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, ИССЛЕДОВАНИЕ ЭФФЕКТИВНОСТИ**

Студент О. Л. Дайнович

Преподаватель Н. Ю. Гриценко

Минск 2023

**СОДЕРЖАНИЕ**

[1 Цель работы 3](#_Toc148080639)

[2 Теоретические сведения 4](#_Toc148080640)

[3 Описание функций программы 6](#_Toc148080641)

[Заключение 8](#_Toc148080642)

[Список использованных источников 9](#_Toc148080643)

[Приложение А (обязательное) Листинг кода 10](#_Toc148080644)

# **1 ЦЕЛЬ РАБОТЫ**

Целью выполнения лабораторной работы является создание оконного приложения на Win32 API, способное совершать манипуляции с процессами, а также исследование различных аспектов управления процессами и потоками в операционной системе Windows и определение оптимальных стратегий для достижения эффективности работы системы.

# **2 ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**

Управление процессами и потоками в операционной системе Windows является важным аспектом для эффективного распределения ресурсов и обеспечения плавной работы системы. Вот некоторые теоретические сведения об управлении процессами и потоками в Windows:

Процесс представляет собой выполняющуюся программу в операционной системе. Каждый процесс имеет свою собственную виртуальную память и набор ресурсов, которые ему выделяются. [1]

В Windows процесс создается с помощью функции CreateProcess или одного из ее вариантов (CreateProcessA, CreateProcessW). При создании процесса задается имя исполняемого файла и другие параметры, такие как командная строка и флаги безопасности.

Каждый процесс имеет уникальный идентификатор (PID), который используется для идентификации процесса в системе.

Поток представляет собой единицу выполнения внутри процесса. Один процесс может содержать несколько потоков, которые выполняются параллельно или конкурируют за ресурсы процесса.

В Windows поток создается с помощью функции CreateThread. При создании потока задается функция, которая будет выполняться внутри потока, и другие параметры, такие как размер стека и флаги безопасности.

Каждый поток имеет уникальный идентификатор (TID), который используется для идентификации потока внутри процесса.

В Windows каждому процессу и потоку назначается приоритет, который определяет его относительную важность и приоритет доступа к процессору.

Приоритеты процессов и потоков могут быть изменены с помощью функций SetPriorityClass и SetThreadPriority. Доступные уровни приоритета включают высшие, обычные, нижние, а также реального времени и фоновые приоритеты.

Изменение приоритетов процессов и потоков может повлиять на их относительное время выполнения и доступ к ресурсам системы.

Завершение процессов и потоков: [2]

Процесс или поток может быть завершен с помощью функции TerminateProcess или TerminateThread. Это приводит к немедленному прекращению выполнения процесса или потока.

При завершении процесса или потока необходимо освободить все выделенные ресурсы и выполнить необходимые действия для корректного завершения работы.

Для изучения эффективности управления процессами и потоками можно использовать различные метрики, такие как время отклика, использование процессора, объем используемой памяти и т.д.

Сравнение различных стратегий управления, таких как порождение процессов и потоков с разными параметрами, изменение приоритетов и т.д., позволяет оценить их влияние на общую производительность системы.

Исследование эффективности также может включать измерение времени, затрачиваемого на операции порождения, завершения иизменения приоритетов процессов и потоков, чтобы определить, какие операции требуют больше ресурсов и могут оказывать большее влияние на производительность системы.

Управление процессами и потоками в операционной системе Windows позволяет достичь более эффективного использования ресурсов и обеспечить более плавное выполнение задач. Исследование эффективности управления процессами и потоками позволяет оптимизировать работу системы и выбрать наилучшие стратегии для различных сценариев использования. [3]

# **3 ОПИСАНИЕ ФУНКЦИЙ ПРОГРАММЫ**

В ходе выполнения лабораторной работы было разработано оконное приложение для вычисления интеграла функции синуса и отслеживания прогресса вычисления в реальном времени (рисунок 1).

