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.253505 Снежко М.А.

Проверил: ассистент кафедры информатики Гриценко Н.Ю.

Минск 2024

СОДЕРЖАНИЕ

[1 Формулировка задачи 3](#_Toc179742261)

[2 Краткие теоритические сведения 4](#_Toc179742262)

[3 Описание функций программы 5](#_Toc179742263)

[Заключение 6](#_Toc179742264)

[Список использованных источников 7](#_Toc179742265)

[Приложение А (обязательное) 8](#_Toc179742266)

# 1 ФОРМУЛИРОВКА ЗАДАЧИ

Целью лабораторной работы является изучение методов работы с системным реестром *Windows*, включая механизмы поиска, извлечения и обработки данных, а также разработка приложений, решающих служебные и технологические задачи. Основной акцент сделан на реализации алгоритмов поиска ключей, ветвей и значений в реестре с использованием заданных критериев. Работа направлена на изучение возможностей интерфейсов взаимодействия с реестром и оптимизации их применения в пользовательских программах.

В процессе выполнения работы необходимо разработать консольное приложение на языке *C++*, предназначенное для поиска в реестре ключей или значений по заданному шаблону. Программа должна поддерживать поиск ключей по имени с выводом иерархической структуры подключей и значений, а также поиск значений по имени и/или содержимому. Результаты выполнения поиска выводятся в консоль или записываются в файл в формате, максимально приближенном к структуре *reg*-файлов, чтобы в перспективе было возможно импортировать данные обратно в реестр.

Реализация приложения включает взаимодействие с *API* *Windows* для работы с реестром, использование функций для открытия, чтения и обхода ветвей реестра. Для повышения удобства и универсальности использования программа должна обеспечивать минимальный пользовательский интерфейс, который может быть представлен в виде командной строки или простейшего графического интерфейса. При запросе на поиск программа принимает параметры, включающие имя ключа или значение, а также дополнительные критерии поиска.

Результаты поиска структурированы таким образом, чтобы обеспечить удобный анализ и возможность экспорта. В случае вывода данных в файл применяется структура *reg*-файлов, где сохраняются имена ключей, значения и их типы. Обработка поиска предполагает анализ всех поддеревьев заданного ключа с целью нахождения совпадений, а также их последующую обработку для формирования финального отчета.

В рамках выполнения лабораторной работы предусмотрена реализация поиска как по точному совпадению имени ключа или значения, так и с использованием шаблонов, задаваемых пользователем. Например, поиск может учитывать подстроки или использовать регулярные выражения. Программа должна быть способна обрабатывать большие объемы данных и обеспечивать стабильность работы при высоких нагрузках.

Таким образом, данная лабораторная работа направлена на освоение принципов сетевого взаимодействия, изучение возможностей программного интерфейса сокетов и применение многопоточности для работы с сетевыми приложениями. Полученные знания могут быть использованы для создания более сложных инструментов, таких как анализаторы трафика, сетевые мониторы или утилиты управления сетью.

2 КРАТКИЕ ТЕОРИТИЧЕСКИЕ СВЕДЕНИЯ

Перехватчик – это механизм, с помощью которого приложение может перехватывать события, такие как сообщения, действия мыши и нажатия клавиш. Функция, перехватывающая событие определенного типа, называется процедурой перехватчика. Процедура перехватчика может действовать для каждого получаемого события, а затем изменять или отменять событие. Система поддерживает отдельную цепочку перехватчиков для каждого типа перехватчика. Цепочка перехватчиков – это список указателей на специальные определяемые приложением функции обратного вызова, называемые процедурами перехватчика. При возникновении сообщения, связанного с определенным типом перехватчика, система передает сообщение каждой процедуре перехватчика, на которую ссылается цепочка перехватчиков, одна за другой. Действие, выполняемое процедурой перехватчика, зависит от типа задействованного перехватчика [1].

Win32 API (Windows API) представляет собой набор функций и интерфейсов, предоставляемых операционной системой Windows для разработки приложений. Этот мощный набор инструментов обеспечивает доступ к различным функциональным возможностям Windows, включая создание и управление окнами, обработку сообщений, работу с файлами и реестром, а также многие другие операции. Некоторые формы IPC упрощают разделение труда между несколькими специализированными процессами. Другие формы IPC упрощают разделение труда между компьютерами в сети. Win32 API играет ключевую роль в разработке приложений для Windows и обеспечивает высокую степень контроля над поведением приложений [2].

