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 Божко Я.Д

Проверил: ассистент кафедры информатики

Гриценко Н.Ю

Минск 2024

**СОДЕРЖАНИЕ**

[Введение 3](#_Toc6512)

[1 Теоретические сведения 4](#_Toc10206)

[1.1 Взаимодействие процессов 4](#_Toc16048)

[1.2 Реестры 6](#_Toc2035)

[1.3 Журналирование 9](#_Toc6435)

[2 Инструментальная языковая среда 12](#_Toc3814)

[3 Работа приложения 13](#_Toc400)

[3.1 Структура приложения 13](#_Toc26015)

[3.2 Функции приложения 13](#_Toc22336)

[3.3 Анализ данных 14](#_Toc29478)

[Список использованных источников 15](#_Toc11910)

[Приложение А (обязательное) 16](#_Toc27357)

**ВВЕДЕНИЕ**

В современном программировании важнейшую роль играют системы управления конфигурациями, механизмы журналирования, а также эффективное взаимодействие с операционной системой через различные интерфейсы. Эти компоненты являются основой разработки приложений, особенно в среде Windows, где операционная система предоставляет широкий спектр инструментов для работы с реестром, журналами событий и динамическими библиотеками.

Одной из ключевых тем в области системного программирования является использование реестра Windows. Реестр представляет собой центральную базу данных, хранящую настройки операционной системы и приложений. Понимание структуры реестра и методов взаимодействия с ним позволяет разработчикам эффективно управлять конфигурациями, а также обеспечивать высокую гибкость и адаптивность программного обеспечения. Кроме того, журналирование в Windows помогает отслеживать события в системе, фиксировать ошибки и обеспечивать безопасность приложений, что является важной составляющей любой сложной системы.

Другим важным аспектом является работа с динамическими библиотеками (DLL), которые позволяют приложениям загружать и использовать код во время выполнения, что значительно повышает гибкость разработки. Применение смешанных сборок (mixed assemblies) в сочетании с .NET и нативным кодом расширяет возможности взаимодействия с операционной системой и облегчает интеграцию различных технологий в одном проекте.

В данной работе рассматриваются подходы к хранению и использованию конфигураций, особенности работы с реестром и журналами событий Windows, а также вопросы, связанные с динамическими и смешанными библиотеками. Эти темы являются неотъемлемой частью разработки программного обеспечения в экосистеме Windows и оказывают значительное влияние на производительность, безопасность и удобство использования приложений.

Целью данной работы является изучение и анализ теоретических аспектов работы с конфигурациями, реестром, журналами и библиотеками в операционной системе Windows, а также рассмотрение практических методов их применения в реальных проектах.

**1 ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**

**1.1 Взаимодействие процессов**

Хранение и использование конфигурационных данных — одна из важнейших задач при разработке программного обеспечения. Конфигурация программы определяет ее поведение, параметры работы и взаимодействие с пользователем и системой. Важно, чтобы разработчик учитывал как хранить конфигурацию, так и как обеспечить ее использование, чтобы система оставалась гибкой и удобной для настройки.

В мире программирования существует несколько подходов к хранению конфигураций, которые можно разделить на несколько категорий: файловые методы, использование баз данных, а также специальные механизмы, такие как реестр в операционной системе Windows. Каждый из этих методов имеет свои преимущества и ограничения, что делает выбор подхода зависимым от специфики приложения.

**1.1.1** Хранение конфигурации в файлах

Один из самых простых и популярных способов хранения конфигурационных данных — использование файлов. Программисты часто используют текстовые файлы различных форматов, таких как .ini, .json, .xml или .yaml, для хранения параметров конфигурации.

INI-файлы — один из самых старых форматов, использовавшийся в первых версиях Windows. Это простые текстовые файлы, в которых конфигурация разделена на секции, а параметры хранятся в виде пар «ключ-значение».

