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

ОТЧЁТ

к лабораторной работе №4

на тему

Управление процессами и потоками (Windows). Порождение, завершение, изменение приоритетов процессов и потоков, исследование эффективности.

Выполнил: студент группы 153503

Киселёва Елизавета Андреевна

Проверил: Гриценко Никита Юрьевич

Минск 2023

СОДЕРЖАНИЕ

[1 Постановка задачи 3](#_Toc146728376)

[2 Краткие теоретические сведения 4](#_Toc146728377)

[3 Результаты выполнения лабораторной работы 7](#_Toc146728378)

[Выводы 7](#_Toc146728380)

[Список использованных исчтоников 8](#_Toc146728381)

[Приложение А (обязательное) Листинг кода 11](#_Toc146728382)

## 1 ПОСТАНОВКА ЗАДАЧИ

Целью выполнения лабораторной работы является создание приложения с использованием управления процессами и потоками (*Windows*), порождения, завершения, изменения приоритетов процессов и потоков, исследование эффективности. Для этого будет создано приложение для многопоточного вычисления экспоненциальной функции.

## 2 КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

## 2.1 Управление процессами и потоками (*Windows*)

Управление процессами и потоками в операционной системе Windows является фундаментальной частью обеспечения функционирования приложений. Процесс представляет собой исполняющуюся программу, а поток – независимую последовательность выполнения внутри процесса.

Иными словами, процесс в операционной системе – это экземпляр выполняемой программы. Он представляет собой абстракцию, которая включает в себя код программы, данные, ресурсы и контекст выполнения. Процесс предоставляет изолированное окружение, в котором программа может выполняться.

Поток (или поток выполнения) в операционной системе – это наименьшая единица выполнения внутри процесса. Потоки представляют собой легковесные подзадачи, работающие параллельно в рамках одного процесса. Они используют общее адресное пространство и ресурсы процесса, но имеют собственный стек вызовов и счетчик программы. [1]

Создание процессов осуществляется с использованием системного вызова *CreateProcess*, который запускает новый процесс. Процессы могут быть завершены с помощью функции *TerminateProcess*.

Внутри процесса действуют потоки, создаваемые функцией *CreateThread*. Завершение потока выполняется через *ExitThread*. Для управления приоритетами процесса и потоков используются функции *SetPriorityClass* и *SetThreadPriority*.

Изучение эффективности процессов и потоков предполагает мониторинг ресурсов. *API* предоставляет функции, такие как *GetProcessTimes* и *GetThreadTimes*, для получения информации о времени использования процесса и потоков. [2]

## 2.2 Многозадачность в *Windows* и синхронизация потоков

Многозадачность в *Windows* реализуется через планирование потоков операционной системой. Синхронизация потоков осуществляется с использованием мьютексов, семафоров и других механизмов, предотвращая возможные гонки данных.

Мьютексы (*Mutex*) и семафоры (*Semaphore*) представляют собой объекты синхронизации в *Win*32 *API*, обеспечивающие взаимоисключение и координацию между процессами или потоками.

Мьютексы используются для обеспечения эксклюзивного доступа к ресурсам. Только один поток (или процесс) может удерживать мьютекс, что предотвращает конфликты доступа к общим ресурсам. Они часто используются для синхронизации доступа к разделяемым данным, когда необходимо гарантировать, что только один поток обладает доступом к ресурсам в конкретный момент времени.

Семафоры обеспечивают управление доступом к количеству одновременно работающих потоков (или процессов). Семафоры имеют счетчик, который уменьшается при запросе на доступ и увеличивается при завершении работы. Используются для ограничения количества потоков, имеющих доступ к определенному ресурсу. Также могут использоваться для синхронизации действий между потоками.

Мьютексы используются, например, при предотвращении конфликтов при записи в файл, где только один поток должен модифицировать данные в определенный момент времени.

Семафоры же могут использоваться при ограничении количества потоков, выполняющих параллельную обработку запросов к серверу, чтобы не перегрузить систему.

Мьютексы и семафоры являются важными инструментами для предотвращения гонок данных и координации между параллельными процессами или потоками. Их неправильное использование может привести к дедлокам (ситуация в многозадачной среде, при которой два или более процесса не могут продолжить выполнение из-за блокировки друг друга, ожидая ресурсы, которые удерживают другие процессы) или низкой производительности, поэтому важно внимательно проектировать их применение в контексте конкретной задачи. [3]

## 2.3 Асинхронные операции ввода-вывода

Асинхронные операции ввода-вывода реализуются с использованием *I/O Completion Ports*, обеспечивая эффективный и отзывчивый обмен данными между процессами и потоками.

Они позволяют программам выполнять операции ввода-вывода без блокировки основного потока выполнения. Это способствует эффективному использованию ресурсов и повышению производительности, так как не блокируется поток во время ожидания завершения операции.

*I/O CP* – это механизм в *Windows* для обработки асинхронных операций ввода-вывода. Программа связывает порт завершения с файловым дескриптором или устройством ввода-вывода. Когда асинхронная операция завершается, информация о завершении операции помещается в очередь завершения порта. Основной поток или рабочие потоки могут ожидать завершения операций, получая уведомления из очереди завершения. Это обеспечивает высокую параллельность и эффективное использование ресурсов.

