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.»

Выполнил:

студент гр. 153504

Скроба А.П.

Проверил:

Гриценко Н.Ю.

Минск 2023

**СОДЕРЖАНИЕ**

[1 Цель работы 3](#_Toc146675523)

[2 Краткие теоретические сведения 4](#_Toc146675524)

[3 Результат выполнения программы 5](#_Toc146675525)

[Список использованных источников 6](#_Toc146675526)

[Приложение А 7](#_Toc146675527)

# 1 ЦЕЛЬ РАБОТЫ

Разработать многозадачное приложение с использованием мьютексов и семафоров для координирования работы потоков. Программа, должна создавать два потока: один поток увеличивает значение переменной, а другой уменьшает. Мьютексы и семафоры должны использоваться для синхронизации доступа к общей переменной.

**2 КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**

Средства синхронизации и взаимного исключения: это механизмы, используемые в операционных системах для обеспечения согласованности выполнения операций и предотвращения конфликтов между потоками.

Поток представляет собой единицу выполнения внутри процесса. Один процесс может содержать несколько потоков, которые выполняются параллельно или конкурируют за ресурсы процесса.

Процесс представляет собой выполняющуюся программу в операционной системе. Каждый процесс имеет свою собственную виртуальную память и набор ресурсов, которые ему выделяются. Каждый процесс имеет уникальный идентификатор (PID), который используется для идентификации процесса в системе.

Семафор — инструмент для управления синхронизацией. Это целочисленная переменная, которую одновременно используют сразу несколько процессов. Основная цель использования семафора — это синхронизация процессов и управление доступом к общему ресурсу в многопроцессорной среде.

Критическая секция – та часть алгоритма, где происходят обращения к критическим ресурсам. Очевидно, число потоков, находящихся в критической секции, ограничивается в соответствии с характеристиками критического ресурса (часто – одним потоком).

Мьютексы (mutex) представляют собой объекты, которые могут находиться в двух состояниях: заблокированном и разблокированном. Когда поток хочет получить доступ к общему ресурсу, он пытается заблокировать мьютекс. Если мьютекс разблокирован, поток получает доступ к ресурсу и блокирует мьютекс, чтобы предотвратить доступ других потоков к ресурсу. Если мьютекс уже заблокирован другим потоком, поток, пытающийся заблокировать его, будет приостановлен до тех пор, пока мьютекс не будет разблокирован. После использования ресурса поток разблокирует мьютекс, чтобы позволить другим потокам получить доступ к ресурсу.

# 3 РЕЗУЛЬТАТ ВЫПОЛНЕНИЯ ПРОГРАММЫ

Результатом выполнения лабораторной работы является многозадачное приложение использующее мьютексы и семафоры для координирования работы потоков. Программа, создает два потока: один поток увеличивает значение переменной, а другой уменьшает (рисунок 3.1).

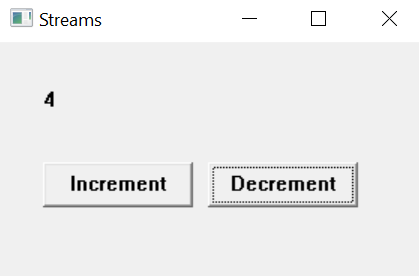
****

Рисунок 3.1 – Графический интерфейс приложения

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Мьютексы [Электронный ресурс]. – Режим доступа: https://www.tstu.ru/book/elib3/mm/2016/evdokimov/site/page41.41.html.

[2] Сообщения окна [Электронный ресурс]. – Режим доступа: <https://learn.microsoft.com/ru-ru/windows/win32/learnwin32/window-messags>.

[3] Справочник по программированию Win32 API. [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/en-us/windows/win32/api.

# ПРИЛОЖЕНИЕ А

**(обязательное)**

**Исходный код программы**

Листинг 1 – Файл Resource.h:

#pragma once

HWND hEditControl; // Объявление глобальной переменной для элемента управления Edit (текстовое поле)

HANDLE mutex; // Объявление глобальной переменной для мьютекса (синхронизация потоков)

HANDLE semaphore; // Объявление глобальной переменной для семафора (синхронизация потоков)

#define OnIncrementClicked 1

#define OnDecrementClicked 2

int sharedValue = 0; // Общая переменная, которую будут изменять потоки

WNDCLASS WindowClass(HBRUSH BGColor, HCURSOR Cursor, HINSTANCE hInst, LPCWSTR Name, WNDPROC wc);

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

void MainWndAddWidget(HWND hwnd);

