**Сервер**

1. **Общее описание работы приложения**

Приложение запускается из командной строки с одним параметром: портом, по которому он будет принимать подключения.

Сервер создает сокет, связывает его с адресом, устанавливает в слушающий режим и ждет подключения клиента.

При подключении клиента сервер выводит сообщение “New client has connected: <IP клиента>”. В структуру записывается IP адрес подключившегося клиента.

Для дальнейшей работы с клиентом создается отдельный поток, а сервер продолжает принимать другие соединения.

В потоке сервер ожидает сообщение от клиента (имя клиента). После получения сообщения сервер проверяет, не достигнуто ли максимальное количество клиентов.

Если максимальное количество клиентов не достигнуто, то сервер выполняет следующие действия:

1. Проверяет имя на уникальность с помощью функции *unique\_name*. Если такого имени нет, сервер выводит сообщение “NEW NAME IS ACCEPTED” и отправляет клиенту сообщение *“accepted”*. Если такое имя уже существует, сервер выводит сообщение “ERROR: name already in use”, отправляет клиенту сообщение *“exists”* и снова ожидает от клиента сообщение с его именем и проверяет на уникальность.
2. Когда имя будет успешно получено (т.е. будет уникальным), сервер ждет сообщения от клиента, где указан порт клиента для связи этого клиента с другими клиентами.
3. В структуру записывается имя клиента, его идентификатор (сокет) и порт, сервер печатает сообщение с этими данными.
4. С помощью функции *display\_users*, клиенту посылается список клиентов, а на сервере выводится список имен активных пользователей.
5. Сервер логгирует в файл событие о подключении нового пользователя. В журнал заносится сообщение *“New user has connected: <имя клиента>”.*
6. Всем остальным клиентам посылается обновленный список клиентов и сообщение о том, что подключился новый пользователь.
7. Сервер ожидает сообщение от клиента об отключении. Если клиент отключился, сервер логгирует это событие. В файл заносится сообщение *“User disconnected: <имя клиента>”*. На сервере выводится сообщение “user removed”, клиент удаляется из структуры. Если клиент отключился, используя команду *“quit”,* сервер печатает “client is normally leaving: <ID>“*.* Если же соединение было потеряно, сервер напечатает “client is abnormally leaving: <ID>”.
8. Всем остальным клиентам отправляется обновленный список клиентов и сообщение об отключившемся пользователе.

Если же достигнуто максимальное количество пользователей, сервер печатает сообщение “We can not support new client!”, отправляет клиенту сообщение *“err”* и закрывает сокет.

1. **Описание функций**
   1. Функция unique\_name

**int** **unique\_name**(**char**\* name) {

**int** counter = 0;

**while** (counter < MAX\_CLIENTS && clients[counter].unique\_id != 0) {

**if** (**strcmp**(clients[counter].name, name) == 0) {

**printf**("User name already exists\n");

**return** -1;

}

counter++;

}

**return** 0;

}

Функция принимает в качестве параметра указатель на имя клиента. Имя сравнивается с уже имеющимися в структуре именами. Если совпадение найдено, функция печатает сообщение “User name already exists” и возвращает -1. Если совпадений не найдено, функция возвращает 0.

* 1. Функция display\_users

**int** **display\_users**(SOCKET sockfd) {

**int** counter = 0;

**char** s[100];

s[0] = 0;

**while** ((counter < MAX\_CLIENTS) && clients[counter].unique\_id != 0) {

**if** (counter == 0) {

**strcpy**(s, clients[counter].name);

} **else** {

**strcat**(s, clients[counter].name);

}

**strcat**(s, "\n");

counter++;

}

**printf**("Printing active users:\n");

**printf**("%s\n", s);

**int** n = my\_send(sockfd, (**char**\*) clients, **sizeof**(clients));

**return** n;

}

Функция принимает в качестве параметра дескриптор сокета. В цикле имена клиентов из структуры записываются в массив, который затем выводится на экран. С помощью функции *my\_send* клиенту посылается структура с данными о пользователях. Функция возвращает положительное значение, если структура была успешно отправлена, и -1 в противном случае.

* 1. Функция my\_send

**int** **my\_send**(SOCKET s, **const** **char**\* buf, **int** size){

**int** n = **send**(s, buf, size, 0);

**if** (n < 0) {

**printf**("Error sending: %d\n", **WSAGetLastError**());

}

**return** n;

}

Функция *my\_send* является оберткой для функции send. Она печатает сообщение “Error sending: <код ошибки>” в случае, если не удалось отравить сообщение. Функция возвращает значение n, которое положительно, если сообщение успешно отправлено, и -1 в противном случае.

