**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

**ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ**

Кафедра информационных систем управления

ЦАРИК ВИТАЛИЙ АЛЕКСАНДРОВИЧ

**СРАВНИТЕЛЬНЫЙ АНАЛИЗ СЕРВЕРНЫХ ПРИЛОЖЕНИЙ (СЛУЖБ/ДЕМОНОВ) В ОС WINDOWS И UNIX-LINUX**

Курсовой проект

студента 3 курса 2 группы

**Руководитель:**

*Безверхий Александр Анатольевич*

старший преподаватель кафедры ИСУ

Минск, 2019

**АННОТАЦИЯ**

Царик В.А. Сравнительный анализ серверных приложений в ОС Windows и Unix-Linux: Курсовой проект / Минск: БГУ, 2019. – 23 с.

В курсовом проекте рассмотрен вопрос разработки серверных приложений в операционных системах Unix-Linux и Windows, а также способы управления этими приложениями. Для сравнительного анализа служб и демонов были разработаны приложения.

**АНАТАЦЫЯ**

Царык В.А. Параўнальны аналіз серверных прыкладанняў у АС Unix-Linux і Windows: Курсавы праэкт / Мінск: БДУ, 2019. - 23 с.

У курсавым праекце разгледжана пытанне распрацоўкі серверных прыкладанняў у аперацыйных сістэмах Unix-Linux і Windows, а таксама спосабы кіравання гэтымі прыкладаннямі. Для параўнальнага аналізу службаў і дэманаў былі распрацаваны праграмы.

**ANNOTATION**

Tsaryk V.A. Comparative analysis of server applications in OS Unix-Linux and Windows: Course project /Minsk: BSU, 2019 – 23 p.

In the course project the question of developing server applications in Unix-Linux and Windows is raised, as well as ways to manage these applications. The application was developed to demonstrate the approaches to development of services and daemons.

**РЕФЕРАТ**

Курсовой проект, 29 с., 3 рис., 2 табл., 6 источников, 4 приложения.

**Ключевые слова:** WINDOWS, LINUX, СЛУЖБА, ДЕМОН

**Объект исследования**: концепция модели клиент-сервер в современных операционных системах.

**Цель работы:** рассмотреть особенности реализации серверных приложений в ОС Unix-Linux и Windows. Провести сравнительный анализ и выявить их общие черты и различия. Разработать один и тот же практический пример приложений-серверов, демонстрирующие работу служб в Windows и демонов в Unix-Linux.

**Методы исследования**: изучение учебного материала, практическое применение полученных знаний, поиск информации, анализ.

**Область применения:** разработка операционных систем или программ, разработка распределённых систем, разработка эффективных приложений клиент-сервер.

