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 Таргонский Д.А.

Проверил:

ассистент кафедры информатики ГриценкоН.Ю.

Минск 2024

**СОДЕРЖАНИЕ**

[1 Цель работы 3](#_Toc177573255)

[2 Теоретические сведегтя 4](#_Toc177573256)

[2.1 Определение зомби процесса 4](#_Toc177573257)

[2.2 Причины появления зомби-процессов 4](#_Toc177573258)

[2.3 Влияние зомби-процессов на эффективность ЭВМ 5](#_Toc177573259)

[2.4 Вспомогательная теория процессов 5](#_Toc177573260)

[3 Инструментальная языковая среда 7](#_Toc177573261)

[4 Результат запуска программы 8](#_Toc177573262)

[4.1 Шаги запуска программы: 8](#_Toc177573263)

[4.2 Иллюстрация запуска и работы программы 8](#_Toc177573264)

[Заключение 11](#_Toc177573265)

[Список использованных источников 12](#_Toc177573266)

[Приложения А (обязательное) исходный код продукта 13](#_Toc177573267)

# 1 ЦЕЛЬ РАБОТЫ

В данной лабораторной работе требуется разработать программу для поиска и идентификации зомби-процессов в операционной системе Windows. Зомби-процессы представляют собой завершённые процессы, которые продолжают находиться в таблице процессов, занимая системные ресурсы до тех пор, пока родительский процесс не обработает информацию о завершении дочернего процесса.

Программным продуктом лабораторной работы является консольное приложение, написанное на языке программирования C++ с использованием API операционной системы Windows. В программе реализована функция запуска дочерних процессов и отслеживания их состояния. После завершения работы дочернего процесса программа проверяет, был ли процесс корректно завершён или он перешёл в состояние зомби. Для этого использовалась функция `WaitForSingleObject`, которая ожидала завершения процесса и позволяла определить его текущее состояние.

Для контроля существования незакрытых (зомби) процессов в системе дополнительно использовался Task Manager, который позволял визуально отслеживать процессы, висящие в системе после их завершения. Это необходимо для проверки правильности работы программы и выявления случаев, когда система не освободила ресурсы после завершения процесса.

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 ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

## 2.1 Определение зомби процесса

Зомби процесс — это процесс, который завершил своё выполнение, но его запись в таблице процессов (Process Table) операционной системы по-прежнему присутствует. Это означает, что сам процесс уже не выполняется и не использует ресурсы процессора или памяти, но его идентификатор процесса (PID) и другие сведения ещё хранятся в системе. Это происходит до тех пор, пока родительский процесс не вызовет функцию ожидания для получения статуса завершения дочернего процесса.

В системе, работающей на основе семейства Unix (Linux и macOS), зомби-процессы — это нормальное явление до тех пор, пока родительский процесс не выполнит нужное действие по сбору информации о завершении дочернего процесса. В Windows аналогом может считаться процесс, который не завершён корректно или имеет проблемы с освобождением ресурсов.

## 2.2 Причины появления зомби-процессов

Основной причиной появления зомби-процессов является то, что родительский процесс не завершил корректную процедуру обработки завершённого дочернего процесса. Это может произойти по следующим причинам:

* Некорректная работа родительского процесса: если родительский процесс не вызывает функцию для получения информации о завершении дочернего процесса (например GetExitCodeProcess в Windows), то дочерний процесс переходит в состояние зомби.
* Аварийное завершение родительского процесса: если родительский процесс завершился аварийно, не успев обработать завершение дочернего, этот дочерний процесс также может стать зомби.
* Умышленное игнорирование родительским процессом дочернего: в некоторых случаях разработчики могут забыть или намеренно не обрабатывать завершение дочернего процесса, что приводит к его зависанию в виде зомби.

Зомби-процесс сохраняется в таблице процессов до тех пор, пока родительский процесс не вызовет соответствующую функцию для его "сбора". После этого система освобождает все оставшиеся ресурсы.

## 2.3 Влияние зомби-процессов на эффективность ЭВМ

Зомби-процессы сами по себе не используют ресурсы процессора или оперативной памяти, но они продолжают занимать запись в таблице процессов. Хотя один зомби-процесс не оказывает значительного влияния на систему, большое количество зомби-процессов может стать проблемой по следующим причинам:

* Ограничение на количество процессов: В операционных системах есть ограничение на количество процессов, которые могут одновременно находиться в системе. Если в таблице процессов будет слишком много зомби-процессов, это может препятствовать созданию новых рабочих процессов.
* Засорение системных ресурсов: Зомби-процессы занимают записи в таблице процессов, что может привести к тому, что система будет тратить ресурсы на поддержку этих мёртвых процессов.
* Усложнение диагностики: Наличие зомби-процессов может усложнить диагностику системы, поскольку администраторы могут затрудниться в понимании текущего состояния системы из-за присутствия множества неактивных процессов.
* Потенциальные ошибки в программном обеспечении: Некорректная обработка завершения процессов может стать причиной программных ошибок или неправильного поведения программ, которые взаимодействуют с процессами.

