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)**

Студент Д. С. Кончик

Преподаватель Н. Ю. Гриценко

Минск 2023

СОДЕРЖАНИЕ

[1 Цель работы 3](#_Toc146752068)

[2 Теоретические сведения 4](#_Toc146752069)

[3 Результат выполнения 5](#_Toc146752070)

[Заключение 6](#_Toc146752071)

[Список использованных источников 7](#_Toc146752072)

[Приложение А (обязательное) Листинг кода 8](#_Toc146752073)

1. **ЦЕЛЬ РАБОТЫ**

Исследовать и овладеть методами управления процессами и потоками в операционной системе Windows, включая порождение, завершение и изменение приоритетов процессов и потоков, а также провести анализ и оценку эффективности данных операций с целью оптимизации работы системы и ресурсов.

1. **ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**

Управление процессами и потоками является одной из фундаментальных функций операционной системы Windows, обеспечивающей многозадачность и эффективное распределение ресурсов компьютера. Процессы и потоки представляют основные строительные блоки для параллельного выполнения задач и обеспечивают отзывчивость и производительность приложений. Рассмотрим ключевые аспекты управления процессами и потоками в операционной системе Windows.

Процесс в операционной системе Windows представляет изолированное выполнение приложения. Каждый процесс имеет своё собственное виртуальное адресное пространство, независимые ресурсы и описатели файлов. Процессы обеспечивают изоляцию и безопасность, так как не имеют прямого доступа друг к другу.

Потоки – это наименьшие единицы выполнения внутри процесса. Они разделяют общее адресное пространство и ресурсы процесса. Потоки позволяют выполнять параллельные задачи в рамках одного процесса и облегчают многозадачность.

Создание процессов: В *Windows* процессы создаются с помощью функции *CreateProcess*, которая запускает новое приложение в отдельном процессе. Это позволяет приложениям работать независимо друг от друга.

Потоки создаются с помощью функции *CreateThread*. Каждый поток имеет свою собственную функцию выполнения, что позволяет параллельно выполнять различные задачи в рамках одного процесса.

Процессы завершаются с использованием функции *ExitProcess*, а потоки завершаются с помощью функции *ExitThread*. Завершение процесса приводит к освобождению всех связанных ресурсов и памяти.

В *Windows* каждому процессу и потоку назначается приоритет выполнения. Это позволяет операционной системе распределять процессорное время между процессами и потоками. Приоритеты включают в себя высокий, нормальный, низкий и другие уровни.

Приоритеты могут быть изменены во время выполнения приложения. Это может быть полезным для управления использованием процессорного времени и ресурсов. Повышение приоритета может ускорить выполнение задачи, а понижение - предоставить больше ресурсов другим задачам.

Правильное управление процессами и потоками обеспечивает отзывчивость системы и возможность параллельного выполнения задач. Это важно для многозадачных приложений и систем с множеством пользователей.

1. **РЕЗУЛЬТАТ ВЫПОЛНЕНИЯ**

Было создано приложение для вычисления площади синусоиды на заданном отрезке (рисунок 1).

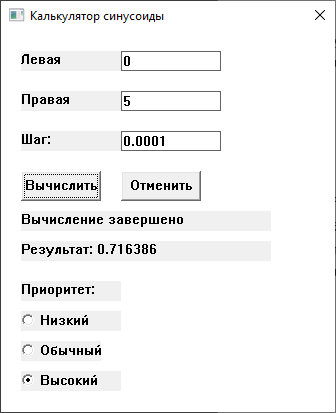


Рисунок 1 – Окно приложения

Предусмотрена возможность выбора левой и правой границы отрезка, на котором располагается функция , для вычисления площади под графиком, а также шага вычисления площади (интеграл Риммана).

Также есть возможность выбора приоритета потока, в котором происходит вычисление (низкий, обычный, высокий).

По нажатию клавиши «Вычислить» начинается вычисление интеграла. В процессе выполнения программы происходит вывод прогресса выполнения в процентном соотношении. По истечении вычисления происходит вывод полученного результата.

По нажатию кнопки «Отменить» происходит отмена текущего вычисления.

# ЗАКЛЮЧЕНИЕ

В рамках выполнения данной лабораторной работы было разработано и рассмотрено приложение, предназначенное для вычисления площади синусоиды на заданном отрезке. Приложение демонстрирует использование основных принципов управления процессами в операционной системе Windows, а также применение механизмов управления приоритетами потоков и обеспечения отзывчивости системы.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Основы программирования для Win32 API [Электронный ресурс]. – Режим доступа: <https://dims.karelia.ru/win32/>.
2. Процессы и потоки [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/ru-ru/windows/win32/api/\_processthreadsapi/.

