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 клиентсерверного взаимодействия в Windows»

Выполнила: Иванова А.А.

Группа: 6206-010302D

Самара, 2024

**Задание на лабораторную работу**

Использовать семафоры (мьютексы) для синхронизации работы процессов (потоков) в процессе реализации задания на лабораторную работу. При необходимости также можно использовать  
события. Требуется реализовать консольный вариант. Базовый вариант реализации – Win32 API в Windows. Дополнительные варианты реализации при использовании других операционных систем  
и языков программирования – на усмотрение студента. Желательно предусмотреть для всех заданий событие, при котором соответствующий процесс завершает свою работу, если это возможно.

Вариант 4.  
Автомобильный дилерский центр. Требуется написать клиентсерверное приложение, моделирующее взаимодействие сервера  
центра и продавцов-клиентов. На сервере хранится база автомобилей с их характеристиками. Требуется передавать часть этой базы  
по запросу клиента и, при необходимости, вносить в нее изменения. Взаимодействовать с базой данный в 1 момент времени может  
только 1 клиент. При изменении базы данных все клиенты должны  
получать сообщение – «Произошло обновления БД!».

**Текст программы**

Сервер

#include <iostream>

#include <string>

#include <vector>

#include <sstream>

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

#include<thread>

#include<winsock2.h>

#pragma warning(disable: 4996)//для inet\_addr

using namespace std;

int n;

vector <SOCKET> clientSocket(n);

HANDLE hMutex = CreateMutexW(NULL, FALSE, //вызывающий поток не получает права владения мьютексом

L"MyMutex");

class Car {

public:

string brand;

string model;

string year;

string country;

string price;

string color;

Car(string brand, string model, string year, string country, string price, string color) {

this->brand = brand; //обращение к полям класса

this->model = model;

this->year = year;

this->country = country;

this->price = price;

this->color = color;

}

};

vector<Car> Database = {

Car("Tayota","Camry","2019","Japan","1000000","Blue"),

Car("Porsche","956","1984","Germany","1100000","Black"),

Car("BMW","M8","2019","Germany","1200000","Green"),

Car("Audi","A4","1994","Germany","1200000","Grey"),

Car("Hyundai","Solaris","2001","Korea","1200000","White"),

Car("Kia","Rio","2005","Korea","1300000","Black"),

Car("Audi","Q5","2008","Germany","1400000","Black"),

Car("Lada","Vesta","2015","Russia","1450000","White"),

};

string getData(string name, string characteristic) {

string msg\_foundCar = "";

if (name == "year") {

for (Car car : Database) {

if (characteristic == car.year) {

msg\_foundCar += "Автомобиль найден: " + car.brand + " " + car.model + " " + car.year + " " + car.country + " " + car.price + " " + car.color + "\n";

}

}

}

else if (name == "price") {

for (Car car : Database) {

if (characteristic == car.price) {

msg\_foundCar += "Автомобиль найден: " + car.brand + " " + car.model + " " + car.year + " " + car.country + " " + car.price + " " + car.color + "\n";

}

}

}

else if (name == "brand") {

for (Car car : Database) {

if (characteristic == car.brand) {

msg\_foundCar += "Автомобиль найден: " + car.brand + " " + car.model + " " + car.year + " " + car.country + " " + car.price + " " + car.color + "\n";

}

}

}

else if (name == "model") {

for (Car car : Database) {

if (characteristic == car.model)

{

msg\_foundCar += "Автомобиль найден: " + car.brand + " " + car.model + " " + car.year + " " + car.country + " " + car.price + " " + car.color + "\n";

}

}

}

else if (name == "country") {

for (Car car : Database) {

if (characteristic == car.country) {

msg\_foundCar += "Автомобиль найден: " + car.brand + " " + car.model + " " + car.year + " " + car.country + " " + car.price + " " + car.color + "\n";

}

}

}

else if (name == "color") {

for (Car car : Database) {

if (characteristic == car.color)

{

msg\_foundCar += "Автомобиль найден: " + car.brand + " " + car.model + " " + car.year + " " + car.country + " " + car.price + " " + car.color + '\n';

}

}

}

else {

msg\_foundCar += "Автомобиль не найден!";

}

return msg\_foundCar;

}

void changeData(string brand, string model, string year, string country, string price, string color) {

Database.push\_back(Car(brand, model, year, country, price, color));

}

void function(int i) {

bool flag = true;

do {

char buffer[1024];