**ОГЛАВЛЕНИЕ**

ВВЕДЕНИЕ 5

1 КОНЦЕПЦИЯ МОДЕЛИ КЛИЕНТ-СЕРВЕР 6

1.1 Клиенты и серверы 6

1.2 Разделение приложений по уровням 6

1.3 Некоторые особенности серверов 6

2 ОСОБЕННОСТИ ПРИЛОЖЕНИЙ-СЛУЖБ В ОС WINDOWS 7

2.1 Основные понятия и определения 7

2.2 Принципы работы служб Windows 7

2.3 Структура приложений-служб Windows 8

2.4 Функции, необходимые для реализации исполняемого файла 9

2.4.1 Основная функция процесса: (w)main или (w)WinMain 9

2.4.2 Функция ServiceMain 10

2.4.3 Функция HandlerEx 11

2.5 Управляющие коды и оповещения о состоянии 12

3 ОСОБЕННОСТИ ПРИЛОЖЕНИЙ-ДЕМОНОВ В ОС UNIX-LINUX 14

3.1 Основные понятия и определения 14

3.2 Создание демона 14

3.3 Функция daemon 15

4 СРАВНИТЕЛЬНЫЙ АНАЛИЗ СЛУЖБ И ДЕМОНОВ НА ОСНОВЕ РАЗРАБОТАННОЙ ПРОГРАММЫ 16

4.1 Постановка задачи 16

4.2 Сравнительный анализ реализации служб и демонов 16

4.3 Запуск приложений-служб и приложений-демонов 17

4.4 Реализация автозагрузки служб и демонов 17

4.5 Передача параметров службам и демонам 18

4.6 Зависимость между службами и демонами 18

4.7 Обработка сигналов службами и демонами 19

4.8 Возможность удалённого администрирования 19

ЗАКЛЮЧЕНИЕ 20

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 21

ПРИЛОЖЕНИЕ А 22

ПРИЛОЖЕНИЕ Б 23

**ВВЕДЕНИЕ**

Развитие технологий привело к появлению распределённых систем. Распределенная система — это набор независимых компьютеров, представляющийся их пользователям единой объединенной системой. Преимуществами такой системы являются:

- разделение задачи между компьютерами, что повысит скорость её выполнения;

- компьютеры системы могут быть расположены очень далеко друг от друга;

- пользователь может даже не подозревать о существовании других компьютеров, кроме своего;

- такие системы, как правило хорошо масштабируемы, то есть можно добавить в систему ещё несколько новых компьютеров;

- система упрощают интеграцию различных приложений, работающих на разных компьютерах, в единую систему.

Таким образом, эта система позволяет объединять компьютеры по всему миру и эффективно выполнять различные задачи. Самым известным примером такой системы является Интернет.

Для каждой распределенной системы важна схема ее внутренней организации. Широко применяется модель клиент-сервер, в которой клиент посылает серверу запрос, сервер его обрабатывает и, как правило, посылает результат клиенту. Обычно приложение-сервер берёт на себя большую часть программы, а приложение-клиент лишь предоставляет пользователю интерфейс программы. Но по причине того, что к одному серверу может подключиться много клиентов или обработка запроса может требовать больших вычислительных затрат, очень важным является вопрос создания быстрых сервер-приложений. При аппаратном решении этой проблемы это приводит к тому, что приложения-серверы часто работают на суперкомпьютерах, которые могут располагаться в отдельных комнатах. Это приводит к трудностям доступа к этим компьютерам и, следовательно, поднимает вопрос об удалённом администрировании этих серверных машин. При программном же решении этой проблемы в первую очередь возникает вопрос, на какой же операционной системе реализовать серверное приложение, что и обосновывает актуальность сравнительного анализа служб и демонов.

**1 КОНЦЕПЦИЯ МОДЕЛИ КЛИЕНТ-СЕРВЕР**

* 1. **Клиенты и серверы**

В базовой модели клиент-сервер все процессы в распределенных системах делятся на две группы. Процессы, реализующие некоторую службу, например, службу файловой системы или базы данных, называются серверами (servers). Процессы, запрашивающие службы у серверов путем посылки запроса и последующего ожидания ответа от сервера, называются клиентами (clients).

Клиент, запрашивая службу, облекает свой запрос в форму сообщения с указанием в нем службы, которой он желает воспользоваться, и необходимых для этого исходных данных. Затем сообщение посылается серверу. Последний, в свою очередь, постоянно ожидает входящего сообщения, получив его, обрабатывает, упаковывает результат обработки в ответное сообщение и отправляет его клиенту.

* 1. **Разделение приложений по уровням**

При разработке модели клиент-сервер возникает вопрос о том, как точно разделить обязанности между клиентом и сервером. Предложено разделение приложений типа клиент-сервер на 3 уровня:

- уровень пользовательского интерфейса;

- уровень обработки;

- уровень данных.

* 1. **Некоторые особенности серверов**

Большинство серверов работает на серверных машинах, расположенных в закрытых помещениях, и, следовательно, в этом случае, если серверное приложение попытается сообщить об ошибке или вывести на экран некоторое сообщение, некому будет реагировать на эту информацию. Также поскольку серверные приложение по своей сути находятся в бесконечном цикле в ожидании входящих запросов от клиента, то им не нужен пользовательский интерфейс. Поэтому в целях экономии ресурсов серверные приложения запускаются в фоновом режиме. Таким образом эти приложения в ОС Unix-Linux реализуются как демоны, а в ОС Windows – как службы. Подробнее реализация таких серверов будет приведена в главах 2 и 3.

