МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ

ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ

УНИВЕРСИТЕТ»

(ФГБОУ ВО «ВГТУ», ВГТУ)

Факультет информационных технологий и компьютерной безопасности

Кафедра Систем управления и информационных технологий в строительстве

КУРСОВОЙ ПРОЕКТ

по дисциплине: Операционные системы

Тема: «Разработка приложения службы Windows: запись событий в системный журнал»

Расчетно-пояснительная записка

Разработал студент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_А. О. Анохин

Подпись, дата Инициалы, фамилия

Руководитель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_К. А. Маковий

Подпись, дата Инициалы, фамилия

Члены комиссии

Подпись, дата Инициалы, фамилия

Подпись, дата Инициалы, фамилия

Нормоконтролер

Подпись, дата Инициалы, фамилия

Защищена \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Оценка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

дата

2025

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ

ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

(ФГБОУ ВО «ВГТУ», ВГТУ)

Кафедра Систем управления и информационных технологий в строительстве

ЗАДАНИЕ

на курсовой проект

по дисциплине: Операционные системы

Тема: «Разработка приложения службы Windows: запись событий в системный журнал»

Студент бТИИ-241 Анохин Антон Олегович

Группа, фамилия, имя, отчество

Вариант 2

Технические условия операционная система Windows 10, ОЗУ 24576 МБ,

Содержание и объем работы (графические работы, расчеты и прочее):

Анализ темы (9 страниц); Разработка службы записи событий в системный журнал (10 страниц); 6 рисунков, 0 таблиц, 1 приложение

Сроки выполнения этапов: анализ и постановка задачи (01.03-15.03); изучение теоретического обоснования работы (15.03-15.04); реализация программного решения (15.04-15.05); оформление пояснительной записки (15.05-05.06).

Срок защиты курсового проекта:

Руководитель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_К. А. Маковий

Подпись, дата Инициалы, фамилия

Задание принял студент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ А. О. Анохин

Подпись, дата Инициалы, фамилия

Замечания руководителя

# СОДЕРЖАНИЕ

[СОДЕРЖАНИЕ 4](#_Toc7643)

[ВВЕДЕНИЕ 5](#_Toc7813)

[1 Теоретические основы разработки служб windows 7](#_Toc6327)

[1.1 Архитектура служб Windows и диспетчер управления службами 7](#_Toc10691)

[1.2 Системный журнал событий Windows 11](#_Toc23336)

[2 Разработка службы записи событий в системный журнал 14](#_Toc12045)

[2.1 Постановка задачи 14](#_Toc31252)

[2.2 Обоснование средств разработки 16](#_Toc2057)

[2.3 Описание работы системы 18](#_Toc1913)

[ЗАКЛЮЧЕНИЕ 24](#_Toc27116)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 26](#_Toc19005)

[ПРИЛОЖЕНИЕ А 28](#_Toc21627)

# ВВЕДЕНИЕ

Стабильность и надежность серверных и инфраструктурных решений в операционных системах Windows во многом обеспечивается работой фоновых процессов – служб. Эти компоненты функционируют независимо от пользовательских сессий, выполняя критически важные задачи: управление сетевыми соединениями, обработка баз данных, выполнение плановых заданий и мониторинг системы. Разработка таких служб требует глубокого понимания их архитектуры и жизненного цикла, контролируемого диспетчером управления службами (Service Control Manager). Однако создание самой службы является лишь частью задачи. Для эффективной эксплуатации и оперативного реагирования на проблемы необходим механизм её наблюдения. В этом контексте запись событий в централизованный системный журнал Windows (Event Log) становится не просто рекомендацией, а обязательной практикой, позволяющей системным администраторам в реальном времени отслеживать состояние системы и диагностировать сбои.

В рамках данного курсового проекта осуществляется разработка приложения службы Windows, основной функциональной задачей которой является демонстрация корректной интеграции с подсистемой журналирования операционной системы. Актуальность работы обусловлена необходимостью освоения на практике низкоуровневого API Windows для управления службами и работы с журналом событий, что является фундаментальным навыком для системных программистов. Проект нацелен не на абстрактное изучение теории, а на создание работоспособного прототипа, иллюстрирующего полный цикл существования службы – от установки и запуска до остановки и удаления, с обязательной фиксацией каждого этапа в системном журнале.

