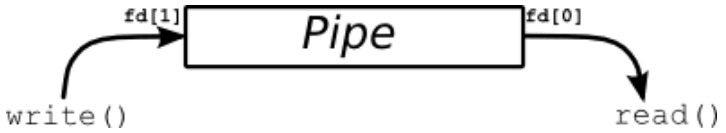
BITS, Pilani - Hyderabad Campus
Operating Systems (CS F372)

Tutorial - 4

The objective in this tutorial is to learn about Unix pipes that is used by bash shell to run multiple commands in a pipleline.

Pipes:

What are pipes and why are they used? Ever used a command similar to the following:
$ `grep string file | wc -l`
In the above program, you are searching for the lines containing a string in file, and counting them. That vertical bar there is *pipe*, and we will learn how to use them.



A call to pipe() returns a pair of file descriptors. A basic example to create and test file descriptors is as follows :

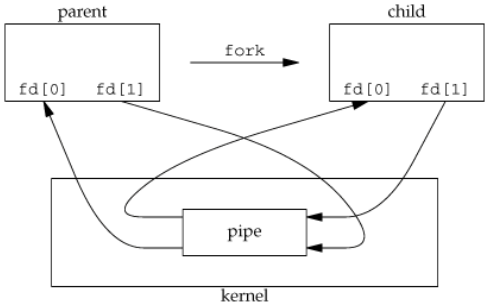
```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <unistd.h>
int main()
{
    int fds[2];
    char buf[30];
    if (pipe(fds) == -1) {
        perror("pipe");
        exit(1);
    }
    printf("writing to file descriptor #%d\n", pfds[1]);
    write(fds[1], "test", 5);
    printf("reading from file descriptor #%d\n", pfds[0]);
    read(fds[0], buf, 5);
    printf("read \"%s\"\n", buf);
    return 0;
}
```

Now let us see how a parent and its child can communicate with each other via a pipe.

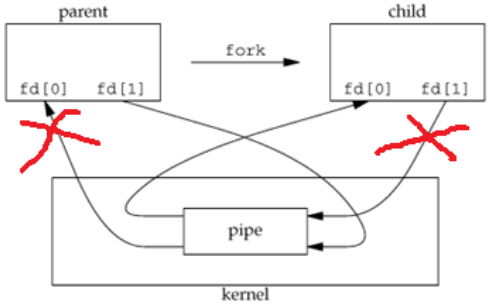
```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <sys/types.h>
#include <unistd.h>
int main()
{
    int fds[2];
    char buf[30];
    pipe(fds);
    if (fork()) {
        close(fds[0]); /*always close the unwanted ends of pipe */
        printf(" Parent: writing to the pipe\n");
        write(fds[1], "test", 5);
        wait(NULL); //discard child status
    } else { //assuming the fork() call is successful
```

```
        close[fds[1]);
        printf("Child: reading from pipe\n");
        read(fds[0], buf, 5);
        printf("Child: read \"%s\"\n", buf);
        printf(" CHILD: exiting\n");
        exit(0);
    }
    return 0;
}
```

After the call to fork() following pipe(), both the parent and child share the pipe as shown in the figure below.



In the program above, we closed the unwanted ends i.e. we did the following:



Write code to implement the following pipeline:

ls | wc –l

You need to use **dup()** or **dup2()** (dup2 is preferred) system call that are used to duplicate a file descriptor. Read man page on how to use them properly. Use the following code fragment as a part of the above program:

```
    pid = fork();
    if (pid == 0) {
        dup2(pfd[1], 1);
        close(pfd[0]);
        close(pfd[1]);
        if (execlp("ls", "ls", NULL) == -1) /* first child runs "ls" */
            perror("execlp ls");
    } else {
        if (fork() == 0) {
            dup2(pfd[0], 0);
            close(pfd[0]);
            close(pfd[1]);
            if (execlp("wc", "wc", "-l", NULL) == -1) /*second child runs wc" */
                perror("execlp wc");
        } else {
                //CLOSE DESCRIPTORS
                // Wait for both children
         }
<<< snipped code >>>
```

Homework: Think about the use of a three process pipeline with two pipes and implement it.