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 №1

на тему

**УПРАВЛЕНИЕ ПРОЦЕССАМИ, ПОТОКАМИ, НИТЯМИ.**

Выполнил: студент гр.253505 Авдошко И.С.

Проверил: ассистент кафедры информатики Гриценко Н.Ю.

Минск 2024

ОГЛАВЛЕНИЕ

[1 Формулировка задачи 3](#_Toc178181893)

[2 Краткие теоритические сведения 4](#_Toc178181894)

[3 Описание функций программы 5](#_Toc178181895)

[Заключение 7](#_Toc178181899)

[Список использованных источников 8](#_Toc178181900)

[Приложение А (обязательное) 9](#_Toc178181901)

# 1 ФОРМУЛИРОВКА ЗАДАЧИ

Цель лабораторной работы – возобновление, закрепление и развитие навыков программирования приложений Windows. В процессе выполнения лабораторной работы будут изучены концепции вычислительных процессов, потоков, нитей и их реализация в Windows, основные этапы жизненного цикла процессов (потоков) и элементарное управление ими, такое как порождение, завершение, получение и изменение состояния, а также типичное (простое) использование многозадачности и многопоточности.

В качестве задачи необходимо разработать самовосстанавливающийся процесс, то есть процесс, возобновляющий работу с сохранением (продолжением) текущих функций после завершения его сообщением WM\_CLOSE. Игнорирование сообщения такого процесса должно приводить либо к объявлению его аварийным (автоматически), либо принудительному безусловному завершению посредством TaskManager. Процесс будет завершаться штатным образом, но при этом порождать свою копию. Копия продолжит выполняться вместо родительского процесса. Требуется решить проблему необходимости продолжить работу с теми же обрабатываемыми данными.

Вариант решения: хранение текущих рабочих данных в структуре («прикладной контекст»), которая может выгружаться на диск или сохраняться в глобальной памяти (родителем) и загружаться оттуда (потомком-«наследником»). [1]

2 КРАТКИЕ ТЕОРИТИЧЕСКИЕ СВЕДЕНИЯ

Приложение может состоять из одного или нескольких процессов. Процессы — это основные единицы выполнения программ в операционных системах. Каждый процесс имеет уникальный идентификатор (PID), собственное пространство памяти и набор системных ресурсов, таких как файловые дескрипторы и сетевые соединения. [2]

Основные этапы жизненного цикла процесса включают:

– Создание процесса: Процесс создается вызовом функции, например, CreateProcess в Windows. Операционная система выделяет необходимые ресурсы и добавляет новый процесс в таблицу процессов.

– Выполнение процесса: На этом этапе процесс получает процессорное время и начинает выполнение, осуществляя операции с памятью, вводом-выводом и взаимодействуя с другими процессами.

– Завершение процесса: После нормального или аварийного выполнения процесс сообщает родительскому процессу о своем завершении. Если эта информация не будет получена, процесс может стать зомби.

В контексте процесса могут выполняться один или несколько потоков. Поток — это базовая единица, которой операционная система выделяет процессорное время. Поток может выполнять любую часть кода процесса, включая те части, которые в настоящее время выполняются другим потоком.

Объект задания позволяет управлять группами процессов как единое целое. Объекты заданий — это именуемые, защищаемые, общие объекты, управляющие атрибутами связанных с ними процессов. Операции, выполняемые с объектом задания, влияют на все процессы, связанные с объектом задания.

Пул потоков представляет собой коллекцию рабочих потоков, предназначенных для эффективного выполнения асинхронных обратных вызовов от имени приложения. Основная цель пула потоков — уменьшение количества потоков в приложении и управление рабочими потоками.

Нить (или поток) — это единица выполнения, которую приложение должно планировать самостоятельно. Нити выполняются в контексте потоков, которые осуществляют их планирование. [3]

3 ОПИСАНИЕ ФУНКЦИЙ ПРОГРАММЫ

Исходя из формулировки задачи, был разработан самовосстанавливающийся процесс в оконном приложении Windows.

Были реализованы следующие особенности:

– при вызове сообщения WM\_CLOSE процесс завершается, но при этом порождает свою копию;

– процесс возобновляет работу с сохранением (продолжением) текущих функций.

