/* one-way-pipe.c - example of using a pipe to communicate data between a
 *                   process and its child process. The parent reads input
 *                   from the user, and sends it to the child via a pipe.
 *                   The child prints the received data to the screen. */

#include <stdio.h>    /* standard I/O routines. */
#include <unistd.h>   /* defines pipe(), amongst other things. */

/* this routine handles the work of the child process. */
void do_child(int data_pipe[]) {
    int c; /* data received from the parent. */
    int rc; /* return status of read(). */

    /* first, close the un-needed write-part of the pipe. */
    close(data_pipe[1]);

    /* now enter a loop of reading data from the pipe, and printing it */
    while ((rc = read(data_pipe[0], &c, 1)) > 0) {
        putchar(c);
    }

    /* probably pipe was broken, or got EOF via the pipe. */
    exit(0);
}

/* this routine handles the work of the parent process. */
void do_parent(int data_pipe[]) {
    int c; /* data received from the user. */
    int rc; /* return status of getchar(). */

    /* first, close the un-needed read-part of the pipe. */
    close(data_pipe[0]);

    /* now enter a loop of read user input, and writing it to the pipe. */
    while ((c = getchar()) > 0) {
        /* write the character to the pipe. */
        rc = write(data_pipe[1], &c, 1);
        if (rc == -1) { /* write failed - notify the user and exit */
            perror("Parent: write");
            close(data_pipe[1]);
            exit(1);
        }
    }

    /* probably got EOF from the user. */
    close(data_pipe[1]); /* close the pipe, to let the child know we're done. */
    close(data_pipe[0]); /* close the pipe, to let the child know we're done. */
    exit(0);
}

/* and the main function. */
int main(int argc, char* argv[]) {
    int data_pipe[2]; /* an array to store the file descriptors of the pipe. */
int pid;       /* pid of child process, or 0, as returned via fork. */
int rc;        /* stores return values of various routines. */

/* first, create a pipe. */
rc = pipe(data_pipe);
if (rc == -1) {
    perror("pipe");
    exit(1);
}

/* now fork off a child process, and set their handling routines. */
pid = fork();

switch (pid) {
    case -1: /* fork failed. */
        perror("fork");
        exit(1);
    case 0: /* inside child process. */
        do_child(data_pipe);
        /* NOT REACHED */
    default: /* inside parent process. */
        do_parent(data_pipe);
        /* NOT REACHED */
}

return 0; /* NOT REACHED */