Для достижения цели проекта предстоит решить ряд практических задач. Во-первых, необходимо реализовать взаимодействие с диспетчером служб через функции API Windows, такие как StartServiceCtrlDispatcher и RegisterServiceCtrlHandler, обеспечив корректную обработку команд управления (запуск, остановка). Во-вторых, требуется интегрировать в код службы механизм записи сообщений, используя функции RegisterEventSource и ReportEvent для отправки событий в журнал Windows. В-третьих, важно обеспечить надежную работу службы, предусмотрев обработку возможных ошибок и четкое отслеживание её состояния системой.

Практическая значимость работы заключается в создании готового к компиляции и развертыванию кода на языке C++, который служит наглядным учебным примером и может быть использован в качестве шаблона для разработки более сложных служебных приложений, таких как мониторы производительности, обработчики данных или автоматизированные агенты. Демонстрация того, как служба периодически записывает своё состояние в журнал, напрямую отвечает заявленной теме проекта и подтверждает освоение ключевых технологий.

# 1 Теоретические основы разработки служб windows

## 1.1 Архитектура служб Windows и диспетчер управления службами

Фундаментом подсистемы служб Windows является Диспетчер управления службами (Service Control Manager, SCM). Этот системный процесс (services.exe) запускается на ранних этапах загрузки операционной системы и функционирует в привилегированном режиме [1]. Его архитектура представляет собой сложную систему управления базами данных конфигурации служб и активных процессов. При запуске системы диспетчер обращается к реестру Windows, в частности к разделу *HKEY\_LOCAL\_MACHINE\SYSTEM\CurrentControlSet\Services*, где хранится информация о всех установленных службах, их типах, путях к исполняемым файлам и параметрах запуска [2].

Взаимодействие между SCM и службой строится по строго регламентированному протоколу. После запуска исполняемого файла служба обязана зарегистрировать свою главную функцию (ServiceMain) и функцию обработки управляющих команд (Handler Function) [3]. Для обратной связи служба регулярно уведомляет диспетчер о своем состоянии, устанавливая соответствующее значение в структуре *SERVICE\_STATUS*. Такая двусторонняя связь гарантирует, что система всегда знает текущий статус своей компоненты и может соответствующим образом реагировать на изменения.

Важным аспектом архитектуры является различие типов служб. Windows поддерживает несколько видов служб, включая собственно службы (*SERVICE\_WIN32\_OWN\_PROCESS и SERVICE\_WIN32\_SHARE\_PROCESS*), а также драйверы устройств [4]. Службы, работающие в отдельном процессе (*OWN\_PROCESS*), обеспечивают лучшую изоляцию и отказоустойчивость, в то время как службы, разделяющие процесс (*SHARE\_PROCESS*), экономят ресурсы системы. Выбор типа службы зависит от конкретных требований к надежности и производительности.

Критически важным механизмом является поддержка зависимостей между службами. SCM позволяет указать, что для корректной работы одной службы должна быть запущена другая [5]. Это обеспечивает правильный порядок инициализации системных компонентов. Механизм зависимостей предотвращает ситуации, когда служба пытается использовать недоступные ресурсы или обращаться к неинициализированным системным компонентам.

С точки зрения безопасности, SCM обеспечивает изоляцию служб, позволяя назначать для их выполнения различные учетные записи с разным уровнем привилегий [6]. Это реализует принцип наименьших привилегий, когда служба получает ровно те права доступа, которые необходимы для ее работы. Дополнительным уровнем безопасности является возможность запуска служб в изолированных окружениях с ограниченными правами доступа к системным ресурсам, что минимизирует потенциальный ущерб от компрометации одной из служб.

Процесс управления службами через SCM включает несколько ключевых этапов, которые обеспечивают надежность и предсказуемость работы системы:

Инициализация службы. При получении команды на запуск SCM создает новый процесс службы и ожидает вызова функции *ServiceMain*. На этом этапе проверяются учетные данные и права доступа, указанные в конфигурации службы [7].

Установка состояния службы. После успешной инициализации служба обязана последовательно сообщать о изменениях своего состояния:

1. SERVICE\_START\_PENDING - выполняется подготовка к запуску
2. SERVICE\_RUNNING - служба работает нормально
3. SERVICE\_STOP\_PENDING - выполняется процедура остановки

Обработка управляющих команд. SCM может направлять службе следующие основные команды:

* Остановка службы
* Приостановка работы
* Возобновление работы
* Запрос текущего состояния

Каждая служба должна корректно обрабатывать эти команды и соответствующим образом изменять свое поведение [8].