После запуска процесса появляется окно приложения с полем для ввода текста, представленное на рисунке 3.1.

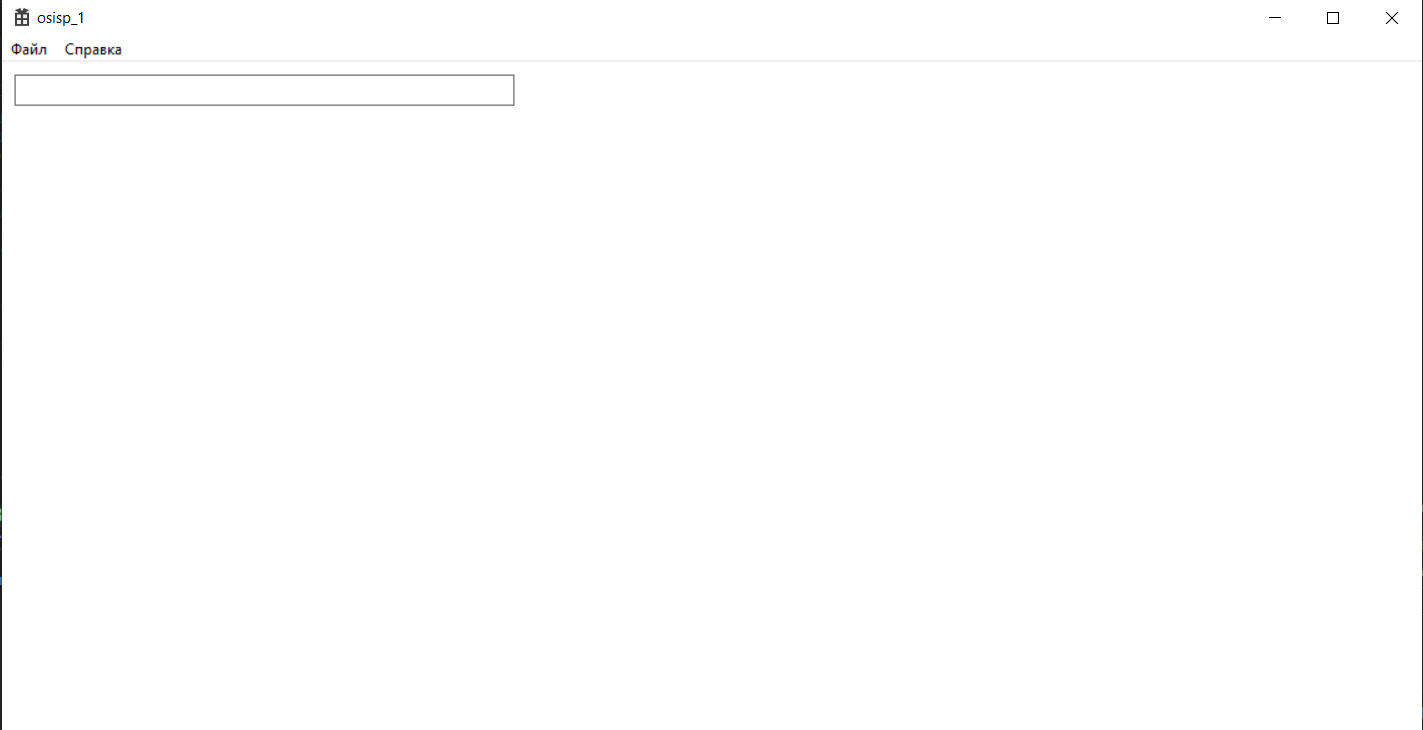


Рисунок 3.1 – Окно приложения

Далее имеется возможность ввести текст в поле и редактировать его, как на рисунке 3.2.

