1. **Что такое служба?**

**Служба** (или сервис) – это процесс, который выполняет вспомогательные функции. Они являются аналогами резидентных программ и часто запускаются при загрузке ОС или по команде, работая до завершения работы ОС или до получения команды на остановку. Важно отметить, что не любая программа, стартующая с ОС, является службой.

1. **Какие бывают службы?**

Выделяются службы по их **функциям**:

* Серверы: Обеспечивают доступ к ресурсам, например, к базам данных.
* Драйверы: Обеспечивают доступ к внешним устройствам.
* Мониторы: Отслеживают работу других приложений.

**Технические типы** служб Windows (определяемые в dwServiceType):

* SERVICE\_WIN32\_OWN\_PROCESS (самостоятельный процесс)
* SERVICE\_WIN32\_SHARE\_PROCESS (разделяемый процесс)
* SERVICE\_KERNEL\_DRIVER (драйвер ядра)
* SERVICE\_FILE\_SYSTEM\_DRIVER (драйвер файловой системы)
* SERVICE\_USER\_OWN\_PROCESS / SERVICE\_USER\_SHARE\_PROCESS (пользовательские процессы)
* Могут быть интерактивными (SERVICE\_INTERACTIVE\_PROCESS)

1. **Чем можно охарактеризовать службы?**

**Характеристики**:

* Работают только в фоновом режиме.
* Не имеют собственного интерфейса управления (ни GUI, ни TUI).
* Управляются специальной программой ОС – менеджером служб (SCM).
* Запускаются/останавливаются при старте/выключении ОС, входе/выходе пользователя или по команде от менеджера служб.
* Предназначены для предоставления услуг другим программам или ОС, а не напрямую пользователям.

1. **Что такое SCM?**

**SCM** – это Service Control Manager (Менеджер Управления Службами). Это специальная программа операционной системы Windows, которая управляет работой сервисов. Её функции включают поддержку базы данных установленных сервисов, запуск/остановку сервисов, передачу им управляющих запросов и предоставление информации об их состоянии.

1. **Где располагается информация о сервисе?**

Информация об установленных сервисах Windows хранится в реестре Windows, в разделе:  
**HKEY\_LOCAL\_MACHINE\SYSTEM\CurrentControlSet\Services**

Внутри этого раздела для каждой службы (и драйвера) существует свой подраздел, содержащий параметры конфигурации, такие как ImagePath (путь к файлу), Type (тип службы), Start (тип запуска), DisplayName, DependOnService, DependOnGroup и другие.

1. **Что такое Linux-демон?**

Демон (daemon) как процесс в Linux, обладающий следующими свойствами:

* Имеет длинный жизненный цикл (часто работает от загрузки до выключения системы).
* Выполняется в фоновом режиме.
* Не имеет контролирующего терминала.

Лабораторная работа добавляет, что демоны обычно обрабатывают сигналы, такие как SIGTERM (для остановки) и SIGHUP (для перезагрузки конфигурации).

1. **Что такое Init и Systemd?**

Init (SysV Init) и Systemd это менеджеры сервисов (системы инициализации) в Linux, которые отвечают за запуск, остановку и управление демонами (службами).

**Init**: Более старая система, использует скрипты в /etc/init.d/. Управляется в основном командой service. Считается устаревшей.

**Systemd**: Более новая и современная система, использует конфигурационные файлы (unit-файлы, например, .service) обычно в /etc/systemd/system/. Управляется командой systemctl.

Обе системы могут выполнять процесс «демонизации» (отсоединение от терминала и т.д.) для запускаемых программ и позволяют настраивать автозапуск служб при загрузке ОС.

**1. Разница между сервисами и демонами**

По сути, "сервис" – это термин, преимущественно используемый в контексте Windows, а "демон" – в контексте Linux. Оба обозначают фоновые процессы, работающие без прямого взаимодействия с пользователем и предоставляющие какую-либо автоматизированную или системную функциональность. Конкретные механизмы управления и детальные характеристики различаются в зависимости от ОС.

**2. Как посмотреть какие процессы являются демонами**

С помощью команды **ps -ef.**

Ключевые признаки демона в выводе:

Колонка **TTY**: Для демона здесь часто будет стоять ?**, что указывает на отсутствие контролирующего терминала.** В примере у daemon\_SIGHUP (PID 1045) в колонке TTY стоит ?.

