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 №6

на тему

**НЕКОТОРЫЕ СЛУЖЕБНЫЕ И ТЕХНОЛОГИЧЕСКИЕ ЗАДАЧИ**

Выполнил: студент гр.253504

Фроленко К.Ю.

Проверил: ассистент кафедры информатики Гриценко Н.Ю.

Минск 2024

СОДЕРЖАНИЕ

[1 Формулировка задачи 3](#_Toc181800310)

[2 Краткие теоритеческие сведения 4](#_Toc181800311)

[3 Описание функция программы 5](#_Toc181800312)

[3.1 Функция parseCommandLine 5](#_Toc181800313)

[3.2 Функция resolveHostname 5](#_Toc181800314)

[3.3 Функция traceroute 5](#_Toc181800315)

[3.4 Функция isValidTTL 6](#_Toc181800316)

[3.5 Функция printHelp 6](#_Toc181800317)

[3.6 Функция main 6](#_Toc181800318)

[4 Пример выполнения программы 7](#_Toc181800319)

[4.1 Запуск программы и процесс выполнения 7](#_Toc181800320)

[Вывод 8](#_Toc181800321)

[Список использованных источников 9](#_Toc181800322)

[Приложение А (обязательное) 10](#_Toc181800323)

1 ФОРМУЛИРОВКА ЗАДАЧИ

Целью лабораторной работы является разработка и анализ служебного приложения для конфигурирования, мониторинга и управления системой, что включает использование различных технологических аспектов, таких как библиотеки, сборки и взаимодействие с реестром *Windows*. Основная задача заключается в изучении подходов к хранению и использованию системных конфигураций, а также методов мониторинга и взаимодействия с операционной системой на уровне системных интерфейсов и *API*.

В ходе работы предполагается создание приложения, обеспечивающего сбор и отображение информации о системе. Состав собираемой информации может включать аппаратное обеспечение, операционную систему и другие параметры, при этом данные могут быть получены из реестра *Windows*, *WMI*, специализированных системных функций и других источников. Особое внимание будет уделено изучению работы с реестром Windows, его структурой, типами хранимых данных и функциями *API* для управления конфигурациями.

Также рассматриваются методы журналирования, задачи которых включают анализ и отображение информации, собранной для диагностики и мониторинга системы. В рамках лабораторной работы изучается использование библиотек и сборок *Windows*, включая динамические библиотеки *(DLL)* и смешанные сборки, что позволит повысить эффективность взаимодействия с операционной системой и расширить возможности приложения.

Результатом выполнения лабораторной работы станет углубленное понимание взаимодействия программ с операционной системой, улучшение навыков конфигурирования и мониторинга, а также практическое освоение использования библиотек и сборок в программировании под *Windows*.

2 КРАТКИЕ ТЕОРИТИЧЕСКИЕ СВЕДЕНИЯ

Системное программирование включает конфигурирование, мониторинг и управление операционной системой, что подразумевает глубокое понимание взаимодействия с её внутренними компонентами, такими как реестр, журналы и библиотеки. Реестр *Windows* представляет собой иерархическую базу данных, содержащую важные сведения о настройках и параметрах операционной системы, программного обеспечения и подключенного оборудования. Он структурирован на основе ключей и значений, что позволяет эффективно хранить и организовывать данные. Реестр играет ключевую роль в конфигурации системы, обеспечивая доступ к информации, необходимой для функционирования приложений и служб [1].

Журналирование в *Windows* позволяет фиксировать события, происходящие в системе, что помогает отслеживать изменения и выявлять проблемы. Существуют различные виды журналов, такие как системные журналы, журналы безопасности и приложений, которые хранят сведения о важных действиях, ошибках и предупреждениях [2]. Журналы предоставляют интерфейсы *API* для чтения, записи и анализа данных, что полезно для мониторинга и диагностики системы.