**2 ОСОБЕННОСТИ ПРИЛОЖЕНИЙ-СЛУЖБ В ОС WINDOWS**

**2.1 Основные понятия и определения**

Операционная система Microsoft Windows предоставляет различные средства, которые упрощают реализацию серверной части клиент-серверных приложений. Microsoft рекомендует реализовывать серверные приложения в виде служб.

Служба – это обычное Windows-приложение, содержащее дополнительную инфраструктуру, которая позволяет ОС рассматривать его особым образом, например, давать возможность администратору запускать и останавливать это приложение с удалённой машины.

У службы не должно быть пользовательского интерфейса. Тогда как настраивать и управлять службой? Всё это позволяет сделать удалённое администрирование. В Windows есть множество утилит для управления службами с других машин, связанных по сети, что позволяет администратору не присутствовать реально за компьютером, на котором работает служба. Примерами таких утилит служат sc и Services.exe.

**2.2 Принципы работы служб Windows**

Работа служб обеспечивается взаимодействием компонент трёх типов:

- диспетчер управления службами (Service Control Manager, SCM). Этот компонент хранится в файле Services.exe и автоматически вызывается при запуске ОС и прерывается при выходе из неё. SCM имеет привилегии системы и обеспечивает унифицированный и безопасный метод управления приложениями-службами. SCM отвечает за взаимодействие с различными службами, указывая им, когда запуститься, остановиться, временно прервать и продолжить работу;

- приложение-служба. Это просто приложение, содержащее инфраструктуру для взаимодействия с SCM, который посылает службе команды запуска, останова, паузы, продолжения или выключения. Служба также вызывает специальные функции, передающие SCM сведения о её состоянии;

- программа управления службой (Service Control Program, SCP). Это приложение, которое обычно имеет пользовательский интерфейс, позволяющий пользователю запустить, остановить, приостановить, продолжить работу и выполнить другие функции управления службами, установленными на машине. SCP вызывает специальные функции Windows, позволяющие ей общаться с SCM. На рисунке 2.1 покахано, как происходит взаимодействие между этими компонентами. RPC – удалённый вызов процедур.

**Рисунок 2.1 – Структурная схема обмена информацией со службами Windows**

SCP взаимодействуют со службами не напрямую, а через SCM. Такая архитектура делает удалённое администирование прозрачным для SCP и приложений-служб.

**2.3 Структура приложений-служб Windows**

Для создания приложения-службы нужно реализовать структуру приложений-служб Windows. В приложении А приведена эта структура для случая нахождения двух служб в одной исполняемом файле. Из этой структуры видно, что каждый процесс службы всегда имеет хотя бы два потока и потоки эти должны взаимодействовать между собой, то есть имеем дело с синхронизацией потоков и обменом информацией между ними. Так же видим, что в одном исполняемом файле может находится несколько служб.

Из приложения А можно выделить следующие основные виды функций:

- основная функция процесса – стандратная функция (w)main или (w)WinMain, с которой и начинается выполнение исполняемого файла. В службах она инициализирует процесс в целом и затем вызывает специальные функции, которые связывают процесс с SCM локальной системы. Теперь SCM управляет основным потоком для собственных нужд. Код снова получит управление, только когда все службы из исполняемого файла прекратят работу.

- функция ServiceMain службы – это функция, которую нужно реализовать для каждой службы исполняемого файла. Чтобы запустить службу, SCM порождает в вашем процессе поток, который выполняет функцию ServiceMain. Когда управление в потоке возвращается из ServiceMain, SCM решает, что служба остановлена. Название этой функции может быть любым.

- функция HandlerEx службы – каждая служба должна иметь связанную с ней функцию HandlerEx. SCM взаимодействует со службой, вызывая её функцию HandlerEx. Её код выполняется в основном потоке процесса. HandleEx или выполняет нужные действия, или, применяя ту или иную форму межпотокового взаимодействия, должна взаимодействовать с той частью потока, управляемого SCM, который выполняет функцию ServiceMain службы. Каждая служба может иметь собственную HandlerEx или несколько служб в одном исполняемом файле могут совместно использовать единственную HandlerEx. Для определения службы, с которой хочет взаимодействовать SCM, используется один из параметров HandlerEx. Название может быть любым.

