CS39002: Operating Systems Lab
Spring 2015
Assignment 4
Due on: March 8, 2015 (EOD)

Implement a chat system using message queue

The chat system consists of one server and multiple clients. Certain protocols are followed, which are described later in the implementation section.

In this chat system, the server starts first. After the relevant initialization, the server runs on a loop and keeps on checking for new messages sent for it. After retrieving a message, according to the type of the message, server takes relevant action. This loop continues.

When a client has to connect to the system, it follows the **joining protocol**. This protocol lets the server and other clients know of the new client, and the new client knows about the other online clients in the system.
After this protocol has finished, it can communicate to any other client using the **communication protocol**. This protocol lets the client send the message to the server, which is relayed to the desired recipient.

In terms of interface, **the server should display** status messages the following way:

- when a client joins, display new client's **chat ID** (described below) and process ID, total number of clients
- whenever message is sent (or received), the same is to be shown along with recipient (or sender), and the number of messages in the "Up" and "Down" message queues (described later).

The client's interface:

- On starting, the client would ask user for a **chat ID.** This can be a string. The string is sent to server in the joining protocol. And the client receives list of other clients.
- Upon completion on the above step, the client would iteratively do the following:
  1. ask user if they want to send a message (y/n); if 'y', go step 2, if 'n', skip to step 6
  2. show list of clients
  3. ask user to pick a client
  4. ask user to type a message
  5. send the message using communication protocol
  6. retrieve a message from message queue
  7. if it is a chat message, show the message, sender and time of message
  8. else, it is a list of clients – so, update the local list of clients

The implementation is as follows:

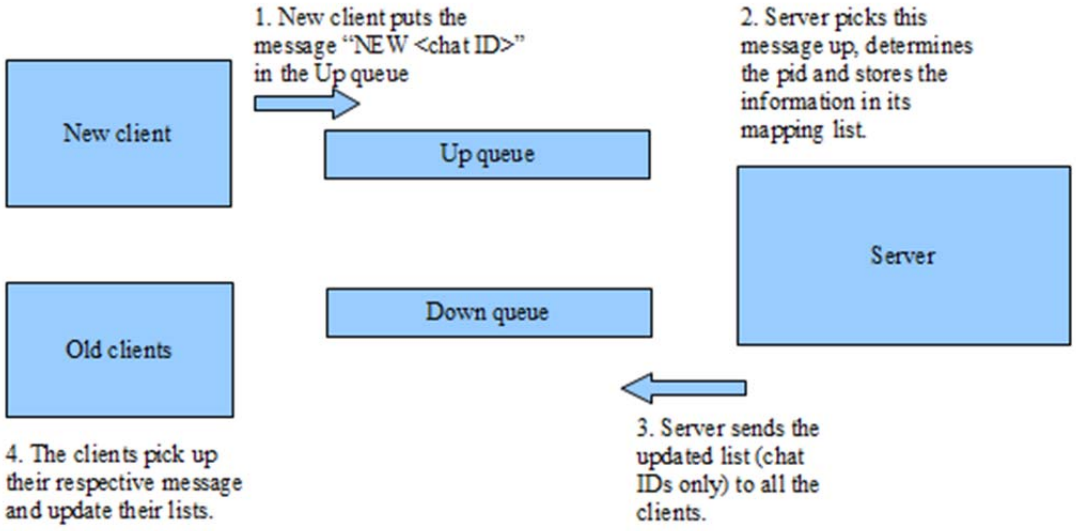
- The clients and server, communicate via two message queues “Up” and “Down” (known to all the processes a priori). “Up” is for client to server communication, “Down” is for server to client messages.
- The messages are sent in particular formats. The client and server parse the messages and execute accordingly.

Joining protocol

- When a new client starts, it first asks user for a **chat ID** and sends it in the format “NEW <chat ID>” to the server.

- On receiving such a message, the server updates its list of clients.
- (The server stores the list as **pid -> chat ID** ‘**mapping list’** about the clients. Clients just store the **chat ID** strings. For determining the pid of a sender, refer msgctl(). Store these lists in arrays.)
- Then, the server sends the list to all the clients as “LIST <chat ID1> <chat ID2> … <chat IDn>” so that they stay updated. (Hence, this list represents all online clients.)
- In this step, put the pids of respective recipient clients in the ‘type’ member of the messages.

- On receiving that message, the clients update their lists.
- This way, at any point after such a transaction, all clients know of all clients’ IDs.



(An illustration of the system executing joining protocol)

Chat communication protocol

(Sender)

- After a new client gets the list, it should present the list to the user in standard output, ask the user to pick a client and enter a message.
- Upon receiving the user's input, the client would send this info (message and recipient ID) to the server in the form "MSG <message string> <recipient chat ID>".
- After sending the message, the client would use msgrcv() to get any new message, and do the appropriate work.
- The above steps would keep iterating and the list of clients displayed should be kept updated.

(Server)

- The server, upon receiving a chat message, would identify the sending client using pid info and the **mapping list**. It would also get the time of the message using msgctl(). This time would be appended to the message. (Refer msgctl())
- And then, send the message to the recipient as "MSG <message string> <time> <sender chat ID>". In this step, put the pid of recipient in the message's 'type' member.

(Receiver)

- The receiving client picks it up with the receiving call in its loop and displays the message, along with the sender client and message time, to the user.

The client code should be named client.c and the server code server.c

Hints and Resources:

- **http://www.cs.cf.ac.uk/Dave/C/node25.html**