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

if (bytesReceived > 0) {

string data(buffer, bytesReceived); //Создание строки data из данных в буфере buffer, используя количество принятых байт

stringstream ss(data); //Создание потока строк ss из строки data

vector<string> words; // Объявление вектора строк words

string word;

while (ss >> word) { //Чтение слов из потока

words.push\_back(word);

}

while (!ss.eof() && isspace(ss.peek())) { //Пропуск пробелов после последнего слова

ss.get();

}

if (words[0] == "getData") {

string buf = getData(words[1], words[2]);

if (buf == "") {

buf = "Автомобиль не найден!\n";

}

char msg\_buffer[1024];

strcpy(msg\_buffer, buf.c\_str()); // Копирование содержимого строки buf в массив символов msg\_buffer

int bytesSent = send(clientSocket[i], msg\_buffer, sizeof(msg\_buffer), NULL);

if (bytesSent == -1) {

cerr << "Ошибка отправки данных\n";

}

}

else if (words[0] == "changeData") {

changeData(words[1], words[2], words[3], words[4], words[5], words[6]);

string buf = "БД обновлена!\n";

char msg\_buffer[1024];

strcpy(msg\_buffer, buf.c\_str());

int bytesSent = send(clientSocket[i], msg\_buffer, sizeof(msg\_buffer), NULL);

if (bytesSent == -1) {

cerr << "Ошибка отправки данных\n";

}

}

else if (words[0] == "finish") {

flag = false;

string msg\_buffer = "До свидания!\n";

cout << msg\_buffer;

int bytesSent = send(clientSocket[i], msg\_buffer.c\_str(), sizeof(msg\_buffer), NULL);

closesocket(clientSocket[i]);

}

else {

char msg\_buffer[1024];

string buf = "Сервер ожидает запроса: ";

strcpy(msg\_buffer, buf.c\_str());

int bytesSent = send(clientSocket[i], msg\_buffer, sizeof(msg\_buffer), NULL);

if (bytesSent == -1) {

cerr << "Ошибка отправки данных\n";

}

}

}

else {

cout << "Ошибка передачи сообщения" << endl;

if (closesocket(clientSocket[i] == SOCKET\_ERROR))

cerr << "Не удалось прервать соединение.\n Код ошибки: " << WSAGetLastError();

}

} while (flag);

}

int main()

{

setlocale(LC\_ALL, "Russian");

//Загрузка библиотеки

WSAData wsaData; //создаем структуру для загрузки

WORD DLLVersion = MAKEWORD(2, 1); // Версия библиотеки winsock

if (WSAStartup(DLLVersion, &wsaData) != 0) { // проверка подключения

cerr << "Error: failed to link library.\n";

return 1;

}

//Заполняем информацию об адресе сокета

SOCKADDR\_IN addr;

int sizeOfAddr = sizeof(addr);

addr.sin\_addr.s\_addr = inet\_addr("127.0.0.1");

addr.sin\_port = htons(1111);

addr.sin\_family = AF\_INET;

SOCKET sListen = socket(AF\_INET, SOCK\_STREAM, 0); //сокет для прослушивания порта,семейство интернет протоколов,протокол

if (bind(sListen, (SOCKADDR\*)&addr, sizeOfAddr) == SOCKET\_ERROR) {

//привязка адреса сокету

printf("Error bind %d\n", WSAGetLastError());

if (closesocket(sListen) == SOCKET\_ERROR)

cerr << "Failed to terminate connection.\n Error code: " << WSAGetLastError();

WSACleanup();

return 1;

}

//подключение прослушивания с максимальной очередью

if (listen(sListen, SOMAXCONN) == SOCKET\_ERROR) {

cout << "Listen failed;\n";

if (closesocket(sListen) == SOCKET\_ERROR)

cerr << "Failed to terminate connection.\n Error code: " << WSAGetLastError();

WSACleanup();

return 1;

}

//задаем информацию для окна открытия

STARTUPINFOW si;//структура

PROCESS\_INFORMATION pi;// структура с информацией о процессе

ZeroMemory(&si, sizeof(si));

si.cb = sizeof(si);// указываем размер

ZeroMemory(&pi, sizeof(pi));

DWORD dwThreadId = 0;

//создаём новые окна для клиентов

cout << "Введите кол-во клиентов: ";

cin >> n;

for (uint8\_t i = 0; i < n; i++)