# ПРИЛОЖЕНИЕ А (обязательное) Листинг кода

Листинг 1 – Файл *Source.cpp*

#include <Windows.h>

#include <cmath>

#include <string>

#include <sstream>

#include <iomanip>

HWND hwnd;

HWND hwndLeftEdit;

HWND hwndRightEdit;

HWND hwndStepEdit;

HWND hwndCancelButton;

HWND hwndProgressLabel;

HWND hwndResultLabel;

HWND hwndLowestPriority;

HWND hwndNormalPriority;

HWND hwndHighestPriority;

const int ID\_CALCULATE\_BUTTON = 1;

const int ID\_CANCEL\_BUTTON = 2;

const int ID\_RESULT\_LABEL = 3;

const int ID\_PROGRESS\_LABEL = 4;

const int ID\_LEFT\_EDIT = 5;

const int ID\_RIGHT\_EDIT = 6;

const int ID\_STEP\_EDIT = 7;

const int ID\_LOWEST\_PRIORITY = 8;

const int ID\_NORMAL\_PRIORITY = 9;

const int ID\_HIGHEST\_PRIORITY = 10;

double leftValue = 0;

double rightValue = 0;

double step = 0.0001;

bool isCalculating = false;

bool isCancelRequested = false;

bool GetDoubleFromEdit(HWND hwndEdit, double& result) {

WCHAR buffer[32];

GetWindowText(hwndEdit, buffer, sizeof(buffer) / sizeof(WCHAR));

try {

result = std::stod(buffer);

return true;

}

catch (...) {

return false;

}

}

DWORD WINAPI CalculateInBackground(LPVOID lpParam) {

double result = 0.0;

for (double x = leftValue; x <= rightValue; x += step) {

if (isCancelRequested) {

SendMessage(hwnd, WM\_USER + 3, 0, 0);

break;

}

result += step \* std::sin(x);

double progress = ((x - leftValue) / (rightValue - leftValue) \* 100);

SendMessage(hwnd, WM\_USER + 1, \*reinterpret\_cast<WPARAM\*>(&progress), 0);

}

if (isCancelRequested)

isCancelRequested = false;

else

SendMessage(hwnd, WM\_USER + 2, \*reinterpret\_cast<WPARAM\*>(&result), 0);

isCalculating = false;

return 0;

}

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

switch (msg) {

case WM\_CREATE: {

// Левая граница.

CreateWindow(L"STATIC", L"Левая граница:", WS\_VISIBLE | WS\_CHILD, 20, 20, 100, 20, hwnd, NULL, NULL, NULL);

hwndLeftEdit = CreateWindow(L"EDIT", L"0", WS\_VISIBLE | WS\_CHILD | WS\_BORDER, 120, 20, 100, 20, hwnd, (HMENU)ID\_LEFT\_EDIT, NULL, NULL);

// Правая граница.

CreateWindow(L"STATIC", L"Правая граница:", WS\_VISIBLE | WS\_CHILD, 20, 60, 100, 20, hwnd, NULL, NULL, NULL);

hwndRightEdit = CreateWindow(L"EDIT", L"1", WS\_VISIBLE | WS\_CHILD | WS\_BORDER, 120, 60, 100, 20, hwnd, (HMENU)ID\_RIGHT\_EDIT, NULL, NULL);

// Шаг.

CreateWindow(L"STATIC", L"Шаг:", WS\_VISIBLE | WS\_CHILD, 20, 100, 100, 20, hwnd, NULL, NULL, NULL);

hwndStepEdit = CreateWindow(L"EDIT", L"0.0001", WS\_VISIBLE | WS\_CHILD | WS\_BORDER, 120, 100, 100, 20, hwnd, (HMENU)ID\_STEP\_EDIT, NULL, NULL);

// Вычислить.

CreateWindow(L"BUTTON", L"Вычислить", WS\_VISIBLE | WS\_CHILD, 20, 140, 80, 30, hwnd, (HMENU)ID\_CALCULATE\_BUTTON, NULL, NULL);

// Отменить.

hwndCancelButton = CreateWindow(L"BUTTON", L"Отменить", WS\_VISIBLE | WS\_CHILD, 120, 140, 80, 30, hwnd, (HMENU)ID\_CANCEL\_BUTTON, NULL, NULL);

// Проценты.

hwndProgressLabel = CreateWindow(L"STATIC", L"", WS\_VISIBLE | WS\_CHILD, 20, 180, 250, 20, hwnd, (HMENU)ID\_PROGRESS\_LABEL, NULL, NULL);

// Результат.

hwndResultLabel = CreateWindow(L"STATIC", L"", WS\_VISIBLE | WS\_CHILD, 20, 210, 250, 20, hwnd, (HMENU)ID\_RESULT\_LABEL, NULL, NULL);

// Приоритет потока.

