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 Я.Д. Божко

Проверил Н.Ю. Гриценко

Минск 2024

**СОДЕРЖАНИЕ**

[Введение 3](#_Toc8086)

[1 Теоретические сведения 4](#_Toc4539)

[1.1 Вычислительные процессы 4](#_Toc6052)

[1.2 Вычислительные потоки 5](#_Toc28098)

[1.3 Создание (порождение) процессов 6](#_Toc20861)

[1.4 Создание (порождение) потоков 7](#_Toc14583)

[1.5 Нити (Fiber) 7](#_Toc19810)

[2 Инструментальная языковая среда 9](#_Toc11757)

[3 Работа приложения 10](#_Toc26054)

[3.1 Структура приложения 10](#_Toc8111)

[3.2 Функции приложения 10](#_Toc13634)

[Список использованных источников 11](#_Toc13203)

[Приложение А 12](#_Toc8377)

**ВВЕДЕНИЕ**

Цель работы: возобновление работы процесса с сохранением (продолжением) текущих функций после завершения его сообщением WM\_CLOSE.

Проблема: игнорирование сообщения приводит, как правило, либо к объявлению его аварийным (автоматически), либо принудительному безусловному завершению посредством TaskManager.

Решение: процесс завершается штатным образом, но перед этим порождает свою копию (аналогичный процесс из того же исполняемого файла). Копия продолжает выполняться вместо родительского процесса.

Проблема: необходимость 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»).

WinAPI (Windows Application Programming Interface) является набором функций и процедур, предоставляемых операционной системой Windows для разработки приложений. В этой статье мы рассмотрим основные принципы работы с WinAPI.

WinAPI обеспечивает доступ к функциям операционной системы Windows, таких как создание окон, управление ресурсами, работа с файлами, реагирование на события и еще многое другое. Чтобы начать разработку приложений с использованием WinAPI, необходимо знать несколько основных понятий и функций [1].

**1 ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**

* 1. **Вычислительные процессы**

Вычислительный процесс представляет собой последовательность операций, выполняемых компьютером для достижения определённой задачи. Он является основным объектом операционной системы, обеспечивающей управление и координацию выполнения программ.

* + 1. Вычислительный процесс как системный объект

В операционных системах, таких как Windows, вычислительный процесс рассматривается как системный объект, который управляется ядром ОС. Процессы могут создавать, уничтожать и взаимодействовать друг с другом.

* + 1. Объект Process в Windows

В Windows каждый процесс представлен объектом Process. Он включает в себя все необходимые атрибуты для управления процессом:

1. Идентификатор процесса (PID): уникальный идентификатор, присваиваемый каждому процессу.
2. Идентификатор родительского процесса: указывает на процесс, создавший текущий процесс.
3. Права доступа: определяют, какие операции может выполнять процесс.
4. Состояние: текущее состояние процесса (например, работающий, приостановленный и т.д.).

**1.1.3** Атрибуты процесса

Атрибуты процесса могут включать:

1. Имя процесса: имя исполняемого файла.
2. Параметры командной строки: аргументы, переданные процессу при его запуске.
3. Используемая память: объём оперативной памяти, который занимает процесс.
4. Приоритет: определяет, как часто процесс получает доступ к CPU по сравнению с другими процессами.

**1.1.4** Образ процесса

Образ процесса — это основа, из которой создаётся процесс. Он включает в себя исполняемый код, данные и ресурсы, необходимые для выполнения процесса. Образ может загружаться в память и запускаться, создавая экземпляр процесса.

**1.1.5** Адресное пространство и пространство дескрипторов

Адресное пространство: это диапазон адресов памяти, доступный процессу. Каждому процессу выделяется отдельное адресное пространство, что позволяет предотвратить конфликт между процессами и обеспечивает безопасность.

Пространство дескрипторов: это набор дескрипторов, которые представляют ресурсы, используемые процессом, такие как файлы, потоки и синхронизационные объекты. Дескрипторы позволяют процессам управлять этими ресурсами.