**PPID** (Parent Process ID): **Демоны часто являются дочерними процессами init** (PID 1) или **аналогичного системного процесса** (например, процесса **systemd**, если он используется в качестве init). В примере daemon\_SIGHUP имеет PPID 673, который, в свою очередь, является дочерним процессом init.

Для Windows-сервисов обычно используется оснастка "Службы" (показана на Слайде 8) или команда sc query (Слайд 38, 40), а не просто просмотр списка процессов, хотя "Диспетчер задач" может отображать процессы, запущенные от системных учетных записей, которые часто являются сервисами.

**3. Процесс ручного создания демонов в линуксе**

Слайды 43-47 описывают шаги, которые программа должна выполнить, чтобы "демонизировать" себя:

**Вызов fork, после которого родитель завершается, а потомок продолжает работать**. В результате этого демон становится **потомком процесса init**. Это делается, чтобы командная оболочка, запустившая программу, получила управление обратно, и чтобы потомок гарантированно не стал лидером группы процессов.

Дочерний процесс вызывает **setsid**, чтобы начать новую сессию, стать лидером этой сессии и группы процессов, и разорвать любые связи с контролирующим терминалом.

Если после этого демон больше не открывает никаких терминальных устройств, можно не волноваться о том, что он восстановит соединение с контролирующим терминалом. В противном случае, нужно сделать так, чтобы терминальное устройство не стало контролирующим. Это можно сделать двумя способами:

* Указывать флаг O\_NOCTTY для любых вызовов open, которые могут открыть терминальное устройство.
* Более простой вариант: после setsid еще раз сделать вызов fork, опять позволив родителю завершиться, а потомку (правнуку) – продолжить работу. Это гарантирует, что потомок не станет лидером сессии, что делает невозможным повторное соединение с контролирующим терминалом.

Очистить атрибут umask процесса (Слайд 45), чтобы файлы и каталоги, созданные демоном, имели запрашиваемые права доступа (umask(0)).

Поменять текущий рабочий каталог процесса (обычно на корневой – /) (Слайд 45). Это необходимо, чтобы демон не блокировал отмонтирование файловой системы, на которой он был запущен.

Закрыть все открытые файловые дескрипторы, которые демон унаследовал от своего родителя (особенно стандартный ввод (0), вывод (1) и ошибку (2), если они ссылаются на терминал) (Слайд 46). (Этот шаг может быть скорректирован, если некоторые дескрипторы нужно оставить открытыми).

Закрыв дескрипторы с номерами 0, 1 и 2, демон обычно перенаправляет их в предварительно открытый файл /dev/null, используя вызов dup2 (или похожий) (Слайд 47). Это позволяет избежать ошибок при вызове библиотечных функций ввода/вывода и исключает возможность повторного открытия файлов с этими дескрипторами.

**4. Что будет если сервис работает и отправить команду delete**

На практике в Windows удаление сервиса во время его работы обычно удаляет его регистрацию, т.е. он не запустится при следующей загрузке и им больше нельзя будет управлять через SCM. Текущий запущенный экземпляр процесса может продолжать работать до тех пор, пока он не будет явно остановлен (другими средствами, например, через Диспетчер задач, или если он сам себя остановит) или до перезагрузки системы. Им больше нельзя будет управлять командами sc start или sc stop.

**5. Рекомендации по созданию демонов**

Корректное завершение (SIGTERM) (Слайд 50): Демон должен обрабатывать сигнал SIGTERM для выполнения очистки (освобождения ресурсов) перед завершением. Процесс init посылает SIGTERM во время выключения системы.

Своевременная очистка (Слайд 51): Процедуру очистки следует выполнять как можно быстрее, поскольку через 5 секунд после SIGTERM процесс init отправляет сигнал SIGKILL (который нельзя перехватить).

Управление ресурсами (Слайд 52): Следует особенно тщательно следить за потенциальными утечками памяти и файловыми дескрипторами из-за длительного жизненного цикла демонов.

Единственный экземпляр (Слайд 52, 53): Часто необходимо убедиться, что только один экземпляр демона активен. Обычно это достигается созданием lock-файла (например, в /var/run/) и применением к нему блокировки для записи. Если новый экземпляр не может получить блокировку, он завершается.

