Parallel and Concurrent Programming - CS3D666 - Hour 4 - FIFOs

- Task 1. Go through the slides and check you understand the examples.

- Task 2. For the following example you will create a client server application. It will use all your skills developed over the last four weeks. The basic idea is that there will be a common pipe the server listens to. Clients then send a name of a pipe the client can use to send messages through to the server, this will be private between the client and the server. The Client then opens a pipe with the given name and listens to it for incoming messages, the incoming messages are then printed out. We will design it so that multiple clients can connect to the server.

Client

- The client should request a pipe name from the user via stdin. It should then create that pipe using mknod.

- The client should then send its pipe name down the pipe `Common` (which will be created in your server app and the server will be listening to). The client must open the pipe for writing and prompt the user for a message to send, if the user types in `q` it should quit.

Server

- The server creates a pipe called `Common` that it reads from. The parent process will continually listens to the Common pipe, if a message is received it then opens a pipe with a name the same as the pipe for reading.

- It then forks, the parent process continues to listen to the Common Pipe, whilst the child process will continually listen to the new pipe and print out whatever message is received appended with the name of the pipe.

- The child process should exit cleanly if the client is closed or quits.

Server

- Does your server continue to listen if the client closes?
  - How can you fix this?

- You should output each time a message is closed.

Server

- A sequence of activity is provided below:

- Step 1: Client 1
  - creates a named pipe `MyPipe1`.

- Step 2: Client 1
  - sends the name of the pipe to the server via `Common`.

Server

- Step 3: The server
  - then opens the received pipe (`MyPipe1`) using a child process for reading and listens to it. If a message is received it is displayed.

- Step 4: The server (via parent process) continues to listen to `Common` to await more clients. The process then continues.

Task 3

- 3. Can you upgrade Task 2 so that the child process is executes a new program where the pipe is redirected to stdin. This program then outputs whatever is received.

Task 4

- 4. Go back over all your work so far (in all weeks) and add in comments if you have not already done so.