**1.1.6** Состояния процесса

Процесс может находиться в различных состояниях:

1. Создание: процесс создаётся, но ещё не запущен.
2. Запуск: процесс выполняется.
3. Приостановка: процесс временно не выполняется, но может быть возобновлён.
4. Завершение: процесс завершил своё выполнение и освобождает все ресурсы.

Эти состояния помогают операционной системе управлять процессами и оптимизировать использование ресурсов.

* 1. **Вычислительные потоки**

Вычислительный поток — это наименьшая единица выполнения, которая может быть запущена и управляется операционной системой. Потоки позволяют программе выполнять несколько операций параллельно, улучшая производительность и отзывчивость.

**1.2.1** Вычислительный поток как системный объект

В операционных системах, таких как Windows, каждый поток является системным объектом, который может быть управляем отдельно от других потоков в рамках одного процесса. Это позволяет реализовывать многопоточность.

**1.2.2** Объект Thread в Windows

В Windows каждый поток представлен объектом Thread. Этот объект содержит всю информацию, необходимую для управления потоком:

1. Идентификатор потока (TID): уникальный идентификатор, присваиваемый каждому потоку.
2. Идентификатор родительского процесса: указывает на процесс, в котором выполняется поток.
3. Стек потока: область памяти, используемая для хранения данных, связанных с выполнением потока.
4. Приоритет: уровень приоритета потока, определяющий, насколько часто он будет получать доступ к CPU.

**1.2.3** Атрибуты потока

Атрибуты потока могут включать:

1. имя потока: имя, присвоенное потоку для идентификации;
2. приоритет: определяет, насколько важен поток по сравнению с другими.

Приоритеты могут быть:

1. низкий: поток будет выполняться реже;
2. нормальный: стандартный уровень выполнения;
3. высокий: поток будет получать больше времени CPU.

Состояние: текущее состояние потока (например, работающий, приостановленный и т.д.).

**1.2.4** Состояния потока

Поток может находиться в различных состояниях:

1. Создание: поток создаётся, но ещё не запущен.
2. Запуск: поток выполняется.
3. Приостановка: поток временно не выполняется и может быть возобновлён.
4. Завершение: поток завершил своё выполнение и освобождает ресурсы.

**1.2.5** Адресное пространство и пространство дескрипторов

Адресное пространство потока является разделяемым. Все потоки внутри одного процесса используют одно и то же адресное пространство. Это позволяет потокам легко обмениваться данными и ресурсами, но также требует осторожности для предотвращения конфликтов и ошибок.

Пространство дескрипторов является общим. Потоки могут использовать общее пространство дескрипторов, что позволяет им управлять общими ресурсами, такими как файлы и синхронизирующие объекты. Это упрощает взаимодействие между потоками, но также требует механизма синхронизации для предотвращения гонок данных.

Таким образом, вычислительные потоки играют важную роль в многопоточности, позволяя программам более эффективно использовать ресурсы системы.

**1.3 Создание (порождение) процессов**

**1.3.1** Иерархия процессов

В операционных системах процессы могут образовывать иерархическую структуру, где:

1. Родительский процесс (parent): процесс, который создает один или несколько дочерних процессов.
2. Дочерний процесс (child): процесс, созданный родительским процессом. Он наследует от родителя определённые атрибуты, такие как открытые файлы и права доступа.

**1.3.2** Завершение процессов

Процесс может завершаться по нескольким причинам:

1. Нормальное завершение: процесс завершает выполнение после выполнения своих задач.
2. Ошибка: процесс завершает выполнение из-за ошибки.
3. Принудительное завершение: процесс завершает выполнение по команде операционной системы.

**1.3.3** Код завершения

Код завершения — это целое число, возвращаемое процессом при его завершении, которое указывает на статус завершения (например, 0 для успешного завершения и ненулевое значение для ошибок).

**1.4 Создание (порождение) потоков**

**1.4.1** Главный поток

Главный поток — это поток, который создаётся при запуске процесса. Он является первым потоком и управляет выполнением других потоков.

**1.4.2** Диспетчер потоков

Диспетчер потоков отвечает за распределение времени процессора между активными потоками. Он использует алгоритмы планирования, чтобы определить, какой поток будет выполняться в данный момент.

