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)**

Выполнил:

студент гр. 153503

Татаринов В.В.

Проверил:

Гриценко Н.Ю.

Минск 2023

**СОДЕРЖАНИЕ**

[1 Цель работы 3](#_Toc146836467)

[2 Теоретические сведения 4](#_Toc146836468)

[3 Полученные результаты 6](#_Toc146836469)

[Выводы 7](#_Toc146836470)

[Список использованных источников 8](#_Toc146836471)

[Приложение А (обязательное) листинг кода 9](#_Toc146836472)

# **1 ЦЕЛЬ РАБОТЫ**

Изучить и использовать средства синхронизации и взаимного исключения. Создать оконное приложение, позволяющее создавать несколько потоков и использовать средства синхронизации для управления доступом к разделяемой памяти.

**2 ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ** 

Синхронизация в многопоточности относится к процессу координации и управления выполнением нескольких потоков в многопоточной среде. Когда несколько потоков выполняются параллельно, могут возникать проблемы с доступом к общим ресурсам (например, памятью или файлами) или соревнование за одну и ту же задачу. Для обеспечения правильного выполнения программы в многопоточной среде необходимо использовать механизмы синхронизации. Они позволяют синхронизировать доступ к общим ресурсам и управлять порядком выполнения потоков.

Важным понятием синхронизации процессов является понятие "критическая секция" программы. 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].

* *EnterCriticalSection* и *LeaveCriticalSection* являются функциями, используемыми для реализации критической секции в программировании многопоточности. EnterCriticalSection блокируется до тех пор, пока поток не сможет стать владельцем критического раздела. Завершив выполнение защищенного кода, поток использует функцию LeaveCriticalSection, чтобы отказаться от владения, что позволяет другому потоку стать владельцем защищенного ресурса и получить доступ к нему. Нет никакой гарантии относительно порядка, в котором ожидающие потоки получат право владения критически важным разделом [2].
* *std::lock\_guardstd::mutex lock(mtx)* захватывает мьютекс в своем конструкторе и освобождает его в своем деструкторе. Это гарантирует, что мьютекс будет автоматически освобожден, когда область действия std::lock\_guard заканчивается или при возникновении исключения внутри этой области действия. Это помогает избежать проблем с блокировкой и разблокировкой мьютекса вручную и обеспечивает безопасность взаимодействия потоков в параллельных программах.
* *CreateSemaphore* cоздает или открывает именованный или неименованный объект семафора. Если функция выполняется успешно, возвращаемое значение является дескриптором для объекта семафора. Если именованный объект семафор существовал до вызова функции, функция возвращает дескриптор существующему объекту [3].

# **3 ПОЛУЧЕННЫЕ РЕЗУЛЬТАТЫ**

В результате выполнения лабораторной работы было создано оконное приложение, производящее запись и чтение из файла, параллельно выполняющее 2 потока, которые используют разделяемую память.

Интерфейс программы представляет собой поля для ввода, и поля, отображающие процент выполнения задачи в каждом из двух потоков (рисунок 1).

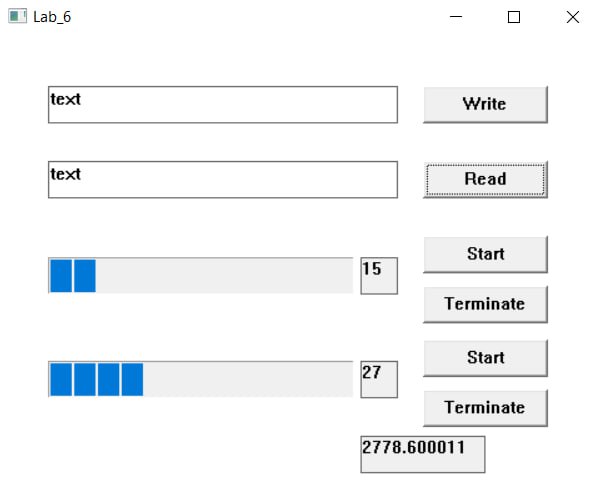


Рисунок 1 – Интерфейс приложения

# **ВЫВОДЫ**

В результате выполнения лабораторной работы были изучены и использованы средства синхронизации и взаимного исключения. Создано оконное приложение, позволяющее создавать несколько потоков и использовать средства синхронизации для управления доступом к разделяемой памяти.