Технологические аспекты системного программирования включают использование библиотек, таких как динамические библиотеки *(DLL),* которые позволяют разделять и повторно использовать код между разными приложениями. Динамические библиотеки могут подключаться к программам как статически, так и динамически, предоставляя возможность загружать и использовать функции только при необходимости [3]. Это упрощает разработку, уменьшает размер исполняемых файлов и улучшает использование ресурсов системы.

Для обеспечения мониторинга и управления системой также важны механизмы перехвата и обработки сообщений, такие как *WinHook*. *WinHook* позволяет обрабатывать оконные сообщения в операционной системе *Windows*, что дает возможность контролировать взаимодействие между приложениями и системой на низком уровне. Это применяется, например, для создания пользовательских интерфейсов или для улучшения функциональности приложений [4].

В лабораторной работе также будет рассмотрен механизм просмотра информации о системе, что включает сбор данных о аппаратном обеспечении и программной среде системы. Для этой цели могут быть использованы специализированные функции *Windows Management Instrumentation (WMI)*, реестр, а также другие доступные источники информации. Это позволит понять структуру операционной системы и особенности её функционирования, а также даст практический опыт взаимодействия с низкоуровневыми системными интерфейсами.

3 ОПИСАНИЕ ФУНКЦИЙ ПРОГРАММЫ

В рамках разработки программы для демонстрации работы с файловой базой данных были реализованы следующие функции:

– Функция *GetOSVersion*;

– функция *GetCPUInfo*;

– функция *GetProcessorModel*;

– функция *GetMemoryInfo*;

– функция *GetSystemDirectoryPath*;

– функция *GetCurrentUser*;

– функция *GetComputerNameFromRegistry*;

– функция *main*.

3.1 Функция GetOSVersion

Функция *GetOSVersion* предназначена для определения версии операционной системы *Windows*. Она использует структуру *OSVERSIONINFOEX* и функцию *GetVersionEx* для получения информации о версии *ОС*, которая затем форматируется и возвращается в виде строки. В случае неудачи функция возвращает сообщение об ошибке.

3.2 Функция GetCPUInfo

Функция *GetCPUInfo* используется для получения информации о процессоре. Сначала с помощью функции *GetSystemInfo* определяется архитектура процессора (например, *x64*, *x86* или *ARM*), а также количество логических процессоров в системе. Эта информация затем форматируется и возвращается пользователю.

3.3 Функция GetProcessorModel

Функция *GetProcessorModel* выполняет чтение модели процессора из реестра *Windows*. Она открывает соответствующий ключ реестра, а затем с помощью функции *RegQueryValueExA* получает значение модели процессора. Если ключ не может быть открыт или значение не может быть прочитано, возвращается соответствующее сообщение об ошибке.

3.4 Функция GetMemoryInfo

Функция *GetMemoryInfo* собирает информацию о состоянии оперативной памяти в системе. Используя структуру *MEMORYSTATUSEX* и функцию *GlobalMemoryStatusEx*, функция возвращает информацию о доступной и общей физической памяти. В случае неудачи возвращается сообщение об ошибке.

3.5 Функция GetSystemDirectoryPath

Функция *GetSystemDirectoryPath* предназначена для получения пути к системному каталогу *Windows*. С использованием функции *GetSystemDirectoryA* она возвращает строку с полным путем к системной директории. В случае ошибки возвращается соответствующее сообщение.

3.6 Функция GetCurrentUser

Функция *GetCurrentUser* используется для получения имени текущего пользователя, вошедшего в систему. С помощью функции *GetUserNameA* функция возвращает имя пользователя. Если получить имя не удается, возвращается сообщение об ошибке.

3.7 Функция GetComputerNameFromRegistry

Функция *GetComputerNameFromRegistry* считывает имя компьютера из реестра *Windows*. Она открывает ключ реестра, в котором хранится имя компьютера, и с помощью функции *RegQueryValueExA* получает его значение. Если ключ не удается открыть или прочитать, возвращается сообщение об ошибке.