**1.4.3** Приостановка и возобновление потоков

Потоки могут быть приостановлены и возобновлены в любой момент. Приостановка может происходить по запросу системы или приложения. Возобновление позволяет потоку продолжить выполнение с того места, где он был приостановлен.

**1.4.4** Завершение потоков

Поток может завершаться по аналогичным причинам, как и процессы:

1. нормальное завершение: поток завершает выполнение;
2. ошибка: поток завершает выполнение из-за ошибки.

**1.4.5** Код завершения потока

Код завершения также как и у процессов возвращает значение, указывающее на статус завершения.

**1.5 Нити (Fiber)**

**1.5.1** Особенности выполнения

Нити (Fiber) представляют собой легковесные потоки, которые требуют от приложения управления их переключением. Они работают в контексте потоков, но имеют собственные стек и состояние.

**1.5.2** Взаимосвязь нитей и потоков

Нити выполняются внутри потоков и могут быть переключены без участия операционной системы, что делает их более эффективными для задач, требующих частого переключения контекста.

Потоки управляются операционной системой, а нити — программным обеспечением на уровне приложения.

**1.5.3** Создание, завершение, переключение нитей

Создание нитей: выполняется с помощью API, который выделяет необходимый стек и устанавливает контекст исполнения.

Завершение нитей: происходит, когда выполнение нити завершается, и ресурсы освобождаются.

Переключение нитей: осуществляется вручную в коде приложения, что позволяет изменять активную нить без вмешательства ОС.

**1.5.4** Программный интерфейс (API)

Для управления процессами, потоками и нитями в Windows существуют различные API, включая:

1. создание процессов: CreateProcess();
2. завершение процессов: TerminateProcess();
3. создание потоков: CreateThread();
4. завершение потоков: ExitThread();
5. создание нитей: CreateFiber();
6. завершение нитей: DeleteFiber();
7. переключение нитей: SwitchToFiber().

Эти функции предоставляют разработчикам возможность эффективно управлять многопоточностью и процессами в приложениях[2].

**2 ИНСТРУМЕНТАЛЬНАЯ ЯЗЫКОВАЯ СРЕДА**

Для выполнения лабораторной работы был выбран язык разработки С++.

Язык программирования С++ представляет высокоуровневый компилируемый язык программирования общего назначения со статической типизацией, который подходит для создания самых различных приложений. На сегодняшний день С++ является одним из самых популярных и распространенных языков.

В качестве среды разраотки Visual Studio.

Win32 API (также называемый Windows API) — это исходная платформа для собственных Windows-приложений на языке C/C++, которым требуется прямой доступ к Windows и оборудованию. Он предоставляет интерфейс разработки первого класса без зависимости от управляемой среды выполнения, такой как .NET и WinRT (для приложений UWP для Windows 10). Благодаря этому API Win32 стает оптимальной платформой для приложений, которым требуется самый высокий уровень производительности и прямой доступ к системному оборудованию.

Файлы заголовков для API Windows позволяют создавать 32-разрядные и 64-разрядные приложения. Они включают объявления для версий API Юникода и ANSI.

Microsoft Visual C++ включает копии файлов заголовков Windows, которые были текущими в момент выпуска Visual C++. Таким образом, если вы устанавливаете обновленные файлы заголовков из пакета SDK, на компьютере может возникнуть несколько версий файлов заголовков Windows. Если вы не уверены, что используете последнюю версию файлов заголовков пакета SDK, при компиляции кода, использующего функции, представленные после выпуска Visual C++, будет получен следующий код ошибки: ошибка C2065: необъявленный идентификатор [3].

**3 РАБОТА ПРИЛОЖЕНИЯ**

* 1. **Структура приложения**

Приложение состоит из нескольких блоков, таких как:

1. структура для хранения передаваемых данных;
2. функция сохранения данных при закрытии;
3. функция загрузки данных в дочерний процесс;
4. создание нового процесса;
5. main, где происходит их вызов для корректной работы приложения.

Также код содержит несколько обработчиков событий для ошибок.

