Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина: Операционные среды и системное программирование

ОТЧЕТ

к лабораторной работе №6

на тему

**СРЕДСТВА СИНХРОНИЗАЦИИ И ВЗАИМНОГО ИСКЛЮЧЕНИЯ (WINDOWS)**

Студент Е. А. Сироткин

Преподаватель Н. Ю. Гриценко

Минск 2023

СОДЕРЖАНИЕ

[1 Цель работы 3](#_Toc146752068)

[2 Теоретические сведения 4](#_Toc146752069)

[3 Результат выполнения 5](#_Toc146752070)

[Заключение 7](#_Toc146752071)

[Список использованных источников 8](#_Toc146752072)

[Приложение А (обязательное) Листинг кода 9](#_Toc146752073)

1. **ЦЕЛЬ РАБОТЫ**

Цель лабораторной работы заключается в изучении структуры и принципов функционирования механизмов синхронизации и взаимного исключения в операционной системе *Windows*, приобретении практических навыков использования средств синхронизации и механизмов взаимного исключения, а также использовании средств синхронизации для управления доступом к разделяемой памяти.

1. **ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**

Средства синхронизации и взаимного исключения в операционных системах *Windows* играют важную роль в обеспечении корректной работы многозадачных приложений и обеспечении безопасного доступа к общим ресурсам. Они используются для синхронизации выполнения потоков и процессов, предотвращения конфликтов доступа к общим данным и обеспечения взаимного исключения, чтобы избежать состояний гонки и обеспечить целостность данных. Одними из основных средств синхронизации и взаимного исключения в операционных системах *Windows* являются мьютексы (*mutexes*), семафоры (*semaphores*), критические секции (*critical* *sections*) и события (*events*).

Критическая секция – это часть программы, в которой осуществляется доступ к разделяемым данным. Чтобы исключить эффект гонок по отношению к некоторому ресурсу, необходимо обеспечить, чтобы в каждый момент в критической секции, связанной с этим ресурсом, находился максимум один процесс. Этот прием называют взаимным исключением [1].

Объект критическая секция обеспечивает синхронизацию. Этим объектом может владеть только один поток, что и обеспечивает синхронизацию. Для работы с критическими секциями есть ряд функций API и тип данных *CRITICAL\_SECTION*. Для использования критической секции нужно создать переменную данного типа, и проинициализировать ее перед использованием с помощью функции *InitializeCriticalSection*. Для того, чтобы войти в секцию нужно вызвать функцию *EnterCriticalSection*, а после завершения работы *LeaveCriticalSection* [2].

Функция *DeleteCriticalSection* используется для удаления (освобождения) объекта критической секции, который используется для синхронизации доступа нескольких потоков к общим ресурсам в многозадачных приложениях. Когда объект критической секции больше не нужен, его можно удалить с помощью *DeleteCriticalSection*, что освобождает ресурсы, занимаемые этим объектом. Это важно для предотвращения утечек ресурсов и обеспечения правильной работ приложений, использующих многозадачность.

1. **РЕЗУЛЬТАТ ВЫПОЛНЕНИЯ**

В ходе выполнения данной лабораторной работы было разработано приложение, производящее вычисления в различных потоках. Интерфейс представляет собой две кнопки, по нажатию на которые происходят вычисления в одном и четырех потоках. При вычислении в четырех потоках используется разделяемая память, доступ к которой обеспечивается средствами синхронизации (рисунок 1).

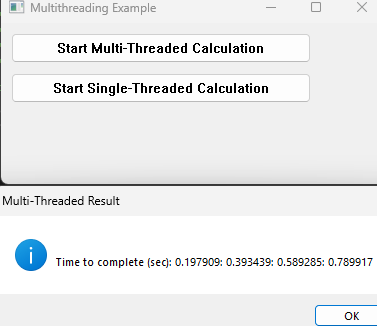


Рисунок 1– Приложение

# ЗАКЛЮЧЕНИЕ

В ходе выполнения данной лабораторной работы были освоены аспекты использования средств синхронизации и взаимного исключения. В рамках работы над приложением была добавлена возможность выполнения вычислений в четырех потоках, которые имеют доступ к разделяемой памяти. Были использованы средства для синхронизации доступа к разделяемой памяти.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Средства синхронизации и взаимодействия процессов [Электронный ресурс]. – Режим доступа: https://studfile.net/preview/7153358/page:11/.
2. Что такое критическая секция [Электронный ресурс]. – Режим доступа: https://firststeps.ru/mfc/msdn/r.php?80.