**6. Чем отличается обычный процесс от сервиса**

Управление жизненным циклом:

Сервисы: Управляются инфраструктурой управления сервисами ОС (SCM для Windows, init/systemd для Linux). Они могут быть настроены на запуск при загрузке, по требованию или при входе пользователя, и останавливаются ОС или специальными командами.

Обычные процессы: Обычно запускаются пользователем напрямую или другим процессом. Их жизненный цикл не управляется выделенным менеджером сервисов ОС таким же образом.

Взаимодействие с пользователем:

Сервисы: Работают в фоновом режиме и обычно не имеют прямого пользовательского интерфейса (GUI или TUI).

Обычные процессы: Могут иметь GUI, TUI или работать в фоне, но часто предполагают некоторую форму взаимодействия с пользователем или инициируются пользователем для конкретной задачи.

**Назначение**:

Сервисы: Предназначены для предоставления непрерывной фоновой функциональности ОС, другим приложениям или для управления системными ресурсами, часто без вмешательства пользователя после запуска.

Обычные процессы: Могут служить широкому спектру целей, включая интерактивные приложения.

**Контекст/Сессия:**

Сервисы (Windows): Могут работать в другой сессии, отличной от сессии вошедшего пользователя, часто под системными учетными записями (LocalSystem, NetworkService и т.д.), что позволяет им работать, даже когда ни один пользователь не вошел в систему.

Обычные процессы: Обычно выполняются в контексте пользователя, который их запустил, и в рамках сессии этого пользователя.

**Запуск**:

Слайд 3 отмечает: "НО! Не каждая программа запускаемая со стартом операционной системы является сервисом". Это подразумевает, что у сервисов есть специфическая структура регистрации и управления, которая отличает их от простых программ автозапуска.

**7. Что такое вообще appdata**

В общих терминах ОС, AppData – это каталог в профилях пользователей Windows, где приложения хранят специфичные для пользователя настройки, данные и конфигурационные файлы.

**8. Файл .service для линукс (что в нём и для чего)**

На основе раздела о Linux-демонах, особенно в контексте systemd:

Назначение: Файл .service – это конфигурационный файл юнита (unit configuration file), используемый systemd, современной системой инициализации и менеджером сервисов в Linux (Слайд 54, 63).

Расположение: Эти файлы обычно размещаются в каталоге /etc/systemd/system/ (Слайд 63). В примере показан testd.service в этом каталоге.

Функция: Он описывает, как systemd должен управлять конкретным сервисом (демоном). Это включает:

* Как запускать сервис (например, команду/исполняемый файл). Слайд 65 показывает /usr/sbin/testd как Main PID для testd.service.
* Пользователя и группу, от имени которых должен работать сервис.
* Зависимости от других сервисов или состояний системы.
* Политики перезапуска.
* Другие операционные параметры.

Лекция показывает, что после размещения этого файла systemctl можно использовать для управления сервисом (запуск, остановка, статус, включение автозапуска) (Слайды 64, 65, 66).

**I. Основы Сервисов/Демонов:**

**Цель и Необходимость:**Зачем вообще нужны службы/демоны? Почему нельзя просто запускать программу в обычном режиме? (Ответ: Для выполнения фоновых задач без участия пользователя, для старта вместе с ОС, для централизованного управления, для работы от имени системы).

**Отличия от Обычных Приложений:** В чем ключевые отличия жизненного цикла, управления и взаимодействия с пользователем у службы/демона по сравнению с обычным консольным или GUI-приложением? (Ответ: Фоновый режим, нет прямого UI, управление через SCM/systemd/init, запуск/остановка системой, часто работают от имени системы).

**Жизненный Цикл (Обобщенно):**Опишите основные этапы жизни службы/демона (установка/регистрация, запуск, выполнение работы, обработка команд/сигналов, остановка, удаление).

**II. Windows Services (Углубленно):**

**Взаимодействие с SCM**: Как именно программа сообщает SCM о своем состоянии? (SetServiceStatus). Как SCM передает команды? (Вызов ServiceCtrlHandler).

**Какова роль StartServiceCtrlDispatcher?** (Связь с SCM, запуск ServiceMain).

**Функции ServiceMain и ServiceCtrlHandler:**Каковы их основные задачи? Как они связаны? Может ли служба работать без одной из них? (Ответ: ServiceMain - инициализация и основной цикл, ServiceCtrlHandler - реакция на команды; обе нужны; ServiceMain запускает ServiceCtrlHandler через RegisterServiceCtrlHandler).

