Министерство науки и высшего образования Российской Федерации

федеральное государственное автономное   
образовательное учреждение высшего образования   
«Самарский национальный исследовательский университет   
имени академика С.П. Королева»

Институт информатики и кибернетики

Кафедра технической кибернетики

Отчет по лабораторной работе №5

Дисциплина: «Операционные системы»

Тема «Использование семафоров и мьютексов для

клиент-серверного взаимодействия»

Выполнил: Белов А.А.

Группа: 6205-010302D

Самара, 2023

**Задание на лабораторную работу**

Вариант 5

Требуется написать клиент-серверное приложение - калькулятор. Вычисления происходят на сервере. Одновременно может подключаться к серверу два клиента. На клиенте вводятся числа в формате «арифметическая операция целое или дробное число пробел целое или дробное число». Достаточно двух чисел для одной операции. На сервере данные от соответствующего клиента отображаются в новой строке, в формате «Получено от клиента №X арифметическая операция «формат операции». Результат - …». Ответ от сервера на клиенте выводится в новой строке.

**Текст программы**

Клиентская часть:

#include <iostream>

#include <winsock2.h>

#include <mutex>

#include <string>

#include <sstream>

using namespace std;

#pragma warning(disable: 4996)

#pragma comment(lib, "ws2\_32.lib") // Для работы с сокетами

SOCKET Connection;

// принятие сообщения от сервера

void ClientHandler() {

char msg[256];

while (true) {

int ds = recv(Connection, msg, sizeof(msg), NULL);

if (ds) {

if (ds == SOCKET\_ERROR || ds == 0) { // получение информации

cout << "Error, failed to receive data from server " << endl;

closesocket(Connection);

WSACleanup();

exit(0);

}

cout << msg << endl;

}

}

}

int main() {

WSADATA wsaData; // создаём структуру wsaData

WORD DLLVersion = MAKEWORD(2, 1); // используется для указания версии Winsock, которая будет запрошена при инициализации библиотеки с помощью функции WSAStartup

if (WSAStartup(DLLVersion, &wsaData) != 0) { // проверка на инициализацию библиотеки Winsock

cout << "Error for inicialization Winsock! " << endl;

exit(1);

}

SOCKADDR\_IN addr; // структура для хранение адреса

int sizeofaddr = sizeof(addr); //размер

addr.sin\_addr.s\_addr = inet\_addr("127.0.0.1"); // IP-адрес сервера (локал хост)

addr.sin\_port = htons(1111); // Порт, на котором будет слушать сервер

addr.sin\_family = AF\_INET; // семейство протоколов, для интерент протоколов: AF\_INET

Connection = socket(AF\_INET, SOCK\_STREAM, NULL);//сокет для соединения с сервером

SOCKET sListen = socket(AF\_INET, SOCK\_STREAM, NULL);//создание сокета для прослушки порта

if (connect(Connection, (SOCKADDR\*)&addr, sizeof(addr)) != 0){ //проверка на подключение к серверу

cout << "Error: failed connect to server.\n";

closesocket(Connection);

WSACleanup();

return 0;

}

cout << "Connected!\n"; //подключился

CreateThread(NULL, NULL, (LPTHREAD\_START\_ROUTINE)ClientHandler, NULL, NULL, NULL);

//Sleep(1000);

char msgl[256];

bool flag;

while (true) {

cin.getline(msgl, sizeof(msgl));

string message(msgl);

if (message == "exit")

return 0;

string str(msgl);

stringstream ss(msgl);

char sign;

int first\_number, second\_number;

try

{

flag = ss >> sign >> first\_number >> second\_number ? 1 : 0;

if (sign == '+' || sign == '-' || sign == '/' || sign == '\*') {

flag = 1;

}

else {

flag = 0;

cout << "Incorrect input" << endl;

}

}

catch (...)

{

cout << "Incorrect input" << endl;

}

if (flag) {

send(Connection, msgl, sizeof(msgl), NULL);

}

}

closesocket(Connection);

WSACleanup();

system("pause");

return 0;

}

Серверная часть:

include <iostream>

#include <string>

#include <thread>

#include <winsock2.h>

#include <sstream>

#include <windows.h>

#include <ws2tcpip.h>

#include <condition\_variable>

#include <queue>

#pragma warning(disable: 4996)

#pragma comment(lib, "ws2\_32.lib") // Для работы с сокетами

using namespace std;

mutex g\_mutex;

condition\_variable g\_condVar; // потоки, ожидающие клиентов, могли быть уведомлены о том, что появился новый клиент в очереди.

queue<SOCKET> g\_clientQueue; // хранения клиентских сокетов в очереди

int g\_numThreads = 0; // отслеживания количества запущенных потоков.

int counter = 0;

const int MAX\_THREADS = 2; // количество клиентов, с которыми одновременно взаимодействует сервер

const int MAX\_QUEUE\_SIZE = 10; // максимальное количество включённых клиентов

bool processClient(SOCKET clientSocket, int index) {

// Получаем данные от клиента

char buffer[1024];

int bytesReceived = recv(clientSocket, buffer, sizeof(buffer), 0);

