd1**. Since descriptors are shared between the parent and child, we should always be sure to close the end of the pipe we aren't concerned with.

Why?

The EOF will never be returned if the unnecessary ends of the pipe are not explicitly closed.

Compile and run lab6.c, **make sure you fully understand what is happening**, then answer the last questions on gradescope. The practice code can be found *here*.