**Состояния Службы и Переходы:**Какие состояния \*\_PENDING вы знаете? Зачем они нужны? Что такое dwWaitHint и dwCheckPoint? (Ответ: Для информирования SCM о длительных операциях старта/остановки/паузы, чтобы SCM не посчитал службу зависшей).

**Контекст Безопасности:**От имени какой учетной записи обычно запускаются службы? (Local System). Какие у нее привилегии и ограничения? Можно ли запустить службу от имени другого пользователя? (Ответ: Local System - высокие привилегии на локальной машине, ограниченный доступ к сети; да, можно настроить при создании или в services.msc).

**Обработка Ошибок при Старте**: Как SCM узнает, что служба не смогла запуститься корректно? (Ответ: ServiceMain не вызвала SetServiceStatus(SERVICE\_RUNNING) в течение таймаута, или установила dwWin32ExitCode в статус SERVICE\_STOPPED).

**Управление** (API vs Утилиты): Какие функции WinAPI используются для программного управления службами (CreateService, StartService, ControlService и т.д.)? Какие стандартные утилиты командной строки есть для этого? (sc.exe, powershell).

**Реестр**: Какие еще параметры службы (кроме упомянутых в вопросах) можно найти/изменить в реестре? (Description, FailureActions, ImagePath, ObjectName и т.д.).

**III. Linux Daemons (Углубленно):**

**Процесс Демонизации** (Подробно): Зачем нужен каждый шаг: fork, setsid, chdir("/"), umask(0), закрытие дескрипторов? Почему иногда делают второй fork? (Ответ: fork - отсоединение от родителя, setsid - новая сессия без терминала, chdir - отвязка от файловой системы запуска, umask - установка нужных прав на создаваемые файлы, close - освобождение ресурсов и отвязка от терминала, второй fork - гарантия, что демон не станет лидером сессии и не сможет снова получить терминал).

**Сигналы** (Подробно): Разница между signal() и sigaction()? (Ответ: sigaction более надежный и портируемый). Почему нельзя делать сложные операции (как fprintf) внутри обработчика сигнала? Какие операции безопасны? (Ответ: Не реентерабельные функции могут вызвать проблемы; безопасно устанавливать флаги volatile sig\_atomic\_t, вызывать некоторые async-signal-safe функции). Как sleep() взаимодействует с сигналами? (Ответ: Прерывается, если не установлен флаг SA\_RESTART для sigaction).

**Systemd vs Init.d** (Сравнение): В чем принципиальные отличия в подходе к управлению демонами? (Ответ: systemd - декларативный (unit-файлы), параллельный запуск, управление через D-Bus, логирование через journald; init.d - процедурный (shell-скрипты), последовательный запуск, управление через сигналы/PID-файлы, логирование через syslog/файлы). Что означает Type=forking в systemd? (Ответ: Указывает systemd, что основной процесс завершится после порождения демона).

**PID-файлы:** Зачем они иногда используются (особенно с init.d или Type=forking в systemd)? Кто их создает/удаляет? (Ответ: Для хранения PID основного процесса демона, чтобы управляющие скрипты могли легко найти его для отправки сигналов; может создавать сам демон или управляющий скрипт/systemd).

**Стандартные Каталоги:**Почему конфигурацию принято хранить в /etc, логи в /var/log, PID-файлы в /run (или /var/run), а исполняемые файлы в /usr/sbin или /usr/local/sbin? (Ответ: Стандартная иерархия файловой системы Linux - FHS).

**Что такое Windows-сервис?**

Представь себе программу, которая должна работать постоянно в фоновом режиме, независимо от того, вошел ли пользователь в систему или нет. Например, антивирус, веб-сервер, служба обновлений или, как в нашем случае, программа для периодического резервного копирования файлов. Вот такие программы и оформляются в виде Windows-сервисов (или служб).