JSON и XML — более современные форматы, которые позволяют хранить более сложные структуры данных, такие как массивы и вложенные объекты. Форматы JSON и XML широко используются в веб-разработке и других областях, так как они хорошо поддерживаются различными языками программирования и являются универсальными.

Преимущества этого подхода:

1. простота реализации;
2. легкость в понимании и редактировании конфигурации;
3. конфигурационные файлы можно легко переносить и изменять.

Недостатки:

1. не всегда безопасно хранить конфигурацию в открытом текстовом виде (например, хранение паролей или других чувствительных данных);
2. проблемы с управлением и версионированием конфигураций в крупных проектах.

**1.1.2** Хранение конфигурации в базах данных

Иногда для более сложных приложений, особенно в корпоративных системах, может возникнуть необходимость в хранении конфигураций в базах данных. Это позволяет централизовать управление параметрами и предоставлять более гибкие возможности для доступа и обновления конфигураций.

Примером может быть использование реляционных баз данных (MySQL, PostgreSQL, Microsoft SQL Server) или NoSQL баз данных (MongoDB). В таких системах конфигурация может быть представлена в виде таблиц или документов, а доступ к данным происходит через SQL-запросы или соответствующие API.

Преимущества этого подхода:

1. централизованное хранение данных, что особенно полезно для распределенных систем;
2. возможность использования транзакций для обеспечения целостности данных;
3. простота масштабирования и обновлений.

Недостатки:

1. более сложная настройка и поддержка;
2. необходимость в дополнительных ресурсах для работы с базой данных.

**1.1.3** Использование реестра Windows

В операционных системах Windows существует специальная система для хранения конфигурационных данных — реестр Windows. Реестр — это иерархическая база данных, в которой хранятся параметры операционной системы, настроек программ и пользовательских данных.

Реестр предоставляет возможность хранить различные типы данных (строки, числа, двоичные данные и т. д.) и имеет удобный механизм для быстрого доступа и изменения этих данных. Программы могут хранить свои конфигурационные данные в реестре в различных разделах, таких как HKEY\_LOCAL\_MACHINE (для системных настроек) и HKEY\_CURRENT\_USER (для настроек конкретного пользователя).

Преимущества реестра:

1. быстрый доступ к данным;
2. центральное хранилище для настроек операционной системы и программ;
3. легкость в интеграции с другими функциями windows (например, автозапуск приложений).

Недостатки:

1. не является кроссплатформенным решением (реестр доступен только в windows);
2. потенциальные проблемы с безопасностью (неправильная настройка реестра может привести к сбоям в системе);
3. сложности с резервным копированием и восстановлением данных.

**1.1.4** Хранение конфигурации в облаке

С развитием облачных технологий многие современные приложения начали использовать облачные сервисы для хранения своих конфигураций. Это может быть удобно для мобильных приложений, веб-сервисов или распределенных систем, где требуется доступ к конфигурационным данным с разных устройств или серверов.

Примером может быть использование облачных баз данных, таких как Amazon DynamoDB, Google Firebase или Microsoft Azure Storage.

Преимущества:

1. конфигурации доступны с любых устройств, подключенных к интернету;
2. удобная синхронизация настроек между различными устройствами;
3. высокая масштабируемость и надежность.

Недостатки:

1. зависимость от интернет-соединения;
2. проблемы с безопасностью и доступом к данным.

**1.1.5** Использование переменных среды

Переменные среды — это еще один способ хранения конфигурационных данных. Они используются для хранения параметров, которые могут изменяться в зависимости от окружения, например, путь к директориям, настройки для системы или для приложений.

Преимущества:

1. быстрое и удобное решение для системных и пользовательских настроек;
2. простота использования в сценариях автоматизации.