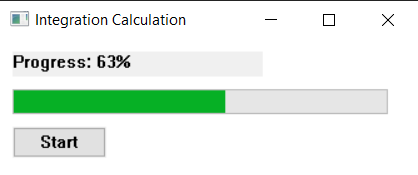


Рисунок 1 — Графический интерфейс приложения

В случае успешного окончания подсчета интеграла поток, в котором происходило вычисление, закрывается, а на экран выводится соответствующее сообщение (рисунок 2).

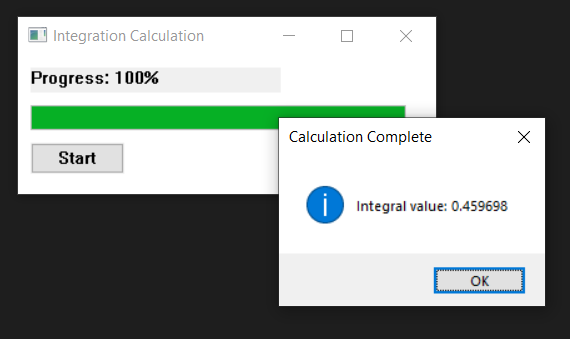


Рисунок 2 — Успешное окончание подсчета

В случае попытки начать еще один процесс вычисления интеграла появится сообщение с предупреждением, однако процесс вычисления интеграла не прервется (рисунок 3).

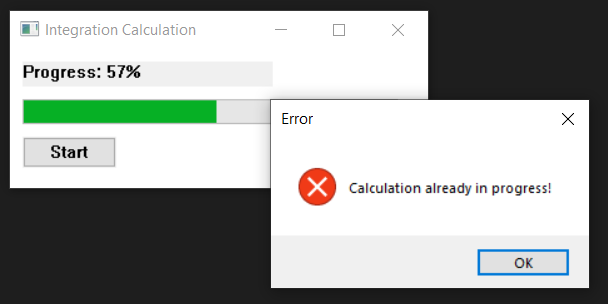


Рисунок 3 — Сообщение с предупреждением

# **ЗАКЛЮЧЕНИЕ**

В результате лабораторной работы были изучены принципы работы с памятью Win32 API. Было создано оконное приложение, позволяющее вычислять математическое выражение и наблюдать за выполнением вычисления в реальном времени, что демонстрирует многопоточную работу данного приложения.

# **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

[1] Microsoft Learn [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/ru-ru/windows/win32/learnwin32/.

[2] Начало работы с классическими приложениями для Windows, которые используют API Win32 [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/ru-ru/windows/win32/desktop-programming.

[3] Сообщения окна [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/ru-ru/windows/win32/learnwin32/window-messages.

# **ПРИЛОЖЕНИЕ А**

# **(обязательное)**

# **Листинг кода**

Листинг 1 – Файл main.cpp

#include <Windows.h>

#include <cmath>

#include <string>

#include <CommCtrl.h>

#pragma comment(lib, "Comctl32.lib")

#pragma comment(linker,"\"/manifestdependency:type='win32' \ name = 'Microsoft.Windows.Common-Controls' version = '6.0.0.0' \

processorArchitecture = '\*' publicKeyToken = '6595b64144ccf1df' language = '\*'\"")

#pragma once

// Объявление глобальных переменных

HWND g\_hMainWindow;

HWND g\_hProgressLabel;

HWND g\_hProgressBar;

HANDLE g\_hCalculationThread;

bool g\_bCalculationInProgress = false;

LRESULT CALLBACK WindowProc(HWND hwnd, UINT uMsg, WPARAM wParam, LPARAM lParam);

double CalculateIntegral();

void UpdateProgress(double progress);

DWORD WINAPI CalculationThreadProc(LPVOID lpParam);

int WINAPI wWinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, PWSTR pCmdLine, int nCmdShow)

{

// Регистрация класса окна

const wchar\_t CLASS\_NAME[] = L"Sample Window Class";

WNDCLASS wc = { };

wc.lpfnWndProc = WindowProc;

wc.hInstance = hInstance;

wc.lpszClassName = CLASS\_NAME;

RegisterClass(&wc);

InitCommonControls();