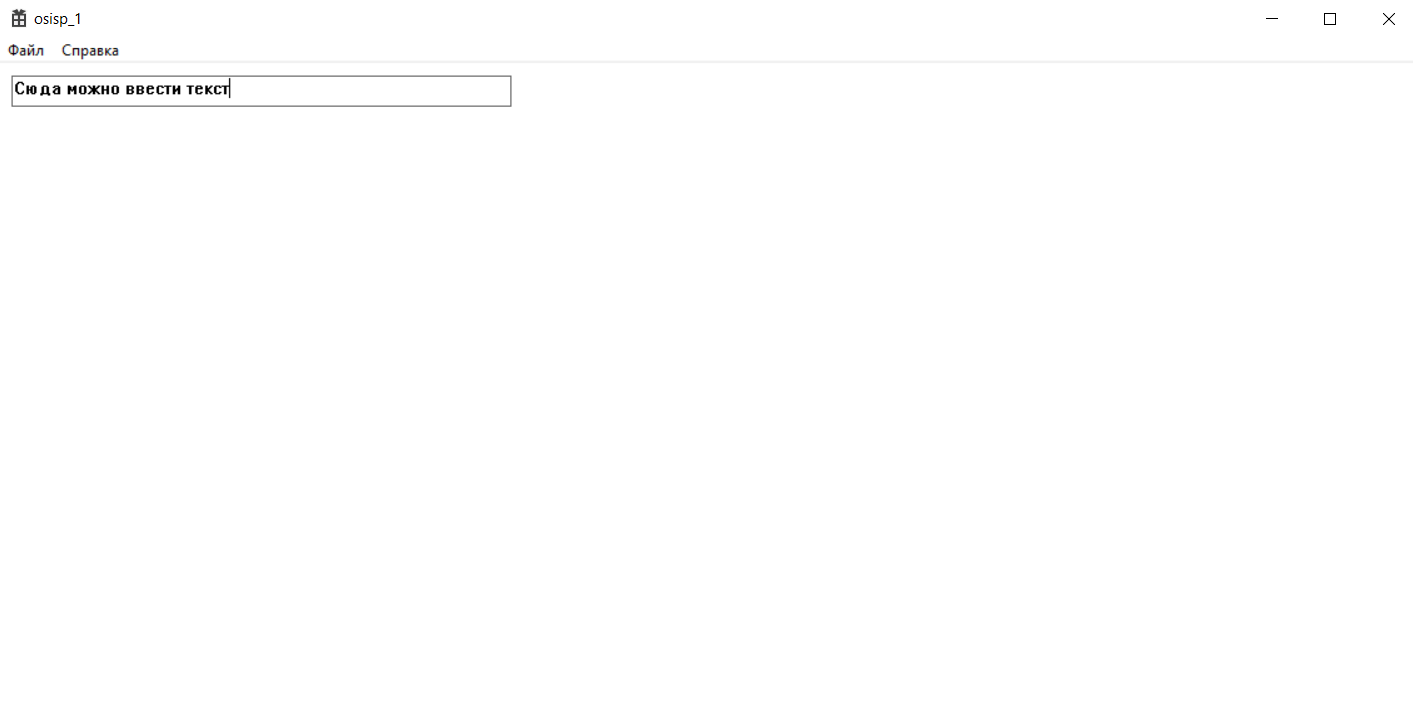


Рисунок 3.2 – Ввод текста в текстовое поле

При закрытии процесса (например, нажатием на крестик в верхнем правом углу окна приложения), данный процесс завершится в штатном режиме, при этом сразу запустится его копия с сохраненными данными (в нашем случае с сохраненными параметрами поля для текстового ввода). Данный процесс продемонстрирован на рисунке 3.3.