{

if (!CreateProcessW(L"C:\\Users\\Asus\\Desktop\\24 инф\\LR\_5\_client\\x64\\Debug\\LR\_5\_client.exe", // module name

NULL, //аргумент командной строки

NULL, //дескриптор процесса не наследуется

NULL, //дескриптор потока не наследуется

FALSE, //будут ли новому процессу наследоваться дескрипторы

CREATE\_NEW\_CONSOLE, //создает новое окно консоли для нового процесса

NULL, //переменные среду окружения

NULL, //полный путь к текущему рабочему каталогу

&si,//указатель на STARTUPINFO

&pi)) { //указатель на PROCESS\_INFORMATION

cout << GetLastError();

}

Sleep(10);

clientSocket.push\_back(accept(sListen, (SOCKADDR\*)&addr, nullptr));

// возвращает дескриптор потока, который действителен только внутри нашего процесса

CreateThread(NULL, //атрибуты защиты

NULL, // начальный размер стека для потока

(LPTHREAD\_START\_ROUTINE)function, // указатель на адрес функции, которая будет выполняться как отдельная задача

(LPVOID)(i), // параметр, который будет передан внутрь функции потока

NULL, // исполнение начинается немедленно

NULL); // значения флагов, которые определяют поведение потока при его создании

}

system("pause");

if (closesocket(sListen) == SOCKET\_ERROR) {

cerr << "The connection could not be terminated.\n Error code: " << WSAGetLastError();

}

WSACleanup();

return 0;

}

Клиент

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

#include <iostream>

#include <thread>

#include<string>

#include<winsock2.h>

#pragma warning(disable: 4996)//для inet\_addr

using namespace std;

void function(SOCKET Connection) {

while (true) {

char buffer[1024];

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

if (bytesReceived > 0) {

if (strcmp(buffer, "До свидания!") == 0) {

return;

}

else {

cout << buffer;

}

}

else return;

}

}

int main()

{

setlocale(LC\_ALL, "Russian");

cout << "Привет! Я Клиент!\n" << endl;

cout << "getData - поиск нужного автомобиля по конкретной характеристике" << endl;

cout << "changeData - внести изменение в базу данных" << endl;

cout << "finish - завершение работы клиента\n" << endl;

cout << "Характеристики базы данных: brand, model, year, country, price, color" << endl;

//загружаем WSAStartup

WSAData wsaData; //создаем структуру для загрузки

WORD DLLVersion = MAKEWORD(2, 1);

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

cout << "Error" << endl;

exit(1);

}

//Заполняем информацию об адресе сокета

SOCKADDR\_IN addr;

addr.sin\_addr.s\_addr = inet\_addr("127.0.0.1"); //адрес

addr.sin\_port = htons(1111); //порт

addr.sin\_family = AF\_INET; //семейство протоколов

//сокет для прослушивания порта

SOCKET Connection = socket(AF\_INET, SOCK\_STREAM, 0);//возвращает целочисленное значение, идентифицирующее созданный сокет аналогично дескриптору файла

//проверка на подключение к серверу

if (connect(Connection, (SOCKADDR\*)&addr, sizeof(addr))) {

string code\_error = to\_string(WSAGetLastError());

string error\_msg = "Установить соединение с сервером не удалось! Код ошибки: ";

string msg\_of\_error = error\_msg + code\_error;

cout << msg\_of\_error;

closesocket(Connection); // Закрыть сокет

WSACleanup(); // Очистить Winsock

return 1;

}

thread t(function, Connection);

HANDLE hMutex = OpenMutexW(SYNCHRONIZE, FALSE, L"MyMutex"); //открывает существующий мьютекс с именем "MyMutex" для синхронизации

WaitForSingleObject(hMutex, INFINITE); //функция, которая ждёт, когда объект, хэндл которого был передан первым аргументом, перейдёт в сигнальное состояние

string message;

while (message != "finish") {

getline(cin, message);

// Отправка сообщения

int bytesSent = send(Connection, message.c\_str(), (int)strlen(message.c\_str()), 0);

if (bytesSent == -1) {

cerr << "Error sending data\n";

}

}

ReleaseMutex(hMutex); // освобождает владение указанным объектом мьютекса

t.join(); // заставляет текущий поток ждать, пока завершит выполнение другой поток

if (closesocket(Connection) == SOCKET\_ERROR)

cerr << "Failed to terminate connection.\n Error code: " << WSAGetLastError();

WSACleanup();

return 0;

}