*DLL* – это библиотека, содержащая код и данные, которые могут использоваться несколькими программами одновременно. Например, в операционных системах *Windows* *DLL*-библиотека *Comdlg32* выполняет общие функции, связанные с диалоговыми окнами. Каждая программа может использовать функции, содержащиеся в этой библиотеке *DLL*, для реализации диалогового окна Открыть. Это способствует повторному использованию кода и эффективному использованию памяти.

С помощью библиотеки *DLL* программу можно разделить на отдельные компоненты. Например, бухгалтерская программа может продаваться по модулям. Каждый модуль можно загрузить в основную программу во время выполнения, если он установлен. Так как модули являются отдельными, время загрузки программы ускоряется. Модуль загружается только тогда, когда запрашивается эта функциональность.

Кроме того, обновления проще применять к каждому модулю, не затрагивая другие части программы. Например, у вас может быть программа расчета заработной платы, и ставки налога меняются каждый год. Если эти изменения изолированы в библиотеке *DLL*, можно применить обновление, не требуя сборки или установки всей программы еще раз [3].

3 ОПИСАНИЕ ФУНКЦИЙ ПРОГРАММЫ

Разработанная программа представляет собой приложение, предназначенное для выполнения задач поиска и экспорта данных реестра в формате .*reg*. Она позволяет производить поиск ключей и значений реестра на основе заданных шаблонов. Пользователь задает шаблон поиска для имени ключа и/или значений, после чего программа осуществляет перебор ключей и значений в указанной ветке реестра. Найденные данные могут быть выведены на консоль либо сохранены в файл в формате, совместимом с импортом обратно в реестр.

Данная программа открывает ветку реестра *HKEY\_CURRENT\_USER* и выполняет рекурсивный обход ее структуры. Каждый найденный ключ проверяется на соответствие заданному шаблону, и, если совпадение найдено, ключ добавляется в результат. Значения в ключах также проходят проверку на соответствие шаблону, после чего выводятся их имена и содержимое. Для корректного формата данных в .*reg* файле программа преобразует содержимое значений в зависимости от их типа (строковые, *DWORD*, бинарные и так далее). Результаты выполнения программы записываются в файл *output.reg*, который содержит найденные ключи и значения в формате, совместимом с редактором реестра *Windows*. Это позволяет пользователю не только анализировать данные реестра, но и при необходимости импортировать их обратно. Результат работы программы представлен на рисунке 3.1.

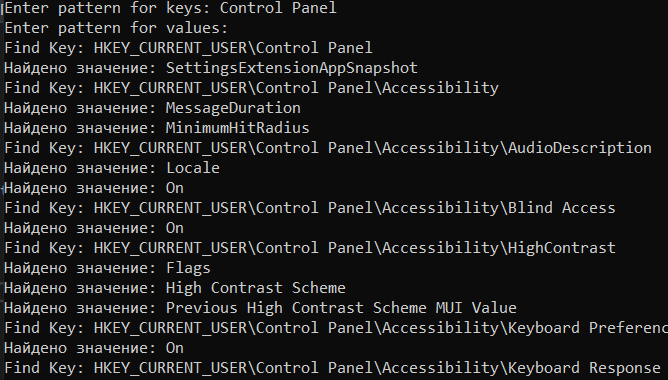


Рисунок 3.1 – Результат работы программы

Таким образом, данная программа демонстрирует использование технологий работы с реестром операционной системы и решения служебных задач. Она позволяет изучить базовые операции поиска ключей и значений реестра, а также экспорт найденных данных. Реализация предоставляет гибкость в настройке поиска по имени ключа, имени значения или его содержимому, что позволяет пользователю получать результаты в виде иерархии подключей или списка значений.