# **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. Средства синхронизации и взаимодействия процессов [Электронный ресурс]. – Режим доступа: https://studfile.net/preview/7153358/page:11/.
2. Критическая секция [Электронный ресурс]. – Режим доступа: <https://learn.microsoft.com/ru-ru/windows/win32/api/synchapi/nf-synchapi-entercriticalsection>.
3. Функции синхронизации [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/ru-ru/windows/win32/api/winbase/nf-winbase-createsemaphorea.

# **ПРИЛОЖЕНИЕ А (обязательное) Листинг кода**

Листинг 1 **–** Файл *Lab\_6\_WINAPI32.cpp*:

#include <windows.h>

#include <cstdlib>

#include <string>

#include <CommCtrl.h>

#include <mutex>

#define ID\_FILE\_WRITE 100

#define ID\_FILE\_READ 101

#define ID\_CALCULATE 102

#define ID\_CALCULATE\_2 103

#define ID\_PROGRESS\_STATE 104

#define ID\_PROGRESS\_STATE\_2 105

#define ID\_PROGRESS\_BAR 106

#define ID\_PROGRESS\_BAR\_2 107

#define ID\_TERMINATE 108

#define ID\_TERMINATE\_2 109

#define ID\_VALUE\_VALUE 110

HMENU hMenu;

HWND hWnd, hResult, hSource, hProgress, hProgressBar, hProgress\_2, hProgressBar\_2, hValueValue;

wchar\_t data[100];

int numBytesToWrite;

wchar\_t res[100];

HANDLE writeCompleteEvent;

bool isInProgress, isInProgress\_2;

HBRUSH hProgressBarBrush;

RECT progressBarRect, progressBarRect\_2;

double value = 0;

CRITICAL\_SECTION cs;

std::mutex mtx;

HANDLE semaphore;

void AddControls(HWND hWnd);

void AddMenus(HWND hWnd);

void WriteToFileAsync(HWND hWnd);

void ReadFromFileAsync(HWND hWnd);

void CalculateAsync(WPARAM wp);

void CalculateAsync\_2(WPARAM wp);

void Terminate();

void Terminate\_2();

void UpdateProgressState(WPARAM wp);

void UpdateProgressState\_2(WPARAM wp);

void UpdateProgressBar(WPARAM wp);

void UpdateProgressBar\_2(WPARAM wp);

void UpdateValueValue(WPARAM wp);

DWORD WINAPI WriteAsync(LPVOID filePointer);

DWORD WINAPI ReadAsync(LPVOID filePointer);

DWORD WINAPI ProgressAsync(LPVOID lpParam);

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

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

WNDCLASSW wc = { 0 };

wc.hbrBackground = CreateSolidBrush(RGB(255, 255, 255));

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

wc.hInstance = hInst;

wc.lpszClassName = L"Class";

wc.lpfnWndProc = WindowProcedure;

if (!RegisterClassW(&wc))

return -1;

isInProgress = false;

isInProgress\_2 = false;

InitializeCriticalSection(&cs);

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

hWnd = CreateWindowW(L"Class", L"Lab\_4", WS\_OVERLAPPEDWINDOW | WS\_VISIBLE, 20, 20, 500, 400, NULL, NULL, NULL, NULL);

MSG msg = { 0 };

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

TranslateMessage(&msg);

DispatchMessage(&msg);

}

DeleteCriticalSection(&cs);

return 0;

}

DWORD WINAPI ProgressAsync(LPVOID lpParam) {

for (int i = 1; i <= 100; i++) {

for (int j = 0; j < 900000; j++) {

if (isInProgress == false) {

SendMessage(hWnd, ID\_PROGRESS\_STATE, 0, 0);

SendMessage(GetDlgItem(hWnd, ID\_PROGRESS\_BAR), PBM\_SETPOS, 0, 0);

return 0;