Недостатки:

1. ограничение на длину значений;
2. не подходит для хранения сложных или больших структур данных. [1]

**1.2 Реестры**

**1.2.1** Назначение реестра Windows

Реестр Windows — это централизованная иерархическая база данных, используемая операционной системой и приложениями для хранения различных настроек и параметров. Он выполняет функцию конфигурационного хранилища и является неотъемлемой частью архитектуры Windows. Реестр служит для хранения информации о системных настройках, установленных приложениях, параметрах пользовательских сессий и других критически важных данных.

В реестре содержатся такие данные, как:

1. настройки операционной системы (например, параметры автозагрузки, настройки безопасности);
2. конфигурации установленных приложений и драйверов;
3. информация о пользователях и группах;
4. системные переменные и их значения.

**1.2.2** Структура реестра Windows

Реестр Windows организован в виде иерархической структуры, напоминающей файловую систему, где «ключи» играют роль каталогов, а «значения» — файлов. Основной элемент реестра — это ключи и значения. Ключи могут содержать несколько значений, а значения могут иметь различные типы данных.

Основные разделы (root keys) реестра:

1. HKEY\_CLASSES\_ROOT (HKCR) — хранит информацию о зарегистрированных типах файлов, ассоциациях с программами и COM-объектах.
2. HKEY\_CURRENT\_USER (HKCU) — содержит настройки и параметры, специфичные для текущего пользователя, такие как настройки рабочего стола, предпочтения приложений.
3. HKEY\_LOCAL\_MACHINE (HKLM) — содержит информацию о компьютере и его аппаратных компонентах, а также параметры, общие для всех пользователей на данном устройстве.
4. HKEY\_USERS (HKU) — хранит данные о всех пользователях, зарегистрированных на компьютере, включая их настройки.
5. HKEY\_CURRENT\_CONFIG (HKCC) — содержит информацию о текущей аппаратной конфигурации устройства.

Структура реестра:

Реестр можно представить как дерево, где корневые разделы — это главные ветви, а подкаталоги (или ключи) могут содержать вложенные ключи и значения.

Каждый ключ может содержать несколько значений — пар «имя-значение», где имя соответствует ключу, а значение — конкретному параметру, тип которого может быть различным.

**1.2.3** Типы и характер хранимых данных

В реестре могут храниться разные типы данных, каждый из которых имеет свое назначение и структуру. Наиболее часто используемые типы данных в реестре:

1. REG\_SZ — строка (строковое значение), например, путь к файлу или имя программы.
2. REG\_DWORD — целое число (32-битное), часто используется для хранения флагов или числовых значений.
3. REG\_BINARY — бинарные данные, могут быть использованы для хранения произвольных данных, например, в виде маски или кодированных значений.
4. REG\_MULTI\_SZ — множество строк, которые представляют собой массив строк.
5. REG\_EXPAND\_SZ — строка, в которой могут использоваться переменные среды, такие как %SystemRoot%.

Каждый тип данных имеет свои особенности хранения и применения, и использование правильного типа данных важно для корректной работы системы.

**1.2.4** Интерфейс и группы функций API для работы с реестром

Windows предоставляет разработчикам API-функции для работы с реестром. Эти функции входят в состав Windows API и обеспечивают доступ к данным реестра, их модификацию и управление ключами и значениями. Основные функции, используемые для работы с реестром, следующие:

1. RegOpenKeyEx — открывает существующий ключ реестра для чтения или записи.
2. RegCloseKey — закрывает ключ, который был открыт функцией RegOpenKeyEx.
3. RegQueryValueEx — извлекает значение, связанное с ключом.
4. RegSetValueEx — задает значение для указанного ключа.
5. RegDeleteKey — удаляет ключ реестра.
6. RegDeleteValue — удаляет значение для ключа реестра.
7. RegCreateKeyEx — создает новый ключ реестра.
8. RegEnumKeyEx — перечисляет все ключи, содержащиеся в указанном разделе реестра.
9. RegEnumValue — перечисляет все значения, связанные с указанным ключом.