# ЗАКЛЮЧЕНИЕ

Таким образом, в рамках данной лабораторной работы была разработана программа для поиска и экспорта данных реестра операционной системы. Основная цель работы заключалась в изучении методов взаимодействия с реестром и реализации эффективного решения служебных и технологических задач, связанных с поиском и обработкой ключей и значений. Реализация программы позволила на практике изучить принципы поиска ключей и значений в реестре, а также способы экспорта данных в формат, пригодный для последующего импорта.

Разработанная программа представляет собой универсальный инструмент, который позволяет пользователю выполнять поиск по имени ключа, предоставляя иерархию подключей и значений в виде поддерева. Кроме того, программа поддерживает поиск по имени и значению, предоставляя результат в виде одного значения или списка значений, удовлетворяющих заданным критериям. Такая гибкость позволяет использовать приложение в различных сценариях, включая анализ конфигурации системы, диагностику ошибок или перенос настроек между компьютерами.

Результаты поиска выводятся на консоль или сохраняются в файл в формате .*reg*, что открывает перспективы для восстановления или переноса данных в реестр. Для корректной работы с реестром и минимизации ошибок взаимодействия реализована обработка ошибок и исключений.

В ходе выполнения лабораторной работы исследовались параметры, влияющие на производительность поиска, включая объем обрабатываемых данных и выбор формата сохранения результатов. Реализация программы продемонстрировала гибкость в настройке поиска и экспорта, а также обеспечила практический опыт в проектировании приложений для взаимодействия с реестром.

Таким образом, в рамках данной лабораторной работы была разработана программа для поиска и экспорта данных реестра операционной системы. Основная цель работы заключалась в изучении методов взаимодействия с реестром и реализации эффективного решения служебных и технологических задач, связанных с поиском и обработкой ключей и значений. Лабораторная работа позволила не только углубить знания о внутренней структуре реестра, но и закрепить навыки использования системных функций для работы с ключами, значениями и их иерархией. Реализация программы дала возможность изучить основные аспекты взаимодействия с системными ресурсами, а также принципы безопасного и корректного обращения с критически важными данными.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Microsoft "Обзор перехватчиков" – Электронный ресурс. Режим доступа: https://learn.microsoft.com/ru-ru/windows/win32/winmsg/about-hooks.

[2] Щупак Ю. Win32 API. Разработка приложений для Windows. – СПб: Питер, 2008. – 592 с.

[3] Microsoft "Библиотека динамической компоновки" – Электронный ресурс. Режим доступа: https://learn.microsoft.com/ru-ru/troubleshoot/windows-client/setup-upgrade-and-drivers/dynamic-link-library.

ПРИЛОЖЕНИЕ А

(обязательное)

**Исходный код программы**

#include <windows.h>

#include <iostream>

#include <fstream>

#include <string>

#include <iomanip>

#include <sstream>

// Функция для рекурсивного вывода дерева реестра и его значений в файл в формате .reg

void PrintRegistryTree(HKEY hKey, const std::wstring& subKey, std::ofstream& outputFile, const std::wstring& searchKeyPattern, const std::wstring& searchValuePattern);

std::wstring ConvertDataToRegFormat(DWORD type, const BYTE\* data, DWORD dataSize);

int main()

{

setlocale(LC\_ALL, "ru");

std::string searchKey, searchValue;

std::cout << "Enter pattern for keys: ";

std::getline(std::cin, searchKey);

std::cout << "Enter pattern for values: ";

std::getline(std::cin, searchValue);

HKEY hKey;

std::wstring str = std::wstring(searchKey.begin(), searchKey.end());

// Открываем ветку реестра HKEY\_CURRENT\_USER

if (RegOpenKeyEx(HKEY\_CURRENT\_USER, NULL, 0, KEY\_READ, &hKey) == ERROR\_SUCCESS)

{

std::ofstream outputFile("output.reg");

if (outputFile.is\_open()) {

outputFile << "Версия редактора реестра Windows 5.00\n\n";

// Рекурсивный поиск ключей и значений

PrintRegistryTree(hKey, str, outputFile, std::wstring(searchKey.begin(), searchKey.end()), std::wstring(searchValue.begin(), searchValue.end()));

outputFile.close();

std::cout << "Результат был записан в output.reg." << '\n';