}

//EnterCriticalSection(&cs);

//std::lock\_guard<std::mutex> lock(mtx);

WaitForSingleObject(semaphore, INFINITE);

value += 0.2;

SendMessage(hWnd, ID\_VALUE\_VALUE, \*reinterpret\_cast<WPARAM\*>(&value), 0);

ReleaseSemaphore(semaphore, 1, NULL);

//LeaveCriticalSection(&cs);

}

int progress = i;

SendMessage(hWnd, ID\_PROGRESS\_STATE, \*reinterpret\_cast<WPARAM\*>(&progress), 0);

SendMessage(GetDlgItem(hWnd, ID\_PROGRESS\_BAR), PBM\_SETPOS, progress, 0);

}

isInProgress = false;

return 0;

}

DWORD WINAPI ProgressAsync\_2(LPVOID lpParam) {

for (int i = 1; i <= 100; i++) {

for (int j = 0; j < 400000; j++) {

if (isInProgress\_2 == false) {

SendMessage(hWnd, ID\_PROGRESS\_STATE\_2, 0, 0);

SendMessage(GetDlgItem(hWnd, ID\_PROGRESS\_BAR\_2), PBM\_SETPOS, 0, 0);

return 0;

}

//EnterCriticalSection(&cs);

//std::lock\_guard<std::mutex> lock(mtx);

WaitForSingleObject(semaphore, INFINITE);

value += 0.000000001;

SendMessage(hWnd, ID\_VALUE\_VALUE, \*reinterpret\_cast<WPARAM\*>(&value), 0);

ReleaseSemaphore(semaphore, 1, NULL);

//LeaveCriticalSection(&cs);

}

int progress = i;

SendMessage(hWnd, ID\_PROGRESS\_STATE\_2, \*reinterpret\_cast<WPARAM\*>(&progress), 0);

SendMessage(GetDlgItem(hWnd, ID\_PROGRESS\_BAR\_2), PBM\_SETPOS, progress, 0);

}

isInProgress\_2 = false;

return 0;

}

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

switch (msg)

{

case WM\_COMMAND:

switch (wp)

{

case ID\_FILE\_WRITE: {

WriteToFileAsync(hWnd);

break;

}

case ID\_FILE\_READ: {

ReadFromFileAsync(hWnd);

break;

}

case ID\_CALCULATE: {

CalculateAsync(wp);

break;

}

case ID\_TERMINATE: {

Terminate();

break;

}

case ID\_CALCULATE\_2: {

CalculateAsync\_2(wp);

break;

}

case ID\_TERMINATE\_2: {

Terminate\_2();

break;

}

default: {

break;

}

}

break;

case ID\_PROGRESS\_STATE: {

UpdateProgressState(wp);

break;

}

case ID\_PROGRESS\_STATE\_2: {

UpdateProgressState\_2(wp);

break;

}

case ID\_VALUE\_VALUE: {

UpdateValueValue(wp);

break;

}

case WM\_PAINT:

{

UpdateProgressBar(wp);

UpdateProgressBar\_2(wp);

break;

}

case WM\_CREATE:

AddMenus(hWnd);

AddControls(hWnd);

break;

case WM\_DESTROY:

PostQuitMessage(0);

break;

default:

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

}

}

void AddMenus(HWND hWnd) {

hMenu = CreateMenu();

SetMenu(hWnd, hMenu);

}

void AddControls(HWND hWnd) {

hSource = CreateWindowW(L"Edit", L"", WS\_VISIBLE | WS\_CHILD | WS\_BORDER, 40, 40, 280, 30, hWnd, NULL, NULL, NULL);

CreateWindowW(L"Button", L"Write", WS\_VISIBLE | WS\_CHILD, 340, 40, 100, 30, hWnd, (HMENU)ID\_FILE\_WRITE, NULL, NULL);

hResult = CreateWindowW(L"Edit", L"", WS\_VISIBLE | WS\_CHILD | WS\_BORDER, 40, 100, 280, 30, hWnd, NULL, NULL, NULL);