В Приложении А. кроме этих функций также показаны пути межпроцессного взаимодействия(IPC, interprocess communication) и межпотокового взаимодействия (ITC, interthread communication).

**2.4 Функции, необходимые для реализации исполняемого файла**

**2.4.1 Основная функция процесса: (w)main или (w)WinMain**

Когда администратор хочет запустить службу, SCM определяет, запущен ли исполняемый файл, содержащий эту службу. Если нет, то SCM запускает его. Основной поток процесса отвечает за инициализацию процесса в целом, а не конкретной службы.

После инициализации процесса его основная функция должна связаться с SCM, который теперь управляет процессом. Для этого основная функция процесса должна прежде всего выделить и проинициализировать массив структур SERVICE\_TABLE\_ENTRY. В каждом элементе этой структуры хранятся внутреннее имя службы и адресс функции ServiceMain, котороя соответсвует этой службе. Каждый элемент этого массива должен соответсвовать какой-то службе из этого исполняемого файла. Последним элементом этого массива является структура с установленными именем и адресом в NULL, что указывает на конец массива.

Далее процесс подсоединяет себя к SCM, вызывая функцию StartServiceCtrlDispatcher:

BOOL StartServiceCtrlDispatcher(CONST SERVICE\_TABLE\_ENTRY\* pServiceTable);

С помощью этой функции исполняемый процесс указывает, какие службы в нём содержаться. Благодаря этому SCM может определить, какие службы он должен пытаться запустить, последовательно просматривая данный массив. Когда он находит службу, для неё создаётся поток и начинается выполнение функции ServiceMain этой службы.

StartServiceCtrlDispatcher не возвращает управление, пока не прекратят работу все службы, содержащиеся в исполняемом модуле. Обычно этот поток просто находится в состоянии ожидания, не расходуя процессорного времени. Если администратор пытается запустить другую службу, реализованную в том же исполняемом модуле, SCM не порождает нового экземпляра исполняемого модуля, а связывается с основным потоком существующего исполняемого модуля и вновь просматривает список служб, в поисках службы, которую нужно запустить. Найдя её, он порождает новый поток, выполняющий функцию ServiceMain соответствующей службы.

Когда какая-либо служба заканчивает работу, система проверяет, остались ли ещё работающие службы. Если нет, то происходит возврат из StartServiceCtrlDispatcher. Затем программа должна освободить основные ресурсы всего процесса и выполнить возврат из основной функции процесса. Если этого не будет сделано за 30 секунд, то SCM уничножит процесс, что может привести к утечке ресурсов.

**2.4.2 Функция ServiceMain**

Для каждой службы в исполняемом файле должен быть определёна функция ServiceMain:

VOID WINAPI ServiceMain(DWORD dwArgc, PTSTR\* pszArgv);

SCM запускает службу, создавая новый поток, который начинает своё выполнение с вызова ServiceMain. В неё есть два параметра. Они позволяют администратору запустить службу с указанием параметров командной строки.

Первое, что должна сделать ServiceMain, - это указать SCM адрес функции обратного вызова HandlerEx службы. Это делается с помощью функции:

SERVICE\_STATUS\_HANDLE RegisterServiceCtrlHandlerEx(

PCTSTR pszServiceName,

LPHANDLER\_FUNCTION\_EX pfnHandler,

PVOID pvContext);

Первый параметр указывает службу, для которой устанавливается функция HandlerEx, а второй - это адрес HandlerEx. Третий параметр – это определяемое пользователем значение, которое передаётся HandlerEx.

Эта функция возвращает описатель SERVICE\_STATUS\_HANDLE – уникальный индетификатор службы, используемый SCM. Всё дальнейшее взаимодействие службы с SCM производится с использованием этого описателя. Этот описатель никогда не должен закрывать пользователем.

Далее поток ServiceMain должен сразу уведомить SCM, что служба продолжает инициироваться. Это должно быть сделано не позднее 80 секунд после запуска, иначе SCM будет считать, что служба не может запуститься и, если в исполняемом модуле нет работающих служб, SCM уничтожит процесс.

