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). ИЗУЧЕНИЕ И ИСПОЛЬЗОВАНИЕ СРЕДСТВ СИНХРОНИЗАЦИИ И ВЗАИМНОГО ИСКЛЮЧЕНИЯ**

Студент А. С. Сорокин

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**

Исследовать и освоить средства синхронизации и взаимного исключения в операционной системе *Windows*. Получить практические навыки использования средств синхронизации, таких как мьютексы, семафоры, и критические секции, для обеспечения взаимной исключительности между потоками.

1. **ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**

Средства синхронизации и взаимного исключения в операционной системе Windows предназначены для координации выполнения потоков, обеспечения правильного порядка доступа к разделяемым ресурсам и предотвращения состояний гонки.

Мьютекс (*mutex*) представляет собой объект, который может находиться в двух состояниях: заблокированном и разблокированном. Он используется для обеспечения эксклюзивного доступа к разделяемым ресурсам. Если мьютекс заблокирован одним потоком, то другие потоки, пытающиеся получить доступ к нему, будут ожидать его освобождения. [1]

Семафор (*semaphore*) является счетчиком, который контролирует доступ к определенному количеству ресурсов. Он может быть использован для ограничения количества потоков, которые могут одновременно получить доступ к разделяемому ресурсу. [2]

Критическая секция (*critical section*) представляет собой блок кода, который может быть выполнен только одним потоком одновременно. Она используется для защиты критических участков кода от одновременного доступа нескольких потоков.

Событие (*event*) представляет собой объект, который может находиться в двух состояниях: сигнальном и несигнальном. Поток может ожидать наступления события и продолжить выполнение только тогда, когда событие станет сигнальным.

Для работы с средствами синхронизации и взаимного исключения в Windows обычно используются функции и структуры из *Windows API*, такие как *CreateMutex, CreateSemaphore, EnterCriticalSection, WaitForSingleObject* и другие.

Основная цель изучения и использования этих средств состоит в том, чтобы обеспечить корректное и безопасное взаимодействие между потоками, избежать состояний гонки и гарантировать правильное использование разделяемых ресурсов в многопоточных приложениях, работающих под управлением операционной системы *Windows*.

1. **РЕЗУЛЬТАТ ВЫПОЛНЕНИЯ**

В результате выполнения лабораторной работы было создано приложение по вычислению логарифма, используя мьютексы для обеспечения целостности (рисунок 1).

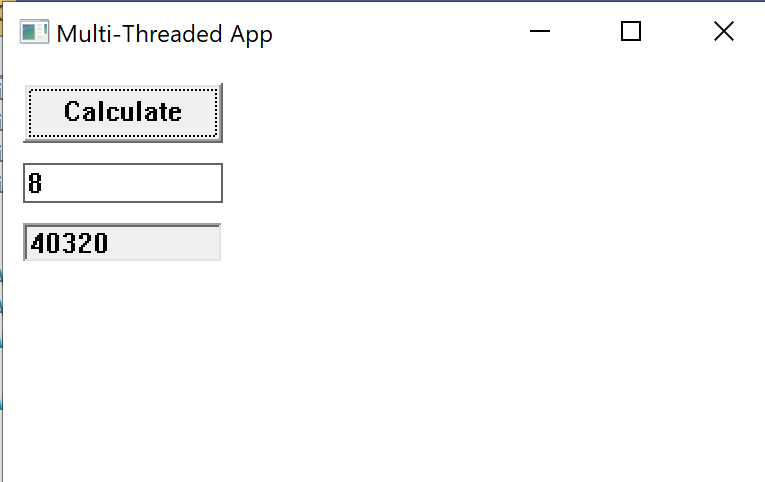


Рисунок 1– Главное окно

# ЗАКЛЮЧЕНИЕ

В ходе выполнения данной лабораторной работы были получены необходимые теоретические сведения и практические навыки по работе с средствами синхронизации и взаимного исключения в операционной системе Windows. Были изучены и использованы различные средства, такие как мьютексы, семафоры и критические секции, для обеспечения правильного взаимодействия и синхронизации потоков.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Объекты мьютексов [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/ru-ru/windows/win32/sync/mutex-objects.
2. Объекты семафора [Электронный ресурс]. – Режим доступа: <https://learn.microsoft.com/ru-ru/windows/win32/sync/using-semaphore-objects>.
3. Критические разделы [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/ru-ru/windows/win32/sync/using-critical-section-objects.