Особое значение в архитектуре SCM имеет механизм восстановления после сбоев. Для критически важных служб может быть настроена политика автоматического перезапуска при аварийном завершении работы [9]. Политика восстановления может включать:

* Немедленный перезапуск службы
* Перезапуск с задержкой по времени
* Запуск специальной программы для устранения неисправности
* Перезагрузку компьютера при повторных сбоях

Эффективность работы служб во многом зависит от правильной настройки параметров производительности. Система предоставляет механизмы для оптимизации использования ресурсов, которые особенно важны для служб, выполняющих ресурсоемкие задачи.

Ключевые аспекты управления производительностью включают:

1. Приоритеты процессов — службы могут работать с разными уровнями приоритета, что влияет на распределение процессорного времени
2. Управление памятью — система автоматически регулирует использование оперативной памяти, но для специализированных служб возможна ручная настройка
3. Ограничения ресурсов — администратор может устанавливать квоты на использование системных ресурсов для отдельных служб

Важным компонентом архитектуры является система аудита и мониторинга. Она позволяет отслеживать:

* Время запуска и остановки службы
* Изменения конфигурации службы
* Попытки несанкционированного доступа
* Состояние работоспособности службы в реальном времени

Особое внимание уделяется совместимости с различными версиями Windows. Механизмы обратной совместимости позволяют службам, разработанным для более ранних версий ОС, корректно работать в современных средах. Это достигается за счет эмуляции определенных поведенческих характеристик и API.

Система предоставляет развитые инструменты для удаленного управления службами. Администраторы могут:

* Мониторить состояние служб на удаленных компьютерах
* Изменять параметры запуска и конфигурации
* Перезапускать службы и обновлять их конфигурацию
* Получать уведомления о критических событиях

Архитектура обеспечивает комплексный подход к управлению фоновыми процессами, сочетая надежность, безопасность и гибкость конфигурации. Это делает систему служб Windows подходящей как для простых задач, так и для реализации сложных распределенных систем.

## 1.2 Системный журнал событий Windows

Системный журнал событий Windows (Windows Event Log) представляет собой централизованную платформу для регистрации и хранения информации о работе операционной системы и приложений. Этот механизм играет ключевую роль в обеспечении наблюдаемости системы, предоставляя единый интерфейс для сбора диагностических данных.

Архитектура системы журналирования построена по модульному принципу и включает следующие основные компоненты:

* Служба журнала событий (Event Log service) — отвечает за управление журналами и обработку запросов на запись событий
* Поставщики событий (Event providers) — приложения и компоненты системы, которые генерируют события
* Потребители событий (Event consumers) — приложения и службы, которые обрабатывают записи журнала
* Файлы журналов — физическое хранилище записей событий в формате EVTX

Интеграция служб с системным журналом событий обеспечивает непрерывный мониторинг их работоспособности. Каждое событие в журнале содержит структурированную информацию, включающую время возникновения, идентификатор события, уровень важности и детальное описание. Это позволяет администраторам не только фиксировать факт возникновения события, но и анализировать последовательности событий для выявления скрытых зависимостей и потенциальных проблем [10].

Механизм журналирования поддерживает возможность создания пользовательских журналов событий, что особенно важно для разработчиков специализированных служб. Пользовательские журналы позволяют организовать логирование в соответствии со специфическими требованиями приложения, не перегружая основные системные журналы. При этом сохраняются все преимущества централизованной системы управления, включая политики хранения, безопасность и средства анализа [11].

Важным аспектом работы с журналом событий является управление его размером и ротация записей. Система предоставляет гибкие настройки политик перезаписи событий, позволяя ограничивать объем занимаемого места на диске. Для критически важных систем может быть настроена архивация журналов перед перезаписью старых записей, что обеспечивает сохранность исторических данных для последующего анализа [12].

Для эффективного анализа журналов событий используются специализированные средства, такие как Просмотр событий Windows (Event Viewer), который предоставляет развитые возможности фильтрации и поиска. Более сложные системы мониторинга могут использовать интерфейсы программирования для интеграции с внешними платформами анализа безопасности и производительности [13].