# ПРИЛОЖЕНИЕ А (обязательное) Листинг кода

Листинг 1 – Файл *Lab6.cpp*

#pragma comment(linker,"\"/manifestdependency:type='win32' \

name='Microsoft.Windows.Common-Controls' version='6.0.0.0' \

processorArchitecture='\*' publicKeyToken='6595b64144ccf1df' language='\*'\"")

#include <windows.h>

#include <vector>

#include <thread>

#include <iostream>

#include <string>

using namespace std;

unsigned long long results[4] = { 0,0,0,0 };

unsigned long long a, b, c, d;

unsigned long long result1 = 0, result2 = 0, result3 = 0, result4 = 0, result = 0;

//unsigned long long result = 0;

const unsigned long long ARRAY\_SIZE = 200000000;

CRITICAL\_SECTION section;

HANDLE writeCompleteEvent;

HANDLE Thread1Complete;

HANDLE Thread2Complete;

HANDLE Thread3Complete;

HANDLE Thread4Complete;

int flag = 0;

vector<unsigned long long> data1(ARRAY\_SIZE);

LRESULT CALLBACK WindowProcedure(HWND, UINT, WPARAM, LPARAM);

void CreateThreads();

void CalculateSingleThread();

double GetTimeElapsed(LARGE\_INTEGER startTime, LARGE\_INTEGER endTime);

DWORD WINAPI SingleThreadCalculation(LPVOID lpParam);

DWORD WINAPI MultiThreadMessage(LPVOID lpParam);

double PCFreq;

double Thread1TimeElapsed, Thread2TimeElapsed, Thread3TimeElapsed, Thread4TimeElapsed;

int WINAPI WinMain(HINSTANCE hInst, HINSTANCE hPrevInst, LPSTR args, int ncmdshow) {

for (unsigned long long i = 0; i < ARRAY\_SIZE; ++i) {

data1[i] = i;

}

a = 0;

b = ARRAY\_SIZE / 4;

c = ARRAY\_SIZE / 2;

d = 3 \* (ARRAY\_SIZE / 4);

LARGE\_INTEGER frequency;

QueryPerformanceFrequency(&frequency);

PCFreq = double(frequency.QuadPart) / 1000.0;

WNDCLASS wc = { 0 };

wc.hbrBackground = (HBRUSH)COLOR\_WINDOW;

wc.hCursor = LoadCursor(NULL, IDC\_ARROW);

wc.hInstance = hInst;

wc.lpszClassName = L"MultithreadingApp";

wc.lpfnWndProc = WindowProcedure;

if (!RegisterClassW(&wc)) {

return -1;

}

CreateWindowW(L"MultithreadingApp", L"Multithreading Example", WS\_OVERLAPPEDWINDOW | WS\_VISIBLE, 100, 100, 400, 200, NULL, NULL, NULL, NULL);

MSG msg = { 0 };

InitializeCriticalSection(&section);

while (GetMessage(&msg, NULL, NULL, NULL)) {

TranslateMessage(&msg);

DispatchMessage(&msg);

}

DeleteCriticalSection(&section);

return 0;

}

LRESULT CALLBACK WindowProcedure(HWND hWnd, UINT msg, WPARAM wp, LPARAM lp) {

switch (msg) {

case WM\_COMMAND:

if (wp == 1) {

CreateThreads();

}

else if (wp == 2) {

CalculateSingleThread();

}

break;

case WM\_CREATE:

CreateWindow(L"button", L"Start Multi-Threaded Calculation", WS\_VISIBLE | WS\_CHILD, 10, 10, 300, 30, hWnd, (HMENU)1, NULL, NULL);

CreateWindow(L"button", L"Start Single-Threaded Calculation", WS\_VISIBLE | WS\_CHILD, 10, 50, 300, 30, hWnd, (HMENU)2, NULL, NULL);

break;

case WM\_DESTROY:

PostQuitMessage(0);

break;

default:

return DefWindowProcW(hWnd, msg, wp, lp);

}

return 0;

}

DWORD WINAPI ThreadFunction1(LPVOID aboba) {

LARGE\_INTEGER startTime, endTime;