Для работы с реестром обычно нужно использовать функции открытия, чтения, изменения и удаления ключей и значений.

**1.2.5** Порядок использования реестра

Реестр используется для хранения и получения настроек программы или операционной системы. Основной порядок работы с реестром следующий:

1. Открытие ключа: Сначала необходимо открыть ключ реестра, используя функцию RegOpenKeyEx, чтобы иметь возможность читать или изменять его содержимое.
2. Чтение значений: Для получения значений, связанных с ключами, используется функция RegQueryValueEx. Это может быть строка, число или бинарные данные.
3. Изменение значений: Для изменения данных в реестре используется функция RegSetValueEx, которая позволяет обновить существующие значения.
4. Удаление значений или ключей: Если необходимо удалить ненужные ключи или значения, используются функции RegDeleteKey или RegDeleteValue.
5. Закрытие ключа: После работы с реестром важно закрыть ключ с помощью функции RegCloseKey.

**1.2.6** Типичные задачи использования реестра

Хранение настроек пользователя и системы: Программы могут сохранять настройки, такие как предпочтения пользователя, пути к файлам или параметры сетевых соединений.

Управление автозагрузкой приложений: В разделе HKEY\_CURRENT\_USER\Software\Microsoft\Windows\CurrentVersion\Run хранятся программы, которые должны автоматически запускаться при старте Windows.

Конфигурация драйверов и системных компонентов: Системные компоненты, включая драйверы устройств, могут хранить свои настройки в реестре.

Реализация лицензирования и защиты от копирования: Некоторые приложения используют реестр для сохранения ключей активации или информации о лицензии.

Мониторинг и отладка: Для диагностики проблем с системой или приложением можно использовать реестр для отслеживания изменений, параметров и поведения программ. [2]

**1.3 Журналирование**

**1.3.1** Назначение журналирования

Журналирование — это процесс записи и хранения информации о событиях и операциях, происходящих в системе, приложениях или на уровне базы данных. В контексте операционных систем, таких как Windows, журналирование используется для мониторинга действий пользователей, системных процессов, ошибок, а также для обеспечения целостности данных в случае сбоев. Основной задачей журналирования является возможность отслеживания событий в системе и восстановления после ошибок.

Журналы могут включать различные типы информации, такие как:

1. Системные события (запуск и завершение процессов, изменения в настройках системы).
2. Ошибки и предупреждения, возникающие при работе приложений.
3. Информация о безопасности (например, вход в систему, изменения прав доступа).
4. История операций с данными (например, операции с файлами или записью в базу данных).

Журналирование играет важную роль в системах с высоким уровнем безопасности, а также в приложениях, где требуется гарантированное восстановление данных после сбоев.

**1.3.2** Решаемые задачи журналирования

Журналирование решает несколько важных задач, которые обеспечивают стабильность и безопасность операционной системы и приложений:

1. Мониторинг и диагностика: Журналирование позволяет отслеживать ход выполнения операций в системе, выявлять потенциальные проблемы и делать выводы о стабильности работы приложений. Например, записи о сбоях могут быть использованы для диагностики и устранения причин ошибок.
2. Аудит и безопасность: Журналы используются для отслеживания действий пользователей, что необходимо для обеспечения безопасности и предотвращения несанкционированного доступа. Это важно в системах с несколькими пользователями и высокими требованиями к безопасности.
3. Восстановление после сбоев: В случае отказа системы или приложения журналирование позволяет восстанавливать данные, фиксируя все операции, которые были выполнены до сбоя. Для этого часто используются методы "журналируемых транзакций" в базах данных.
4. Соответствие стандартам и требованиям: Во многих отраслях (например, в банковской сфере или медицинских приложениях) существует требование для ведения журналов с целью соответствия стандартам безопасности и защиты данных. Журналирование помогает убедиться, что действия пользователей и системы записываются в соответствии с регламентами.