CreateWindow(L"STATIC", L"Приоритет:", WS\_VISIBLE | WS\_CHILD, 20, 250, 100, 20, hwnd, NULL, NULL, NULL);

hwndLowestPriority = CreateWindow(L"BUTTON", L"Низкий", WS\_VISIBLE | WS\_CHILD | BS\_AUTORADIOBUTTON, 20, 280, 100, 20, hwnd, (HMENU)ID\_LOWEST\_PRIORITY, NULL, NULL);

hwndNormalPriority = CreateWindow(L"BUTTON", L"Обычный", WS\_VISIBLE | WS\_CHILD | BS\_AUTORADIOBUTTON, 20, 310, 80, 20, hwnd, (HMENU)ID\_NORMAL\_PRIORITY, NULL, NULL);

hwndHighestPriority = CreateWindow(L"BUTTON", L"Высокий", WS\_VISIBLE | WS\_CHILD | BS\_AUTORADIOBUTTON, 20, 340, 100, 20, hwnd, (HMENU)ID\_HIGHEST\_PRIORITY, NULL, NULL);

SendMessage(hwndNormalPriority, BM\_SETCHECK, BST\_CHECKED, 0);

break;

}

case WM\_COMMAND: {

if (LOWORD(wParam) == ID\_CALCULATE\_BUTTON) {

if (isCalculating) {

MessageBox(hwnd, L"Идет вычисление!", L"Предупреждение", MB\_ICONWARNING);

}

else {

if (GetDoubleFromEdit(hwndLeftEdit, leftValue) &&

GetDoubleFromEdit(hwndRightEdit, rightValue) &&

GetDoubleFromEdit(hwndStepEdit, step)) {

if (leftValue > rightValue) {

MessageBox(hwnd, L"Левое значение не может быть больше правого!", L"Предупреждение", MB\_ICONWARNING);

}

else {

isCalculating = true;

SetWindowText(hwndResultLabel, L"");

SetWindowText(hwndProgressLabel, L"Идет вычисление...");

int threadPriority = THREAD\_PRIORITY\_NORMAL;

if (SendMessage(hwndLowestPriority, BM\_GETCHECK, 0, 0) == BST\_CHECKED) {

threadPriority = THREAD\_PRIORITY\_LOWEST;

}

else if (SendMessage(hwndHighestPriority, BM\_GETCHECK, 0, 0) == BST\_CHECKED) {

threadPriority = THREAD\_PRIORITY\_HIGHEST;

}

HANDLE hThread = CreateThread(NULL, 0, CalculateInBackground, NULL, 0, NULL);

SetThreadPriority(hThread, threadPriority);

}

}

else {

MessageBox(hwnd, L"Неверный ввод. Пожалуйста, введите корректное число с плавающей запятой.", L"Ошибка", MB\_ICONERROR);

}

}

}

else if (LOWORD(wParam) == ID\_CANCEL\_BUTTON) {

if (isCalculating) {

SendMessage(hwnd, WM\_USER + 3, 0, 0); // Запрос на отмену

}

}

break;

}

case WM\_USER + 1: {

double percent = \*reinterpret\_cast<double\*>(&wParam);

std::wstringstream ss;

ss << L"Прогресс: " << std::fixed << std::setprecision(3) << percent << L"%";

SetWindowText(hwndProgressLabel, ss.str().c\_str());

break;

}

case WM\_USER + 2: {

double result = \*reinterpret\_cast<double\*>(&wParam);

std::wstring resultText = L"Результат: " + std::to\_wstring(result);

SetWindowText(hwndResultLabel, resultText.c\_str());

SetWindowText(hwndProgressLabel, L"Вычисление завершено");

break;

}

case WM\_USER + 3: {

isCancelRequested = true;

SetWindowText(hwndProgressLabel, L"Вычисление отменено");

break;

}

case WM\_DESTROY: {

PostQuitMessage(0);

break;

}

default: {

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

}

}

return 0;

}

int WINAPI WinMain(HINSTANCE hInst, HINSTANCE, LPSTR, int nCmdShow) {

WNDCLASSEX wc = {

sizeof(WNDCLASSEX),

CS\_CLASSDC,

MainWindowProcedure,

0L,

0L,

GetModuleHandle(NULL),

NULL,

NULL,

NULL,

NULL,

L"CalculatorClassName",

NULL

};

RegisterClassEx(&wc);

hwnd = CreateWindow(wc.lpszClassName,

L"Калькулятор синусоиды",

WS\_OVERLAPPED | WS\_CAPTION | WS\_SYSMENU,

0, 0, 350, 420,

NULL,

NULL,

wc.hInstance,

NULL);

ShowWindow(hwnd, nCmdShow);

UpdateWindow(hwnd);

MSG msg;

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

TranslateMessage(&msg);

DispatchMessage(&msg);

}

return 0;

}