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 Сенько Н.С.

Проверил: ассистент кафедры информатики Гриценко Н.Ю.

Минск 2024

**СОДЕРЖАНИЕ**

[1 Формулировка задачи 3](#_Toc177575297)

[2 Краткие теоретические сведения 4](#_Toc177575298)

[3 Описание функций программы 5](#_Toc177575299)

[3.1 Инициализация процесса 5](#_Toc177575300)

[3.2 Перезапуск процесса 5](#_Toc177575301)

[4 Заключение 7](#_Toc177575302)

[Список использованных источников 8](#_Toc177575303)

[Приложение А (обязательное) 9](#_Toc177575304)

# **1 ФОРМУЛИРОВКА ЗАДАЧИ**

Цель лабораторной работы заключается в восстановлении, закреплении и развитии навыков программирования приложений для Windows. В процессе выполнения работы необходимо изучить концепции вычислительных процессов, потоков и нитей, а также их реализацию в Windows. Важно ознакомиться с основными этапами жизненного цикла процессов (потоков) и основами управления ими, включая создание, завершение, получение и изменение состояния. Также следует рассмотреть типичное применение многозадачности и многопоточности.

Задачей является разработка диспетчерского процесса, который должен возобновлять работу процесса с сохранением текущих функций после получения сообщения WM\_CLOSE. Игнорирование этого сообщения обычно приводит к аварийному завершению процесса или принудительному закрытию через Диспетчер задач. Решение состоит в том, чтобы процесс завершался корректно, но перед этим создавал свою копию, аналогичный процесс из того же исполняемого файла. Эта копия продолжает выполнение вместо родительского процесса. Однако возникает проблема сохранения работы с теми же обрабатываемыми данными. Возможное решение заключается в хранении текущих рабочих данных в структуре, называемой «прикладным контекстом», которая может быть записана на диск или сохранена в глобальной памяти родительского процесса, а затем загружена потомком.

2 КРАТКИЕ ТЕОРИТИЧЕСКИЕ СВЕДЕНИЯ

Приложение состоит из одного или нескольких процессов. Процесс, в простейших терминах, — это выполняющаяся программа. Один или несколько потоков выполняются в контексте процесса. 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.[1]

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. [2]

Пул потоков — это коллекция рабочих потоков, которые эффективно выполняют асинхронные обратные вызовы от имени приложения. Пул потоков в основном используется для уменьшения количества потоков приложения и обеспечения управления рабочими потоками.

Нить — это единица выполнения, которую приложение должно запланировать вручную. Нити выполняются в контексте потоков, которые планируют их. [3]

3 ОПИСАНИЕ ФУНКЦИЙ ПРОГРАММЫ

Согласно формулировке задачи, были спроектированы функции программы, которые обеспечивают выполнение ключевых операций. Эти функции включают:

– обработка сообщений WM\_CLOSE процесса;

– создание копии процесса;

– сохранение состояния процесса;

– восстановление состояния процесса;

– закрытие открытого программой процесса.

## **3.1** **Инициализация процесса**

Окно приложения представлено на рисунке 3.1, на нем виден счетчик попыток закрытия приложения.



Рисунок 3.1 – Экран приложения

## **3.2 Перезапуск процесса**

При закрытии программы вызовется новое окно, в котором будет увеличен счетчик на 1 (рисунок 3.2).



Рисунок 3.2 – Экран приложения после перезапуска

При последующем запуске и закрытии приложения счетчик будет увеличиваться на 1 на каждом закрытии.

ЗАКЛЮЧЕНИЕ

В заключение, данная лабораторная работа предоставила ценную возможность для глубокого изучения и практического применения принципов программирования приложений для Windows. Мы сосредоточились на ключевых аспектах, таких как управление процессами и потоками, что является основополагающим для создания эффективных и стабильных приложений. В ходе работы мы познакомились с жизненным циклом процессов, включая их создание, завершение, а также управление состоянием. Эти знания стали основой для проектирования функций программы, которые обеспечивают надежное функционирование в условиях многозадачности. Мы рассмотрели, как обработка системных сообщений, таких как WM\_CLOSE, может влиять на поведение приложения и как важно правильно реагировать на такие события, чтобы избежать потери данных и обеспечить плавный пользовательский опыт. Особое внимание было уделено разработке диспетчерского процесса. Мы реализовали функции, которые позволяют создавать копии процессов, что обеспечивает продолжение выполнения работы после завершения родительского процесса. Это решение не только иллюстрирует возможности многозадачности, но и подчеркивает важность сохранения состояния приложения. Хранение текущих рабочих данных в структуре, способной сохраняться на диск или в глобальной памяти, позволяет избежать потери информации и обеспечивает стабильность работы приложения. Проектирование и реализация таких функций, как сохранение и восстановление состояния, продемонстрировало, как можно эффективно управлять данными в многопоточной среде. Этот опыт углубил наши навыки программирования и подготовил нас к решению более сложных задач в области разработки программного обеспечения. Таким образом, лабораторная работа не только укрепила теоретические знания, но и предоставила практические навыки, необходимые для успешной работы с многозадачными и многопоточными приложениями. Полученные знания и навыки будут полезны в дальнейших исследованиях и разработках, открывая новые горизонты в области программирования.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