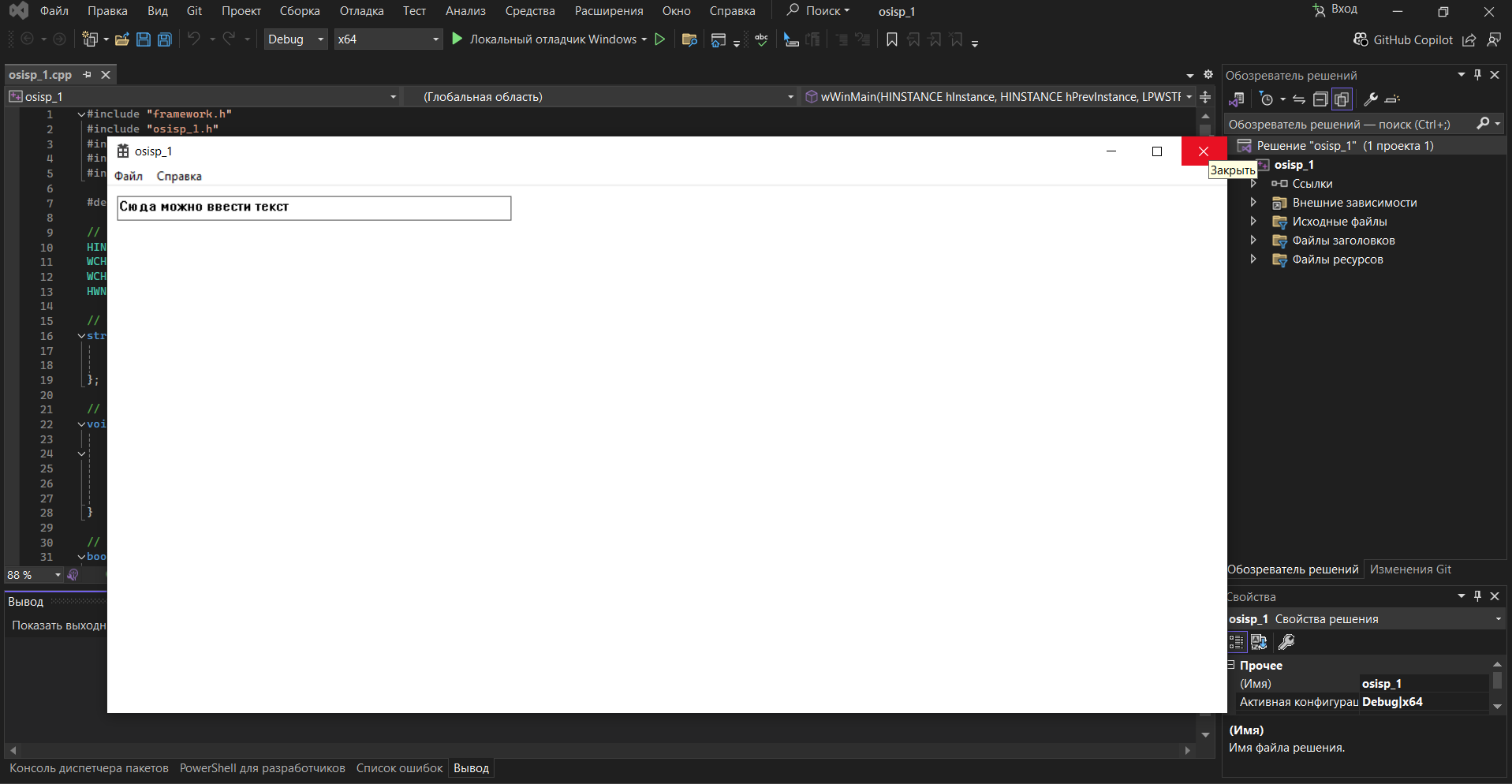


Рисунок 3.3 – Завершение текущего процесса

Открытие копии процесса продемонстрировано на рисунке 3.4.