Уведомление SCM делает функция:

BOOL SetServiceStatus(

SERVICE\_STATUS\_HANDLE hService,

LPSERVICE\_STATUS pServiceStatus);

Первым параметром её является описатель службы, а второй параметр – это адрес структуры SERVICE\_STATUS, состоящий из семи элементов, которые отражают текущее состояние службы:

* тип исполняемого модуля – одна служба на весь модуль или несколько;
* текущее состояние службы – служба запускается, запущена, останавливается, остановлена, приостанавливается, приостановлена, продолжается;
* какие управляющие сигналы может воспринимать служба – может ли быть приостановлена, продолжена и тому подобное.

Также здесь предусмотрены элементы для выдачи кодов ошибок и для уведомления о ходе своей работы.

После инициализации служба должна снова вызвать функцию SetServiceStatus, указав состояние службы SERVICE\_RUNNING. Теперь служба может работать.

**2.4.3 Функция HandlerEx**

Каждая служба, содержащая в исполняемом модуле, должна иметь связанную с ней функцию HandlerEx:

DWORD WINAPI HandlerEx(

DWORD dwControl,

DWORD dwEventType,

DWORD pvEventData,

PVOID pvContext);

Большую часть времени основной поток процесса находится в состоянии ожидания в результате обращения к функции StartServiceCtrlDispatcher. Когда SCP требуется управление службой, SCM связывается с основным потоком процесса, в результате чего этот поток разблокируется и вызывает функцию HandlerEx соответствующей службы.

Следует реализовывать HandlerEx так, чтобы он работал как можно быстрее, чтобы поскорее освободился основной поток и он снова мог принимать управляющие коды от SCP.

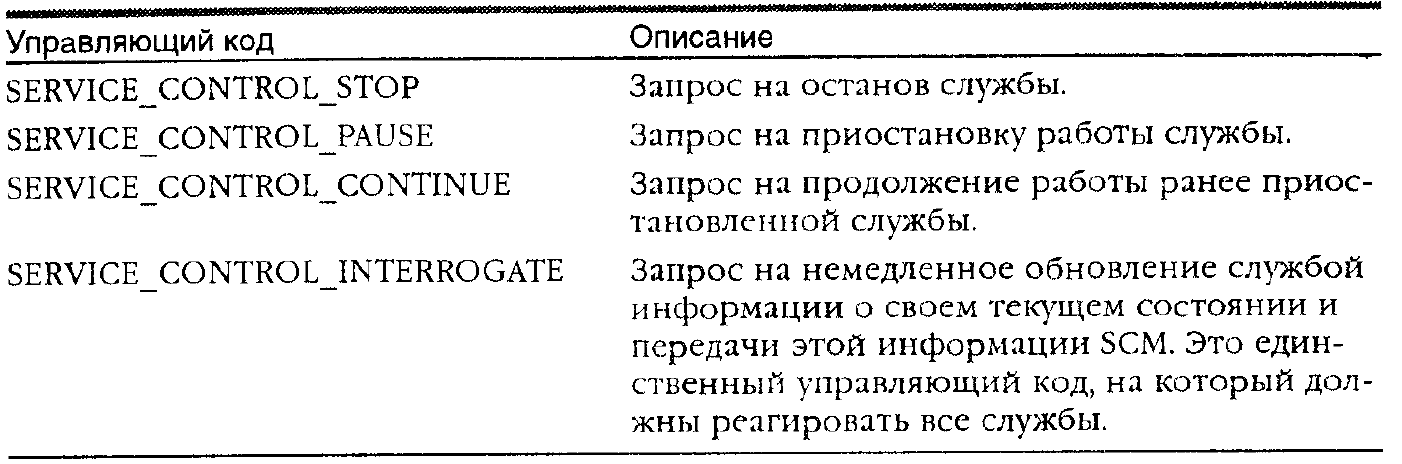
Поскольку HandlerEx исполняется основным потоком, а служба – другим, может возникнуть необходимость взаимодействия HandlerEx с потоком службы. Стандартного способа такого взаимодействия нет, поэтому сам программист должен решать эту проблему. Например, в моей практической части курсового проекта это достигнуто использованием порта завершения ввода-вывода.