Ключевые особенности сервисов (из лекции, слайды 3-6):

* **Работают в фоне:**У них нет стандартного пользовательского интерфейса (окон, кнопок).
* **Управляются системой:** Запуск, остановка, управление ими осуществляется специальной программой – Менеджером Управления Службами (Service Control Manager - SCM). Ты взаимодействуешь с SCM через оснастку "Службы" (services.msc), утилиту командной строки sc.exe или программно через Windows API.
* **Жизненный цикл:** Могут запускаться автоматически при старте Windows, вручную пользователем (с нужными правами) или по требованию другого приложения. Работают до остановки или выключения системы.
* **Контекст безопасности:** Могут работать от имени специальных учетных записей (Local System, Network Service, Local Service) или указанной пользовательской учетной записи. Это определяет их права доступа к ресурсам.
* **Регистрация в системе:** Чтобы SCM "узнал" о сервисе, его нужно установить (зарегистрировать) в системе. Информация о нем хранится в реестре (HKEY\_LOCAL\_MACHINE\SYSTEM\CurrentControlSet\Services).

**Структура кода Windows-сервиса (Лекция, слайды 16-21):**

Обычно сервис – это консольное приложение, но со специальной структурой:

1. **main() функция:** Ее главная задача – не выполнять основную логику сервиса, а зарегистрировать одну или несколько служб, которые реализует это приложение, в SCM. Это делается с помощью функции **StartServiceCtrlDispatcher**(). Ей передается массив структур **SERVICE\_TABLE\_ENTRY**, где каждая структура связывает имя твоего сервиса с функцией, которая будет его точкой входа.
2. **ServiceMain() функция (точка входа сервиса):** Эта функция вызывается SCM после успешного вызова **StartServiceCtrlDispatcher**. Именно здесь начинается реальная работа сервиса. Основные шаги внутри ServiceMain:

* **Зарегистрировать обработчик команд:** Вызвать **RegisterServiceCtrlHandler**(), передав ей имя функции-обработчика (ServiceCtrlHandler). Этот обработчик будет принимать команды от SCM (стоп, пауза и т.д.).
* **Сообщить SCM о статусе:** Использовать **SetServiceStatus**(), чтобы информировать SCM о текущем состоянии сервиса (запускается, работает, останавливается и т.д.). Это важно, чтобы SCM не подумал, что сервис завис. Используется структура SERVICE\_STATUS.
* **Выполнить инициализацию:** Настроить логирование, прочитать конфигурацию, создать необходимые события синхронизации (например, событие для сигнала остановки).
* **Сообщить SCM о запуске:**Снова вызвать **SetServiceStatus**(), установив статус SERVICE\_RUNNING.
* **Основной цикл работы:**Выполнять главную задачу сервиса (в нашем случае – периодическое копирование). Этот цикл должен проверять, не пришла ли команда остановиться.
* **Завершение:** При получении команды на остановку – освободить ресурсы, завершить цикл и выйти из ServiceMain.

1. **ServiceCtrlHandler() функция (обработчик команд)**: Эта функция вызывается SCM, когда сервису нужно передать управляющую команду (стоп, пауза, продолжить, пользовательская команда). Она принимает код команды (dwControl). Внутри обычно стоит switch по этому коду:

* **SERVICE\_CONTROL\_STOP / SERVICE\_CONTROL\_SHUTDOWN**: Установить статус **SERVICE\_STOP\_PENDING**, дать сигнал основному циклу в ServiceMain на завершение (например, через **SetEvent**()).
* **SERVICE\_CONTROL\_PAUSE**: Установить статус **SERVICE\_PAUSE\_PENDING**, приостановить работу основного цикла, установить статус **SERVICE\_PAUSED**.
* **SERVICE\_CONTROL\_CONTINUE:** Установить статус **SERVICE\_CONTINUE\_PENDING**, возобновить работу основного цикла, установить статус SERVICE\_RUNNING.
* **Пользовательские команды (128-255):**Выполнить специфическое действие (в нашем случае – записать тестовое сообщение в лог).
* **SERVICE\_CONTROL\_INTERROGATE**: Просто сообщить текущий статус через SetServiceStatus().
* **SERVICE\_CONTROL\_PARAMCHANGE:** Перечитать конфигурационный файл (согласно требованию ЛР про сигнал SIGHUP/аналог).

**Точка входа (main):**

**Суть**: Это самая первая функция, которая выполняется при запуске DemoService.exe. Ее единственная задача — связать программу с Диспетчером Управления Службами (Service Control Manager, SCM) Windows.

**Алгоритм**:

* Определяется структура SERVICE\_TABLE\_ENTRY, которая говорит SCM: "Для службы с именем DemoService используй функцию ServiceMain как точку входа".
* Вызывается StartServiceCtrlDispatcher. Эта функция блокирует выполнение main и передает управление SCM. SCM найдет службу DemoService в таблице и вызовет ServiceMain в отдельном потоке.
* Перед этим вызывается InitializePaths для определения базовых путей (каталог exe и системный AppData), но логирование на этом этапе еще не настроено.