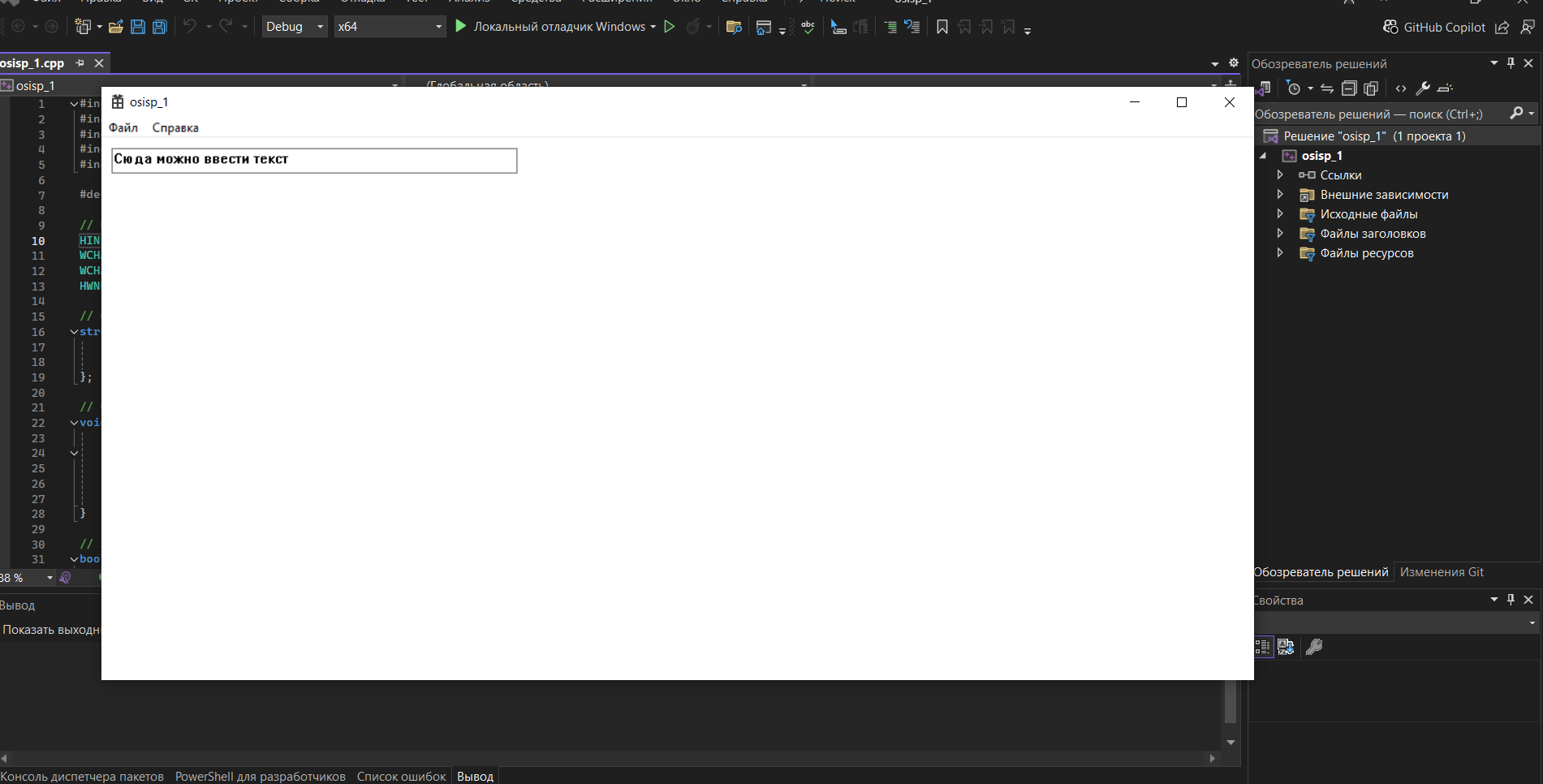


Рисунок 3.4 – Открытие копии родительского процесса

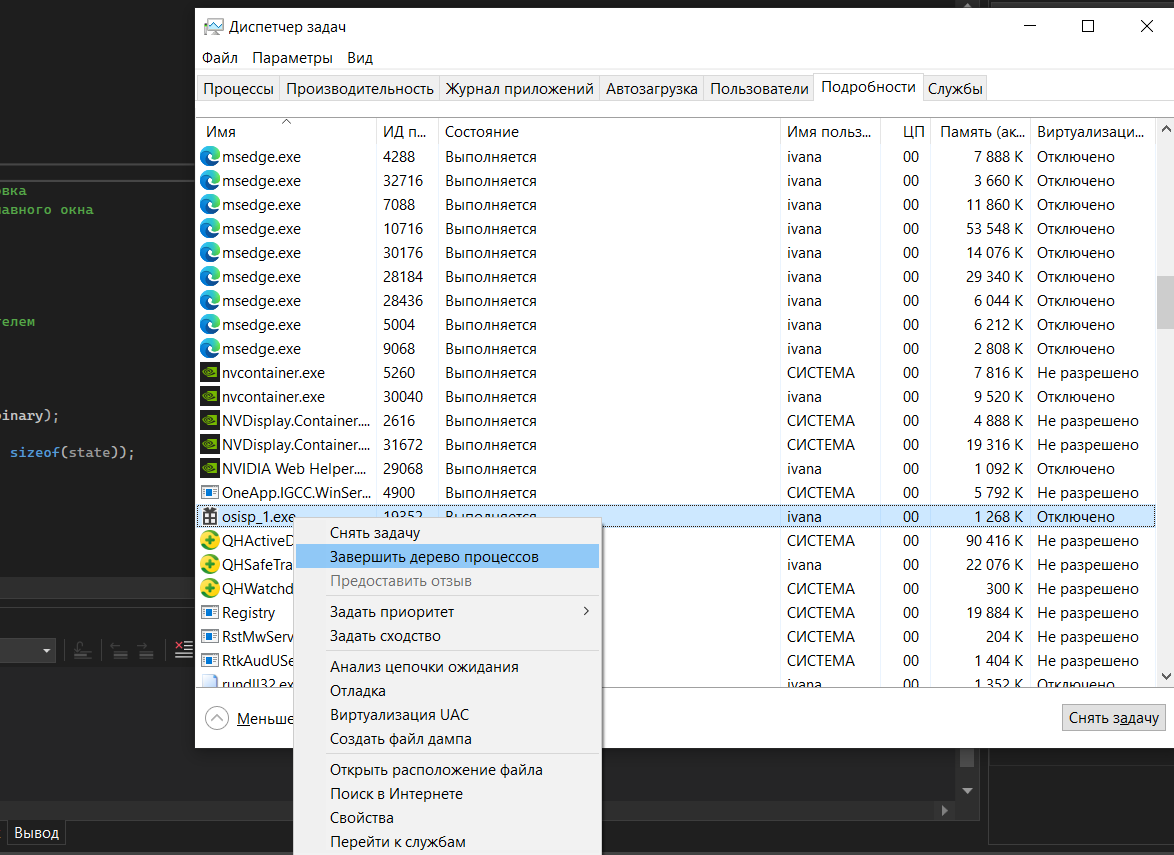


Рисунок 3.5 – Завершение дерева процессов

Для принудительного безусловного завершения процесса необходимо завершить дерево процессов через TaskManager, как на рисунке 3.5.

# ЗАКЛЮЧЕНИЕ

В ходе выполнения лабораторной работы по данной дисциплине была разработана программа, реализующая самовосстанавливающийся процесс, который при завершении порождает свою копию и сохраняет текущее состояние для продолжения работы. Основной задачей было добиться того, чтобы процесс мог восстанавливаться без потери данных и сохранять состояние предыдущего процесса, обеспечивая бесшовный переход между завершением старого и запуском нового процесса.

Для реализации этой задачи были использованы функции WinAPI для управления процессами, такие как CreateProcess, позволяющая порождать новые процессы, и GetWindowText с последующей записью состояния в файл, чтобы сохранить текущее состояние процесса. Функция LoadState позволила восстанавливать состояние после создания нового процесса, что обеспечило непрерывность работы программы даже после ее завершения.

