

COSC 350 System Software

Lab #9

How to submit

- For each program, you need write detailed comments for each statement.
- You need demonstrate each task in front of me during the next lab hours.

Task 1: A simple pipe

This task is intended to help your understanding of how pipes work.

- Copy `pipe1.c` from BLP, page 532-533.
- Compile and run it.
- Reverse the order of the `write` and `read` and run the program again.
- Briefly describe what happened with the reversed order and why the program behaved that way.
- Restore the read and write to their original order before proceeding.
- Modify `pipe1.c` in the following ways:
 - It's a good idea to define two global integer constants
 - `READ_END=0`
 - `WRITE_END=1`

just to help keep straight which is which in the code. For example, when you refer to the write end of the pipe `p`, you would then use `p[WRITE_END]`. Makes the code much more readable.

- Dynamically allocate `buffer` so it's exactly the right size for `some_data`.
- Copy the string from `some_data` into `buffer`.
- Modify the read so its third argument is the exact size of `buffer` (rather than the large `BUFSIZ`).

Task 2: Pipes across a fork/exec

- Copy `pipe3.c` from BLP, page 535.
- Copy `pipe4.c` from BLP, pages 536.
- Compile and run the program just to see what it does.
- Modify `pipe3.c` as follows:
 - Have the parent wait for the child.
 - Close the file descriptor of the write end of the pipe on the parent's side.
 - Pass the write end of the pipe to the child as a command-line argument.

- d. Close the file descriptor of the read end of the pipe on the child's side.

Task 3: Multiple pipes across a fork/exec

In this task, you will write a program that has two-way communication between parent and child. Use Task 2 as a starting point, and add a second pipe to it.

- A. Copy `pipe3.c` to `twoPipesParent.c`
- B. Copy `pipe4.c` to `twoPipesChild.c`
- C. Modify `twoPipesParent.c` as follows:
 - a. Add a second pipe that will be used for a message from child to parent.
 - b. Close the appropriate file descriptors on both pipes.
 - c. Pass both pipes' file descriptors to child in the `exec`
 - d. Send the message "Hi there, Kiddo" to child over one pipe (as in Task 2). Print the pid and byte count as in Task 2.
 - e. Then, read a message from child over the second pipe. Print the pid, byte count, and message text as in Task 2.
- D. Modify `twoPipesChild.c` as follows:
 - a. Grab the file descriptors of both pipes from the argument list.
 - b. Close the appropriate file descriptors.
 - c. Read a message from parent over the first pipe (as in Task 2). Print the pid, byte count, and message text as in Task 2.
 - d. Then, send the message "Hi, Mom" to parent over the second pipe. Print the pid and byte count as in Task 2.
- E. Print your `twoPipesParent.c` and `twoPipesChild.c` to hand in.
- F. Run the program multiple times.

Task 4: FIFOs

In this task, you will modify `pipe1.c` from Task 1 so it uses a fifo instead of a pipe.

- A. Copy `pipe1.c` from Task 1 and name it `pipeFifo.c`.
- B. From your shell, create a fifo `/tmp/task4_fifo` with appropriate file permissions and ownership by using `mkfifo` command.
- C. Briefly describe how you created the fifo (the exact command you used). Also describe another command you could have used.
- D. Modify `pipeFifo.c` so it uses the fifo you created instead of using a pipe.