**Правительство Российской Федерации**

**Федеральное государственное автономное образовательное**

**учреждение высшего образования «Национальный исследовательский университет «Высшая школа экономики»**

Факультет компьютерных наук

Департамент программной инженерии

**Отчет к домашнему заданию**

**По дисциплине**

**«Архитектура вычислительных систем»**

Работу выполнил:

Студент группы БПИ-194 Романюк А.С

Вариант 21

**Москва 2020**

**Задание**

Преподаватель проводит экзамен у группы студентов. Каждый студент заранее знает свой билет и готовит по нему ответ. Подготовив ответ, он передает его преподавателю. Преподаватель просматривает ответ и сообщает студенту оценку. Требуется создать многопоточное приложение, моделирующее действия преподавателя и студентов. При решении использовать парадигму «клиент-сервер».

**Модель**

Клиенты и серверы – способ взаимодействия неравноправных потоков. Клиентский поток запрашивает сервер и ждет ответа. Серверный поток ожидает запроса от клиента, затем действует в соответствии с поступившим запросом.

**Решение**

Для реализации данной задачи была использована библиотека “windows.h” для взаимодействия с WINAPI. Так как разработка велась на ОС Windows, то было принято решение пользоваться библиотекой WINAPI, а не POSIX Threads.

При разработке были использованы критические секции для обеспечения корректного вывода на экран, а также события для синхронизации действий между потоками сервера и клиентом.

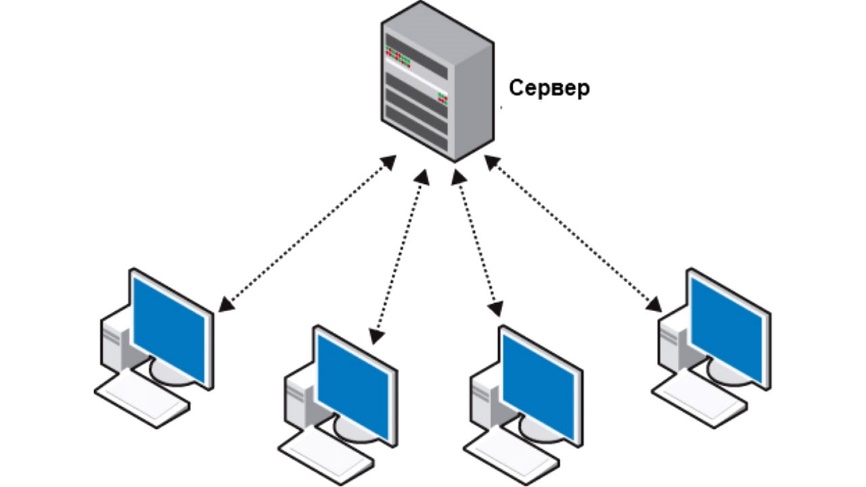


Рисунок 1 – Пример клиент-серверного приложения.

Чтобы полностью продемонстрировать работу с потоками была сымитирована следующая ситуация: студенты входят один за одним в аудиторию к преподавателю, причём входят они за случайное время [*minStudentEnters*; *maxStudentEnters*) мс. Сам же преподаватель готовится принимать, то есть создаётся отдельный поток под преподавателя. После того как студент вошёл в аудиторию, он получает номер билета и начинает готовиться. Причём он может готовиться во время того, как другие студенты входят в аудиторию. Это было сделано для того, чтобы продемонстрировать работу главного потока и потока-студента.

Далее каждый студент смотрит, открыты ли события “serverReady”, “exchange1”, “exchange2”, “serverAnswer”, а также выполняется проверка адресации в пространство процессора данных о студенте. Запросы выполняются с задержкой в *defaultWait* мс. То есть клиент смотрит, готов ли сервер к работе.

Как только сервер стал доступен – студент берёт билет и начинает готовиться случайное время [*minStudentPrep*; *maxStudentPrep*) мс. Дальше студент ждёт готовности преподавателя, если преподаватель готов – он ожидает, пока студент подготовит все данные и выполнит событие “exchange1”. После чего преподаватель принимает у студента экзамен за случайное время [*minTeacherCheck*; *minTeacherCheck*) мс.

