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 Снежко М.А.

Проверил: ассистент кафедры информатики Гриценко Н.Ю.

Минск 2024

**СОДЕРЖАНИЕ**

[1 Цель работы 3](#_Toc178181510)

[2 Краткие теоретические сведения 4](#_Toc178181511)

[3 Инструментальная языковая среда 5](#_Toc178181512)

[4 Описание функций программы 6](#_Toc178181513)

[Заключение 8](#_Toc178181514)

[Список использованных источников 9](#_Toc178181515)

[Приложение А (обязательное) 10](#_Toc178181516)

1 ЦЕЛЬ РАБОТЫ

Цель данной лабораторной работы заключается в восстановлении, закреплении и развитии навыков программирования приложений для Windows. В процессе выполнения работы необходимо изучить концепции вычислительных процессов, потоков и их реализацию в Windows, а также основные этапы жизненного цикла процессов (потоков) и базовые методы управления ими: создание, завершение, получение и изменение состояния. Также следует ознакомиться с типичным использованием многозадачности и многопоточности.

В качестве задачи необходимо разработать процесс-диспетчер, а именно процесс, выполняющий:

– выбор исполняемого файла и запуск процесса из него;

– хранение списка порожденных процессов;

– отображение состояния процессов (достаточно различать состояния «выполняется», «завершился»);

– возможность послать сообщение `WM\_CLOSE` выбранному процессу;

– отображение возникающих ошибок.

В качестве контролируемых процессов можно использовать произвольные подходящие программы либо специально написанный процесс: оконное приложение, способное наглядно показывать свое выполнение.

Программным продуктом лабораторной работы является консольное приложение, написанное на языке программирования C++ с использованием API операционной системы Windows

Для разработки программного продукта использовалась среда разработки Microsoft Visual Studio 2022, предоставляющая инструменты для написания, компиляции, отладки и тестирования приложений на C++. В Visual Studio был реализован весь цикл разработки — от написания исходного кода до отладки программы и её запуска на тестовых данных. Использование Visual Studio позволило автоматизировать процесс компиляции и управления проектом, а также отследить ошибки на этапе разработки.

В отчёте по лабораторной работе были полностью описаны ключевые понятия, касающиеся зомби-процессов, в том числе их определение, процесс их появления и основные причины, по которым они возникают в системе. В дополнение к этому, было подробно рассмотрено использование инструментальной языковой среды, включая особенности языка C++ и среды Visual Studio, которые использовались для разработки и тестирования программы.

2 КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Процессы — это основные единицы выполнения программ в операционных системах. Каждый процесс имеет свой уникальный идентификатор (PID), собственное пространство памяти и набор системных ресурсов (например, дескрипторы файлов, сетевые соединения). Основные этапы жизни процесса включают:

* Создание процесса: Процесс создается в результате вызова функции создания (например CreateProcess в Windows). При этом операционная система выделяет ресурсы для нового процесса и помещает его в таблицу процессов.
* Выполнение процесса: Процесс переходит в состояние выполнения, получая процессорное время. В это время он может выполнять операции с памятью, вводом-выводом и взаимодействовать с другими процессами.

– Завершение процесса: после завершения выполнения (нормального или аварийного) процесс должен передать родительскому процессу информацию о завершении. Если родительский процесс не получает эту информацию, то процесс может стать зомби.

Объект задания позволяет управлять группами процессов как единым целым. Объекты заданий — это именованные, защищенные, общие объекты, которые контролируют атрибуты связанных с ними процессов. Операции, выполняемые с объектом задания, оказывают влияние на все процессы, связанные с этим объектом.

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.[1]

Таким образом, управление процессами в операционных системах требует корректного завершения и очистки после выполнения, чтобы не оставлять неиспользуемых зомби-процессов, засоряющих системные ресурсы.

3 ИНСТРУМЕНТАЛЬНАЯ ЯЗЫКОВАЯ СРЕДА

Для разработки программы был выбран язык программирования C++. Это компилируемый, высокопроизводительный язык программирования общего назначения, поддерживающий как низкоуровневые, так и высокоуровневые парадигмы программирования. C++ используется для разработки системного программного обеспечения, приложений, драйверов устройств, а также других программ, требующих высокой эффективности и прямого доступа к системным ресурсам. [2]

В качестве интегрированной среды разработки (IDE) был выбран Microsoft Visual Studio. Это мощная среда разработки, которая предоставляет инструменты для написания, отладки, тестирования и оптимизации программ на C++. Операционной системой для разработки выступает Microsoft Windows 10, что обеспечивает совместимость программы с управлением процессами и API Windows. Вся разработка и тестирование программы производится на ноутбуке. Для документирования кода и разработки пользовательского интерфейса используется подход, ориентированный на пользователя, что позволит создать интуитивно понятное и удобное приложение. Предусмотрены регулярные рецензии кода, что поможет поддерживать стандарты качества и соответствие современным практикам разработки. В процессе разработки будут регулярно проводиться тестирования на различных этапах, включая модульное тестирование и интеграционное тестирование, чтобы обеспечить стабильность и надежность приложения. В результате проект будет не только функциональным, но и легко поддерживаемым в будущем, что позволит минимизировать время на обслуживание и обновления программы. [3]