**Инициализация Путей (InitializePaths):**

**Суть**: Определить два важных базовых пути до начала основной работы службы.

**Алгоритм**:

* GetModuleFileName(NULL, ...): Получить полный путь к DemoService.exe.
* PathRemoveFileSpec: Убрать имя файла (DemoService.exe), оставив только путь к каталогу, где он лежит (executable\_dir). Именно здесь будет искаться config.ini.
* SHGetFolderPath(..., CSIDL\_APPDATA, ...): Получить путь к папке AppData\Roaming для той учетной записи, от имени которой запущена служба. По умолчанию это Local System, поэтому путь будет системным (например, C:\Windows\System32\config\systemprofile\AppData\Roaming), а не вашим пользовательским.
* PathCombine: Создать путь к базовому каталогу данных службы в системном AppData (например, ...\Roaming\DemoService). Сохранить его в base\_app\_data\_dir.

**Основная Логика Службы (ServiceMain):**

**Суть**: Это "сердце" службы. Здесь происходит инициализация, запуск основного цикла работы и очистка при остановке. SCM вызывает эту функцию после StartServiceCtrlDispatcher.

**Алгоритм** (Шаги Инициализации):

* Регистрация Обработчика Команд: RegisterServiceCtrlHandler сообщает SCM, какую функцию (ServiceCtrlHandler) нужно вызывать, когда службе приходят команды (стоп, пауза и т.д.).
* Установка Статуса START\_PENDING: Служба сообщает SCM, что она запускается, но еще не готова к работе (service\_status.dwCurrentState = SERVICE\_START\_PENDING). Устанавливаются флаги принимаемых команд (dwControlsAccepted). SetServiceStatus отправляет эту информацию SCM.
* Инициализация Путей: Снова вызывается InitializePaths (на всякий случай).
* Чтение Конфигурации (ReadConfig): Загружаются настройки из config.ini (находящегося рядом с exe). Определяются source\_dir, reserved\_dir, copy\_interval. Если config.ini нет или SourceDir не указан, устанавливается флаг критической ошибки.

**Настройка Логирования:**

* Создается путь к каталогу логов (log\_dir\_full) внутри системного base\_app\_data\_dir.
* Вызывается CreateOrCheckDir для создания/проверки этого каталога логов.
* Формируется имя лог-файла с датой и временем (log\_path).
* Открывается файл лога (CreateFile) и сохраняется его дескриптор (hLogFile). С этого момента возможно логирование.

**Настройка Каталога Резервирования:**

* Вызывается CreateOrCheckDir для создания/проверки каталога reserved\_dir (путь к которому был определен в ReadConfig).
* Проверка Исходного Каталога: GetFileAttributes проверяет, существует ли source\_dir (указанный в конфиге) и является ли он каталогом. Если нет - критическая ошибка.
* Проверка на Критические Ошибки: Если на предыдущих шагах был установлен флаг critical\_error, служба логирует ошибку, устанавливает статус SERVICE\_STOPPED и завершает работу ServiceMain.
* Успешный Запуск: Если ошибок не было, логируется сообщение об успешном запуске с параметрами. Статус службы устанавливается в SERVICE\_RUNNING.

**Алгоритм (Основной Цикл Работы):**

* while (service\_status.dwCurrentState == SERVICE\_RUNNING || service\_status.dwCurrentState == SERVICE\_PAUSED): Цикл продолжается, пока служба должна работать или приостановлена.
* if (service\_status.dwCurrentState == SERVICE\_RUNNING && !is\_paused): Если служба активна (не на паузе), вызывается функция CopyFiles().
* Sleep(copy\_interval \* 60 \* 1000): Служба "засыпает" на указанное в copy\_interval количество минут (переведенное в миллисекунды).

**Алгоритм (Завершение):**

* Когда цикл прерывается (например, получен сигнал СТОП), логируется сообщение о завершении.
* Закрывается дескриптор лог-файла (CloseHandle(hLogFile)).
* Функция ServiceMain завершается. SCM увидит, что поток службы завершился.