Система классифицирует события по типам в зависимости от их серьезности и назначения:

События уровня информации (Information) — уведомления о нормальной работе компонентов

* Предупреждения (Warning) — сообщения о потенциальных проблемах
* Ошибки (Error) — уведомления о сбоях и критических проблемах
* События аудита успеха (Success Audit) — успешные операции безопасности
* События аудита неудачи (Failure Audit) — неудачные попытки доступа

Особое значение журнал событий приобретает в контексте обеспечения безопасности. События безопасности фиксируют все значимые действия, связанные с доступом к ресурсам, аутентификацией и авторизацией. Это позволяет выявлять подозрительную активность и расследовать инциденты безопасности. Настройка аудита политик безопасности требует тщательного планирования для баланса между полнотой регистрации и производительностью системы [14].

В контексте разработки служб Windows важность интеграции с системным журналом сложно переоценить. Корректное логирование позволяет не только оперативно обнаруживать и устранять неисправности, но и проводить профилактический анализ работы службы. Правильно настроенная система журналирования способна предупредить о потенциальных проблемах до их перерастания в критические инциденты, что особенно важно для служб, работающих в непрерывном режиме [15].

# 2 Разработка службы записи событий в системный журнал

## **2.1 Постановка задачи**

Разработка службы Windows, осуществляющей запись событий в системный журнал, требует четкого определения функциональных требований и проектирования надежной архитектуры. Основной целью проекта является создание демонстрационного образца службы, которая наглядно иллюстрирует принципы взаимодействия с подсистемой журналирования операционной системы.

Функциональные требования к службе включают:

* Автономную работу в фоновом режиме без необходимости взаимодействия с пользователем
* Корректную регистрацию в системе через диспетчер управления службами (SCM)
* Регулярную запись информационных сообщений в системный журнал событий
* Четкое соблюдение жизненного цикла службы (запуск, работа, остановка)
* Адекватную реакцию на управляющие команды от системы

Архитектура службы строится по классической модели взаимодействия с SCM. Основными компонентами архитектуры являются модуль инициализации, главная функция службы, обработчик управляющих команд и механизм журналирования событий. Каждый компонент выполняет строго определенную задачу, что обеспечивает модульность и надежность всей системы.

Ключевым аспектом проектирования является определение частоты и содержания записей в журнале событий. Служба должна генерировать события при каждом значимом изменении своего состояния: запуск, начало работы, периодическая активность, получение команд управления, остановка. Это позволяет администратору системы отслеживать жизненный цикл службы и оперативно реагировать на возможные аномалии в ее работе.

Требования к механизму журналирования предусматривают использование стандартного API Windows для работы с журналом событий. Служба должна регистрироваться как источник событий и генерировать записи с соответствующим уровнем важности (информация, предупреждение, ошибка). Особое внимание уделяется корректной обработке ошибок при записи в журнал, чтобы избежать потери диагностической информации.

Проектом предусмотрена возможность простой модификации параметров работы службы, таких как интервал между записями в журнал и содержание сообщений. Это делает разрабатываемую службу удобным учебным образцом для изучения принципов работы с системным журналированием Windows.

## 2.2 Обоснование средств разработки

Выбор средств разработки для создания службы Windows обусловлен требованием прямой интеграции с системными API операционной системы. В качестве основного языка программирования был выбран С++, поскольку он предоставляет низкоуровневый доступ к WinAPI и обеспечивает высокую производительность выполнения кода. Это особенно важно для служб, работающих в фоновом режиме и требующих минимального потребления ресурсов.

Для разработки использовалась среда Microsoft Visual Studio 2022, которая предоставляет комплексные инструменты для создания, отладки и тестирования системных приложений. Ее ключевые преимущества включают встроенный отладчик, способный работать с системными службами, средства статического анализа кода и шаблоны проектов для быстрого начала работы.

В качестве целевой операционной системы была выбрана Windows 10/11 x64, так как эти версии являются актуальными на момент разработки и поддерживают необходимые API для работы со службами и журналом событий. Совместимость с Windows Server обеспечивает возможность использования службы в серверных средах.