**1.3.3** Основные используемые подходы журналирования

Существует несколько подходов к журналированию, каждый из которых выбирается в зависимости от специфики задачи и требуемого уровня надежности.

Журналирование событий (Event Logging): Это самый распространенный метод, при котором все системные и пользовательские события записываются в специальные журналы. Например, в Windows используется журнал событий (Event Log), в который записываются различные типы сообщений, такие как ошибки, предупреждения и информационные сообщения. Этот журнал помогает администраторам отслеживать состояние системы.

Журналируемые транзакции (Transactional Logging): В базе данных или файловой системе это подход, при котором все изменения данных записываются в журнал в виде транзакций. Каждое изменение данных (например, добавление, удаление или изменение записи) фиксируется в журнале, и в случае сбоя система может откатить данные до предыдущего согласованного состояния. Это важно для обеспечения целостности данных и их восстановления после сбоев.

Циклическое журналирование (Circular Logging): В этом подходе старые записи в журнале автоматически заменяются новыми, когда достигается максимальный размер журнала. Это экономит место на диске и полезно для длительного мониторинга, когда важно хранить только актуальную информацию.

Удаленное журналирование (Remote Logging): Когда система настроена на отправку журналов на удаленный сервер или облачное хранилище для дальнейшего анализа, хранения и архивирования. Это особенно полезно для распределенных систем и крупных организаций, где нужно централизованно собирать логи с разных машин.

Журналирование в реальном времени (Real-Time Logging): Это подход, при котором все события или изменения данных записываются в журнал немедленно, что позволяет оперативно отслеживать происходящие изменения. Этот подход часто используется в приложениях для мониторинга или для повышения безопасности (например, в системах обнаружения вторжений).

**1.3.4** Журналы Windows: структура, виды хранимых данных и интерфейс

В операционной системе Windows существует встроенная система журналирования, называемая Журнал событий Windows (Windows Event Log). Этот механизм записывает различные системные события, ошибки и предупреждения, а также информацию о безопасности и приложениях.

Журналы событий Windows организованы по категориям, таким как:

1. System — события, связанные с операционной системой и ее компонентами.
2. Application — события, связанные с работой приложений.
3. Security — события, касающиеся безопасности (например, попытки входа в систему).
4. Setup — события, связанные с установкой или обновлением системы.
5. Forwarded Events — события, полученные с удаленных компьютеров.

Каждое событие в журнале содержит:

1. Идентификатор события (Event ID) — уникальный номер, который помогает классифицировать событие.
2. Уровень (Level) — важность события (например, ошибка, предупреждение, информация).
3. Источник (Source) — приложение или компонент, который сгенерировал событие.
4. Описание (Description) — текстовое описание события, которое может включать дополнительную информацию.
5. Дата и время — время, когда событие было зарегистрировано.

Интерфейс работы с журналами в Windows включает использование таких инструментов, как Просмотр событий (Event Viewer), который позволяет просматривать, фильтровать и анализировать записи журналов, а также API Windows, с помощью которого разработчики могут программно работать с журналами событий. [3]

**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. [4]

В качестве среды разработки 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. [5]

**3 РАБОТА ПРИЛОЖЕНИЯ**

Данный код представляет собой программу, которая выполняет несколько операций с реестром Windows, таких как получение, сохранение, сравнение состояний и генерация файлов с изменениями в реестре. Программа использует API Windows для работы с реестром и выполняет эти операции с ключами реестра, связанными с автозапуском программ. Все эти действия могут быть полезны для мониторинга изменений в реестре, например, для отслеживания изменений в настройках автозагрузки.