Порты завершения обеспечивают высокую производительность, масштабируемость и уменьшение потребления ресурсов. Использование портов завершения улучшает масштабируемость приложения при обработке большого числа клиентов или операций ввода-вывода. Они позволяют обрабатывать множество операций одновременно, минимизируя ожидание. [4]

## 3 РЕЗУЛЬТАТЫ ВЫПОЛНЕНИЯ ЛАБОРАТОРНОЙ РАБОТЫ

В ходе выполнения лабораторной работы было разработано приложение для многопоточного вычисления экспоненциальной функции. Приложение позволяет пользователю посчитать значение нескольких экспоненциальных функций в точке и отобразить полученные значения на графике функции. Также пользователь может видеть название приложения и его иконку (Рисунок 1).

****

Рисунок 1 – Главное окно

Пользователь может ввести число, экспоненциальную функцию от которого хочет найти (Рисунок 2).



Рисунок 2 – Введенное число для нахождения функции

Пользователь видит результат подсчета, а также соответствующую точку на графике экспоненциальной функции (Рисунок 3).

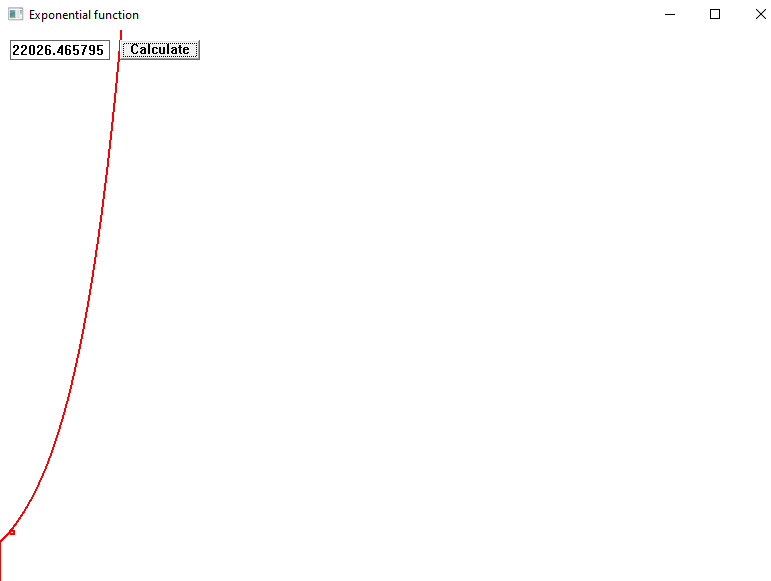


Рисунок 3 – Отображение результата

## ВЫВОДЫ

В ходе выполнения данной лабораторной работы было разработано приложение для многопоточного вычисления экспоненциальной функции. Проект включает в себя создание графического окна приложения, обработку оконных сообщений, пользовательский интерфейс с кнопками, использование управления потоками *Windows*, мьютексом. Результатом стало рабочее приложение для многопоточного вычисления экспоненциальной функции, имеющее понятное отображение экспоненциальной функции и способное считать значения этой функции, а также выводить на график соответствующие точки.

## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Щупак Ю. *Win*32 *API*. Разработка приложений для *Windows*. – СПб: Питер, 2008. – 592 с.: ип.

[2] *Microsoft* *Learn* [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/ru-ru/windows/win32/sync/interprocess-synchronization – Дата доступа 11.10.2023

[3] Многопоточность и синхронизация[Электронный ресурс]. – Режим доступа: http://www.kurzenkov.com/software\_development/multithreading3.html – Дата доступа 11.10.2023

[4] Порт завершения[Электронный ресурс]. – Режим доступа: https://habr.com/ru/articles/59282/ – Дата доступа 11.10.2023

## ПРИЛОЖЕНИЕ А

## (обязательное)

## Листинг кода

**main.cpp**

#include <windows.h>

#include <tchar.h>

#include <cmath>

#include <vector>

#include <thread>

#include <string>

#include <atomic>

std::vector<std::thread> threads;

double result = 0.0;

double inputX = 0.0; // Добавлено для хранения введенного значения x

HANDLE mutex;

std::atomic<bool> exitFlag(false);

std::vector<double> inputPoints;

// Переменная для отслеживания предыдущего результата

double previousResult = 0.0;

void CalculateExponential(double x, HWND hInputEdit, HWND hWnd) {

while (true) {

// Захватываем мьютекс

WaitForSingleObject(mutex, INFINITE);

// Проверяем флаг завершения

if (exitFlag) {

ReleaseMutex(mutex);

break;

}

// Имитация долгой операции

result = exp(x);

// Сохраняем введенное значение x

inputX = x;

// Сохраняем введенное значение x

inputPoints.push\_back(x);

// Отправляем сообщение для установки текста в поле ввода только при изменении результата

if (result != previousResult) {

previousResult = result;

// Обновляем текст в поле ввода

SetWindowText(hInputEdit, std::to\_wstring(result).c\_str());