3.8 Функция main

Функция *main* является входной точкой программы и выполняет следующие основные действия:

1 Сбор системной информации с помощью вызова всех функций, описанных выше, таких как *GetOSVersion*, *GetCPUInfo*, *GetProcessorModel* и других.

2 Запись собранной информации в файл *system\_info.txt* для дальнейшего использования. Сначала открывается файл для записи, затем в него записывается вся системная информация.

3 Вывод собранной информации на экран и в файл, чтобы пользователь мог сразу видеть результаты работы программы.

Функция *main* также включает обработку ошибок при записи в файл и выводит сообщение об успешном завершении операции. Весь процесс, включая сбор данных и их запись, организован для удобства использования и предоставляет пользователю полную информацию о системе в виде текстового файла.

4 ПРИМЕР ВЫПОЛНЕНИЯ ПРОГРАММЫ

4.1 Запуск программы и процесс выполнения

В результате выполнения программы была успешно разработана утилита для просмотра системной информации, которая собирает и отображает данные о параметрах операционной системы, процессора, памяти и других ключевых компонентах. Утилита обеспечивает удобный способ получения этой информации как на экране, так и в виде текстового файла system\_info.txt. Данный файл содержит полную системную информацию, что упрощает анализ и диагностику системы, файл на рисунке 4.1.

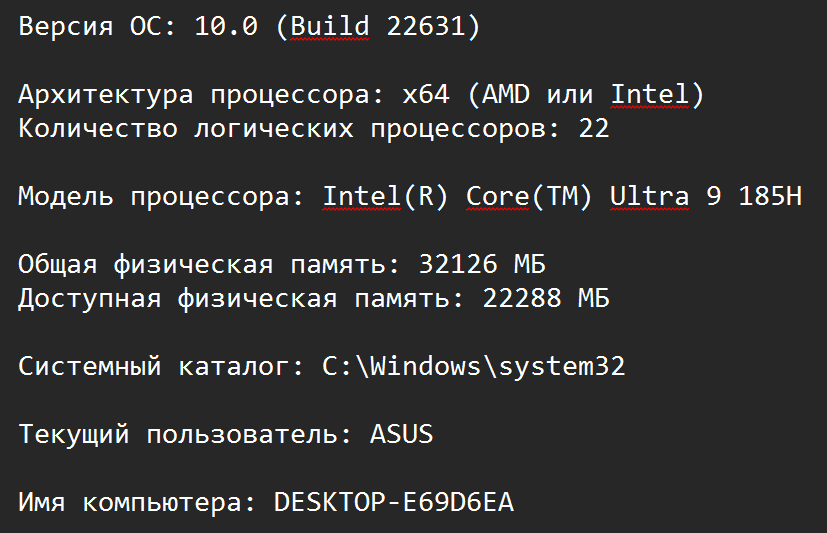
****

Рисунок 4.1 – Консоль с результатами выполнения трассировки маршрута

Программа обеспечивает обработку ошибок, таких как невозможность доступа к данным реестра или запись в файл, что делает её более надежной и удобной для пользователя. Все ошибки обрабатываются, а пользователю выводятся понятные сообщения, информирующие о возникших проблемах.

Функция *main* включает обработку ошибок при записи в файл и выводит сообщение об успешном завершении операции, что гарантирует пользователю уверенность в корректности выполнения программы. Весь процесс, включая сбор данных и их запись, организован для удобства использования и предоставляет пользователю полную информацию о системе в виде текстового файла, который может быть использован для анализа, диагностики или документирования конфигурации системы.

Таким образом, разработанная утилита является простым и эффективным инструментом для получения системной информации, что позволяет улучшить понимание текущих характеристик системы и упростить процесс мониторинга.

# ВЫВОД

В ходе выполнения лабораторной работы была успешно разработана утилита для конфигурирования, мониторинга и управления системой, основанная на использовании системных интерфейсов и *API Windows*. Программа позволила реализовать сбор и отображение системной информации, включая сведения о версии операционной системы, архитектуре процессора, модели процессора, оперативной памяти, системном каталоге, текущем пользователе и имени компьютера. Реализация данной функциональности помогла закрепить теоретические знания и получить практический опыт работы с такими системными компонентами, как реестр *Windows*, системные библиотеки и интерфейсы *API*.