**3.1 Структура приложения**

Программа состоит из нескольких частей, каждая из которых выполняет конкретную задачу:

1. Получение данных из реестра (GetRegistryKeysAndValues) — извлекает ключи и их значения из указанного раздела реестра.
2. Сохранение снимка реестра (SaveRegistrySnapshot) — сохраняет текущее состояние реестра в файл для дальнейшего анализа.
3. Сравнение снимков реестра (CompareRegistrySnapshots) — сравнивает два снимка состояния реестра и находит различия между ними.
4. Отображение изменений (DisplayDifferences) — выводит на экран изменения, найденные в процессе сравнения снимков.
5. Генерация reg-файлов (GenerateRegFile, GenerateInverseRegFile) — создает файлы для внесения изменений в реестр, например, для восстановления предыдущего состояния.

**3.2 Функции приложения**

GetRegistryKeysAndValues. Открывает реестр по заданному пути и извлекает все ключи и значения. Для этого используется функция RegOpenKeyEx для открытия ключа реестра и RegEnumValue для извлечения всех значений этого ключа. Возвращает список пар "ключ-значение".

SaveRegistrySnapshot. Принимает список ключей и значений, а также имя файла, в который нужно записать этот список в формате .reg. Формат записи соответствует формату, который Windows использует для сохранения изменений в реестре.

CompareRegistrySnapshots. Принимает два списка ключей и значений — "эталонный" и "текущий". Сравнивает их и возвращает список изменений. Если ключ был удален или его значение изменилось, это будет отражено в результирующем списке. Для каждого различия создается пара: ключ и описание изменения (например, "Deleted" или новое значение).

DisplayDifferences. Отображает изменения между снимками на экране, выводя для каждого различия ключ и его новое значение или статус "Удалено".

GenerateRegFile и GenerateInverseRegFile. Создают .reg файлы с изменениями, которые можно использовать для восстановления состояния реестра (например, для удаления или изменения ключей).

GenerateRegFile создает файл, который применяет изменения (добавляет или изменяет ключи).

GenerateInverseRegFile генерирует файл с инвертированными изменениями, то есть для каждого изменения будет предложено его удаление.

**3.3 Анализ данных**

Программа выполняет несколько ключевых операций с данными.

Сбор данных:

Программа использует функцию RegEnumValue, чтобы собрать все ключи и значения из заданного раздела реестра. Эти данные сохраняются в векторе пар pair<wstring, wstring>, где первый элемент пары — это имя ключа, а второй — его значение.

Сравнение состояний:

После получения базового и текущего снимка состояния реестра, программа сравнивает их. Для каждого ключа из эталонного снимка проверяется, существует ли он в текущем снимке, и если существует, то проверяется, совпадают ли его значения. Если ключ был удален или изменен, это будет отражено в списке различий.

Формирование выходных данных:

На основе полученных различий программа создает два файла:

registry\_changes.reg — файл, который можно использовать для применения изменений (например, добавления или изменения значений).

registry\_inverse\_changes.reg — инвертированный файл, который можно использовать для отмены изменений (удаления или восстановления значений).

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Основные понятия реестров. – [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/ru-ru/vcpkg/concepts/registries.

[2] Создание реестров. – [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/ru-ru/vcpkg/maintainers/registries.

[3] Просмотр событий и ошибок. – [Электронный ресурс]. – Режим доступа: <https://learn.microsoft.com/ru-ru/defender-endpoint/event-error-codes>.

[4] C++ Tutorial. – [Электронный ресурс]. – Режим доступа: https://www.w3schools.com/cpp/.

[5] Visual Studio. – [Электронный ресурс]. – Режим доступа: <https://visualstudio.microsoft.com/ru/.>

**ПРИЛОЖЕНИЕ** **А**

**(обязательное)**

**Реализация программы на языке С++**

#include <vector>

#include <windows.h>

#include <tlhelp32.h>

#include <iostream>

#include <fstream>

#include <string>

#include <algorithm>

using namespace std;

vector<pair<wstring, wstring>> GetRegistryKeysAndValues(HKEY hKey, const wstring& subkey) {

vector<pair<wstring, wstring>> keysAndValues;

HKEY hSubKey;

