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 № 4

на тему

**УПРАВЛЕНИЕ ПРОЦЕССАМИ И ПОТОКАМИ (WINDOWS). ПОРОЖДЕНИЕ, ЗАВЕРШЕНИЕ, ИЗМЕНЕНИЕ ПРИОРИТЕТОВ ПРОЦЕССОВ И ПОТОКОВ**

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 управление процессами и потоками в *Windows.* Освоить возможности порождения, завершения, изменения приоритета процессов и потоков. Создать оконное приложение, выполняющее чтение, запись данных в файл, отрисовку окна загрузки в разных потоках.

**2 ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**

*Windows 32 API* (*Application Programming Interface*) предоставляет разработчикам мощные средства для управления процессами и потоками в операционной системе *Windows*. Процесс является независимой программой, а поток - базовой единицей выполнения внутри процесса.

Процесс представляет собой выполняющуюся программу или задачу. Каждый активный процесс имеет собственное адресное пространство и ресурсы, которые позволяют ему работать независимо от других процессов. Процессы работают независимо друг от друга. Если один процесс завершается или сталкивается с ошибкой, это не влияет на работу других процессов. Операционная система *Windows* управляет выделением ресурсов для каждого процесса, таких как центральный процессор (*CPU*) и оперативная память. Это позволяет системе равномерно распределять ресурсы между активными процессами.

Поток (*thread*) представляет собой базовую единицу выполнения внутри процесса. Потоки являются частью многозадачности и позволяют процессам выполнять несколько задач параллельно или асинхронно. Потоки принадлежат конкретному процессу. Каждый процесс имеет по крайней мере один поток, называемый главным потоком, который создается при запуске процесса. Дополнительные потоки могут быть созданы внутри процесса для выполнения различных задач.

Многопоточность (*multithreading*) в операционной системе *Windows* означает использование нескольких потоков внутри одного процесса для параллельного выполнения задач. Многопоточные приложения позволяют выполнять несколько задач одновременно, что может увеличить производительность и отзывчивость приложения. Например, один поток может обрабатывать пользовательский интерфейс, а другой - фоновые вычисления.

В *Win32* *API* операции ввода и вывода (*I/O*) играют ключевую роль в разработке приложений, так как они позволяют программам взаимодействовать с файлами, устройствами и сетевыми ресурсами. Операции *I/O* предоставляют широкий набор функций и механизмов для эффективной и гибкой работы с данными.

В ходе выполнения данной лабораторной работы использовались следующие инструменты и концепции:

* *CreateThread* предназначена для создания новых потоков внутри процесса и позволяет разработчикам реализовать параллельное выполнение задач в приложениях под операционной системой *Windows*.
* *WaitForSingleObject* позволяет потоку ожидать завершения выполнения другого потока или объекта синхронизации. Она блокирует выполнение текущего потока, пока целевой поток или объект синхронизации не завершат свою работу или не перейдут в состояние, которое удовлетворяет заданным условиям ожидания.

# **3 ПОЛУЧЕННЫЕ РЕЗУЛЬТАТЫ**

Было создано оконное приложение, позволяющее вводить данные и асинхронно записывать их в файл, запускать в отдельном потоке вычисление и отображение процесса вычисления в окне загрузки. В качестве входных данных пользователем вводится любая строка. При нажатии на кнопку записи, происходит запись данных в файл, при нажатии на кнопку чтения, происходит чтение из файла, при нажатии на кнопку начала или конца выполнения вычисления запускается или останавливается вычислительный поток.

Интерфейс программы представляет собой поля для ввода кнопки чтения, записи, управления потоком вычисления, окно загрузки (рисунок 1).

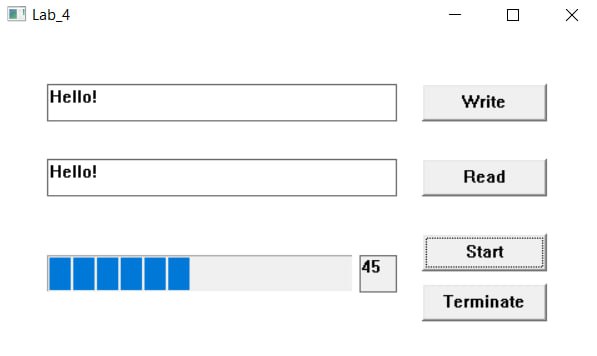


Рисунок 1 – Интерфейс приложения

# 

# **ВЫВОДЫ**

В результате выполнения лабораторной работы было изучено управление процессами и потоками в *Windows,* освоены возможности порождения, завершения, изменения приоритета процессов и потоков. Было создано оконное приложение, выполняющее чтение, запись данных в файл, отрисовку окна загрузки в разных потоках.

# **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. Управление потоками в *WinAPI* [Электронный ресурс]. – Режим доступа: https://ppt-online.org/799671.
2. Потоки в *Win32* [Электронный ресурс]. – Режим доступа: https://club.shelek.ru/viewart.php?id=71.

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

Листинг 1 **–** Файл *Lab\_4\_WINAPI32.cpp*:

#include <windows.h>

#include <cstdlib>

#include <string>

#include <CommCtrl.h>

#define ID\_FILE\_WRITE 100

#define ID\_FILE\_READ 101

#define ID\_CALCULATE 102

#define ID\_PROGRESS\_STATE 103

#define ID\_PROGRESS\_BAR 104

#define ID\_TERMINATE 105

HMENU hMenu;

HWND hWnd, hResult, hSource, hProgress, hProgressBar;

wchar\_t data[100];

int numBytesToWrite;

wchar\_t res[100];

HANDLE writeCompleteEvent;

bool isInProgress;

HBRUSH hProgressBarBrush;

RECT progressBarRect;

void AddControls(HWND hWnd);

void AddMenus(HWND hWnd);

void WriteToFileAsync(HWND hWnd);

void ReadFromFileAsync(HWND hWnd);

void CalculateAsync(WPARAM wp);

void Terminate();

void UpdateProgressState(WPARAM wp);

void UpdateProgressBar(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;

hWnd = CreateWindowW(L"Class", L"Lab\_4", WS\_OVERLAPPEDWINDOW | WS\_VISIBLE, 20, 20, 500, 300, NULL, NULL, NULL, NULL);

MSG msg = { 0 };

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

TranslateMessage(&msg);

DispatchMessage(&msg);

}

return 0;

}

DWORD WINAPI ProgressAsync(LPVOID lpParam) {

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

for (int j = 0; j < 70000000; j++) {

if (isInProgress == false) {

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

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

return 0;

}

int a = 6;

int b = 7;

int c = a + b;

}

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;

}

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;

}

}

break;

case ID\_PROGRESS\_STATE: {

UpdateProgressState(wp);

break;

}

case WM\_PAINT:

{

UpdateProgressBar(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);

}

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 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 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);

}