МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ государственное БЮДЖЕТНОЕ образовательное учреждениевысшего образования

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра вычислительной техники

**ОТЧЁТ по ПРАКТИЧЕСКОЙ работе №2**

**по дисциплине «ПРОГРАММИРОВАНИНЕ»**

**«Разработка сетевого интерфейса системы управления базой данных»**

|  |  |
| --- | --- |
| Факультет: АВТ  Группа: АБ-220  Студент(ы): Лазарев Кирилл | Преподаватель: Исаев Г.А. |

Новосибирск, 2023 г.

**Задание к работе:**

Требуется реализовать сетевой интерфейс для СУБД из практики 1.

При запуске приложения из первой практики СУБД должна ожидать соединение по протоколу tcp на порту 6379. При подключении требуется обработать запрос либо в отдельном потоке, либо в отдельном процессе, либо асинхронной задачей и отдать результат в ответ на запрос.

Также необходимо позаботится о блокировках на структуре БД если используются потоки или процессы.

**Постановка задачи:**

Написать приложение сервера которое будет принимать запросы по протоколу TCP на порту 6379 и обрабатывать их в соответствии с поставленной задачей. После обработки и выполнения запроса возвращать результат работы клиенту.

Так же для удобства напишем клиентское приложение которое будет обращаться к серверу, передавать команду для работы с СУБД в виде запроса и получать ответ от сервера.

1. **Алгоритм решения задачи**

Серверное приложение создает сокет и привязывает его к адресу сервера (localhost или локальный адрес) и нужному порту. Запустить прослушивание порта (LISTENING) и ожидать входящее соединение от клиента. При установлении соединения проверять есть ли входящее сообщение и при его наличии обработать его. Обработку сделать через консоль (system()) и сохранить поток вывода в файл (\*Команда\* **>** \*Имя файла\*). Считать содержимое файла и сохранить его в переменную для отправки, в нашем случае переменная output. Вернуть клиенту ответ от сервера и закрыть соединение. Все должно происходить в бесконечном цикле и сервер не должен закрываться или падать.

**Текст программы**

**server.c:**

#include <stdio.h>

#include <stdlib.h>

#include <winsock2.h>

#pragma comment(lib, "ws2\_32.lib")

#define IP "127.0.0.1"

#define PORT 6379

#define MAX\_CONNECTIONS 3

**typedef** **struct** sockaddr\_in sockaddr\_in;

**typedef** **struct** sockaddr sockaddr;

// Структура для хранения данных клиента

**typedef** **struct** {

SOCKET sockfd;

sockaddr\_in addr;

} ClientData;

// Функция обработки запросов клиента

DWORD WINAPI **handleClient**(LPVOID arg) {

ClientData\* client = (ClientData\*)arg;

**char** buffer[**1024**]; // Запрос к серверу -> сmd

**char** output[**4096**]; // Ответ сервера

**int** bytesRead;

**char** cmd[**512**]; // Команда для работы с NO-SQL СУБД из 1й практики

// Прием данных от клиента

**while** ((bytesRead = recv(client->sockfd, buffer, **sizeof**(buffer), **0**)) > **0**) {

// Обработка данных и формирование ответа

snprintf(cmd, **sizeof**(cmd), "..**\\**practice1**\\**main.exe %s > cout", buffer);

printf("buffer=**\"**%s**\"**, cmd=**\"**%s**\"\n**", buffer, cmd);

printf("Производится вызов команды **\"**%s**\"\n**", cmd);

system(cmd); // Сохраняем вывод консоли в файл "... > cout"

printf("Окончание вызова команды.**\n**");

printf("Начало работы с файлом!**\n**");

**FILE**\* fr = fopen("cout", "r");

**if** (fr == NULL) {

snprintf(output, **sizeof**(output), "Файл не найден или у программы нет к нему доступа!**\n**");

send(client->sockfd, output, **sizeof**(output), **0**);

**break**;

} **else** {

**char** line[**512**];

**while** (fgets(line, **sizeof**(line), fr)) { // Считываем файл по строкам

snprintf(output, **sizeof**(output), "%s%s", output, line);

}

}

fclose(fr);

printf("Конец работы с файлом!**\n**");

// Отправка ответа клиенту

send(client->sockfd, output, **sizeof**(output), **0**);

}

// Закрытие соединения с клиентом

closesocket(client->sockfd);

free(client);

**return** **0**;

}

**int** **main**(**void**) {

system("chcp 1251"); // Кодировка Cyrillic1251

SOCKET sockfd, newsockfd;

sockaddr\_in serv\_addr, cli\_addr;

**int** clilen;

HANDLE **thread**;

// Создание сокета

WSADATA wsaData;

WSAStartup(MAKEWORD(**2**, **2**), &wsaData);

sockfd = socket(AF\_INET, SOCK\_STREAM, **0**);

**if** (sockfd < **0**) {

printf("Ошибка при создании сокета");

exit(**1**);