Программа продемонстрировала эффективное взаимодействие с операционной системой, обеспечивая корректное управление ресурсами и сохранение состояния при закрытии. Такой подход может быть полезен в приложениях, требующих устойчивости к сбоям или аварийным завершениям. Это решение может быть применено в сценариях, связанных с долгосрочными вычислительными задачами или критически важными системами, где требуется непрерывное выполнение процесса и минимизация потерь данных.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Build desktop Windows apps using the Win32 API – Электронный ресурс. – Режим доступа: https://learn.microsoft.com/en-us/windows/win32/

[2] Разработка приложений с помощью WinAPI. – Электронный ресурс. – Режим доступа: https://shorturl.at/BDJW8

[3] Основные сообщения ОС Windows (Win32 API). Программирование в ОС Windows. Лекция 1. – Электронный ресурс. – Режим доступа: https://www.youtube.com/watch?v=wTArIolxch0

ПРИЛОЖЕНИЕ А

(обязательное)

**Исходный код программы**

#include "framework.h"

#include "osisp\_1.h"

#include <windows.h>

#include <iostream>

#include <fstream>

#define MAX\_LOADSTRING 100

// Глобальные переменные:

HINSTANCE hInst; // текущий экземпляр

WCHAR szTitle[MAX\_LOADSTRING]; // Текст строки заголовка

WCHAR szWindowClass[MAX\_LOADSTRING]; // имя класса главного окна

HWND hEdit; // Дескриптор текстового поля

// Структура для сохранения рабочего состояния

struct AppState {

int someData; // Пример данных

char userText[256]; // Текст, введенный пользователем

};