## 2.4 Вспомогательная теория процессов

Процессы — это фундаментальные 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.

Процессы могут быть как родительскими, так и дочерними. Родительский процесс может создать один или несколько дочерних процессов для выполнения различных задач. После завершения дочерний процесс должен сообщить своему родителю о своем завершении, чтобы родитель мог освободить его ресурсы.

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 по обнаружению зомби-процессов 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.

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.

## 4 РЕЗУЛЬТАТ ЗАПУСКА ПРОГРАММЫ

Программа для поиска и идентификации зомби-процессов была успешно разработана и запущена в среде разработки Visual Studio 2022.

## 4.1 Шаги запуска программы:

* Открываем проект в Visual Studio 2022.
* Нажимаем Ctrl + F5 для запуска программы без отладки.

В процессе работы программы пользователю предлагается ввести выбор:

* Сохранить процесс в системе для дальнейшего мониторинга.
* Завершить выполнение программы.

После ввода соответствующей команды программа выполняет запуск дочернего процесса, отслеживает его состояние с использованием функции WaitForSingleObject и выводит результат на экран.

Результаты выполнения программы:

* Если дочерний процесс завершён корректно, программа сообщает, что процесс завершён успешно.
* Если дочерний процесс остаётся в системе в виде зомби-процесса (когда родительский процесс не обработал завершение дочернего), программа выводит сообщение о том, что был обнаружен зомби-процесс.

## 4.2 Иллюстрация запуска и работы программы

На рисунке 4.1 показана структура проекта ZombieProcessManager.sln.

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание

Рисунок 4.1 – Структура проекта ZombieProcessManager.sln

Главным (родительским) запускаемым проектом в решении является ZombieProcess Manager. Проект Children нужны для отработки появления зомби-процессов путём вызова родительским проектом дочернего с или без закрытия дескриптора дочернего проекта.

На рисунке 4.2 показано главное меню консольного приложения.

Изображение выглядит как текст, снимок экрана, Шрифт, дисплей

Автоматически созданное описание

Рисунок 4.2 – Меню консольного приложения

При сохранении дескриптора и завершении процесса вернётся статус завершения дочернего процесса отличный от нуля. Это означает, что дочерний процесс был сохранён с ошибкой, он неактивен в TaskManager вследствие особенности системы Windows, но ресурсы, выделенные на процесс, будут некоторое время недоступны. Следовательно, был создан зомби-процесс.

На рисунке 4.3 представлен результат выполнения программы с сохранённым дескриптором.

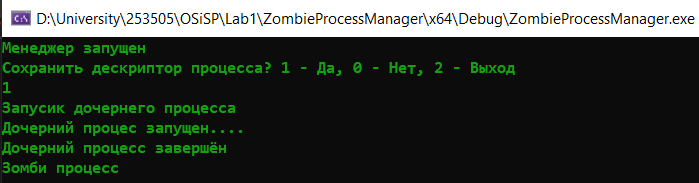


Рисунок 4.3 – Результат выполнения программы с сохранённым дескриптором

На рисунке 4.4 представлен код переменной result при сохранении дескриптора.

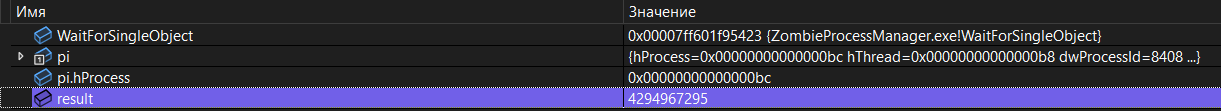


Рисунок 4.4 – Код переменной result при сохранении дескриптора

На рисунке 4.5 представлен результат выполнения программы с закрытым дескриптором.