4 ОПИСАНИЕ ФУНКЦИЙ ПРОГРАММЫ

Согласно формулировке задачи, были спроектированы следующие функции программы:

– выбор процесса для запуска (с обработкой ошибок при выборе некорректного файла);

– отображение статуса открытого процесса;

– закрытие открытого программой процесса.

Для запуска нового процесса необходимо выбрать пункт меню "Запустить процесс" (опция 1 в консольном интерфейсе). После выбора программа попросит указать путь к исполняемому файлу, который требуется запустить. Пример изображен на рисунке 4.1.

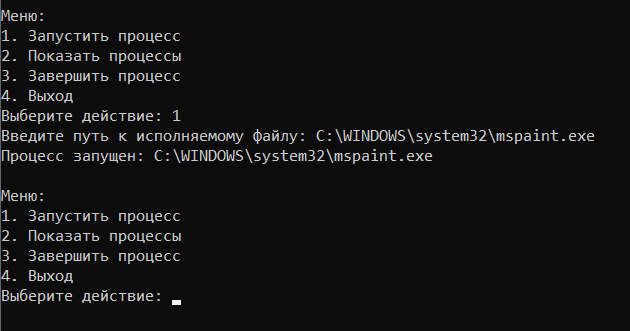


Рисунок 4.1 – Экран приложения и запуска процесса

Таким образом, программа попытается создать новый процесс, используя введенный путь к файлу. Если файл не является исполняемым, программа выведет сообщение об ошибке, если процесс успешно запущен, на экран будет выведено сообщение.

Чтобы отобразить список всех запущенных процессов, нужно выбрать пункт "Показать процессы" (опция 2 в консольном меню). На экране отобразятся запущенные через диспетчер процессы, включая их идентификатор и текущий статус.

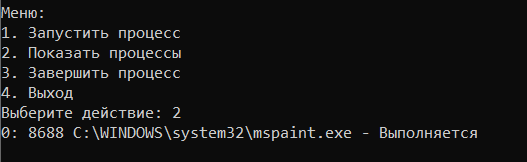


Рисунок 4.2 – Отображение информации о порожденном процессе

Чтобы завершить процесс, необходимо выбрать пункт меню "Завершить процесс" (опция 3) и ввести индекс процесса из списка. Пример программы изображен на рисунке 4.3.

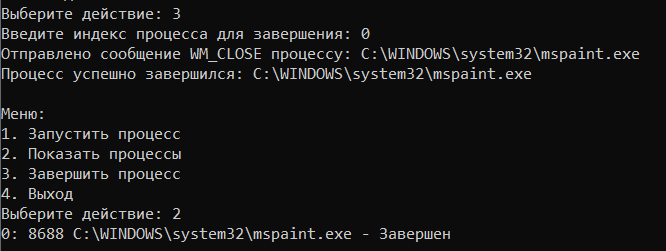


Рисунок 4.3 – Отображение информации о завершенном процессе

Таким образом, программа разработана для управления процессами, обеспечивая возможность их запуска п статуса и завершения. Она включает три ключевые функции: выбор исполняемого файла для запуска, отображение информации о запущенных процессах и их завершение по запросу пользователя.

ЗАКЛЮЧЕНИЕ

В ходе выполнения данной лабораторной работы по данной дисциплине была разработана программа, выступающая в роли процесс-диспетчера. Основной задачей было реализовать возможность выбора исполняемого файла, его запуска, а также отслеживания состояния порожденных процессов. Программа обеспечила хранение списка активных процессов, отображение их статуса (выполняется или завершился) и возможность отправки сообщения WM\_CLOSE для завершения выбранного процесса.

Реализация программы включала использование функций WinAPI для управления процессами, таких как CreateProcess для запуска новых процессов и WaitForSingleObject для проверки их состояния. Эти функции позволили организовать эффективное взаимодействие с операционной системой, обеспечивая надлежащее управление ресурсами.

Лабораторная работа позволила закрепить знания о работе с процессами, потоками и функциями WinAPI, а также понять, как важно управлять ресурсами операционной системы для обеспечения корректной работы приложений и избегания утечек памяти и некорректных состояний процессов, таких как зомби-процессы.

