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 И. В. Бобко

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 был создан симулятор работы банкомата, где несколько потоков одновременно обращаются к счету, используя мьютексы для обеспечения целостности (рисунок 1).

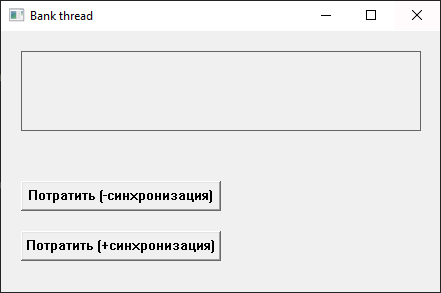


Рисунок 1– Главное окно

При трате 100000 единиц, без средств синхронизации, результат случайный (рисунок 2).

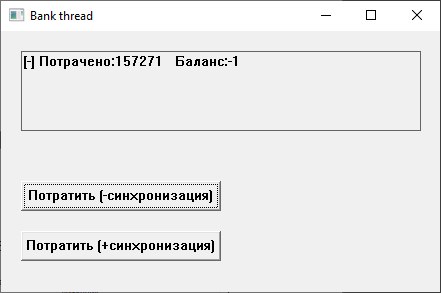


Рисунок 2 – Результат без синхронизации

При трате 100000 единиц, используя средства синхронизации (*mutex*), результат корректный (рисунок 3).

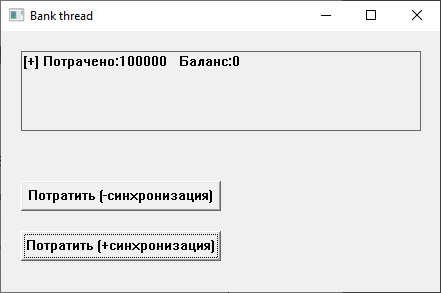


Рисунок 3 – Результат с синхронизацией

# 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 *Source.cpp*

#include <windows.h>

#include <tchar.h>

#include <string>

#include <vector>

struct Account {

int balance = 0;

bool Spend(int value) {

if (value > balance)

return false;

balance -= value;

return true;

}

};

Account account{100000};

HWND hEditResult;

HWND hButtonSpendBalanceA;

HWND hButtonSpendBalanceS;

HANDLE hMutex;

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

void SpendBalanceA(int\* total\_spent);

void SpendBalanceS(int\* total\_spent);

DWORD WINAPI ThreadFuncA(LPVOID lpParam);

DWORD WINAPI ThreadFuncS(LPVOID lpParam);

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

MSG msg;

HWND hWnd;

WNDCLASSEX wc;

wc.cbSize = sizeof(WNDCLASSEX);

wc.cbClsExtra = 0;

wc.cbWndExtra = 0;

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

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

wc.hIcon = LoadIcon(NULL, IDI\_APPLICATION);

wc.hIconSm = LoadIcon(NULL, IDI\_APPLICATION);

wc.hInstance = hInstance;

wc.lpfnWndProc = WndProc;

wc.lpszClassName = L"Bank";

wc.lpszMenuName = NULL;

wc.style = CS\_HREDRAW | CS\_VREDRAW;

if (!RegisterClassEx(&wc))

{

MessageBox(NULL, L"Failed to register window class.", L"Error", MB\_ICONERROR | MB\_OK);

return EXIT\_FAILURE;

}

hMutex = CreateMutex(NULL, FALSE, NULL);

hWnd = CreateWindow(wc.lpszClassName, L"Bank thread", WS\_OVERLAPPEDWINDOW, CW\_USEDEFAULT, CW\_USEDEFAULT, 500, 300, NULL, NULL, wc.hInstance, NULL);

if (hWnd == INVALID\_HANDLE\_VALUE)

{

MessageBox(NULL, L"Failed to create a window.", L"Error", MB\_ICONERROR);

return EXIT\_FAILURE;

}

ShowWindow(hWnd, nCmdShow);

UpdateWindow(hWnd);

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

{

TranslateMessage(&msg);

DispatchMessage(&msg);

}

return (int)msg.wParam;

}

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

switch (msg)

{

case WM\_CREATE:

{

hEditResult = CreateWindow(L"EDIT", L"", WS\_CHILD | WS\_VISIBLE | ES\_READONLY | WS\_BORDER, 20, 20, 400, 80, hWnd, (HMENU)1, NULL, NULL);

hButtonSpendBalanceA = CreateWindow(L"BUTTON", L"Потратить (-синхронизация)", WS\_CHILD | WS\_VISIBLE, 20, 150, 200, 30, hWnd, (HMENU)101, NULL, NULL);

hButtonSpendBalanceS = CreateWindow(L"BUTTON", L"Потратить (+синхронизация)", WS\_CHILD | WS\_VISIBLE, 20, 200, 200, 30, hWnd, (HMENU)102, NULL, NULL);

break;

}

case WM\_COMMAND:

{

switch (LOWORD(wParam))

{

case 101:

{

account.balance = 100000;

SetWindowText(hEditResult, L"");

int res1=0;

int res2=0;

HANDLE hThread1 = CreateThread(NULL, 0, ThreadFuncA, &res1, 0, NULL);

HANDLE hThread2 = CreateThread(NULL, 0, ThreadFuncA, &res2, 0, NULL);

WaitForSingleObject(hThread1, INFINITE);

WaitForSingleObject(hThread2, INFINITE);

CloseHandle(hThread1);

CloseHandle(hThread2);

SetWindowText(hEditResult, (L"[-] Потрачено:" + std::to\_wstring(res1 + res2) + L" Баланс:" + std::to\_wstring(account.balance)).c\_str());

break;

}

case 102:

{

account.balance = 100000;

SetWindowText(hEditResult, L"");

int res1 = 0;

int res2 = 0;

HANDLE hThread1 = CreateThread(NULL, 0, ThreadFuncS, &res1, 0, NULL);

HANDLE hThread2 = CreateThread(NULL, 0, ThreadFuncS, &res2, 0, NULL);

WaitForSingleObject(hThread1, INFINITE);

WaitForSingleObject(hThread2, INFINITE);

CloseHandle(hThread1);

CloseHandle(hThread2);

SetWindowText(hEditResult, (L"[+] Потрачено:" + std::to\_wstring(res1 + res2) + L" Баланс:" + std::to\_wstring(account.balance)).c\_str());

break;

}

break;

}

break;

}

case WM\_DESTROY:

{

PostQuitMessage(0);

break;

}

default:

{

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

}

}

return 0;

}

void SpendBalanceA(int\* total\_spent) {

\*total\_spent = 0;

for (int i = 0; i < 100000; i++) {

if(account.Spend(1))

(\*total\_spent)++;

}

}

void SpendBalanceS(int\* total\_spent) {

WaitForSingleObject(hMutex, INFINITE);

\*total\_spent = 0;

for (int i = 0; i < 100000; i++) {

if (account.Spend(1))

(\*total\_spent)++;

}

ReleaseMutex(hMutex);

}

DWORD WINAPI ThreadFuncA(LPVOID lpParam) {

int\* result = reinterpret\_cast<int\*>(lpParam);

SpendBalanceA(result);

return 0;

}

DWORD WINAPI ThreadFuncS(LPVOID lpParam) {

int\* result = reinterpret\_cast<int\*>(lpParam);

SpendBalanceS(result);

return 0;

}