}

// Заполнение структуры адреса сервера

memset(&serv\_addr, **0**, **sizeof**(serv\_addr));

serv\_addr.sin\_family = AF\_INET;

serv\_addr.sin\_addr.s\_addr = inet\_addr(IP);

serv\_addr.sin\_port = htons(PORT);

// Привязка сокета к адресу сервера

**if** (bind(sockfd, (sockaddr\*)&serv\_addr, **sizeof**(serv\_addr)) < **0**) {

printf("Ошибка при привязке сокета к адресу");

exit(**1**);

}

// Ожидание подключений

listen(sockfd, MAX\_CONNECTIONS);

printf("Сервер запущен и ожидает подключений...**\n**");

// Обработка подключений клиентов

**while** (**1**) {

clilen = **sizeof**(cli\_addr);

// Принятие подключения

newsockfd = accept(sockfd, (sockaddr\*)&cli\_addr, &clilen);

**if** (newsockfd < **0**) {

printf("Ошибка при принятии подключения");

exit(**1**);

}

// Создание структуры с данными клиента

ClientData\* client = calloc(**1**, **sizeof**(ClientData));

client->sockfd = newsockfd;

client->addr = cli\_addr;

// Создание нового потока для обработки клиента

**thread** = CreateThread(NULL, **0**, handleClient, (LPVOID)client, **0**, NULL);

**if** (**thread** == NULL) {

printf("Ошибка при создании потока");

free(client);

exit(**1**);

}

}

// Закрытие сокета сервера

closesocket(sockfd);

WSACleanup();

**return** **0**;

}

**client.c:**

#include <stdio.h>

#include <stdlib.h>

#include <winsock2.h>

#pragma comment(lib, "ws2\_32.lib")

#define IP "127.0.0.1"

#define PORT 6379

**typedef** **struct** sockaddr\_in sockaddr\_in;

**typedef** **struct** sockaddr sockaddr;

**int** **main**(**int** argc, **char**\* argv[]) {

system("chcp 1251"); // Кодировка 1251

WSADATA wsa;

**if** (WSAStartup(MAKEWORD(**2**, **2**), &wsa) != **0**) {

perror("WSAStartup");

exit(EXIT\_FAILURE);

}

SOCKET sockfd = socket(AF\_INET, SOCK\_STREAM, **0**); // Создаем сокет

**if** (sockfd == INVALID\_SOCKET) {

perror("socket");

WSACleanup();

exit(EXIT\_FAILURE);

}

sockaddr\_in serv\_addr;

memset(&serv\_addr, **0**, **sizeof**(serv\_addr)); // Обнуляем структуру

serv\_addr.sin\_family = AF\_INET;

serv\_addr.sin\_port = htons(PORT);

serv\_addr.sin\_addr.s\_addr = inet\_addr(IP); // Адрес сервера

// Подключаемся к серверу

**if** (connect(sockfd, (sockaddr\*)&serv\_addr, **sizeof**(serv\_addr)) == SOCKET\_ERROR) {

perror("connect");

closesocket(sockfd);

WSACleanup();

exit(EXIT\_FAILURE);

}

**char**\* message = argv[**1**]; // Сообщение для отправки

**if** (send(sockfd, message, strlen(message), **0**) == SOCKET\_ERROR) { // Отправляем сообщение

perror("send");

closesocket(sockfd);

WSACleanup();

exit(EXIT\_FAILURE);

}

// Получаем ответ от сервера

**char** buffer[**4096**];

**int** bytesReceived = recv(sockfd, buffer, **sizeof**(buffer), **0**);

**if** (bytesReceived < **0**) {

perror("recv");

closesocket(sockfd);

WSACleanup();

exit(EXIT\_FAILURE);

}

// Печатаем ответ от сервера

printf("Ответ от сервера: %s**\n**", buffer);

closesocket(sockfd); // Закрываем сокет

WSACleanup();

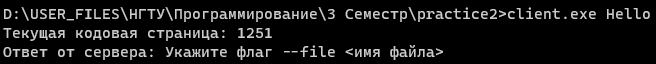
**return** **0**;

}

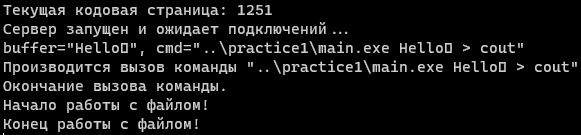
**Результат работы программы**



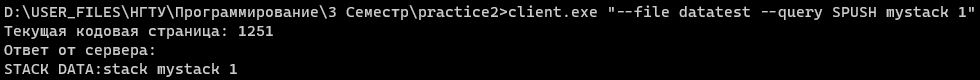
Запуск сервера



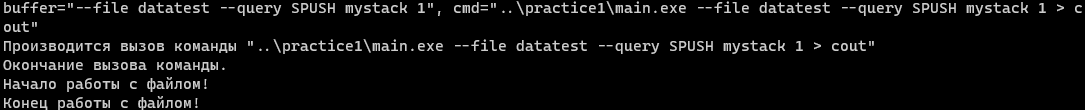
Отправка случайного запроса на сервер



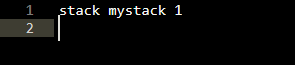
Тем временем на сервере



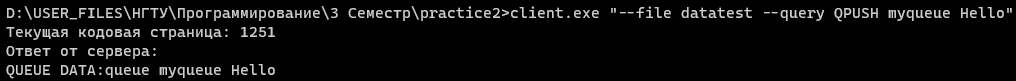
Отправка корректного запроса для работы с СУБД и получение ответа от сервера