CreateWindowW(L"Button", L"Read", WS\_VISIBLE | WS\_CHILD, 340, 100, 100, 30, hWnd, (HMENU)ID\_FILE\_READ, NULL, NULL);

hProgressBar = CreateWindowEx(0, PROGRESS\_CLASS, NULL, WS\_CHILD | WS\_VISIBLE, 40, 177, 245, 30, hWnd, (HMENU)ID\_PROGRESS\_BAR, NULL, NULL);

hProgress = CreateWindowW(L"Static", L"", WS\_VISIBLE | WS\_CHILD | WS\_BORDER, 290, 177, 30, 30, hWnd, NULL, NULL, NULL);

CreateWindowW(L"Button", L"Start", WS\_VISIBLE | WS\_CHILD, 340, 160, 100, 30, hWnd, (HMENU)ID\_CALCULATE, NULL, NULL);

CreateWindowW(L"Button", L"Terminate", WS\_VISIBLE | WS\_CHILD, 340, 200, 100, 30, hWnd, (HMENU)ID\_TERMINATE, NULL, NULL);

hProgressBar\_2 = CreateWindowEx(0, PROGRESS\_CLASS, NULL, WS\_CHILD | WS\_VISIBLE, 40, 260, 245, 30, hWnd, (HMENU)ID\_PROGRESS\_BAR\_2, NULL, NULL);

hProgress\_2 = CreateWindowW(L"Static", L"", WS\_VISIBLE | WS\_CHILD | WS\_BORDER, 290, 260, 30, 30, hWnd, NULL, NULL, NULL);

CreateWindowW(L"Button", L"Start", WS\_VISIBLE | WS\_CHILD, 340, 243, 100, 30, hWnd, (HMENU)ID\_CALCULATE\_2, NULL, NULL);

CreateWindowW(L"Button", L"Terminate", WS\_VISIBLE | WS\_CHILD, 340, 283, 100, 30, hWnd, (HMENU)ID\_TERMINATE\_2, NULL, NULL);

hValueValue = CreateWindowW(L"Static", L"", WS\_VISIBLE | WS\_CHILD | WS\_BORDER, 290, 320, 100, 30, hWnd, NULL, NULL, NULL);

}

void WriteToFileAsync(HWND hWnd) {

LPCWSTR filename = L"mapped\_file.txt";

HANDLE fileHandle = CreateFile(

filename,

GENERIC\_READ | GENERIC\_WRITE,

FILE\_SHARE\_READ | FILE\_SHARE\_WRITE,

NULL,

CREATE\_ALWAYS,

FILE\_ATTRIBUTE\_NORMAL | FILE\_FLAG\_OVERLAPPED,

NULL

);

GetWindowText(hSource, data, 100);

numBytesToWrite = (wcslen(data) + 1) \* sizeof(wchar\_t);

DWORD fileSize = numBytesToWrite;

SetFilePointer(fileHandle, fileSize, NULL, FILE\_BEGIN);

SetEndOfFile(fileHandle);

HANDLE fileMapping = CreateFileMapping(

fileHandle,

NULL,

PAGE\_READWRITE,

0,

fileSize,

NULL

);

LPVOID filePointer = MapViewOfFile(fileMapping, FILE\_MAP\_WRITE, 0, 0, 0);

HANDLE writeThread = CreateThread(NULL, 0, WriteAsync, filePointer, 0, NULL);

WaitForSingleObject(writeThread, INFINITE);

CloseHandle(writeThread);

UnmapViewOfFile(filePointer);

CloseHandle(fileMapping);

CloseHandle(fileHandle);

}

void ReadFromFileAsync(HWND hWnd) {

LPCWSTR filename = L"mapped\_file.txt";

HANDLE fileHandle = CreateFile(

filename,

GENERIC\_READ | GENERIC\_WRITE,

FILE\_SHARE\_READ | FILE\_SHARE\_WRITE,

NULL,

OPEN\_EXISTING,

FILE\_ATTRIBUTE\_NORMAL | FILE\_FLAG\_OVERLAPPED,

NULL

);

DWORD fileSize = numBytesToWrite;

SetFilePointer(fileHandle, fileSize, NULL, FILE\_BEGIN);