DWORD WINAPI IncrementThread(LPVOID lpParam);

DWORD WINAPI DecrementThread(LPVOID lpParam);

Листинг 2 **–** Файл Source.cpp:

#include <Windows.h>

#include <string>

#include "Resource.h"

int WINAPI WinMain(\_In\_ HINSTANCE hInstance, \_In\_opt\_ HINSTANCE hPrevInstance, \_In\_ LPSTR lpCmdLine, \_In\_ int nShowCmd) {

// Создание мьютекса

mutex = CreateMutex(NULL, FALSE, NULL);

if (mutex == NULL) {

MessageBox(NULL, L"Failed to create mutex", L"Error", MB\_OK | MB\_ICONERROR);

return 1;

}

// Создание семафора

semaphore = CreateSemaphore(NULL, 1, 1, NULL);

if (semaphore == NULL) {

MessageBox(NULL, L"Failed to create semaphore", L"Error", MB\_OK | MB\_ICONERROR);

CloseHandle(mutex);

return 1;

}

WNDCLASS MainClass = WindowClass((HBRUSH)COLOR\_WINDOW, LoadCursor(NULL, IDC\_ARROW), hInstance, L"MainWndClass", WndProc);

if (!RegisterClassW(&MainClass)) {

return -1;

}

MSG MainMsg = {};

CreateWindow(L"MainWndClass", L"Streams", WS\_OVERLAPPEDWINDOW | WS\_VISIBLE, 500, 250, 300, 200, NULL, NULL, NULL, NULL);

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

TranslateMessage(&MainMsg);

DispatchMessage(&MainMsg);

}

// Закрытие мьютекса и семафора

CloseHandle(mutex);

CloseHandle(semaphore);

return 0;

}

WNDCLASS WindowClass(HBRUSH BGColor, HCURSOR Cursor, HINSTANCE hInst, LPCWSTR Name, WNDPROC wc) {

WNDCLASS nwc = {};

nwc.hbrBackground = BGColor;

nwc.hCursor = Cursor;

nwc.hInstance = hInst;

nwc.lpszClassName = Name;

nwc.lpfnWndProc = wc;

return nwc;

}

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

switch (uMsg) {

case WM\_CREATE:

MainWndAddWidget(hwnd);

break;

case WM\_DESTROY:

PostQuitMessage(0);

break;

case WM\_COMMAND:

switch (wParam) {

case OnIncrementClicked:

// Создание потока для инкремента

CreateThread(NULL, 0, IncrementThread, hwnd, 0, NULL);

break;

case OnDecrementClicked:

// Создание потока для декремента

CreateThread(NULL, 0, DecrementThread, hwnd, 0, NULL);

break;

}

break;

default:

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

}

return 0;

}

void MainWndAddWidget(HWND hwnd) {

// Создание элементов управления в окне

hEditControl = CreateWindow(L"edit", L"0", WS\_VISIBLE | WS\_CHILD | ES\_MULTILINE | ES\_READONLY, 30, 30, 100, 30, hwnd, NULL, NULL, NULL);

CreateWindowA("button", "Increment", WS\_VISIBLE | WS\_CHILD, 30, 80, 100, 30, hwnd, (HMENU)OnIncrementClicked, NULL, NULL);

CreateWindowA("button", "Decrement", WS\_VISIBLE | WS\_CHILD, 140, 80, 100, 30, hwnd, (HMENU)OnDecrementClicked, NULL, NULL);

}

DWORD WINAPI IncrementThread(LPVOID lpParam) {

// Ожидание мьютекса и семафора

WaitForSingleObject(mutex, INFINITE);

WaitForSingleObject(semaphore, INFINITE);

// Инкремент общей переменной

sharedValue++;

std::wstring value = std::to\_wstring(sharedValue);

// Обновление текстового поля

SetWindowText(hEditControl, value.c\_str());

// Освобождение семафора и мьютекса

ReleaseSemaphore(semaphore, 1, NULL);

ReleaseMutex(mutex);

return 0;

}

DWORD WINAPI DecrementThread(LPVOID lpParam) {

// Ожидание мьютекса и семафора

WaitForSingleObject(mutex, INFINITE);

WaitForSingleObject(semaphore, INFINITE);

// Декремент общей переменной

sharedValue--;

std::wstring value = std::to\_wstring(sharedValue);

// Обновление текстового поля

SetWindowText(hEditControl, value.c\_str());

// Освобождение семафора и мьютекса

ReleaseSemaphore(semaphore, 1, NULL);

ReleaseMutex(mutex);

return 0;

}