Как только преподаватель принял у студента экзамен, он выставляем ему оценку равную случайному целому числу из диапазона [*minMark*; *maxMark*] и выполняет событие “serverAnswer”, и ждёт, пока студент-клиент выполнит событие “exchange2”, то есть заберёт свой студенческий билет и уйдёт. После чего студент подчищает за собой все данные и выходит из аудитории, а сервер уже готов принимать следующего студента.

Всё это происходит до тех пор, пока все студенты не будет опрошены, после чего последует сообщение о том, что экзамен окончен и программа завершится.

Стоит также упомянуть работу с критической секцией.

CRITICAL\_SECTION cs; // Критическая секция для синхронизации вывода на экран

void showMessage(char \*msg) {

EnterCriticalSection(&cs);

cout << msg << endl;

LeaveCriticalSection(&cs);

}

Каждый вывод на экран в потоках сопровождается работой с критической секцией для корректного отображения данных.

Помимо этого, после выполнения работы каждого из созданных потоков выполняется очищение данных.

// Освобождение ресурсов, занятых сервером

UnmapViewOfFile(pData);

CloseHandle(mapFile);

CloseHandle(serverAnswer);

CloseHandle(exchange1);

CloseHandle(exchange2);

CloseHandle(serverReady);

Проецирование данных происходит через MapFile:

// Создание общей области памяти для общения клиентов с сервером

HANDLE mapFile = CreateFileMapping(INVALID\_HANDLE\_VALUE, nullptr, PAGE\_READWRITE, 0, sizeof(int) \* 2, LPCSTR("MyShared")); //Создать проекцию на файл в памяти (используется файл подкачки)

// Проецируем файл в адресное пространство процесса

int \*pData = (int\*) MapViewOfFile(mapFile, FILE\_MAP\_READ | FILE\_MAP\_WRITE, 0, 0, 0);

**Код программы**

#include <iostream>

#include <windows.h>

#include <utility>

#include <vector>

using namespace std;

bool stopExam = false; // Флаг остановки экзамена

CRITICAL\_SECTION cs; // Критическая секция для синхронизации вывода на экран

static const unsigned int defaultWait = 100; // Задержка между запросами в мс.

static const unsigned int minStudentEnters = 100; // Минимальное время мс для входа студента в аудиторию в мс.

static const unsigned int maxStudentEnters = 250; // Максимальное время для входа студента в аудиторию в мс.

static const unsigned int minStudentPrep = 4000; // Минимальное время подготовки студента к ответу в мс.

static const unsigned int maxStudentPrep = 10000; // Максимальное время подготовки студента к ответу в мс.

static const unsigned int minTeacherCheck = 3000; // Минимальное время проверки преподавателем студента в мс.

static const unsigned int maxTeacherCheck = 5000; // Максимальное время проверки преподавателем студента в мс.

static const unsigned int minMark = 0; // Минимальная оценка.

static const unsigned int maxMark = 10; // Максимальная оценка.

static const unsigned int countTickets = 100; // Количество билетов.

static const unsigned int maxStudents = 273; // Максимальное количество студентов. Реальные цифры с ФКН ПИ 2 курс :)

// Вывод сообщения

void showMessage(char \*msg) {

EnterCriticalSection(&cs);

cout << msg << endl;

LeaveCriticalSection(&cs);

}

// Функция потока студента(клиент)

DWORD WINAPI studentThread(PVOID param) {

char buf[256];

auto p = (DWORD) param; // Переданный параметр

int ticket = p >> 16; // Номер билета

int studNumber = p & 0xffff; // Номер студенческого билета

srand(ticket);

// События для синхронизации с сервером

HANDLE serverReady, exchange1, exchange2, serverAnswer;

// Открытие событий сервера

while ((serverReady = OpenEvent(EVENT\_ALL\_ACCESS, FALSE, LPCSTR("serverReady"))) == nullptr ||

(exchange1 = OpenEvent(EVENT\_ALL\_ACCESS, FALSE, LPCSTR("exchange1"))) == nullptr ||