SetEndOfFile(fileHandle);

HANDLE fileMapping = CreateFileMapping(

fileHandle,

NULL,

PAGE\_READWRITE,

0,

fileSize,

NULL

);

LPVOID filePointer = MapViewOfFile(fileMapping, FILE\_MAP\_WRITE, 0, 0, 0);

HANDLE readThread = CreateThread(NULL, 0, ReadAsync, filePointer, 0, NULL);

WaitForSingleObject(readThread, INFINITE);

CloseHandle(readThread);

SetWindowText(hResult, res);

UnmapViewOfFile(filePointer);

CloseHandle(fileMapping);

CloseHandle(fileHandle);

}

DWORD WINAPI WriteAsync(LPVOID filePointer) {

CopyMemory(filePointer, data, numBytesToWrite);

SetEvent(writeCompleteEvent);

return 0;

}

DWORD WINAPI ReadAsync(LPVOID filePointer) {

WaitForSingleObject(writeCompleteEvent, INFINITE);

wchar\_t\* Buffer = new wchar\_t[numBytesToWrite / sizeof(wchar\_t)];

CopyMemory(Buffer, filePointer, numBytesToWrite);

wcscpy\_s(res, Buffer);

wcscat\_s(res, L"\0");

return 0;

}

void CalculateAsync(WPARAM wp) {

if (isInProgress) {

return;

}

else

{

isInProgress = true;

HANDLE CalculateThread = CreateThread(NULL, 0, ProgressAsync, NULL, 0, NULL);

SetThreadPriority(CalculateThread, THREAD\_PRIORITY\_HIGHEST);

}

}

void Terminate() {

isInProgress = false;

}

void CalculateAsync\_2(WPARAM wp) {

if (isInProgress\_2) {

return;

}

else

{

isInProgress\_2 = true;

HANDLE CalculateThread\_2 = CreateThread(NULL, 0, ProgressAsync\_2, NULL, 0, NULL);

SetThreadPriority(CalculateThread\_2, THREAD\_PRIORITY\_HIGHEST);

}

}

void Terminate\_2() {

isInProgress\_2 = false;

}

void UpdateProgressState(WPARAM wp) {

int progress\_int = \*reinterpret\_cast<int\*>(&wp);

WCHAR progress\_wchar[4];

swprintf\_s(progress\_wchar, L"%d", progress\_int);

LPCWSTR progress = progress\_wchar;

SetWindowText(hProgress, progress);

}

void UpdateProgressState\_2(WPARAM wp) {

int progress\_int = \*reinterpret\_cast<int\*>(&wp);

WCHAR progress\_wchar[4];

swprintf\_s(progress\_wchar, L"%d", progress\_int);

LPCWSTR progress = progress\_wchar;

SetWindowText(hProgress\_2, progress);

}

void UpdateValueValue(WPARAM wp) {

double value\_value = \*reinterpret\_cast<double\*>(&wp);

WCHAR value\_value\_wchar[20];

swprintf\_s(value\_value\_wchar, L"%f", value\_value);

LPCWSTR value = value\_value\_wchar;

SetWindowText(hValueValue, value);

}

void UpdateProgressBar(WPARAM wp) {

PAINTSTRUCT ps;

HDC hdc = BeginPaint(hWnd, &ps);

int progress = \*reinterpret\_cast<int\*>(&wp);

progressBarRect.right = progressBarRect.left + progress;

hProgressBarBrush = CreateSolidBrush(RGB(0, 0, 255));

FillRect(hdc, &progressBarRect, hProgressBarBrush);

EndPaint(hWnd, &ps);

}

void UpdateProgressBar\_2(WPARAM wp) {

PAINTSTRUCT ps;

HDC hdc = BeginPaint(hWnd, &ps);

int progress = \*reinterpret\_cast<int\*>(&wp);

progressBarRect\_2.right = progressBarRect\_2.left + progress;

hProgressBarBrush = CreateSolidBrush(RGB(0, 0, 255));

FillRect(hdc, &progressBarRect\_2, hProgressBarBrush);

EndPaint(hWnd, &ps);

}