Problems for 17 July 2019

Please see /u/csc209h/summer/pub/ex/09/forkpipe-starter.c (which also appears below).

This is a simple example of sending data on a pipe. It does not involve i/o redirection like last week’s example, because we aren’t running any external programs.

It creates a pipe, then forks, then the parent process should send some data to the child.

1. Call write() in the parent and read() in the child so as to do this data transmission (search for the string “TODO”).

2. This code doesn’t close “the wrong side of the pipe” in the two processes. This doesn’t hurt here because the reader never waits for end-of-file; nevertheless, insert the appropriate close() calls as an exercise.

3. If you have further time, create a second pipe so that the data transmission can be two-way (one pipe from the child to the parent, one pipe from the parent to the child). The parent transmits "0". The child reads this, calls sleep(1), and replies with "1", that being what the parent sent plus 1. Also output something suitable to stdout so that you can see what’s happening. The parent does the same, so they keep sending increasing numbers back and forth, indefinitely.

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

int main()
{
    int pipefd[2];
    
    if (pipe(pipefd)) {
        perror("pipe");
        return(1);
    }
    switch (fork()) {
        case -1:
            perror("fork");
            return(1);
        case 0:
            /* child */
            printf("I am the child. FEED ME\n");
            char buf[20];
            TODO: read into buf from the pipe, terminating the string based on how many bytes read() says we read
            printf("Parent says ’%s’\n", buf);
            return(0);
        default:
            /* parent */
            printf("I am the parent. Have some data.\n");
            TODO: write a string into the pipe. Do not write any extra bytes (such as a ‘0’) 
            printf("Ok, sent some data. Hope the child likes it.\n");
            return(0);
    }
}