// Создание главного окна

g\_hMainWindow = CreateWindowEx(0, L"Sample Window Class", L"Integration Calculation",

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

350, 150, NULL, NULL, hInstance, NULL);

if (g\_hMainWindow == NULL)

{

return 0;

}

ShowWindow(g\_hMainWindow, nCmdShow);

UpdateWindow(g\_hMainWindow);

// Цикл обработки сообщений

MSG msg;

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

{

TranslateMessage(&msg);

DispatchMessage(&msg);

}

return 0;

}

// Функция обработки сообщений главного окна

LRESULT CALLBACK WindowProc(HWND hwnd, UINT uMsg, WPARAM wParam, LPARAM lParam)

{

switch (uMsg)

{

case WM\_CREATE:

{

// Создание надписи прогресса

g\_hProgressLabel = CreateWindowEx(0, L"STATIC", L"Progress: 0%",

WS\_VISIBLE | WS\_CHILD, 10, 10, 200, 20, hwnd, NULL, NULL, NULL);

// Создание прогресс-бара

g\_hProgressBar = CreateWindowEx(0, PROGRESS\_CLASS, NULL,

WS\_VISIBLE | WS\_CHILD, 10, 40, 300, 20, hwnd, NULL, NULL, NULL);

// Создание кнопки "Start"

CreateWindowEx(0, L"BUTTON", L"Start", WS\_VISIBLE | WS\_CHILD,

10, 70, 75, 25, hwnd, reinterpret\_cast<HMENU>(1), NULL, NULL);

}

return 0;

case WM\_COMMAND:

{

if (LOWORD(wParam) == 1) // ID кнопки "Start"

{

if (!g\_bCalculationInProgress)

{

// Запуск потока вычисления

g\_hCalculationThread = CreateThread(NULL, 0, CalculationThreadProc, NULL, 0, NULL);

}

else

{

MessageBox(hwnd, L"Calculation already in progress!", L"Error", MB\_OK | MB\_ICONERROR);

}

}

}

return 0;

case WM\_DESTROY:

{

if (g\_bCalculationInProgress)

{

MessageBox(hwnd, L"Calculation in progress. Please wait...", L"Warning", MB\_OK | MB\_ICONWARNING);

return 0;

}

if (g\_hCalculationThread)

{

CloseHandle(g\_hCalculationThread);

g\_hCalculationThread = NULL;

}

PostQuitMessage(0);

}

return 0;

}

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

}

// Функция вычисления интеграла

double CalculateIntegral()

{

const double lowerBound = 0.0;

const double upperBound = 1;

const int numSteps = 500000000; // Количество шагов

double stepSize = (upperBound - lowerBound) / numSteps;

double integral = 0.0;

double proc = 0.0;

for (int i = 0; i < numSteps; ++i)

{

double x = lowerBound + (i + 0.5) \* stepSize;

integral += std::sin(x) \* stepSize;

if (i % 50000 == 0) {

proc = double(i) / 500000000;

UpdateProgress(proc);

}

}

UpdateProgress(1);

return integral;

}

// Функция обновления прогресса вычисления

void UpdateProgress(double progress)

{

int progressValue = static\_cast<int>(progress \* 100);

SendMessage(g\_hProgressBar, PBM\_SETPOS, progressValue, 0);

std::wstring progressText = L"Progress: " + std::to\_wstring(progressValue) + L"%";

SetWindowText(g\_hProgressLabel, progressText.c\_str());

}

// Функция вычисления интеграла в отдельном потоке

DWORD WINAPI CalculationThreadProc(LPVOID lpParam)

{

g\_bCalculationInProgress = true;

double integral = CalculateIntegral();

g\_bCalculationInProgress = false;

MessageBox(g\_hMainWindow, (L"Integral value: " + std::to\_wstring(integral)).c\_str(), L"Calculation Complete", MB\_OK | MB\_ICONINFORMATION);

SendMessage(g\_hProgressBar, PBM\_SETPOS, 0, 0);

SetWindowText(g\_hProgressLabel, L"Progress: 0%");

return 0;

}