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. 153503

Вергасов В.М.

Проверил:

Гриценко Н.Ю.

Минск 2023

**СОДЕРЖАНИЕ**

[1 Цели работы 3](#_Toc146883343)

[2 Краткие теоретические сведения 4](#_Toc146883344)

[3 Полученные результаты 5](#_Toc146883345)

[Выводы 8](#_Toc146883346)

[Список литературы 9](#_Toc146883347)

# 1 ЦЕЛИ РАБОТЫ

1. Ознакомиться с основными понятиями и концепциями синхронизации и взаимного исключения в Windows.
2. Изучить различные алгоритмы и механизмы синхронизации в Windows, такие как «семафор» и «мьютекс».
3. Разработать программный продукт, реализующий механизм синхронизации в Windows.

**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.

Семафор: это переменная, которая может принимать значения от 0 до N-1, где N - максимальное количество потоков, которые могут одновременно использовать семафор. Семафор используется для управления доступом к общему ресурсу.

Использование средств синхронизации и взаимного исключения в Windows: в Windows реализованы различные средства синхронизации и взаимного исключения, такие как мьютекс, критические секции, семафоры и события.

Мьютекс используется в многопоточных программах для работы с общими данными и памятью, такими как чтение или запись в общий файл, использование разделяемой памяти и т.д. При получении доступа к общему ресурсу поток блокирует мьютекс, чтобы предотвратить доступ других потоков к этому ресурсу, пока первый поток не завершит свою работу и не освободит мьютекс.

Мьютексы обеспечивают правильный доступ к общим ресурсам и предотвращают конфликты, которые могут возникнуть при доступе нескольких потоков к одному и тому же ресурсу. Они также позволяют избежать фрагментации памяти благодаря контролю потоков.

Применение средств синхронизации и взаимного исключения в разработке программного обеспечения: использование средств синхронизации и взаимного исключения позволяет обеспечить корректное и согласованное выполнение операций, предотвращая конфликты между потоками и обеспечивая безопасность и стабильность работы приложения.

# 3 ПОЛУЧЕННЫЕ РЕЗУЛЬТАТЫ

Результат работы программы после запуска, показан на рисунке 3.1.

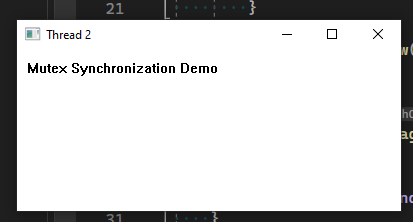


Рисунок 3.1 – Результат работы программы после запуска

Результат работы программы после запуска, когда активен первый поток показан на рисунке 3.2.

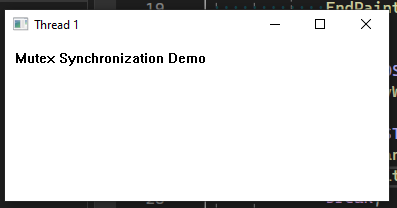


Рисунок 3.2 – Результат работы программы после запуска

Листинг 1 **–** Код исходной программы:

#include <windows.h>

#include <tchar.h>

#include <thread>

#include <string>

HANDLE hMutex;

HWND hwnd;

std::thread thread1, thread2;

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

switch (msg) {

case WM\_PAINT:

{

PAINTSTRUCT ps;

HDC hdc = BeginPaint(hwnd, &ps);

// Рисование в окне

TextOut(hdc, 10, 10, \_T("Mutex Synchronization Demo"), \_tcslen(\_T("Mutex Synchronization Demo")));

EndPaint(hwnd, &ps);

break;

}

case WM\_CLOSE:

DestroyWindow(hwnd);

break;

case WM\_DESTROY:

CloseHandle(hMutex);

PostQuitMessage(0);

break;

default:

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

}

return 0;

}

void WorkerFunction(const wchar\_t \*threadName, int delay) {

for (int i = 0; i < 5; ++i) {

WaitForSingleObject(hMutex, INFINITE);

SetWindowText(hwnd, threadName);

ReleaseMutex(hMutex);

Sleep(delay);

}

OutputDebugString((L"End: " + std::wstring(threadName) + L"\n").c\_str());

}

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

hMutex = CreateMutex(NULL, FALSE, NULL);

if (hMutex == NULL) {

MessageBox(NULL, \_T("Mutex creation failed."), \_T("Error"), MB\_ICONERROR);

return 1;

}

WNDCLASS wc = { 0 };

wc.lpfnWndProc = WndProc;

wc.hInstance = hInstance;

wc.hbrBackground = (HBRUSH)(COLOR\_WINDOW + 1);

wc.lpszClassName = \_T("MutexSyncDemo");

RegisterClass(&wc);

hwnd = CreateWindow(\_T("MutexSyncDemo"), \_T("Lab6, OEaSP"),

WS\_OVERLAPPEDWINDOW, CW\_USEDEFAULT, CW\_USEDEFAULT,

400, 200, NULL, NULL, hInstance, NULL);

if (hwnd == NULL) {

MessageBox(NULL, \_T("Window creation failed."), \_T("Error"), MB\_ICONERROR);

return 1;

}

ShowWindow(hwnd, nCmdShow);

UpdateWindow(hwnd);

thread1 = std::thread(WorkerFunction, L"Thread 1", 500);

thread2 = std::thread(WorkerFunction, L"Thread 2", 700);

MSG msg;

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

TranslateMessage(&msg);

DispatchMessage(&msg);

}

thread1.join();

thread2.join();

return 0;

}}

# ВЫВОДЫ

Синхронизация и взаимное исключение являются ключевыми элементами в разработке многопоточных программ, особенно в операционных системах, таких как Windows.

Семафоры представляют собой более гибкие и сложные средства синхронизации, которые позволяют контролировать доступ к общим ресурсам и обеспечивать взаимное исключение.

В Windows представлены различные средства синхронизации и взаимного исключения, такие как мьютекс, критические секции, семафоры, события и блокировки.

Использование синхронизирующих и исключающих средств позволяет обеспечить корректное и согласованное выполнение операций, предотвращать конфликты между потоками и обеспечивать безопасность и стабильность работы приложения.

Была сделана программа, которая показывает работу мьютекса через изменения заголовка программы.

# СПИСОК ЛИТЕРАТУРЫ

1. Programming reference for the Win32 API [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/en-us/windows/win32/api/ – Дата доступа: 21.09.2023.