**Клиент**

1. **Общее описание работы приложения**

Приложение взаимодействует с сервером, а также напрямую с другими клиентами.

Приложение запускается из командной строки со следующими параметрами: IP-адрес (или имя хоста) и порт сервера, к которому будет выполняться подключение.

Клиент создает сокет и устанавливает соединение с сервером. После успешного соединения клиент вводит свое имя. Это происходит внутри функции *get\_name.* Эта функция вызывает функцию *trim*, которая обрезает лишние пробелы в начале и в конце имени, и проверяет, чтобы имя не было пустым.

После успешного ввода имя отправляется на сервер.

Клиент ожидает ответа от сервера. Возможны несколько вариантов ответа:

1. Если пришло сообщение *“err”*, клиент выводит сообщение *“The server is full, please try later!”* и завершает свою работу.
2. Если пришло сообщение *“accepted”*, то имя успешно добавлено.
3. Если пришло сообщение *“exists”*, значит имя уже занято, и будет предложено заново ввести имя.

После успешной отправки имени, на клиенте создается новый сокет для связи с другими клиентами. Он связывается с адресом, устанавливается в слушающий режим и отправляет на сервер номер своего порта.

С сервера клиент принимает структуру, в которой содержатся сведения о других клиентах, и выводит их имена.

Далее выводится сообщение “You are ready” и список доступных команд с их описанием:

1. *“type”* – для того, чтобы начать писать сообщение;
2. *“quit”* – для выхода из программы;
3. *“user list”* – для просмотра списка пользователей;
4. *“help”* – для получения помощи по программе.

Создается три потока для дальнейшего взаимодействия клиента с сервером и другими клиентами.

Первый поток ожидает сообщений от сервера. Он принимает обновленный список клиентов и сообщение об отключившемся или подключившемся клиенте и выводит это сообщение. Если соединение с сервером потеряно, выводится сообщение “Server disconnected!” и клиент завершает свою работу.

Второй поток ожидает подключение другого клиента и ожидает сообщение от него.

Третий поток принимает команды от пользователя:

1. если пользователь ввел команду *“quit”*, вызывается функция *quit,* и программа завершает свою работу;
2. если пользователь ввел команду *“user list”*, выводится список имен активных пользователей;
3. если пользователь ввел команду *“help”*, выводится сообщение с доступными командами и их описанием;
4. если пользователь ввел команду *“type”*, ему предлагается ввести сообщение и имя получателя. Имя ищется в списке клиентов с помощью функции *find\_user*. Если пользователь не найден, выводится сообщение “User is not found!”*.* Если же такой пользователь существует, создается новый сокет, который соединяется со слушающим сокетом указанного клиента-получателя и ему отправляется сообщение.
5. **Описание функций**
   1. Функция find\_user

**int** **find\_user**(**char**\* name) {

**int** counter = 0;

**while** (counter < MAX\_CLIENTS && clients[counter].unique\_id != 0) {

**if** (**strcmp**(clients[counter].name, name) == 0) {

**return** counter;

}

counter++;

}

**return** -1;

}

Функции принимает в качестве параметра указатель на имя клиента-получателя. Имя сравнивается с имеющимися в структуре именами. Если имя найдено, функция возвращает порядковый номер клиента. Если же имя не найдено, функция возвращает -1.

* 1. Функция quit

**void** **quit**(**int** sock) {

my\_send(sock, "quit", **strlen**("quit"));

**printf**("\nYou are now quitting!\n");

**close**(sock);

**WSACleanup**();

**exit**(1);

}

Функции принимает в качестве параметра дескриптор сокета. В этой функции вызывается функция *my\_send,* которая отправляет серверу сообщение. Далее печатается сообщение “You are now quitting!», закрывается сокет и осуществляется выход из программы.

* 1. Функция my\_send

**int** **my\_send**(SOCKET s, **const** **char**\* buf, **int** size){

**int** n = **send**(s, buf, size, 0);

**if** (n < 0) {

**printf**("Error sending: %d\n", **WSAGetLastError**());

}

**return** n;

}

Функция *my\_send* является оберткой для функции send. Она печатает сообщение “Error sending: <код ошибки>” в случае, если не удалось отравить сообщение. Функция возвращает значение n, которое положительно, если сообщение успешно отправлено, и -1 в противном случае.