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

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

[3] Разработка приложений с помощью WinAPI. – Электронные данные. – Режим доступа: https://shorturl.at/BDJW8

ПРИЛОЖЕНИЕ А

(обязательное)

**Исходный код программы**

#include <iostream>

#include <ostream>

#include <string>

#include <windows.h>

struct ApplicationContext {

int someData; // Example field for application state

};

LPSTR \_lpstr;

ApplicationContext appContext;

void SaveContext();

void LoadContext();

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

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE, LPSTR lpstr, int nShowCmd) {

LoadContext();

// Initialize application context (example data)

\_lpstr = lpstr;

const char CLASS\_NAME[] = "SampleWindowClass";

// Register the window class

WNDCLASS wc = {};

wc.lpfnWndProc = WindowProc;

wc.hInstance = hInstance;

wc.lpszClassName = CLASS\_NAME;

RegisterClass(&wc);

// Create the window

HWND hwnd = CreateWindowA(

// 0, // Optional window styles

CLASS\_NAME, // Window class

"HAHA", // Window text

WS\_OVERLAPPEDWINDOW, // Window style

CW\_USEDEFAULT, CW\_USEDEFAULT, CW\_USEDEFAULT, CW\_USEDEFAULT,

NULL, // Parent window

NULL, // Menu

hInstance, // Instance handle

NULL // Additional application data

);

if (hwnd == NULL) {

return 0;

}

ShowWindow(GetConsoleWindow(), SW\_HIDE);

ShowWindow(hwnd, nShowCmd);

// ShowWindow(fuck, nShowCmd);

UpdateWindow(hwnd);

MSG msg = {};

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

TranslateMessage(&msg);

DispatchMessage(&msg);

}

return 0;

}

void SaveContext() {

DeleteFile("context.dat");

HANDLE hFile = CreateFile("context.dat", GENERIC\_WRITE, 0, NULL, CREATE\_ALWAYS, FILE\_ATTRIBUTE\_NORMAL, NULL);

if (hFile != INVALID\_HANDLE\_VALUE) {

DWORD bytesWritten;

WriteFile(hFile, &appContext, sizeof(appContext), &bytesWritten, NULL);

CloseHandle(hFile);

}

}

void LoadContext() {

HANDLE hFile = CreateFile("context.dat", GENERIC\_READ, 0, NULL, OPEN\_EXISTING, FILE\_ATTRIBUTE\_NORMAL, NULL);

if (hFile != INVALID\_HANDLE\_VALUE) {

DWORD bytesRead;

ReadFile(hFile, &appContext, sizeof(appContext), &bytesRead, NULL);

appContext.someData += 1;

CloseHandle(hFile);

}

}

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

switch (uMsg) {

case WM\_CLOSE: {

SaveContext(); // Save the application state

// Create a new process

STARTUPINFO si;

PROCESS\_INFORMATION pi;

ZeroMemory(&si, sizeof(si));

si.cb = sizeof(si);

ZeroMemory(&pi, sizeof(pi));

if (CreateProcess(NULL, GetCommandLineA(), NULL, NULL, FALSE, 0, NULL,

NULL, &si, &pi)) {

CloseHandle(pi.hProcess);

CloseHandle(pi.hThread);

}

DestroyWindow(hwnd); // Destroy the window

return 0;

}

case WM\_PAINT: {

PAINTSTRUCT ps;

HDC hdc = BeginPaint(hwnd, &ps);

TextOut(hdc, 10, 10, TEXT(std::to\_string(appContext.someData).c\_str()), strlen(std::to\_string(appContext.someData).c\_str()));

EndPaint(hwnd, &ps);

ReleaseDC(hwnd, hdc);

break;

}

case WM\_DESTROY: {

PostQuitMessage(0);

TerminateProcess(GetCurrentProcess(), 0);

return 0;

}

}

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

}