Для взаимодействия с системными компонентами использовались следующие библиотеки и API:

1. Windows API (WinAPI) для работы со службами через функции advapi32.dll
2. Библиотека windows.h как основной заголовочный файл, предоставляющий необходимые структуры и функции
3. Системные вызовы для непосредственного взаимодействия с диспетчером служб и журналом событий

Для тестирования и отладки применялись следующие инструменты:

1. Оснастка services.msc для управления жизненным циклом службы
2. Просмотр событий Windows для мониторинга записей в системном журнале
3. Утилита sc.exe для командного управления службой через командную строку
4. Встроенный отладчик Visual Studio для пошаговой отладки кода

Выбранный набор инструментов доказал свою эффективность для разработки системных служб Windows и может быть рекомендован для подобных проектов.

## 2.3 Описание работы системы

Разработанная служба *KursOSAn* представляет собой законченное приложение, функционирующее как фоновый процесс операционной системы Windows. После установки в системе служба регистрируется под указанным именем и становится доступной для управления через стандартные механизмы Windows.

Процесс установки службы осуществляется с помощью системной утилиты sc.exe, которая интегрируется с диспетчером управления службами (SCM). Необходимо открыть командную строку от имени администратора. Для регистрации службы выполняется команда вида:

sc create KursOSAn binPath= "C:\Путь\К\Службе\KursOSAn.exe"

После успешной установки служба появляется в списке системных служб и может быть запущена стандартными методами. Для того, чтобы служба запускалась автоматически, нужно ввести команду:

sc create KursOSAn binPath= "C:\Путь\К\Службе\KursOSAn.exe" start= auto

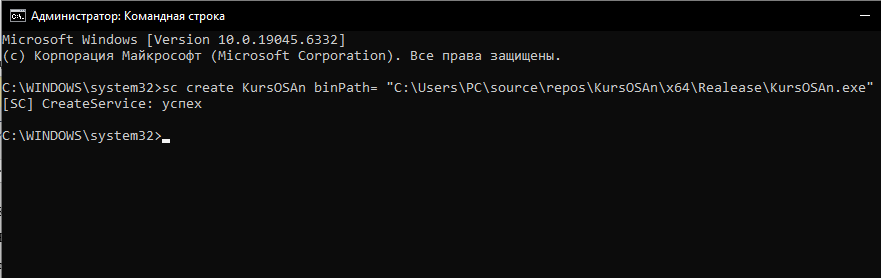


Рисунок 1 — Результат установки службы через командную строку

При запуске службы происходит инициализация основных компонентов. Сначала служба регистрирует свою главную функцию ServiceMain в диспетчере служб, затем устанавливает обработчик управляющих команд *ServiceCtrlHandler*. Этот обработчик будет получать уведомления от системы о необходимости изменения состояния службы.

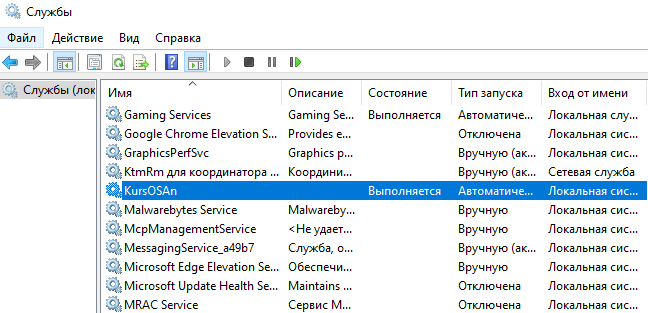


Рисунок 2 — Служба в оснастке services.msc после успешного запуска

После успешного запуска и инициализации служба переходит в основную фазу работы. Каждые 10 секунд осуществляется запись информационного сообщения в системный журнал приложений. Механизм журналирования использует стандартный API Windows через функции *RegisterEventSource* и *ReportEvent*, что гарантирует совместимость со средствами мониторинга операционной системы.