* 1. Функция get\_name

**void** **get\_name**() {

**do** {

**printf**("\nEnter your name: ");

**fgets**(name, NAME\_LEN + 1, stdin);

name[NAME\_LEN] = '\n';

name[NAME\_LEN + 1] = '\0';

trim(name);

} **while** (**strcmp**(name, "\n") == 0);

}

В функции *get\_name* осуществляется ввод имени пользователя, к имени добавляются символы новой строки и конца строки, затем вызывается функция trim. Ввод имени происходит до тех пор, пока имя не будет корректно введено, т.е. не будет пустым.

* 1. Функция trim

**void** **trim**(**char** \*s) {

**unsigned** **int** i = 0, j;

**while** ((s[i] == ' ') || (s[i] == '\t')) {

i++;

}

**if** (i > 0) {

**for** (j = 0; j < **strlen**(s); j++) {

s[j] = s[j + i];

}

s[j] = '\0';

}

i = **strlen**(s) - 2;

**while** ((s[i] == ' ') || (s[i] == '\t')) {

i--;

}

**if** (i < (**strlen**(s) - 2 )) {

s[i + 2] = '\0';

}

}

Функция принимает в качестве параметра указатель строку. Функция удаляет пробелы и символы табуляции в начале и в конце строки.

**Схемы передачи сообщений между программами**

На рисунке 1 изображена схема передачи сообщений между клиентом и сервером при подключении клиента к серверу.

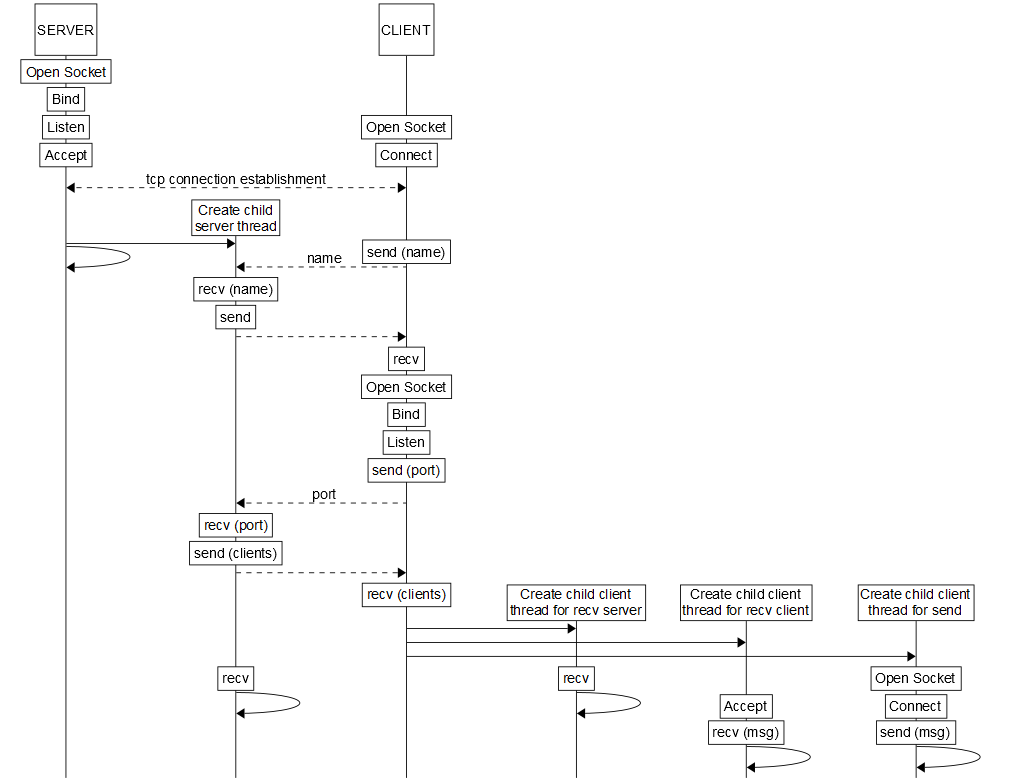


Рис. 1 – Запуск сервера и соединение клиента с сервером

На рисунке 2 изображена схема передачи сообщений от сервера клиенту при подключении еще одного клиента.