Особое внимание в работе уделено обработке ошибок и созданию удобного интерфейса взаимодействия с пользователем, что значительно повышает надежность и применимость утилиты. Программа была разработана таким образом, чтобы пользователю была доступна вся необходимая информация как в консоли, так и в виде текстового файла, что делает её удобной в использовании и применимой в задачах мониторинга и диагностики системы.

Программа также позволила на практике исследовать особенности работы с динамическими библиотеками *(DLL)*, изучить принципы их использования и подключения, что способствует более глубокому пониманию технологических аспектов системного программирования под *Windows*. Работа с библиотеками, журналами и реестром предоставила возможность изучить различные подходы к хранению конфигураций и управлению системными ресурсами.

Таким образом, целью лабораторной работы, которая заключалась в разработке и анализе служебного приложения для управления системой, достигнута. Полученные знания и навыки в области взаимодействия с системными интерфейсами и компонентами *Windows* позволяют улучшить понимание архитектуры операционной системы и её функционирования. Разработанная утилита служит простым и эффективным инструментом для конфигурирования и мониторинга, что является важным этапом в процессе создания надежных системных решений.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Microsoft Docs: Introduction to the Windows Registry [Электронный ресурс]. – Режим доступа: <https://learn.microsoft.com/en-us/windows/win32/sysinfo/registry>. – Дата доступа: 06.11.2024.

[2] Microsoft Docs: Event Logging [Электронный ресурс]. – Режим доступа: <https://learn.microsoft.com/en-us/windows/win32/eventlog/event-logging>. – Дата доступа: 06.11.2024.

[3] Microsoft Docs: Dynamic-Link Libraries (DLLs) [Электронный ресурс]. – Режим доступа: <https://learn.microsoft.com/en-us/windows/win32/dlls/dynamic-link-libraries>. – Дата доступа: 06.11.2024.

[4] Microsoft Docs: Using Hooks [Электронный ресурс]. – Режим доступа: <https://learn.microsoft.com/en-us/windows/win32/winmsg/using-hooks>. – Дата доступа: 06.11.2024.

ПРИЛОЖЕНИЕ А

(обязательное)

Исходный код программы

#include <windows.h>

#include <Lmcons.h>

#include <bits/stdc++.h>

#define vs vector<string>

using namespace std;

string GetOSVersion()

{

OSVERSIONINFOEX osvi;

ZeroMemory(&osvi, sizeof(OSVERSIONINFOEX));

osvi.dwOSVersionInfoSize = sizeof(OSVERSIONINFOEX);

if (!GetVersionEx((OSVERSIONINFO \*)&osvi))

{

return "Не удалось определить версию ОС.";

}

string version = "Версия ОС: " + to\_string(osvi.dwMajorVersion) + "." +

to\_string(osvi.dwMinorVersion) + " (Build " +

to\_string(osvi.dwBuildNumber) + ")";

return version;

}

string GetCPUInfo()

{

SYSTEM\_INFO sysInfo;

GetSystemInfo(&sysInfo);

string cpuInfo = "Архитектура процессора: ";

switch (sysInfo.wProcessorArchitecture)

{

case PROCESSOR\_ARCHITECTURE\_AMD64:

cpuInfo += "x64";

break;

case PROCESSOR\_ARCHITECTURE\_INTEL:

cpuInfo += "x86";

break;

case PROCESSOR\_ARCHITECTURE\_ARM:

cpuInfo += "ARM";

break;

case PROCESSOR\_ARCHITECTURE\_ARM64:

cpuInfo += "ARM64";

break;

default:

cpuInfo += "Неизвестная архитектура";

break;

}

cpuInfo += "\nКоличество логических процессоров: " + to\_string(sysInfo.dwNumberOfProcessors);

return cpuInfo;