# ПРИЛОЖЕНИЕ А (обязательное) Листинг кода

Листинг 1 – Файл *Lab\_6.cpp*

#include <windows.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

HANDLE hCalculationMutex; // Мьютекс для синхронизации доступа к переменной-результату

HANDLE hCalculationComplete; // Событие для сигнализации о завершении вычислений

DWORD calculationResult; // Переменная для хранения результата вычислений

DWORD WINAPI FactorialThread(LPVOID param)

{

int n = \*((int\*)param);

DWORD result = 1;

for (int i = 1; i <= n; i++)

{

result \*= i;

}

// Захват мьютекса перед записью результата

WaitForSingleObject(hCalculationMutex, INFINITE);

calculationResult = result;

// Освобождение мьютекса и сигнализация о завершении вычислений

ReleaseMutex(hCalculationMutex);

SetEvent(hCalculationComplete);

return 0;

}

LRESULT CALLBACK WindowProc(HWND hwnd, UINT uMsg, WPARAM wParam, LPARAM lParam)

{

switch (uMsg)

{

case WM\_CREATE:

{

CreateWindow(TEXT("BUTTON"), TEXT("Calculate"), WS\_CHILD | WS\_VISIBLE, 10, 10, 100, 30, hwnd, (HMENU)1, NULL, NULL);

CreateWindow(TEXT("EDIT"), NULL, WS\_CHILD | WS\_VISIBLE | ES\_NUMBER | WS\_BORDER, 10, 50, 100, 20, hwnd, (HMENU)2, NULL, NULL);

CreateWindowEx(WS\_EX\_CLIENTEDGE, TEXT("EDIT"), NULL, WS\_CHILD | WS\_VISIBLE | ES\_READONLY, 10, 80, 100, 20, hwnd, (HMENU)3, NULL, NULL);

hCalculationMutex = CreateMutex(NULL, FALSE, NULL);

hCalculationComplete = CreateEvent(NULL, FALSE, FALSE, NULL);

}

return 0;

case WM\_COMMAND:

{

if (LOWORD(wParam) == 1)

{

HWND hInput = GetDlgItem(hwnd, 2);

char input[10];

GetWindowTextA(hInput, input, sizeof(input));

int n = atoi(input);

// Запуск вычисления в отдельном потоке

DWORD threadId;

HANDLE hThread = CreateThread(NULL, 0, FactorialThread, &n, 0, &threadId);

// Ожидание завершения вычислений

WaitForSingleObject(hCalculationComplete, INFINITE);

// Захват мьютекса перед чтением результата

WaitForSingleObject(hCalculationMutex, INFINITE);

// Получение результата

DWORD result = calculationResult;

// Освобождение мьютекса

ReleaseMutex(hCalculationMutex);

// Вывод результата в текстовое поле

HWND hOutput = GetDlgItem(hwnd, 3);

char output[20];

snprintf(output, sizeof(output), "%lu", result);

SetWindowTextA(hOutput, output);

// Закрытие дескриптора потока

CloseHandle(hThread);

}

}

return 0;

case WM\_CLOSE:

DestroyWindow(hwnd);

return 0;

case WM\_DESTROY:

// Закрытие мьютекса и события

CloseHandle(hCalculationMutex);

CloseHandle(hCalculationComplete);

PostQuitMessage(0);

return 0;

}

return DefWindowProc(hwnd, uMsg, wParam, lParam);

}

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int nCmdShow)

{

WNDCLASSEX wc = {};

wc.cbSize = sizeof(WNDCLASSEX);

wc.lpfnWndProc = WindowProc;

wc.hInstance = hInstance;

wc.lpszClassName = TEXT("MyWindowClass");

RegisterClassEx(&wc);

HWND hwnd = CreateWindowEx(0, TEXT("MyWindowClass"), TEXT("Multi-Threaded App"), WS\_OVERLAPPEDWINDOW, CW\_USEDEFAULT, CW\_USEDEFAULT, 400, 250, NULL, NULL, hInstance, NULL);

ShowWindow(hwnd, nCmdShow);

MSG msg;

while (GetMessage(&msg, NULL, 0, 0))

{

TranslateMessage(&msg);

DispatchMessage(&msg);

}

return (int)msg.wParam;

}