**Обработчик Управляющих Команд (ServiceCtrlHandler):**

**Суть**: Эта функция вызывается SCM (в отдельном потоке), когда пользователь или система отправляет команду службе (через services.msc, sc.exe и т.д.).

**Алгоритм (switch (dwControl)):**

* Определяется, какая команда (dwControl) пришла.
* STOP, SHUTDOWN: Установить статус SERVICE\_STOPPED. Закрыть лог-файл. (Это прервет цикл в ServiceMain).
* PAUSE: Если служба RUNNING, установить статус SERVICE\_PAUSED, установить флаг is\_paused = TRUE.
* CONTINUE: Если служба PAUSED, установить статус SERVICE\_RUNNING, установить флаг is\_paused = FALSE.
* PARAMCHANGE (код 6): Вызвать ReadConfig() для перезагрузки настроек. Проверить новые настройки на корректность (особенно SourceDir), если некорректно - установить флаг trigger\_stop. Пересоздать/проверить ReservedDir.
* Пользовательские Коды (например, 138): Выполнить специфическое действие (здесь - просто запись в лог).
* INTERROGATE: Ничего не делать, SCM просто запрашивает текущий статус.
* В конце: Логируется изменение состояния (если оно произошло). Вызывается SetServiceStatus, чтобы сообщить SCM новый статус службы (или подтвердить старый после INTERROGATE). Если был установлен trigger\_stop, статус меняется на STOPPED.

**Чтение Конфигурации (ReadConfig):**

**Суть**: Загрузить параметры из config.ini.

**Алгоритм**:

* Определить путь к config.ini (рядом с DemoService.exe в executable\_dir).
* Установить значения по умолчанию (интервал=60, SourceDir=пусто, ReservedDir=путь в системном AppData).
* Если config.ini существует:
* Прочитать SourceDir (GetPrivateProfileString). Если пустой - логировать ошибку.
* Прочитать ReservedDir. Если путь относительный - сделать его абсолютным относительно executable\_dir. Если не указан - использовать значение по умолчанию (если системный AppData доступен).
* Прочитать Interval (GetPrivateProfileInt). Проверить, что значение > 0.
* Залогировать финальные загруженные параметры.

**Копирование Файлов (CopyFiles):**

**Суть**: Выполнить основную работу - копирование.

**Алгоритм**:

* Проверить, что source\_dir и reserved\_dir заданы.
* Перечислить все файлы (не каталоги) в source\_dir (FindFirstFile/FindNextFile).
* Для каждого найденного файла:
* Сформировать полные пути источника и назначения.
* Вызвать CopyFile(..., FALSE) (FALSE означает перезапись, если файл назначения существует).
* Залогировать успех (LogSuccess) или ошибку (LogError) копирования для этого файла.
* Залогировать завершение цикла копирования.

**Создание/Проверка Каталога** (CreateOrCheckDir):

**Суть**: Убедиться, что каталог существует, и создать его при необходимости.

**Алгоритм**:

* GetFileAttributes: Проверить, существует ли путь.
* Если не существует (INVALID\_FILE\_ATTRIBUTES): Попробовать создать (\_mkdir). Залогировать успех или ошибку создания (LogSuccess/LogError).
* Если существует и это каталог (FILE\_ATTRIBUTE\_DIRECTORY): Залогировать успешное обнаружение (LogSuccess).
* Если существует, но не каталог: Залогировать ошибку (LogError).

**Логирование** (WriteLog, LogSuccess, LogError, LogInfo):

**Суть**: Записывать информацию о работе службы в файл.

**Алгоритм**:

* WriteLog: Базовая функция. Проверяет, открыт ли файл лога (hLogFile). Если нет (или закрылся) - пытается открыть его снова по пути log\_path. Добавляет метку времени HH:MM:SS к переданному сообщению. Записывает строку в файл (WriteFile). Сбрасывает буфер (FlushFileBuffers).
* Остальные функции (LogSuccess, LogError, LogInfo) просто форматируют строку с нужным префиксом ("Успех! ", "Попытка... провалена! ", или без префикса) и передают ее в WriteLog.
* В итоге: Служба запускается SCM, инициализируется (читает конфиг, настраивает логи и каталоги), входит в цикл ожидания/копирования, реагирует на команды управления (стоп, пауза, перезагрузка конфига) и логирует свои действия в системную папку AppData. Конфигурация при этом читается из файла рядом с исполняемым файлом.