(exchange2 = OpenEvent(EVENT\_ALL\_ACCESS, FALSE, LPCSTR("exchange2"))) == nullptr ||

(serverAnswer = OpenEvent(EVENT\_ALL\_ACCESS, FALSE, LPCSTR("serverAnswer"))) == nullptr)

Sleep(defaultWait);

HANDLE mapFile;

// Открытие общей области памяти для общения клиента с сервером

while ((mapFile = OpenFileMapping(FILE\_MAP\_ALL\_ACCESS, FALSE, LPCSTR("MyShared"))) == nullptr)

Sleep(defaultWait);

//Проецируем файл в адресное пространство процесса

int \*pData = (int\*)MapViewOfFile(mapFile, FILE\_MAP\_READ | FILE\_MAP\_WRITE, 0, 0, 0);

wsprintfA(buf, "[Client] Студент №%d получил билет %d и начал готовиться.", studNumber, ticket);

showMessage(buf);

// Студент готовится:

Sleep(minStudentPrep + rand() % (maxStudentPrep - minStudentPrep));

// Когда готов, ожидает готовности сервера (преподавателя):

WaitForSingleObject(serverReady, INFINITE);

// Если сервер готов - садится отвечать:

wsprintfA(buf, "[Client] Студент №%d c билетом %d садится отвечать.",

studNumber, ticket);

showMessage(buf);

// Говорит серверу, кто это и какой билет:

pData[0] = studNumber;

pData[1] = ticket;

// Подает событие, что данные готовы:

SetEvent(exchange1);

// Ожидает ответа сервера:

WaitForSingleObject(serverAnswer, INFINITE);

wsprintfA(buf, "[Client] Студент №%d получил оценку %d.", pData[0], pData[1]);

showMessage(buf);

//Подает сигнал серверу, что область общей памяти свободна и можно принимать следующего

SetEvent(exchange2);

// Овобождение ресурсов, занятых клиентом

UnmapViewOfFile(pData);

CloseHandle(mapFile);

CloseHandle(serverAnswer);

CloseHandle(exchange1);

CloseHandle(exchange2);

CloseHandle(serverReady);

wsprintfA(buf, "[Client] Студент №%d выходит из аудитории.", studNumber);

showMessage(buf);

return 0;

}

// Функция потока преподавателя (сервер)

DWORD WINAPI teacherThread(PVOID param) {

char buf[256];

// Создание событий для синхронизации

HANDLE serverReady = CreateEvent(nullptr, FALSE, FALSE, LPCSTR("serverReady"));

HANDLE exchange1 = CreateEvent(nullptr, FALSE, FALSE, LPCSTR("exchange1"));

HANDLE exchange2 = CreateEvent(nullptr, FALSE, FALSE, LPCSTR("exchange2"));

HANDLE serverAnswer = CreateEvent(nullptr, FALSE, FALSE, LPCSTR("serverAnswer"));

// Создание общей области памяти для общения клиентов с сервером

HANDLE mapFile = CreateFileMapping(INVALID\_HANDLE\_VALUE, nullptr, PAGE\_READWRITE, 0, sizeof(int) \* 2,

LPCSTR("MyShared")); //Создать проекцию на файл в памяти (используется файл подкачки)

int \*pData = (int\*) MapViewOfFile(mapFile, FILE\_MAP\_READ | FILE\_MAP\_WRITE,

0, 0, 0); //Проецируем файл в адресное пространство процесса

// Пока экзамен не кончился:

while (!stopExam) {

// Установить событие что сервер готов

SetEvent(serverReady);

// Ожидаем пока кто-то из ожидающих клиентов заполнит общую область памяти

WaitForSingleObject(exchange1, INFINITE);

wsprintfA(buf, "[Server] Преподаватель начал принимать билет %d у студента №%d", pData[1], pData[0]);

showMessage(buf);

// Преподаватель принимает у студента рандомное время

Sleep(rand() % (maxTeacherCheck - minTeacherCheck) + minTeacherCheck);

pData[1] = rand() % (maxMark - minMark + 1) + minMark; // Выставляет оценку за экзамен

wsprintfA(buf, "[Server] Преподаватель поставил оценку %d студенту №%d", pData[1], pData[0]);

showMessage(buf);