// Обработка выражения

if (bytesReceived) {

if (bytesReceived == SOCKET\_ERROR || bytesReceived == 0) {

closesocket(clientSocket);

cout << "Client: " << index << " disconnect" << endl;

Sleep(500);

return 0;

}

else {

cout << "from " << index << " : " << buffer << endl;

string str(buffer);

int res; // результат

stringstream ss(str);

char sign;

int first\_number, second\_number;

ss >> sign >> first\_number >> second\_number;

string inf;

if (sign == '+') {

res = first\_number + second\_number;

}

else if (sign == '-') {

res = first\_number - second\_number;

}

else if (sign == '/') {

if (second\_number == 0)

inf = "inf";

else

res = first\_number / second\_number;

}

else if (sign == '\*') {

res = first\_number \* second\_number;

}

string m;

if (second\_number == 0 && sign == '/') {

m = "To client: inf";

}

else

m = "To client: " + to\_string(res);

char msg[sizeof(m)];

strcpy\_s(msg, m.c\_str()); // Преобразование std::string в char[]

cout << "The result sent to " << index << " client" << endl;

int bytesSent = send(clientSocket, msg, sizeof(msg), 0);

}

}

else {

closesocket(clientSocket);

return 0;

}

}

void threadFunc() {

int i = 1;

while (true) {

SOCKET clientSocket;

{

unique\_lock<mutex> lock(g\_mutex); // захватываем мьютекс

// Ожидаем клиента, если все потоки заняты

while (g\_clientQueue.empty()) {

g\_condVar.wait(lock);

}

// Извлекаем следующего клиента из очереди

clientSocket = g\_clientQueue.front();

g\_clientQueue.pop();

}

// Обрабатываем запрос клиента

bool fd;

do {

fd = processClient(clientSocket, i);

} while (fd != 0);

++i;

// Закрываем сокет клиента

closesocket(clientSocket);

{

// Освобождаем один слот для нового клиента

lock\_guard<mutex> lock(g\_mutex);

--g\_numThreads;

--counter;

}

//cout << "counter: " << counter << endl;

if (counter == 0) {

closesocket(clientSocket);

WSACleanup();

exit (0);

}

g\_condVar.notify\_one(); // Уведомляем потоки, что есть свободный слот

}

}

int main() {

WSADATA wsaData; // создаём структуру wsaData

WORD DLLVersion = MAKEWORD(2, 1); // используется для указания версии Winsock, которая будет запрошена при инициализации библиотеки с помощью функции WSAStartup

if (WSAStartup(DLLVersion, &wsaData) != 0) { // проверка на инициализацию библиотеки Winsock

cout << "Error for inicialization Winsock! " << endl;

exit(1);

}

SOCKET serverSocket = socket(AF\_INET, SOCK\_STREAM, IPPROTO\_TCP);

SOCKADDR\_IN addr; // структура для хранение адреса

int sizeofaddr = sizeof(addr); //размер

addr.sin\_addr.s\_addr = inet\_addr("127.0.0.1"); // IP-адрес сервера (локал хост)

addr.sin\_port = htons(1111); // Порт, на котором будет слушать сервер

addr.sin\_family = AF\_INET; // семейство протоколов, для интерент протоколов: AF\_INET

if (bind(serverSocket, (SOCKADDR\*)&addr, sizeof(addr)) != 0) { //проверка на привязку адреса сокету

closesocket(serverSocket);

WSACleanup();

}

listen(serverSocket, SOMAXCONN); // прослушивание, сколько запросов ожидается, (остальные получат ошибку)

//HANDLE semaphore = CreateSemaphore(nullptr, MAX\_THREADS, MAX\_THREADS, nullptr);

cout << "Waiting for client connection..." << endl;

std::vector<std::thread> threads;

for (int i = 0; i < MAX\_THREADS; ++i) {

threads.emplace\_back(threadFunc);

}

int o = 1;

// Ожидаем клиентов и добавляем их в очередь

while (true) {

SOCKET clientSocket = accept(serverSocket, nullptr, nullptr);

++counter;

string mes = "Client " + to\_string(o) + " is connected";

char msg[sizeof(mes)];

strcpy\_s(msg, mes.c\_str());

send(clientSocket, msg, sizeof(msg), 0);

if (clientSocket == INVALID\_SOCKET) {

cout << "All clients are disconected" << endl;

for (auto& thread : threads)

thread.join();

// Закрываем сокеты и освобождаем ресурсы

//CloseHandle(semaphore);

closesocket(serverSocket);

WSACleanup();

return 0;

}

else {

Sleep(1000);

unique\_lock<mutex> lock(g\_mutex);

// Ожидаем свободный слот

while (g\_numThreads == MAX\_THREADS) {

g\_condVar.wait(lock);

send(clientSocket, "Client to wait ", 64, 0);

}

// Добавляем клиента в очередь

g\_clientQueue.push(clientSocket);

send(clientSocket, "Client to queue ", 64, 0);

++g\_numThreads;

}

g\_condVar.notify\_one(); // Уведомляем потоки о новом клиенте

++o;

}

// Ожидаем завершения всех потоков

for (auto& thread : threads) {

thread.join();

}

// Закрываем сокеты и освобождаем ресурсы

//CloseHandle(semaphore);

closesocket(serverSocket);

WSACleanup();

return 0;

}