В HandlerEx передаются четыре параметра. Первый указывает запрашиваемую операцию или уведомление. Параметры dwEventType и pvEventData содержат более конкретную информацию об операции или уведомлении. Последний параметр – это определяемое пользователем значение, которое изначально передавалось функции RegisterServiceCtrlHandlerEx. Используя его, можно выбрать конкретную службу, с которой должна взаимодействовать HandlerEx, если HandlerEx был зарегистрирован для нескольких служб.

Значение, возвращаемое HandlerEx, позволяет службе вернуть данные SCM. Если HandlerEx не обрабатывает некоторый управляющий код, то нужно вернуть значение ERROR\_CALL\_NOT\_IMPLEMENTED. Для всех других управляющих кодов HandlerEx должна возвращать NO\_ERROR.

**2.5 Управляющие коды и оповещения о состоянии**

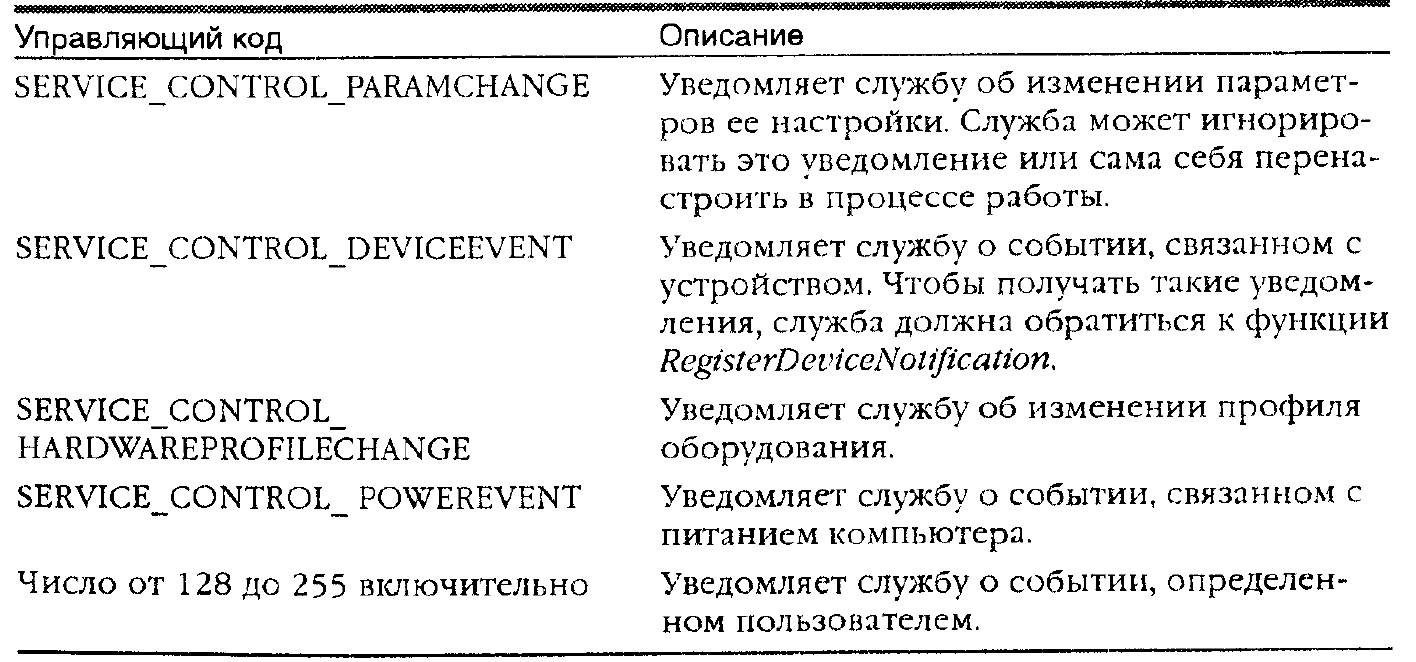
В функции HandlerEx первый параметр – это код, указывающий запрашиваемую операцию или уведомление. В Таблице 2.1 приведены коды, указывающие запросы операций.



**Таблица 2.1 – Управляющие коды, указывающие запросы операций**

При получении этих кодов HandlerEx должен вызвать функцию SetServiceStatus, чтобы подтвердить получение кода и указать, сколько времени может потребоваться службе для изменения своего состояния на запрашиваемое. Кроме того, HandlerEx должна вернуть управление в течении 30 секунд, иначе SCP-приложение сочтёт, что служба не отвечает. Тогда SCM не уничтожает, а лишь возвращает ошибку SCP.