// Устанавливает событие, что сервер ответил (экзамен сдан):

SetEvent(serverAnswer);

// Ожидает, пока клиент не подаст сигнал, что он прочитал переданные данные

WaitForSingleObject(exchange2, INFINITE);

}

// Освобождение ресурсов, занятых сервером

UnmapViewOfFile(pData);

CloseHandle(mapFile);

CloseHandle(serverAnswer);

CloseHandle(exchange1);

CloseHandle(exchange2);

CloseHandle(serverReady);

return 0;

}

int main() {

DWORD idThread, tmp;

setlocale(LC\_ALL, "Russian");

InitializeCriticalSection(&cs);

// Создать поток-сервер

HANDLE prep = CreateThread(nullptr, 0, teacherThread, nullptr, 0, &idThread);

int n;

cout << "Введите количество студентов: ";

do {

cin >> n;

if (n <= 0)

cout << "Правильно. Какие очные экзамены во время пандемии?" << endl;

else if (n >= maxStudents)

cout << "У нас на програмной инженерии точно не больше " << maxStudents << " человек. Видимо, Вы ошиблись."

<< endl;

cout << "Введите повторно, пожалуйста: ";

} while (n <= 0 || n >= maxStudents);

auto \*st = new HANDLE[n];

//Создать потоки студентов

cout << "Студенты начинают входить в аудиторию и готовиться." << endl;

for (int i = 0; i < n; i++) {

tmp = (rand() % (countTickets + 1) + 1) << 16; //случайные номера билетов

tmp |= (i + 1); //и номера студентов упаковать в передаваемый параметр

st[i] = CreateThread(NULL, 0, studentThread, (LPVOID) tmp, NULL, &idThread);

Sleep(rand() % (maxStudentEnters - minStudentEnters) + minStudentEnters);

}

// Дожидаемся завершения всех потоков-клиентов

if (n <= MAXIMUM\_WAIT\_OBJECTS) // Либо всех сразу, если возможно

WaitForMultipleObjects(n, st, TRUE, INFINITE);

else { // либо по частям

int j = n;

int k = 0;

while (j) {

if (j <= MAXIMUM\_WAIT\_OBJECTS) {

WaitForMultipleObjects(j, &st[k], TRUE, INFINITE);

j = 0;

} else {

WaitForMultipleObjects(MAXIMUM\_WAIT\_OBJECTS, &st[k], TRUE, INFINITE);

k += MAXIMUM\_WAIT\_OBJECTS;

j -= MAXIMUM\_WAIT\_OBJECTS;

}

}

}

// Выставляем флаг окончания экзамена

stopExam = true;

cout << "Экзамен окончен. Всем спасибо." << endl;

// Подчищаем за собой

DeleteCriticalSection(&cs);

delete[] st;

system("pause");

return 0;

}

**Тестирование**

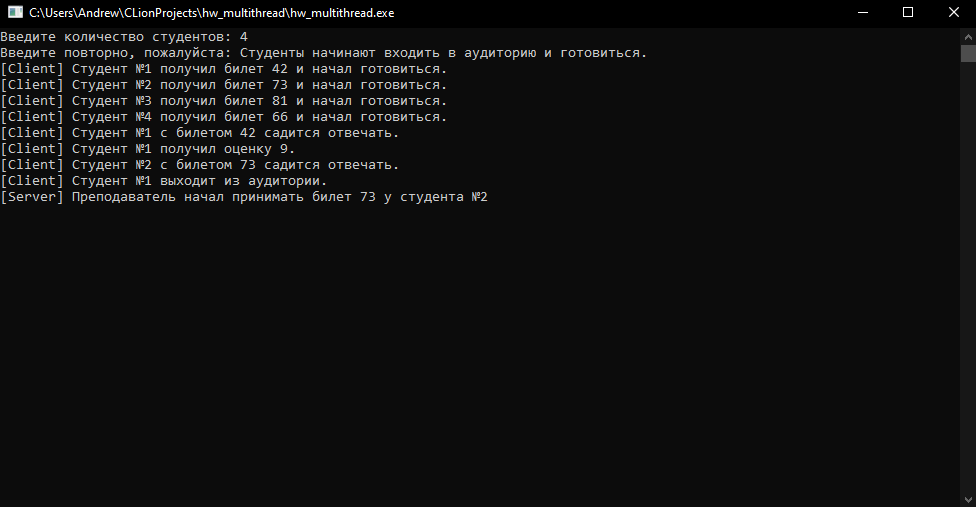


Рисунок 2 ­– Результат создания потоков-клиентов.