}

string GetProcessorModel()

{

HKEY hKey;

const char \*subKey = "HARDWARE\\DESCRIPTION\\System\\CentralProcessor\\0";

const char \*valueName = "ProcessorNameString";

if (RegOpenKeyExA(HKEY\_LOCAL\_MACHINE, subKey, 0, KEY\_READ, &hKey) != ERROR\_SUCCESS)

{

return "Не удалось открыть реестр для получения модели процессора.";

}

char processorName[256];

DWORD bufferSize = sizeof(processorName);

DWORD type = 0;

if (RegQueryValueExA(hKey, valueName, NULL, &type, (LPBYTE)processorName, &bufferSize) != ERROR\_SUCCESS or type != REG\_SZ)

{

RegCloseKey(hKey);

return "Не удалось получить модель процессора из реестра.";

}

RegCloseKey(hKey);

return string("Модель процессора: ") + processorName;

}

string GetMemoryInfo()

{

MEMORYSTATUSEX memInfo;

memInfo.dwLength = sizeof(MEMORYSTATUSEX);

if (!GlobalMemoryStatusEx(&memInfo))

{

return "Не удалось получить информацию об оперативной памяти.";

}

DWORDLONG totalPhys = memInfo.ullTotalPhys / (1024 \* 1024);

DWORDLONG availPhys = memInfo.ullAvailPhys / (1024 \* 1024);

string memoryInfo = "Общая физическая память: " + to\_string(totalPhys) + " МБ\n" +

"Доступная физическая память: " + to\_string(availPhys) + " МБ";

return memoryInfo;

}

string GetSystemDirectoryPath()

{

char systemDir[MAX\_PATH];

UINT len = GetSystemDirectoryA(systemDir, MAX\_PATH);

if (len == 0 or len > MAX\_PATH)

{

return "Не удалось получить системный каталог.";

}

return string("Системный каталог: ") + systemDir;

}

string GetCurrentUser()

{

char username[UNLEN + 1];

DWORD username\_len = UNLEN + 1;

if (GetUserNameA(username, &username\_len))

{

return string("Текущий пользователь: ") + username;

}

return "Не удалось получить имя текущего пользователя.";

}

string GetComputerNameFromRegistry()

{

HKEY hKey;

const char \*subKey = "SYSTEM\\CurrentControlSet\\Control\\ComputerName\\ComputerName";

const char \*valueName = "ComputerName";

if (RegOpenKeyExA(HKEY\_LOCAL\_MACHINE, subKey, 0, KEY\_READ, &hKey) != ERROR\_SUCCESS)

{

return "Не удалось открыть реестр для получения имени компьютера.";

}

char computerName[256];

DWORD bufferSize = sizeof(computerName);

DWORD type = 0;

if (RegQueryValueExA(hKey, valueName, NULL, &type, (LPBYTE)computerName, &bufferSize) != ERROR\_SUCCESS or type != REG\_SZ)

{

RegCloseKey(hKey);

return "Не удалось получить имя компьютера из реестра.";

}

RegCloseKey(hKey);

return string("Имя компьютера: ") + computerName;

}

int main()

{

vs systemInfo;

systemInfo.push\_back(GetOSVersion());

systemInfo.push\_back(GetCPUInfo());

systemInfo.push\_back(GetProcessorModel());

systemInfo.push\_back(GetMemoryInfo());

systemInfo.push\_back(GetSystemDirectoryPath());

systemInfo.push\_back(GetCurrentUser());

systemInfo.push\_back(GetComputerNameFromRegistry());

ofstream outFile("system\_info.txt");

if (!outFile.is\_open())

{

cerr << "Не удалось открыть файл для записи.\n";

return 1;

}

for (const auto &info : systemInfo)

{

cout << info << endl

<< endl;

outFile << info << endl

<< endl;

}

outFile.close();

cout << "Информация успешно записана в файл 'system\_info.txt'.\n";

return 0;

}