Centralized Chat System

Cao Anh Quan

April 8, 2018

1 Process and Thread Design

- When client connect to the server (Server Main), then Server Main will create a new Server Fork to handle the connection from that client.
- At the beginning, Server Main also create a separate thread to handle the data passing between all Server Forks when clients send the data.
- Each server fork creates a separate thread to send the data to client while the main thread will be used to receive data from Server Main.

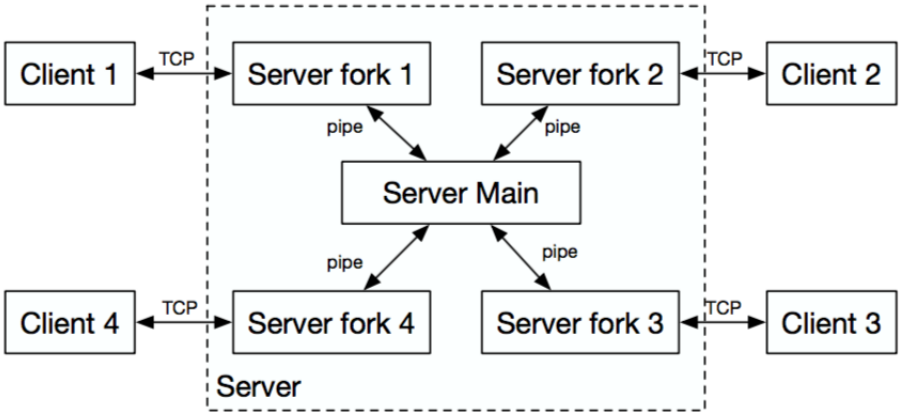
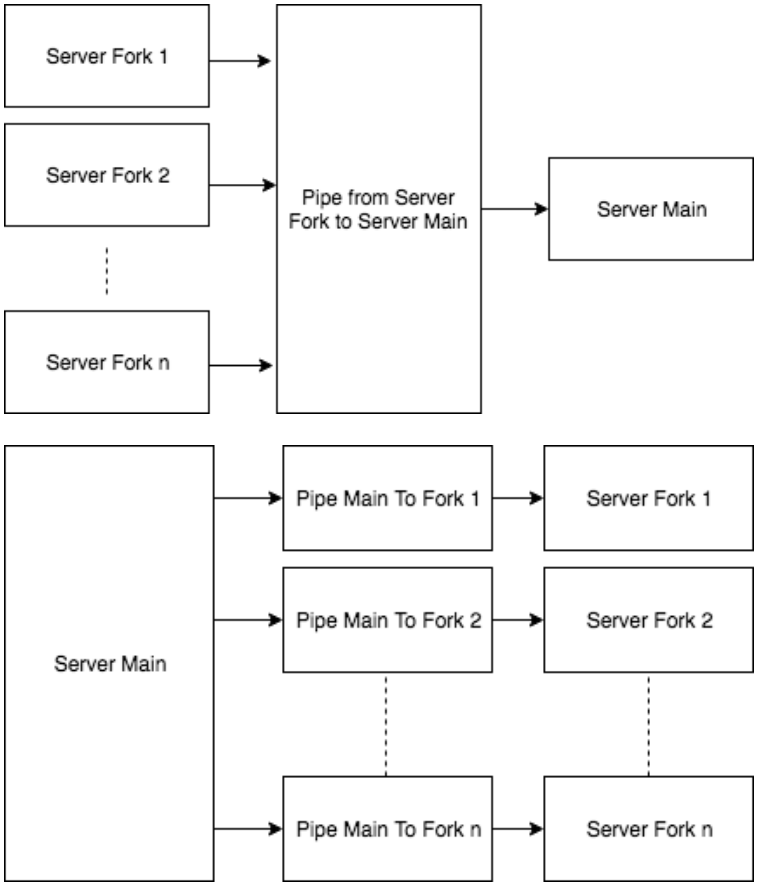


Figure 2: Detailed server structure

2 Pipe Design

We use pipe to handle the communication between Server Main and all Server Forks. There are two types of pipe we will create to handle the communication.

- Pass data from Server Forks to Server Main. For this type of pipe, we will create 1 pipe. Server Main keeps the 0 head while all Server Forks keep the 1 heads.
- Pass data from Server Main to Server Forks. For this type of pipe, we will create one pipe for each Server Fork. Each Server Fork will keep different 0 head and Server Main will keep all the 1 head.



3 Protocol Design

We will define 2 types of signal: DATA_SIGNAL and DISCONNECTED_SIGNAL. Both of them will help Server Main to know whether Server Fork received the data from client or client disconnect.

3.1 Client Send Data

1. When client connect to Server Main, Server Main will assign an id to that client and the corresponding pipe will be created. We denote it as **client n** and **pipeMainToFork n**. Then, Server Main create a Server Fork for that client, we call it **Server Fork n**.

2. When **client n** send the data, the **Server Fork n** read the data. Then it write **3 pieces of data** to the **pipeForkToMain[1]**: the signal (DATA_SIGNAL), the clientId and the data.

3. The Server Main read the data from **pipeForkToMain[0]**. It got the DATA_SIGNAL, clientId and data. Thus, It will write the data to all **pipeMainToFork** except the one corresponding to the clientId.

4. After that, Server Forks get the data and forward to their corresponding clients.

3.2 Client disconnect

1. When **client n** disconnect, the **Server Fork n** write **2 pieces of data** to the **pipeForkToMain[1]**: the signal (DISCONNECTED_SIGNAL), and the clientId. Then, the server

fork exit.

2. The Server Main read the data from **pipeForkToMain[0]**. It got the DISCONNECTED_SIGNAL and clientId. Then, the Server Main will close the **pipeMainToFork** to the sent Server Fork.

4 Result

In this work, I have completed all the requirements. When 1 client send the message, all other clients get that message. Our server serves each new client with a separated fork(). Both our clients and servers separate a thread for read and a thread for write. You can see the working servers clients in the video **demo.mov** in the repository.