}

else {

std::cerr << "Ошибка при открытии файла для записи" << '\n';

}

RegCloseKey(hKey);

}

else {

std::cerr << "Ошибка при открытии реестра" << '\n';

}

return 0;

}

// Рекурсивная функция для вывода дерева реестра и значений в файл в формате .reg

void PrintRegistryTree(HKEY hKey, const std::wstring& subKey, std::ofstream& outputFile, const std::wstring& searchKeyPattern, const std::wstring& searchValuePattern) {

HKEY hSubKey;

//std::cout << RegOpenKeyEx(hKey, subKey.c\_str(), 0, KEY\_READ, &hSubKey);

if (RegOpenKeyEx(hKey, subKey.c\_str(), 0, KEY\_READ, &hSubKey) == ERROR\_SUCCESS) {

wchar\_t keyName[256];

DWORD keyNameSize;

DWORD index = 0;

FILETIME lastWriteTime;

// Перебор значений текущего ключа

wchar\_t valueName[256];

DWORD valueNameSize;

BYTE data[16384]; // Буфер для данных

DWORD dataSize;

DWORD type;

DWORD valueIndex = 0;

// Если это не корневой ключ, выводим информацию о нём

if (!subKey.empty()) {

std::wstring fullKeyPath = L"HKEY\_CURRENT\_USER\\" + subKey;

if (searchKeyPattern.empty() || fullKeyPath.find(searchKeyPattern) != std::wstring::npos) {

outputFile << "[HKEY\_CURRENT\_USER\\" << std::string(subKey.begin(), subKey.end()) << "]\n";

std::cout << "Find Key: HKEY\_CURRENT\_USER\\" << std::string(subKey.begin(), subKey.end()) << std::endl;

// Перебор значений в ключе

while (true) {

valueNameSize = sizeof(valueName) / sizeof(valueName[0]);

dataSize = sizeof(data);

if (RegEnumValue(hSubKey, valueIndex++, valueName, &valueNameSize, NULL, &type, data, &dataSize) == ERROR\_SUCCESS) {

std::wstring valueStr(valueName);

// Если имя значения или его содержимое совпадает с шаблоном

if (searchValuePattern.empty() || valueStr.find(searchValuePattern) != std::wstring::npos)

{

std::cout << "Найдено значение: " << std::string(valueStr.begin(), valueStr.end()) << '\n';

// Преобразование данных значения в правильный формат для reg-файла

std::wstring dataString = ConvertDataToRegFormat(type, data, dataSize);

// Запись значения в файл

outputFile << "\"" << std::string(valueStr.begin(), valueStr.end()) << "\"=" << std::string(dataString.begin(), dataString.end()) << "\n";

}

}

else

{

break;

}

}

outputFile << "\n";

}

}

while (true)

{

keyNameSize = sizeof(keyName) / sizeof(keyName[0]);

if (RegEnumKeyEx(hSubKey, index++, keyName, &keyNameSize, NULL, NULL, NULL, &lastWriteTime) == ERROR\_SUCCESS)

{

std::wstring fullSubKey = subKey.empty() ? keyName : subKey + L"\\" + keyName;

PrintRegistryTree(hKey, fullSubKey, outputFile, searchKeyPattern, searchValuePattern);

}

else

{

break;

}

}

RegCloseKey(hSubKey);

}

}

// Функция для преобразования данных значения в формат, подходящий для reg-файлов

std::wstring ConvertDataToRegFormat(DWORD type, const BYTE\* data, DWORD dataSize) {

std::wstringstream ss;

switch (type)

{

case REG\_SZ: // Строковое значение

ss << L"\"" << std::wstring((wchar\_t\*)data) << L"\"";

break;

case REG\_DWORD: // DWORD значение

ss << L"dword:" << std::hex << std::setw(8) << std::setfill(L'0') << \*(DWORD\*)data;

break;

case REG\_BINARY: // Бинарное значение

ss << L"hex:";

for (DWORD i = 0; i < dataSize; i++)

{

if (i > 0) ss << L",";

ss << std::hex << std::setw(2) << std::setfill(L'0') << (int)data[i];

}

break;

default:

ss << L"неизвестно";

break;

}

return ss.str();

}