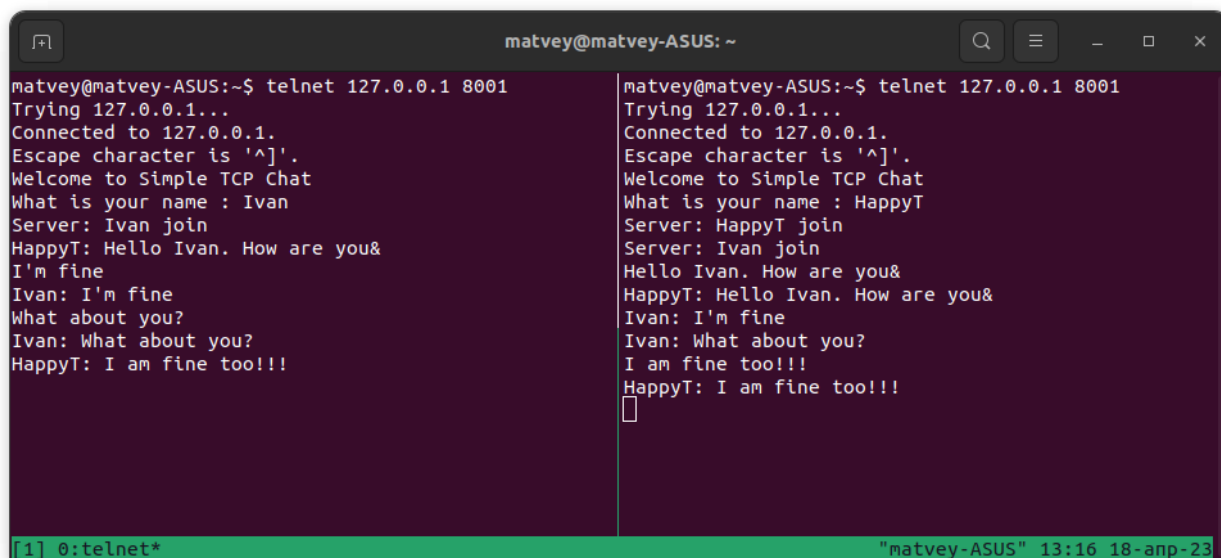


Краткий гайд по работе сервера SimpleTCPChat

!!!В коде таже есть подробные поясняющие комментарии

!!!Запуск, сборка и тд — на гитхабе

Пример работы:



```
matvey@matvey-ASUS:~$ telnet 127.0.0.1 8001
Trying 127.0.0.1...
Connected to 127.0.0.1.
Escape character is '^]'.
Welcome to Simple TCP Chat
What is your name : Ivan
Server: Ivan join
HappyT: Hello Ivan. How are you&
I'm fine
Ivan: I'm fine
What about you?
Ivan: What about you?
HappyT: I am fine too!!!

matvey@matvey-ASUS:~$ telnet 127.0.0.1 8001
Trying 127.0.0.1...
Connected to 127.0.0.1.
Escape character is '^]'.
Welcome to Simple TCP Chat
What is your name : HappyT
Server: HappyT join
Server: Ivan join
Hello Ivan. How are you&
HappyT: Hello Ivan. How are you&
Ivan: I'm fine
Ivan: What about you?
I am fine too!!!
HappyT: I am fine too!!!
^
```

Сервер построен на 2 параллельных потоках — `accept_thread` и `handle_clients_thread`.

`accept_thread`

В потоке `accept_thread` создаются новые пользователи:

```
std::shared_ptr<Client> client(new Client(messages_, service_, last_id_));
last_id_++;
acceptor.accept(client->sock());
client->update_ping();
```

Также в этом потоке новые пользователи добавляются в `clientsList`:

```
clientsList_.push_back(client);
```

`handle_clients_thread`

Если упростить, то потоке `handle_clients_thread` происходит следующее:

Сначала для каждого пользователя проверяется, отправлял ли он сообщение

```
for (const auto &x: clientsList_) {
    x->answer_to_client();
}
```

Каждый экземпляр класса `Client` сам добавит сообщение в очередь `messages_`

После этого из `clientsList` удаляются пользователи, которые слишком долго не пинговались, или при их обработке произошла ошибка.

```
clientsList_.erase(
    std::remove_if(
        clientsList_.begin(),
        clientsList_.end(),
        [&](const std::shared_ptr<Client> &Client) -> bool {
            if (Client->get_user_exit()) {
                messages_.emplace("Server", Client->get_username() + " leave the chat");
            };
            return Client->get_user_exit();
        }),
    clientsList_.end());
```

В конце каждому пользователю отправляются новые сообщения из очереди

```
while (!messages_.empty()) {  
    std::string msg = messages_.front().first + ": " + messages_.front().second;  
    for (const auto &x: clientsList_) {  
        if (x->user_is_ok())  
            x->write(msg);  
    }  
    messages_.pop();  
}
```

Вот по такой логике работает сервер.

Мелкие пояснения:

clientsList — общий для двух потоков. Для разграничения используется mutex
messages хранит пару значений (отправитель:сообщение)