LONG result = RegOpenKeyEx(hKey, subkey.c\_str(), 0, KEY\_READ, &hSubKey);

if (result == ERROR\_SUCCESS) {

DWORD index = 0;

DWORD nameSize = 1024;

DWORD dataSize = 1024;

wchar\_t nameBuffer[1024];

wchar\_t dataBuffer[1024];

while (RegEnumValue(hSubKey, index, nameBuffer, &nameSize, NULL, NULL, (BYTE\*)dataBuffer, &dataSize) == ERROR\_SUCCESS) {

wstring key(nameBuffer, nameSize);

wstring value(dataBuffer, dataSize / sizeof(wchar\_t));

keysAndValues.push\_back(make\_pair(key, value));

index++;

}

RegCloseKey(hSubKey);

}

return keysAndValues;

}

void SaveRegistrySnapshot(const vector<pair<wstring, wstring>>& keysAndValues, const wstring& filename) {

wofstream file(filename);

file << L"[HKEY\_CURRENT\_USER]\n";

for (const auto& kv : keysAndValues) {

file << L"\"" << kv.first << L"\"=\"" << kv.second << L"\"\n";

}

file.close();

}

vector<pair<wstring, wstring>> CompareRegistrySnapshots(

const vector<pair<wstring, wstring>>& baseline,

const vector<pair<wstring, wstring>>& current) {

vector<pair<wstring, wstring>> differences;

for (const auto& baselineKeyValue : baseline) {

auto it = find\_if(current.begin(), current.end(), [&baselineKeyValue](const pair<wstring, wstring>& currentKeyValue) {

return currentKeyValue.first == baselineKeyValue.first;

});

if (it == current.end()) {

differences.push\_back(make\_pair(baselineKeyValue.first, L"Deleted"));

}

else if (it->second != baselineKeyValue.second) {

differences.push\_back(make\_pair(baselineKeyValue.first, it->second));

}

}

return differences;

}

void DisplayDifferences(const vector<pair<wstring, wstring>>& differences) {

if (differences.empty()) {

wcout << L"No changes detected in the registry.\n";

}

else {

wcout << L"Detected the following changes:\n";

for (const auto& diff : differences) {

wcout << L"Key: " << diff.first << L" => " << diff.second << endl;

}

}

}

void GenerateRegFile(const vector<pair<wstring, wstring>>& differences, const wstring& filename) {

wofstream file(filename);

for (const auto& diff : differences) {

if (diff.second == L"Deleted") {

file << L"[-" << diff.first << L"]\n";

}

else {

file << L"[" << diff.first << L"]\n";

file << L"\"" << diff.second << L"\"\n";

}

}

file.close();

}

void GenerateInverseRegFile(const vector<pair<wstring, wstring>>& differences, const wstring& filename) {

wofstream file(filename);

for (const auto& diff : differences) {

if (diff.second == L"Deleted") {

file << L"[" << diff.first << L"]\n";

file << L"\"" << diff.second << L"\"\n";

}

else {

file << L"[-" << diff.first << L"]\n";

}

}

file.close();

}

int main() {

wstring path = L"SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Run";

vector<pair<wstring, wstring>> baselineKeysAndValues = GetRegistryKeysAndValues(HKEY\_CURRENT\_USER, path);

SaveRegistrySnapshot(baselineKeysAndValues, L"baseline\_snapshot.reg");

wcout << L"Waiting for changes...\n";

Sleep(10000);

vector<pair<wstring, wstring>> currentKeysAndValues = GetRegistryKeysAndValues(HKEY\_CURRENT\_USER, path);

vector<pair<wstring, wstring>> differences = CompareRegistrySnapshots(baselineKeysAndValues, currentKeysAndValues);

DisplayDifferences(differences);

GenerateRegFile(differences, L"registry\_changes.reg");

GenerateInverseRegFile(differences, L"registry\_inverse\_changes.reg");

return 0;

}