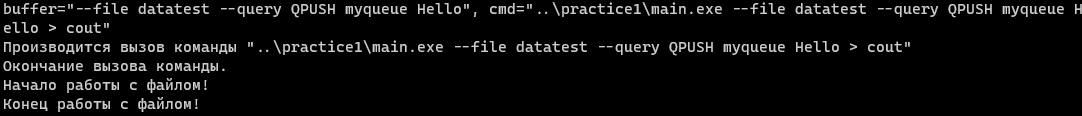
Тем временем на сервере



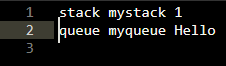
Содержимое файла на сервере после обработки и выполнения запроса



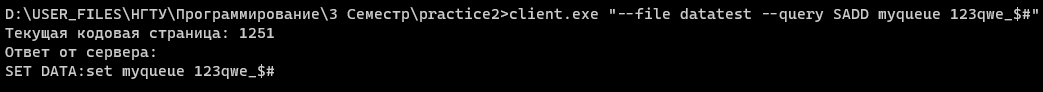
Отправка корректного запроса для СУБД для добавления элемента ‘Hello’ в очередь “myqueue”



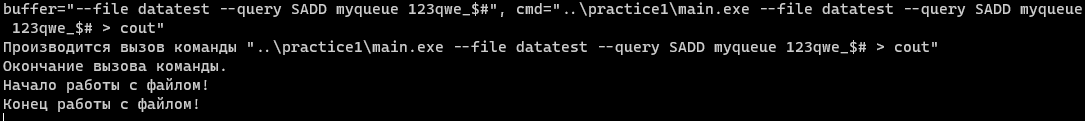
Тем временем на сервере



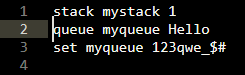
Файл на сервере обновился



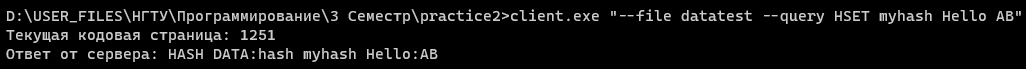
Отправка запроса на добавление нового элемента в множество



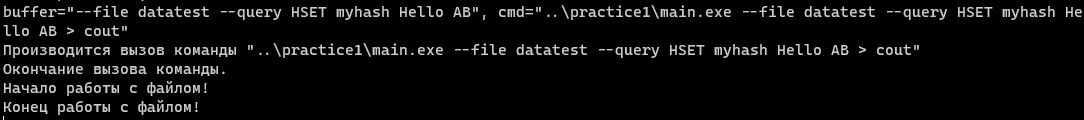
За кулисами



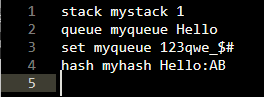
Содержимое файла на сервере после работы



Отправка запроса на добавление значения ‘AB’ по ключу ‘Hello' в хэш-таблицу ‘myhash’



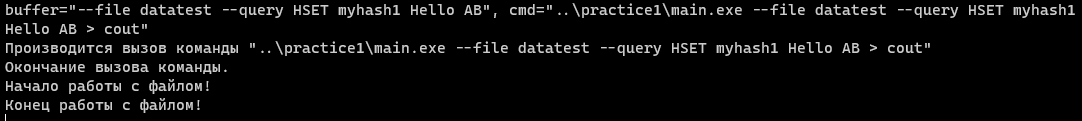
Сервер



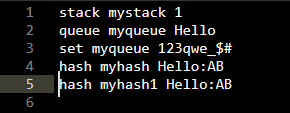
Файл после добавления



Отправка запроса на добавление значения ‘AB’ по ключу ‘Hello' в хэш-таблицу ‘myhash1’



Сервер



Файл после добавления

*Программа СУБД взята в точности из первой практики и работает соответственно точно так же. Для примера мы попробовали по одной операции с каждой структурой данных и проверили как ведет себя программа при добавлении одной и той же структуры с разными именами*

**Вывод**: Результат работы программы совпадает с ожидаемыми результатами, что говорит о корректной работе программы. Мы изучили как работать с сокетами, принимать, отправлять и обрабатывать запросы по протоколу TCP и научились полностью реализовывать это на C.

У данной программы есть одна серьезная уязвимость в виде *command injection* если передать на сервер запрос в виде “\*первая команда\* **&** \*вредоносная команда\*”