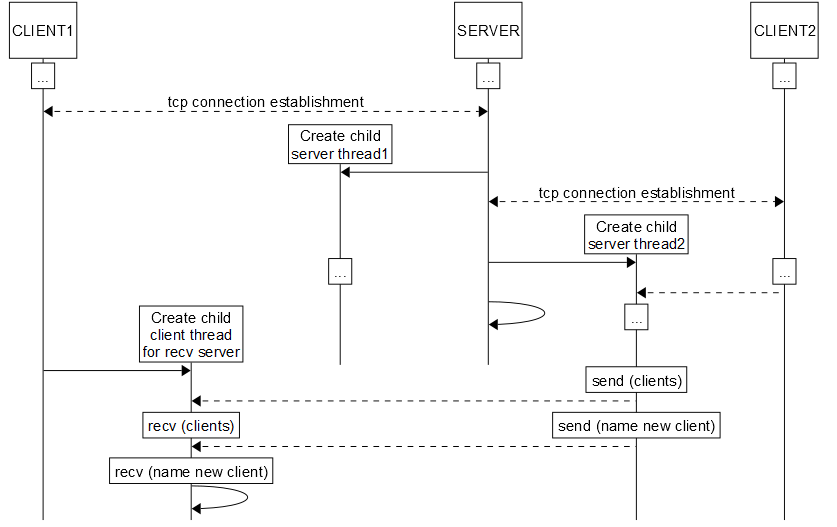


Рис. 2 – Подключение еще одного клиента

На рисунке 3 изображена схема передачи сообщений между клиентами и сервером при отключении одного из клиентов.

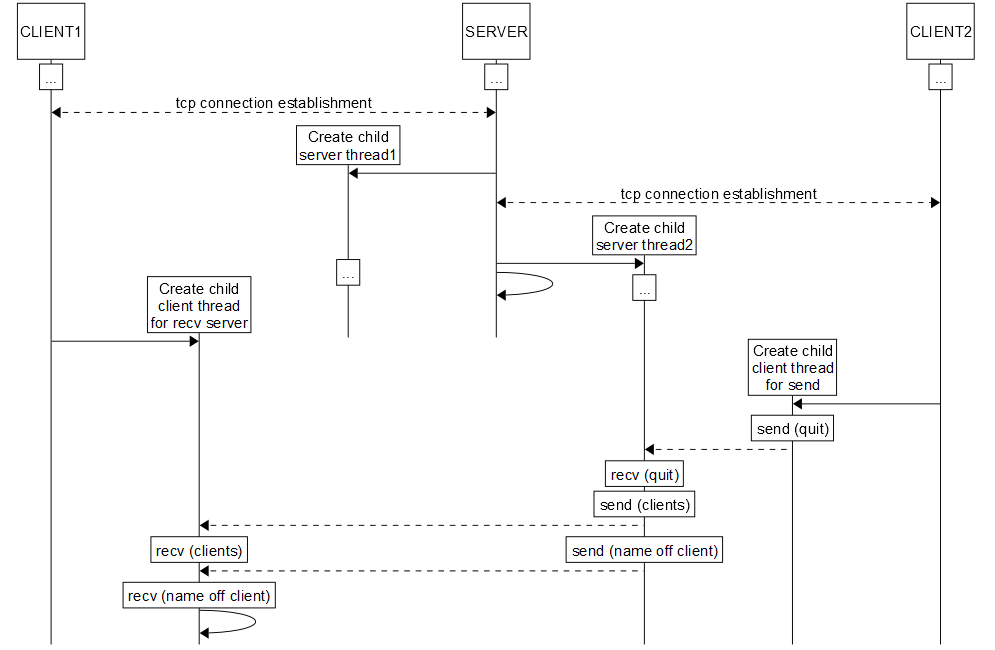


Рис. 3 – Отключение клиента

На рисунке 4 изображена схема передачи сообщений между клиентами. Общение клиентов осуществляется напрямую от клиента к клиенту, минуя сервер.

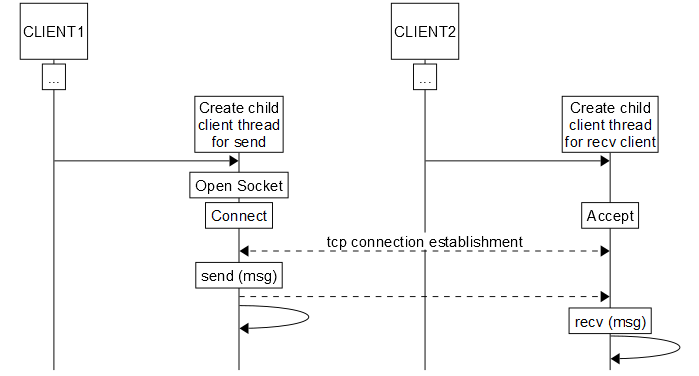


Рис. 4 – Взаимодействие клиента с другим клиентом