* 1. **Функции приложения**

Само приложение создает и открывает окно, которое при закрытии создает новое, передавая ему данные через структуру, при этом программа увеличивает счётчик на 1 при каждом закрытии окна.

Структура ApplicationContext:

1. Определяет структуру для хранения необходимых данных.
2. Добавьте в нее все необходимые поля для сохранения состояния вашего приложения.

Функции SaveContext и LoadContext:

1. SaveContext: Сохраняет структуру ApplicationContext в файл в двоичном формате.
2. LoadContext: Загружает структуру ApplicationContext из файла.

Обработчик сообщения WM\_CLOSE:

1. Сохраняет текущий контекст в файл.
2. Создает новый процесс, используя CreateProcessA для запуска того же исполняемого файла.
3. Передает аргументы командной строки (через GetCommandLineA) новому процессу.
4. Закрывает дескрипторы нового процесса.
5. Вызывает PostQuitMessage для завершения родительского процесса.

Блок main:

1. регистрирует класс окна;
2. создает окно;
3. отображает окно;
4. запускает цикл сообщений;
5. после завершения цикла сообщений (в потомке), загружает контекст из файла;
6. продолжает работу приложения с использованием загруженного контекста.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Разработка приложений с помощью WinAPI. – Электронный ресурс.– Режим доступа: https://habr.com/ru/articles/742396/.

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

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

**ПРИЛОЖЕНИЕ** **А**

**(обязательное)**

**Реализация программы на языке С++**

#include <windows.h>

#include <iostream>

#include <fstream>

#include <string>

struct ApplicationContext {

int counter = 0;

};

void saveContext(const ApplicationContext& context, const std::string& filename) {

std::ofstream file(filename, std::ios::binary);

if (file.is\_open()) {

file.write(reinterpret\_cast<const char\*>(&context), sizeof(context));

file.close();

}

}

ApplicationContext loadContext(const std::string& filename) {

ApplicationContext context;

std::ifstream file(filename, std::ios::binary);

if (file.is\_open()) {

file.read(reinterpret\_cast<char\*>(&context), sizeof(context));

file.close();

}

return context;

}

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

if (message == WM\_CLOSE) {

ApplicationContext context;

// Создание нового процесса

STARTUPINFOA si;

PROCESS\_INFORMATION pi;

ZeroMemory(&si, sizeof(si));

ZeroMemory(&pi, sizeof(pi));

si.cb = sizeof(si);

if (CreateProcessA(NULL,

(LPSTR)GetCommandLineA(),

NULL,

NULL,

FALSE,

0,

NULL,

NULL,

&si,

&pi)) { // Информация о процессе

CloseHandle(pi.hThread);

CloseHandle(pi.hProcess);

}

else {

MessageBoxA(hwnd, "Ошибка создания процесса", "Ошибка", MB\_OK);

}

PostQuitMessage(0);

return 0;

}

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

}

int main() {

WNDCLASSEX wcex;

ZeroMemory(&wcex, sizeof(wcex));

wcex.cbSize = sizeof(WNDCLASSEX);

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

wcex.lpfnWndProc = WindowProc;

wcex.hInstance = GetModuleHandle(NULL);

wcex.lpszClassName = L"MyWindowClass";

if (!RegisterClassEx(&wcex)) {

MessageBoxA(NULL, "Ошибка регистрации класса окна", "Ошибка", MB\_OK);

return 1;

}

HWND hwnd = CreateWindowExA(0, "MyWindowClass",

"Окно",

WS\_OVERLAPPEDWINDOW,

CW\_USEDEFAULT, CW\_USEDEFAULT,

300, 200,

NULL, NULL,

wcex.hInstance, NULL);

if (!hwnd) {

MessageBoxA(NULL, "Ошибка создания окна", "Ошибка", MB\_OK);

return 1;

}

ShowWindow(hwnd, SW\_SHOW);

UpdateWindow(hwnd);

MSG msg;

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

TranslateMessage(&msg);

DispatchMessage(&msg);

}

ApplicationContext context = loadContext("context.dat");

std::cout << "counter: " << context.counter << std::endl;

context.counter++;

saveContext(context, "context.dat");

return 0;

}