В Таблице 2.2 приведены управляющие коды, указывающие уведомления.



**Таблица 2.2 – Управляющие коды, указывающие уведомления**

Получив коды уведомления, HandlerEx должна их обработать и вернуть управление. Вызывать SetServiceStatus здесть не требуется. Так же из Таблицы 2.2 видно, что можно определять свои уведомления, что может оказаться очень полезным при разработке служб.

**3 ОСОБЕННОСТИ ПРИЛОЖЕНИЙ-ДЕМОНОВ В ОС UNIX-LINUX**

**3.1 Основные понятия и определения**

Демоном называется процесс, который запущен в фоновом режиме и не привязан ни к какому управляющему терминалу. Демоны обычно запускаются во время загрузки с правами root или другими специфическими пользовательскими правами (например, apache или postfix) и выполняют задачи системного уровня.

Из определения ясно, что две обязательные особенности демона таковы: он должен запускаться как потомок процесса init и не должен быть связан с терминалом.

Как правило, демоны занимаются такими вещами, как сетевые запросы, работой аппаратного обеспечения и прочими заданиями типа «жди, получи запрос, обработай, жди».

**3.2 Создание демона**

Демоны появляются двумя способами. Первый заключается в том, что их может создать процесс init. Второй же способ подразумевает выполнение некоторой последовательности шагов.

Традиционно демоны реализуются по схеме, возникшей в SysV Unix. Когда запускается традиционный демон SysV, он должен выполнить следующие шаги как часть инициализации.

- закройте все дескрипторы открытого файла, кроме стандартного ввода, вывода и вывода ошибок. Это гарантирует, что ни один случайно переданный файловый дескриптор не останется в процессе демона;

- сбросьте все обработчики сигналов на их значения по умолчанию;

- сбросьте маску сигнала;

- очистите блок среды, удалив или сбросив переменные среды, которые могут оказать негативное влияние во время выполнения демона;

- вызовите fork(), чтобы создать фоновый процесс;

- у ребёнка вызовите setsid(), чтобы отсоединиться от любого терминала и создать независимый сеанс с лидером ребёнком;

- в дочернем файле вызовите fork() снова, чтобы демон никогда больше не мог повторно получить терминал;

- вызовите exit() первого потомка, чтобы остался только второй потомок (фактический процесс демона). Это гарантирует, что родителем демона станет init с pid 1, как и у всех демонов;

- в процессе демона подключите /dev/null к стандартному вводу, выводу и выводу ошибок.

- в демоне, сбросить umask на 0;

- в демоне измените текущий каталог на корневой каталог (/), чтобы избежать случаев, когда демон невольно блокирует точки монтирования от размонтирования;

- в демоне, запишите pid демона в pid-файл, чтобы гарантировать, что демон не может быть запущен более одного раза;

- в процессе демона удалите привилегии, если это возможно;

- из процесса-демона уведомите исходный процесс о завершении инициализации. Это может быть реализовано через безымянный канал или аналогичный канал связи, который создается до первого fork() и, следовательно, доступен как в исходном процессе, так и в процессе демона;

- вызовите exit() в оригинальном процессе. Процесс, который вызвал демон, должен иметь возможность полагаться на то, что это exit() происходит после завершения инициализации.

Демон, который должен обеспечивать совместимость с системами SysV, должен реализовывать схему, указанную выше. Однако рекомендуется сделать это поведение необязательным и настраиваемым с помощью аргумента командной строки, чтобы упростить отладку, а также упростить интеграцию в системы, использующие systemd.

Практически все эти шаги были реализованы мной в ходе выполнения курсового проекта и с этой реализацией можно ознакомиться в приложении Б.

**3.3 Функция daemon**

Большинство систем UNIX предоставляют функцию daemon():

#include <unistd.h>

int daemon (int nochdir, int noclose);

Если nochdir не равен нулю, демон не изменяет свою рабочую директорию на root. Если noclose не равен нулю, демон не закроет все открытые дескрипторы файлов.