// Сохранение состояния в файл

void SaveState(AppState& state) {

std::ofstream outFile("app\_state.dat", std::ios::binary);

if (outFile.is\_open()) {

outFile.write(reinterpret\_cast<char\*>(&state), sizeof(state));

outFile.close();

}

}

// Загрузка состояния из файла

bool LoadState(AppState& state) {

std::ifstream inFile("app\_state.dat", std::ios::binary);

if (inFile.is\_open()) {

inFile.read(reinterpret\_cast<char\*>(&state), sizeof(state));

inFile.close();

return true;

}

return false;

}

// Функция для создания копии процесса

void CreateNewProcess() {

STARTUPINFO si = { sizeof(si) };

PROCESS\_INFORMATION pi;

// Порождаем новый процесс

if (!CreateProcess(NULL, GetCommandLine(), NULL, NULL, FALSE, 0, NULL, NULL, &si, &pi)) {

std::cerr << "Ошибка создания процесса: " << GetLastError() << std::endl;

}

else {

CloseHandle(pi.hProcess);

CloseHandle(pi.hThread);

}

}

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

static AppState appState = { 0 }; // Начальное состояние

switch (msg) {

case WM\_CREATE: {

// Создаем текстовое поле

hEdit = CreateWindowEx(0, L"EDIT", NULL, WS\_CHILD | WS\_VISIBLE | WS\_BORDER | ES\_LEFT,

10, 10, 400, 25, hwnd, NULL, hInst, NULL);

// Загрузка состояния после создания текстового поля

if (LoadState(appState)) {

SetWindowTextA(hEdit, appState.userText); // Восстановление текста

}

break;

}

case WM\_CLOSE: {

// Сохранение состояния

GetWindowTextA(hEdit, appState.userText, sizeof(appState.userText));

SaveState(appState);

// Порождение новой копии процесса

CreateNewProcess();

// Завершение текущего окна

DestroyWindow(hwnd);

break;

}

case WM\_DESTROY:

PostQuitMessage(0);

break;

default:

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

}

return 0;

}

ATOM MyRegisterClass(HINSTANCE hInstance) {

WNDCLASSEXW 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, MAKEINTRESOURCE(IDI\_OSISP1));

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

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

wcex.lpszMenuName = MAKEINTRESOURCEW(IDC\_OSISP1);

wcex.lpszClassName = szWindowClass;

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

return RegisterClassExW(&wcex);

}

BOOL InitInstance(HINSTANCE hInstance, int nCmdShow) {

hInst = hInstance; // Сохранить маркер экземпляра в глобальной переменной

HWND hWnd = CreateWindowW(szWindowClass, szTitle, WS\_OVERLAPPEDWINDOW,

CW\_USEDEFAULT, 0, CW\_USEDEFAULT, 0, nullptr, nullptr, hInstance, nullptr);

if (!hWnd) {

return FALSE;

}

ShowWindow(hWnd, nCmdShow);

UpdateWindow(hWnd);

return TRUE;

}

// Точка входа

int APIENTRY wWinMain(\_In\_ HINSTANCE hInstance, \_In\_opt\_ HINSTANCE hPrevInstance,

\_In\_ LPWSTR lpCmdLine, \_In\_ int nCmdShow) {

UNREFERENCED\_PARAMETER(hPrevInstance);

UNREFERENCED\_PARAMETER(lpCmdLine);

// Инициализация глобальных строк

LoadStringW(hInstance, IDS\_APP\_TITLE, szTitle, MAX\_LOADSTRING);

LoadStringW(hInstance, IDC\_OSISP1, szWindowClass, MAX\_LOADSTRING);

MyRegisterClass(hInstance);

// Выполнить инициализацию приложения:

if (!InitInstance(hInstance, nCmdShow)) {

return FALSE;

}

HACCEL hAccelTable = LoadAccelerators(hInstance, MAKEINTRESOURCE(IDC\_OSISP1));

MSG msg;

// Цикл основного сообщения:

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

if (!TranslateAccelerator(msg.hwnd, hAccelTable, &msg)) {

TranslateMessage(&msg);

DispatchMessage(&msg);

}

}

return (int)msg.wParam;

}