Программа успешно продемонстрировала управление процессами, включая их запуск, состояние и завершение. В зависимости от действий пользователя, приложение обеспечивало корректное завершение процессов и информировало о возникших ошибках. Это подчеркивает важность правильного управления процессами для поддержания стабильности и эффективности работы операционной системы.  
 Лабораторная работа позволила углубить знания о взаимодействии с процессами в Windows, а также укрепить навыки работы с WinAPI. Понимание механизмов управления процессами является ключевым для разработки надежных и эффективных приложений, способных корректно обрабатывать ресурсы операционной системы.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

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

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

# ПРИЛОЖЕНИЕ А

# (обязательное)

# Исходный код программного продукта

#include <windows.h>

#include <iostream>

#include <vector>

#include <string>

struct ProcessInfo

{

PROCESS\_INFORMATION procInfo;

std::string processPath;

bool isRunning;

};

std::vector<ProcessInfo> processes;

std::wstring ConvertToWideString(const std::string& str)

{

int size\_needed = MultiByteToWideChar(CP\_UTF8, 0, str.c\_str(), (int)str.size(), NULL, 0);

std::wstring wstr(size\_needed, 0);

MultiByteToWideChar(CP\_UTF8, 0, str.c\_str(), (int)str.size(), &wstr[0], size\_needed);

return wstr;

}

bool CreateNewProcess(const char\* processPath)

{

STARTUPINFO si = { sizeof(si) };

PROCESS\_INFORMATION pi;

std::wstring wideProcessPath = ConvertToWideString(processPath);

if (!CreateProcess(

wideProcessPath.c\_str(),

NULL, // Аргументы командной строки

NULL, // Атрибуты процесса

NULL, // Атрибуты потока

FALSE, // Наследование дескрипторов

0, // Флаги создания

NULL, // Переменные окружения

NULL, // Рабочая директория

&si, // Структура STARTUPINFO

&pi)) { // Структура PROCESS\_INFORMATION

std::cerr << "Ошибка создания процесса: " << GetLastError() << '\n';

return false;

}

CloseHandle(pi.hThread);

ProcessInfo newProcess = { pi, processPath, true };

processes.push\_back(newProcess);

std::cout << "Процесс запущен: " << processPath << '\n';

return true;

}

BOOL CALLBACK EnumWindowsProc(HWND hwnd, LPARAM lParam)

{

DWORD procId;

GetWindowThreadProcessId(hwnd, &procId);

if (procId == (DWORD)lParam)

{

PostMessage(hwnd, WM\_CLOSE, 0, 0);

return FALSE;

}

return TRUE;

}

void CloseProcessWindow(DWORD processId)

{

EnumWindows(EnumWindowsProc, (LPARAM)processId);

}

void TerminateProcessById(int processIndex)

{

if (processIndex < 0 || processIndex >= processes.size())

{

std::cerr << "Неверный индекс процесса." << '\n';

return;

}

ProcessInfo& process = processes[processIndex];

if (process.isRunning)

{

DWORD processId = GetProcessId(process.procInfo.hProcess);

CloseProcessWindow(processId);

std::cout << "Отправлено сообщение WM\_CLOSE процессу: " << process.processPath << '\n';

// Ожидаем завершение процесса в течение 5 секунд

DWORD waitResult = WaitForSingleObject(process.procInfo.hProcess, 5000);

if (waitResult == WAIT\_OBJECT\_0)

{

process.isRunning = false;

std::cout << "Процесс успешно завершился: " << process.processPath << '\n';

CloseHandle(process.procInfo.hProcess);

}

else if (waitResult == WAIT\_TIMEOUT)

{

std::cerr << "Процесс не завершился за отведенное время." << '\n';

}

else

{

std::cerr << "Ошибка при ожидании завершения процесса: " << GetLastError() << '\n';

}

}

else

{

std::cerr << "Процесс уже завершен." << '\n';

}

}

void ShowProcesses()

{

for (int i = 0; i < processes.size(); ++i)

{

std::cout << i << ": " << processes[i].procInfo.dwProcessId << " " << processes[i].processPath

<< " - " << (processes[i].isRunning ? "Выполняется" : "Завершен") << '\n';

}

}

int main()

{

setlocale(LC\_ALL, "ru");

std::string input;

while (true)

{

std::cout << "\nМеню:\n"

<< "1. Запустить процесс\n"

<< "2. Показать процессы\n"

<< "3. Завершить процесс\n"

<< "4. Выход\n"

<< "Выберите действие: ";

std::cin >> input;

if (input == "1")

{

std::string processPath;

std::cout << "Введите путь к исполняемому файлу: ";

std::cin >> processPath;

if (!CreateNewProcess(processPath.c\_str())) {

std::cerr << "Не удалось запустить процесс." << '\n';

}

}

else if (input == "2")

{

ShowProcesses();

}

else if (input == "3")

{

break;

}

else

{

std::cout << "Неверный выбор, попробуйте снова." << '\n';

}

}

return 0;

}