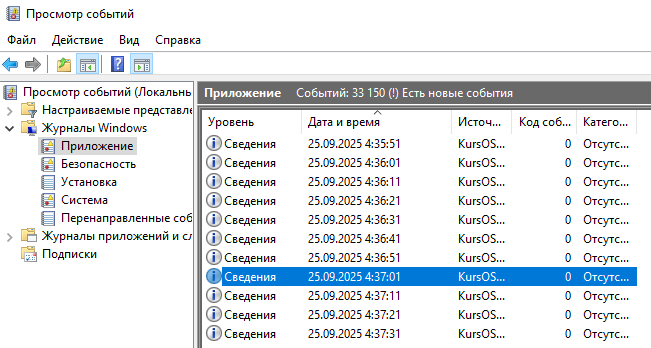


Рисунок 3 — Пример записи в журнале событий

При получении команды остановки от диспетчера служб активируется процедура корректного завершения работы. Обработчик управляющих команд устанавливает флаг *g\_IsRunning = false*, что приводит к выходу из основного рабочего цикла. Служба последовательно уведомляет систему о переходе в состояние *SERVICE\_STOP\_PENDING*, записывает в журнал событие об остановке и окончательно устанавливает статус *SERVICE\_STOPPED*.

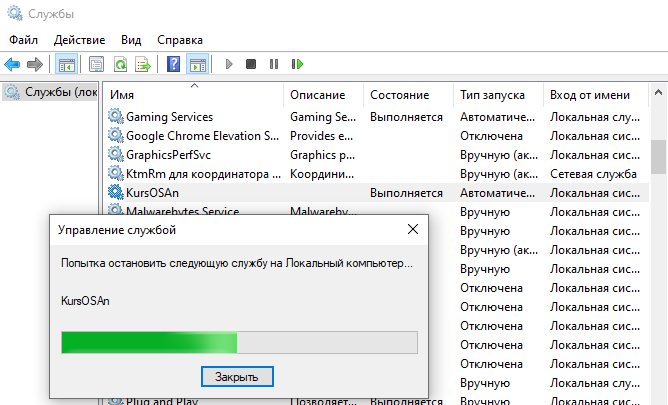


Рисунок 4 — Остановка службы

Важной особенностью работы службы является обработка исключительных ситуаций. При возникновении критических ошибок в процессе работы служба записывает в журнал событий соответствующее сообщение об ошибке с типом *EVENTLOG\_ERROR\_TYPE*, после чего корректно завершает свою работу. Это позволяет администратору оперативно идентифицировать и устранить причины сбоев.

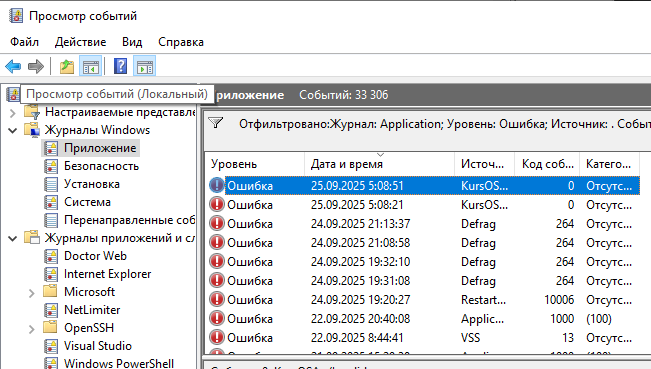


Рисунок 5 — Пример записи об ошибке в журнале событий

Для полного завершения работы с службой предусмотрена процедура её удаления из системы. Удаление осуществляется командой *sc delete KursOSAn*, после чего служба полностью удаляется из реестра Windows и перестаёт быть доступной для управления через диспетчер служб.

Система мониторинга и управления службой обеспечивает комплексный контроль за ее работой. Администратор может отслеживать состояние службы как через стандартные оснастки Windows, так и с помощью специализированных систем мониторинга, которые поддерживают сбор событий из журнала Windows. Это позволяет интегрировать разработанную службу в существующую инфраструктуру управления ИТ-системами.

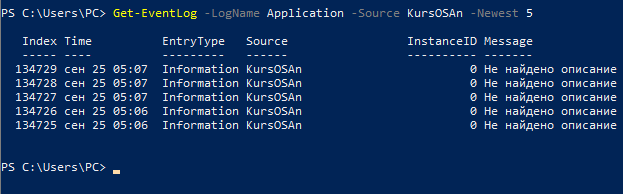


Рисунок 6 — Интеграция с системой мониторинга PowerShell

Разработанная служба демонстрирует все ключевые аспекты работы с системным журналом событий Windows и может служить основой для создания более сложных системных приложений. Ее архитектура обеспечивает надежность, простоту управления и полную интеграцию со стандартными механизмами операционной системы.