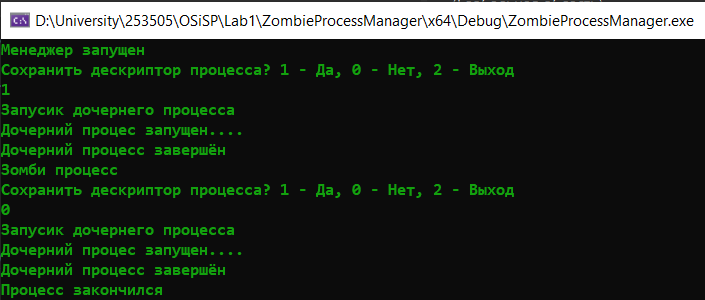


Рисунок 4.5 – Результат выполнения программы с закрытым дескриптором

На рисунке 4.6 представлен код переменной result с закрытым дескриптором.

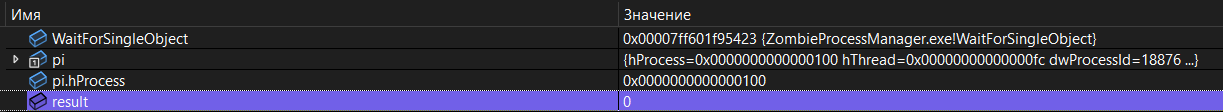


Рисунок 4.6 – Код переменной result c закрытым дескриптором.

# ЗАКЛЮЧЕНИЕ

В ходе выполнения данной лабораторной работы по дисциплине Операционные системы и системное программирование была разработана программа для управления процессами, в частности для создания и отслеживания так называемых зомби-процессов. Основная задача заключалась в создании дочернего процесса из родительского с возможностью управления дескрипторами и проверкой состояния дочернего процесса. Также было необходимо продемонстрировать возможность сохранения или закрытия дескрипторов и показать влияние этих действий на корректное завершение дочернего процесса.

Реализация программы включала в себя использование функций WinAPI для управления процессами, таких как CreateProcess для создания новых процессов, WaitForSingleObject для ожидания завершения процесса, и CloseHandle для закрытия дескрипторов процессов и потоков. Эти функции являются основными инструментами для работы с процессами в Windows, обеспечивая возможность прямого взаимодействия с операционной системой на уровне управления процессами.

Во время выполнения работы был рассмотрен механизм создания дочернего процесса и его завершения. Основное внимание было уделено изучению ситуации, когда дескрипторы процесса не закрываются после завершения дочернего процесса, что приводит к возникновению зомби-процесса.

Программа успешно продемонстрировала возможность создания и управления процессами в соответствии с заданными условиями. В зависимости от решения пользователя сохранять или не сохранять дескриптор дочернего процесса, программа либо корректно завершала дочерний процесс, либо оставляла его в состоянии зомби. Это позволило наглядно показать, как важно корректное закрытие дескрипторов процессов для освобождения ресурсов операционной системы.

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.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Build desktop Windows apps using the Win32 API [Электронный ресурс]. – 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 <tchar.h>

#include <iostream>

using namespace std;

STARTUPINFO si;

PROCESS\_INFORMATION pi;

void RunChildProcess(bool saveHandle)

{

ZeroMemory(&si, sizeof(si));

si.cb = sizeof(si);

ZeroMemory(&pi, sizeof(pi));

wstring CommandLine(L"Child.exe");

LPWSTR lpwCmdLine = &CommandLine[0];

// Start the child process.

if (!CreateProcess(NULL, // No module name (use command line)

lpwCmdLine, // Command line

NULL, // Process handle not inheritable

NULL, // Thread handle not inheritable

FALSE, // Set handle inheritance to FALSE

0, // No creation flags

NULL, // Use parent's environment block

NULL, // Use parent's starting directory

&si, // Pointer to STARTUPINFO structure

&pi) // Pointer to PROCESS\_INFORMATION structure

) {

std::cout << "Ошибка создания процесса\n";

return;

}

WaitForSingleObject(pi.hProcess, INFINITE);

if (saveHandle) {

CloseHandle(pi.hProcess);

CloseHandle(pi.hThread);

}

}

void CheckZombie()

{

auto result = WaitForSingleObject(pi.hProcess, 0);

if (result == WAIT\_OBJECT\_0) {

cout << "Процесс закончился\n";

}

else {

cout << "Зомби процесс\n";

}

}

int main() {

setlocale(LC\_ALL, "ru");

cout << "Менеджер запущен\n";

int savehandle;

while (true) {

cout << "Сохранить дескриптор процесса? 1 - Да, 0 - Нет, 2 - Выход\n";

cin >> savehandle;

if (savehandle == 2) {

return 0;

}

RunChildProcess(savehandle);

CheckZombie();

}

}