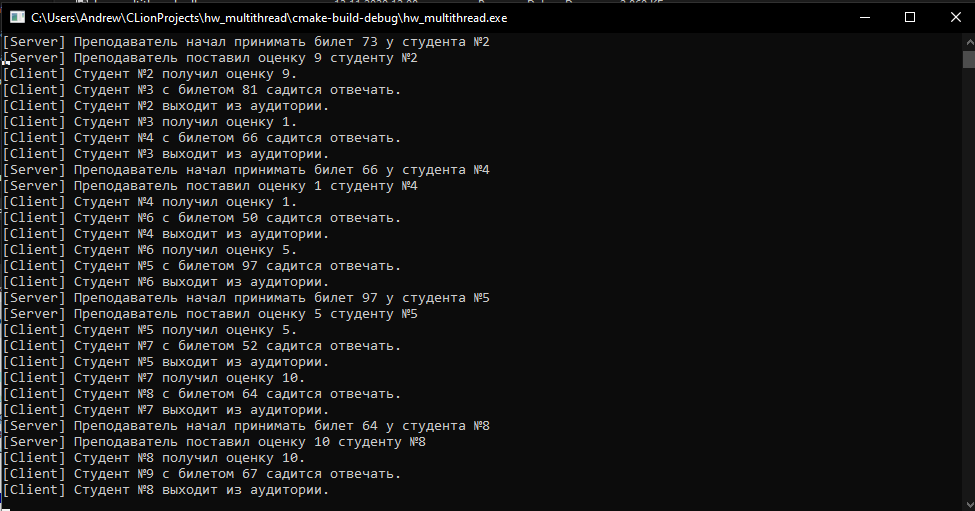


Рисунок 3 ­– Процесс принятия экзамена

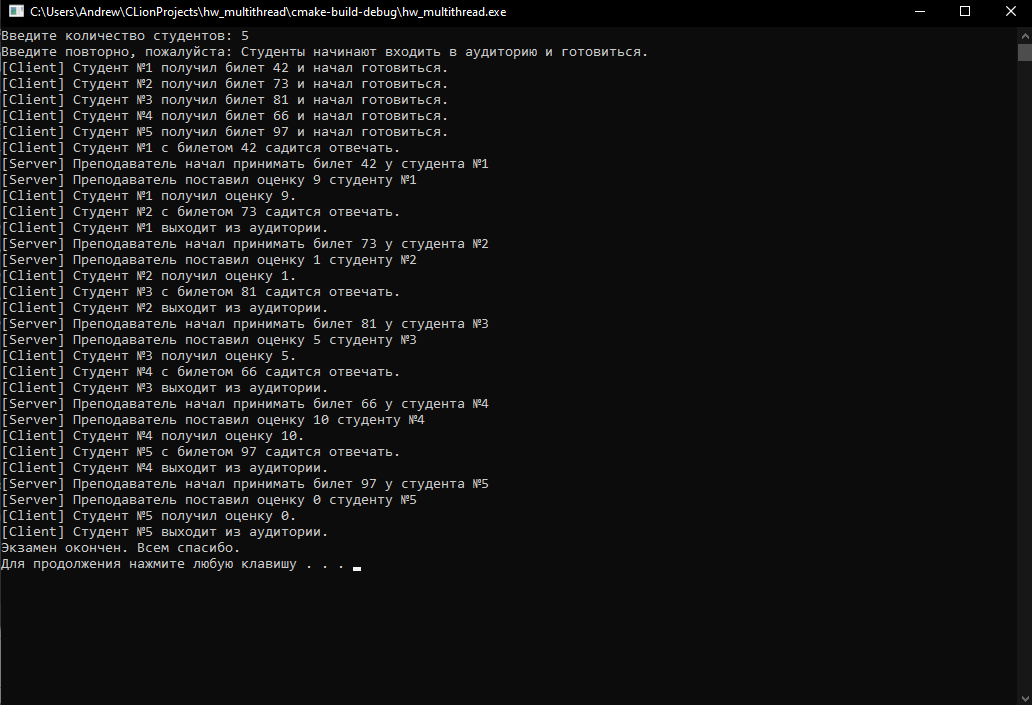


Рисунок 4 ­– Результат выполнения программы

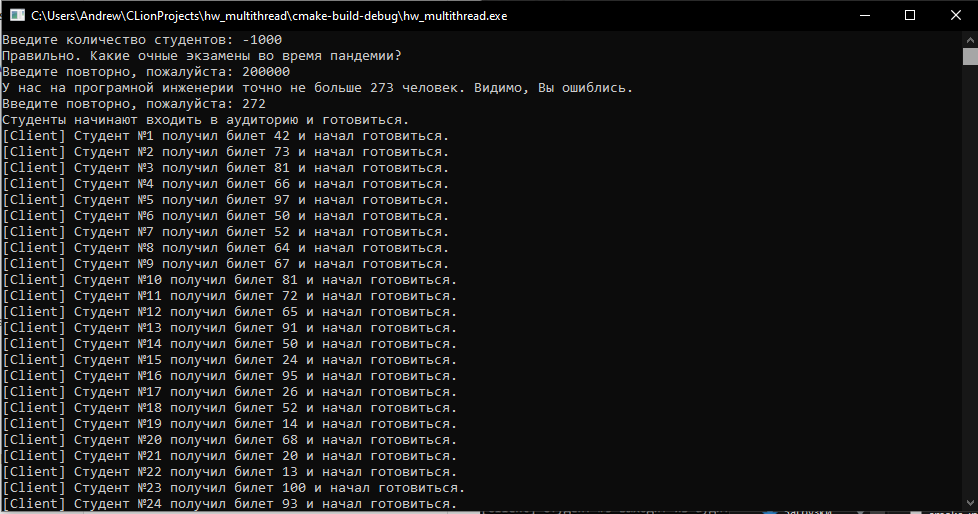


Рисунок 5 ­– Обработка некорректного ввода

**Список используемых источников**

1. Википедия (2020) «Клиент-сервер» ([https://ru.wikipedia.org/wiki/Клиент\_—\_сервер](https://ru.wikipedia.org/wiki/%D0%9A%D0%BB%D0%B8%D0%B5%D0%BD%D1%82_%E2%80%94_%D1%81%D0%B5%D1%80%D0%B2%D0%B5%D1%80)). Просмотрено: 10.11.2020
2. Habr (2020) «Клиент-сервер шаг — за — шагом, от однопоточного до многопоточного (Client-Server step by step)» (<https://habr.com/ru/post/330676/>). Просмотрено: 10.11.2020
3. Metanit (2020) «Многопоточное клиент-серверное приложение TCP» (<https://metanit.com/sharp/net/4.3.php>). Просмотрено: 10.11.2020
4. Cyberforum (2020) «Простой клиент-сервер с многопоточностью» (<https://www.cyberforum.ru/java-networks/thread1557122.html>). Просмотрено: 10.11.2020
5. Docs Microsoft (2020) «Creating Threads» (<https://docs.microsoft.com/en-us/windows/win32/procthread/creating-threads>). Просмотрено: 10.11.2020
6. Shalatov Grost17 «Потоки (threads) в WinAPI» (<http://shatalov.ghost17.ru/winapi/threads.html>). Просмотрено: 10.11.2020
7. Легалов А.И.(2020) «Многопоточность. Простая многопоточная программа. Основные функции» (<http://softcraft.ru/edu/comparch/practice/thread/01-simple/>). Просмотрено: 18.10.2020
8. Легалов А.И.(2020) «Многопоточность. Синхронизация потоков. Методы синхронизации» (<http://softcraft.ru/edu/comparch/practice/thread/02-sync/>). Просмотрено: 18.10.2020