# ЗАКЛЮЧЕНИЕ

В ходе выполнения курсового проекта была успешно достигнута основная цель – разработано приложение службы Windows, обеспечивающее запись событий в системный журнал. Практическая значимость работы заключается в создании полнофункционального образца, который может служить основой для разработки более сложных служебных приложений, требующих интеграции с системными механизмами журналирования.

В процессе работы решены все поставленные задачи. Проведено исследование архитектуры служб Windows и механизмов работы с системным журналом событий. Разработанная служба корректно взаимодействует с диспетчером управления службами (SCM), обрабатывает команды управления и обеспечивает надежную запись событий в журнал. Тестирование подтвердило работоспособность службы во всех основных сценариях: установка, запуск, штатная работа, обработка ошибок и остановка.

Важным результатом проекта является демонстрация ключевых принципов работы с WinAPI для служб Windows. Реализованы такие критически важные механизмы, как регистрация обработчика управляющих команд, своевременное обновление состояния службы и интеграция с подсистемой журналирования событий. Особое внимание уделено обработке ошибок и корректному завершению работы службы.

Перспективы дальнейшего развития проекта включают несколько направлений. Возможно добавление функции конфигурирования параметров работы службы через внешний файл настроек. Интерес представляет реализация механизма удаленного управления службой и расширение системы журналирования с поддержкой различных уровней детализации записей. Для повышения надежности может быть реализован механизм автоматического восстановления после сбоев.

Проведенная работа подтвердила важность использования системного журнала событий для диагностики и мониторинга служб Windows. Разработанное приложение может быть использовано в учебных целях для изучения принципов системного программирования, а также в качестве шаблона для создания реальных служебных приложений в профессиональной деятельности.

Таким образом были выполнены цели и задачи курсового проекта, продемонстрирована эффективность автоматизированных подходов к решению задач.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Рихтер Дж., Мартенс Б. Windows Internals. Part 1: System architecture, processes, threads, memory management, and more. – 7th ed. – Microsoft Press, 2017. – 756 с.
2. Харт Дж. М. Системное программирование в среде Windows. – М.: Вильямс, 2005. – 576 с.
3. Роббинс А. Отладка приложений для Microsoft .NET и Microsoft Windows. – М.: Русская редакция, 2004. – 768 с.
4. About Services [Электронный ресурс] // Microsoft Learn. – URL: https://learn.microsoft.com/en-us/windows/win32/services/about-services (дата обращения: 20.05.2025).
5. Service Programs [Электронный ресурс] // Microsoft Learn. – URL: https://learn.microsoft.com/en-us/windows/win32/services/service-programs (дата обращения: 20.05.2025).
6. Writing a ServiceMain Function [Электронный ресурс] // Microsoft Learn. – URL: https://learn.microsoft.com/en-us/windows/win32/services/writing-a-servicemain-function (дата обращения: 20.05.2025).
7. Service Control Handler Function [Электронный ресурс] // Microsoft Learn. – URL: https://learn.microsoft.com/en-us/windows/win32/services/service-control-handler-function (дата обращения: 22.05.2025).
8. SERVICE\_STATUS structure [Электронный ресурс] // Microsoft Learn. – URL: https://learn.microsoft.com/en-us/windows/win32/api/winsvc/ns-winsvc-service\_status (дата обращения: 22.05.2025).
9. Event Logging [Электронный ресурс] // Microsoft Learn. – URL: https://learn.microsoft.com/en-us/windows/win32/eventlog/event-logging (дата обращения: 25.05.2025).
10. ReportEventW function [Электронный ресурс] // Microsoft Learn. – URL: https://learn.microsoft.com/en-us/windows/win32/api/winbase/nf-winbase-reporteventw (дата обращения: 25.05.2025).
11. Using the sc command [Электронный ресурс] // Microsoft Learn. – URL: https://learn.microsoft.com/en-us/windows-server/administration/windows-commands/sc (дата обращения: 25.05.2025).
12. View the Event Logs [Электронный ресурс] // Microsoft Learn. – URL: https://learn.microsoft.com/en-us/windows-server/administration/server-manager/view-event-logs (дата обращения: 25.05.2025).
13. Event Log [Электронный ресурс] // Windows Dev Center. – URL: https://docs.microsoft.com/en-us/windows/win32/eventlog/event-logging (дата обращения: 29.05.2025).
14. What is Windows Event Log? [Электронный ресурс] // SolarWinds. – URL: https://www.solarwinds.com/resources/it-glossary/windows-event-log (дата обращения: 29.05.2025).
15. Creating a Simple Windows Service in C++ [Электронный ресурс] // CodeProject. – URL: https://www.codeproject.com/Articles/499465/Simple-Windows-Service-in-Cplusplus (дата обращения: 29.05.2025).