QueryPerformanceCounter(&startTime);

EnterCriticalSection(&section);

for (unsigned long long i = a; i < b; ++i) {

result += data1[i];

}

results[0] = result;

LeaveCriticalSection(&section);

QueryPerformanceCounter(&endTime);

Thread1TimeElapsed = GetTimeElapsed(startTime, endTime);

flag++;

MultiThreadMessage(NULL);

return result;

}

DWORD WINAPI ThreadFunction2(LPVOID aboba) {

LARGE\_INTEGER startTime, endTime;

QueryPerformanceCounter(&startTime);

EnterCriticalSection(&section);

for (unsigned long long i = b; i < c; ++i) {

result += data1[i];

}

results[1] = result;

LeaveCriticalSection(&section);

QueryPerformanceCounter(&endTime);

Thread2TimeElapsed = GetTimeElapsed(startTime, endTime);

flag++;

MultiThreadMessage(NULL);

return result;

}

DWORD WINAPI ThreadFunction3(LPVOID aboba) {

LARGE\_INTEGER startTime, endTime;

QueryPerformanceCounter(&startTime);

EnterCriticalSection(&section);

for (unsigned long long i = c; i < d; ++i) {

result += data1[i];

}

results[2] = result;

LeaveCriticalSection(&section);

QueryPerformanceCounter(&endTime);

Thread3TimeElapsed = GetTimeElapsed(startTime, endTime);

flag++;

MultiThreadMessage(NULL);

return result;

}

DWORD WINAPI ThreadFunction4(LPVOID aboba) {

LARGE\_INTEGER startTime, endTime;

QueryPerformanceCounter(&startTime);

EnterCriticalSection(&section);

for (unsigned long long i = d; i < ARRAY\_SIZE; ++i) {

result += data1[i];

}

results[3] = result;

LeaveCriticalSection(&section);

QueryPerformanceCounter(&endTime);

Thread4TimeElapsed = GetTimeElapsed(startTime, endTime);

flag++;

MultiThreadMessage(NULL);

return result;

}

void CreateThreads() {

HANDLE threads[4];

threads[0] = CreateThread(NULL, 0, ThreadFunction1, NULL, 0, NULL);

threads[1] = CreateThread(NULL, 0, ThreadFunction2, NULL, 0, NULL);

threads[2] = CreateThread(NULL, 0, ThreadFunction3, NULL, 0, NULL);

threads[3] = CreateThread(NULL, 0, ThreadFunction4, NULL, 0, NULL);

}

void CalculateSingleThread() {

HANDLE SingleThread = CreateThread(NULL, 0, SingleThreadCalculation, NULL, 0, NULL);

}

double GetTimeElapsed(LARGE\_INTEGER startTime, LARGE\_INTEGER endTime) {

return (double(endTime.QuadPart - startTime.QuadPart) / PCFreq) / 1000.0;

}

DWORD WINAPI SingleThreadCalculation(LPVOID lpParam) {

LARGE\_INTEGER startTime, endTime;

QueryPerformanceCounter(&startTime);

unsigned long long result\_2 = 0;

for (unsigned long long i = 0; i < ARRAY\_SIZE; ++i) {

result\_2 += data1[i];

}

QueryPerformanceCounter(&endTime);

double timeElapsed = GetTimeElapsed(startTime, endTime);

wstring resultText = L"\nTime to complete (sec): " + to\_wstring(timeElapsed);

MessageBoxW(NULL, resultText.c\_str(), L"Single-Threaded Result", MB\_OK | MB\_ICONINFORMATION);

return 0;

}

DWORD WINAPI MultiThreadMessage(LPVOID lpParam) {

if (flag != 4) {

return 0;

}

unsigned long long totalSum = 0;

for (int i = 0; i < 4; i++) {

totalSum += results[i];

}

wstring resultText = L"\nTime to complete (sec): " + to\_wstring(Thread1TimeElapsed) + L": " + to\_wstring(Thread2TimeElapsed) + L": " + to\_wstring(Thread3TimeElapsed) + L": " + to\_wstring(Thread4TimeElapsed);

MessageBoxW(NULL, resultText.c\_str(), L"Multi-Threaded Result", MB\_OK | MB\_ICONINFORMATION);

result = 0;

flag = 0;

for (int i = 0; i < 4; i++) {

results[i] = 0;

}

return 0;

}