// Перерисовываем окно

InvalidateRect(hWnd, NULL, TRUE);

}

// Освобождаем мьютекс

ReleaseMutex(mutex);

// Даем немного времени, чтобы обновление успело произойти перед следующей итерацией

std::this\_thread::sleep\_for(std::chrono::milliseconds(2000));

}

}

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

switch (message) {

case WM\_COMMAND:

if (HIWORD(wParam) == BN\_CLICKED) {

if (LOWORD(wParam) == 1) {

// Получаем значение из поля для ввода

TCHAR buffer[256];

GetWindowText(GetDlgItem(hWnd, 0), buffer, sizeof(buffer) / sizeof(buffer[0]));

double x = \_ttof(buffer);

// Запускаем потоки для вычисления

threads.emplace\_back(CalculateExponential, x, GetDlgItem(hWnd, 0), hWnd);

}

}

break;

case WM\_DESTROY:

// Устанавливаем флаг завершения

exitFlag.store(true);

// Ждем завершения потоков

for (std::thread& t : threads) {

t.join();

}

// Закрываем дескриптор мьютекса

CloseHandle(mutex);

// Завершаем программу

PostQuitMessage(0);

break;

case WM\_PAINT:

{

PAINTSTRUCT ps;

HDC hdc = BeginPaint(hWnd, &ps);

// Получаем размеры клиентской области окна

RECT clientRect;

GetClientRect(hWnd, &clientRect);

// Создаем красное перо для рисования линии

HPEN hPen = CreatePen(PS\_SOLID, 2, RGB(255, 0, 0));

HGDIOBJ hOldPen = SelectObject(hdc, hPen);

// Рисуем график экспоненты

MoveToEx(hdc, 0, clientRect.bottom, NULL);

for (int i = 0; i < clientRect.right; ++i) {

double x = static\_cast<double>(i);

double y = exp(x / 50.0) \* 50.0; // Экспоненциальная функция, масштабированная для лучшей видимости

LineTo(hdc, i, clientRect.bottom - static\_cast<int>(y));

}

// Рисуем точки введенных значений

for (double x : inputPoints) {

Ellipse(hdc, static\_cast<int>(x), clientRect.bottom - static\_cast<int>(exp(x / 50.0) \* 50.0),

static\_cast<int>(x) + 5, clientRect.bottom - static\_cast<int>(exp(x / 50.0) \* 50.0) + 5);

}

// Очищаем ресурсы

SelectObject(hdc, hOldPen);

DeleteObject(hPen);

EndPaint(hWnd, &ps);

}

break;

default:

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

}

return 0;

}

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

UNREFERENCED\_PARAMETER(hPrevInstance);

UNREFERENCED\_PARAMETER(lpCmdLine);

const TCHAR szWindowClass[] = \_T("Win32App");

WNDCLASSEX wcex;

wcex.cbSize = sizeof(WNDCLASSEX);

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

wcex.lpfnWndProc = WndProc;

wcex.cbClsExtra = 0;

wcex.cbWndExtra = 0;

wcex.hInstance = hInstance;

wcex.hIcon = LoadIcon(hInstance, IDI\_APPLICATION);

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

wcex.hbrBackground = (HBRUSH)(COLOR\_WINDOW + 1);

wcex.lpszMenuName = NULL;

wcex.lpszClassName = szWindowClass;

wcex.hIconSm = LoadIcon(wcex.hInstance, IDI\_APPLICATION);

if (!RegisterClassEx(&wcex)) {

MessageBox(NULL, \_T("Call to RegisterClassEx failed!"), \_T("Win32 Guided Tour"), NULL);

return 1;

}

// Создание окна

HWND hWnd = CreateWindow(szWindowClass, \_T("Exponential function"), WS\_OVERLAPPEDWINDOW,

CW\_USEDEFAULT, CW\_USEDEFAULT, 800, 600, NULL, NULL, hInstance, NULL);

if (!hWnd) {

MessageBox(NULL, \_T("Call to CreateWindow failed!"), \_T("Win32 Guided Tour"), NULL);

return 1;

}

// Создание мьютекса

mutex = CreateMutex(NULL, FALSE, NULL);

if (mutex == NULL) {

MessageBox(NULL, \_T("Mutex creation failed."), \_T("Error"), NULL);

return 1;

}

// Создаем поле для ввода

HWND hEdit = CreateWindow(L"EDIT", L"", WS\_CHILD | WS\_VISIBLE | WS\_BORDER | ES\_LEFT,

10, 10, 100, 20, hWnd, NULL, hInstance, NULL);

// Создаем кнопку

HWND hButton = CreateWindow(\_T("BUTTON"), \_T("Calculate"), WS\_CHILD | WS\_VISIBLE,

120, 10, 80, 20, hWnd, (HMENU)1, hInstance, NULL);

// Отображение окна

ShowWindow(hWnd, nCmdShow);

UpdateWindow(hWnd);

// Цикл обработки сообщений

MSG msg;

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

TranslateMessage(&msg);

DispatchMessage(&msg);

}

return (int)msg.wParam;

}