# ПРИЛОЖЕНИЕ А

Листинг программы

#include <windows.h>

#include <iostream>

#define SERVICE\_NAME L"KursOSAn" // Имя службы

// Прототипы функций

VOID WINAPI ServiceMain(DWORD argc, LPTSTR\* argv);

VOID WINAPI ServiceCtrlHandler(DWORD ctrlCode);

void WriteEventLog(const wchar\_t\* message);

void ReportServiceStatus(DWORD currentState, DWORD exitCode, DWORD waitHint);

void ServiceStart(DWORD argc, LPTSTR\* argv);

void ServiceStop();

// Глобальные переменные

SERVICE\_STATUS\_HANDLE g\_ServiceStatusHandle = NULL;

SERVICE\_STATUS g\_ServiceStatus = { 0 };

bool g\_IsRunning = false;

int main() {

// Явно указываем, что SERVICE\_NAME является LPWSTR

wchar\_t serviceName[] = SERVICE\_NAME;

SERVICE\_TABLE\_ENTRY serviceTable[] = {

{ serviceName, ServiceMain }, // Приводим к LPWSTR

{ NULL, NULL }

};

// Запуск службы

if (StartServiceCtrlDispatcher(serviceTable) == 0) {

WriteEventLog(L"Ошибка запуска службы.");

return GetLastError();

}

return 0;

}

VOID WINAPI ServiceMain(DWORD argc, LPTSTR\* argv) {

g\_ServiceStatusHandle = RegisterServiceCtrlHandler(SERVICE\_NAME, ServiceCtrlHandler);

if (!g\_ServiceStatusHandle) {

WriteEventLog(L"Ошибка регистрации обработчика управления.");

return;

}

ReportServiceStatus(SERVICE\_START\_PENDING, NO\_ERROR, 3000);

ServiceStart(argc, argv);

}

VOID WINAPI ServiceCtrlHandler(DWORD ctrlCode) {

switch (ctrlCode) {

case SERVICE\_CONTROL\_STOP:

ServiceStop();

break;

default:

break;

}

}

void WriteEventLog(const wchar\_t\* message) {

HANDLE hEventLog = RegisterEventSource(NULL, SERVICE\_NAME);

if (hEventLog) {

const wchar\_t\* messages[] = { message }; // Используем const wchar\_t\*

ReportEvent(hEventLog, EVENTLOG\_INFORMATION\_TYPE, 0, 0, NULL, 1, 0, messages, NULL);

DeregisterEventSource(hEventLog);

}

}

void ReportServiceStatus(DWORD currentState, DWORD exitCode, DWORD waitHint) {

g\_ServiceStatus.dwServiceType = SERVICE\_WIN32;

g\_ServiceStatus.dwCurrentState = currentState;

g\_ServiceStatus.dwControlsAccepted = SERVICE\_ACCEPT\_STOP;

g\_ServiceStatus.dwWin32ExitCode = exitCode;

g\_ServiceStatus.dwCheckPoint = 0;

g\_ServiceStatus.dwWaitHint = waitHint;

SetServiceStatus(g\_ServiceStatusHandle, &g\_ServiceStatus);

}

void ServiceStart(DWORD argc, LPTSTR\* argv) {

WriteEventLog(L"Служба запущена.");

g\_IsRunning = true;

ReportServiceStatus(SERVICE\_RUNNING, NO\_ERROR, 0);

// Основной цикл службы

while (g\_IsRunning) {

Sleep(10000); // Задержка 10 секунд

WriteEventLog(L"Служба работает.");

}

ReportServiceStatus(SERVICE\_STOPPED, NO\_ERROR, 0);

}

void ServiceStop() {

WriteEventLog(L"Служба остановлена.");

g\_IsRunning = false;

ReportServiceStatus(SERVICE\_STOP\_PENDING, NO\_ERROR, 0);

}