the Description of Problem 4

Server and Client

∗ Name: Gong Zheng Student ID: 5130309210 Email: 573965625@qq.com

1. **Files**
   - server.c
   - client.c
2. **How to operate**
   - Use command *gccserver.c* − *oserver* − *lpthread* to generate server.
   - Use command *gccclient.c* − *oclient* generate client.
   - Use command *./server* to run server.
   - Use many commands *./client* to run many clients. You can send message in cmd. Then you while receive 2 types of answer:
     - the transfered message.
     - Please wait, which means there has been 2 other clients used. You can send message to ensure whether you can use it.
   - Warning! only allow : *q* to end a client.
3. **Programming instructions**

   Client.c
   - bzero(buffer, 256); fgets(buffer, 255, stdin); n = write(sockfd, buffer, strlen(buffer));

     Instruction.

     To clear the buffer and get and send message from you to server.
   - if (buffer[0] == ':' && buffer[1] == 'q' && strlen(buffer) == 3)

     Instruction.

     To probe quit signal.
   - bzero(buffer,256); n = read(sockfd,buffer,255); printf("%s",buffer);

     Instruction.

     To clear the buffer and receive and show message from server to you.

   Server.c
   - main()
     - **listen(sockfd, 5);** To listen pthread.
     - **newsockfd = accept(sockfd, (struct sockaddr)&cli_addr, &clilen)** If one pthread has been listened, accept it and store filede scriptor to newsockfd.
     - **if(pthread_create(&serve_thread, NULL, serve, &newsockfd) != 0)** Use newsockfd to call serve function to work for client.
   - serve()
     - **if(size_server < 2)**client can run directly.

    * **pthread_mutex_lock; size_server++; pthread_mutex_unlock;** a new client added, serve_mutex used to protect data.
    * **n = read(newsockfd, buffer, 255);** receive message from client to server.
    * **if (buffer[0] == ':' && buffer[1] == 'q' && strlen(buffer) == 3)** To probe quit signal. In this judgment statement, you can sub size_server during protection of serve_mutex.
    * **solve1(buffer, n -1)** transfer the message.
    * **write(newsockfg, buffer, n)** send message from server to client.
  - **if(size_server == 2)**client need to wait.
    This part is similar with the previous. The diffrent is that there are another semaphore: wait_mutex.
    Each time server receive message from client, it will lock the semaphore and judge that if ther size_server < 2. If right, then add the size_server and unlock the semaphore and do whatever like previous part, which means another wait client cannot coming at same time. If not, just tell client "Please wait!" and then releaser the semaphore.

- solve1() Just for transfer.
- solve2() Just for wait message.