**4 СРАВНИТЕЛЬНЫЙ АНАЛИЗ СЛУЖБ И ДЕМОНОВ НА ОСНОВЕ РАЗРАБОТАННОЙ ПРОГРАММЫ**

**4.1 Постановка задачи**

В дальнейшем я буду ссылаться на результаты разработанного мной приложения. Суть которого состоит в следующем:

Служба/Демон отслеживает процесс, с заданным именем, и логирует информацию о нём: запущен ли процесс, текущий pid, суммарное время работы и др.

**4.2 Сравнительный анализ реализации служб и демонов**

Из главы 2 и приложения А имеем, что для реализации службы необходимо реализовать три функции: основную функцию исполняемого файла, ServiceMain, HandlerEx. В первой функции мы сообщаем SCM о том какие службы есть в данном исполняемом файле и какие им соответствуют функции ServiceMain, в которых и происходит работа службы. HandlerEx же служит для обработки сигналов, поступающих от SCP. При этом известно, что ServiceMain и HandlerEx выполняются в разных потоках, то есть нужно дополнительно осуществлять межпотоковое взаимодействие.

В свою очередь из главы 3 имеем, что для реализации демона нужно лишь выполнить некоторую последовательность шагов, причём не обязательно выполнять их все. Из приложения Б видно также, что большинство шагов являются шаблонными, и их можно смело выполнять для любых приложений-демонов. Например, закрытие файловых дескрипторов, сброс описателей сигналов, маски сигналов, изменение текущего каталога, fork(), exit для родителей демона.

Таким образом в ходе проделанной мной работы были получены следующие результаты:

- службы и демоны оба работают в фоновом режиме и не имеют пользовательского интерфейса;

- реализовать приложение-службу сложнее, чем приложение-демон, по причине организации межпотокового взаимодействия и организации обязательной обработки некоторый управляющих кодов в службе. При этом во время обработки этих кодов нужно не забывать постоянно сообщать SCM о состоянии службы, а также необходимо учитывать временные ограничения, которые даёт SCM на обработку или инициализацию, хотя их и можно увеличивать с помощью вызовов функции SetServiceStatus. В то же время код, отведенный для выполнения шагов инициализации демона, может вообще свестись к выполнению одной команды daemon;

- запуск приложений-демонов существенно отличается от запуска приложений-служб;

- приложения-службы предоставляют более мощный инструмент для удалённого администрирования, по причине того, что запуск служб и управление ими осуществляется через специальный процесс SCM, в то время как демон – это просто процесс, который потерял родителя и управляющий терминал.

**4.3 Запуск приложений-служб и приложений-демонов**

Для запуска демона достаточно запустить его исполняемый файл в терминале. Можно также написать скрипт, запустить который можно через тот же терминал, в котором можно записать команду запуска демона.

Что же касается запуска служб, то перед его запуском нужно добавить эту службу в базу данных SCM, чтобы он знал о существовании этой службы. Внести в базу данных SCM службу можно несколькими способами, начиная от программы на C/C++ и заканчивая использованием различных утилит.

На С/С++ этого можно достичь с помощью следующей последовательности команд: сначала вызываем OpenSCManager, затем CreateService. Первая команда подсоединяется к базе данных SCM и указывает с какой целью она это делает. Например, можно захотеть прочитать какую-то запись в базе данных, удалить запись, создать новую. Вторая же команда создаёт новую службу и помещает её в эту базу данных. В параметрах CreateService указывается внутреннее и отображаемое имена службы, адрес, где лежит исполняемый файл этой службы, уточняется, одна или несколько служб в файле, зависимости и так далее.

С помощью утилиты sc этого можно достичь с помощью следующей команды в командной строке:

sc <сервер> create [имя службы] [binPath= ] <параметр1> <параметр2>, где в параметрах могут быть указаны тип исполняемого файла, зависимости и так далее. binPath – адрес исполняемого файла.

После внесения службы в базу данных, её можно запускать. Это можно сделать, например, с помощью всё той же утилиты sc:

sc <сервер> start [имя службы] <арг1> <арг2> …

**4.4 Реализация автозагрузки служб и демонов**

Для реализации этой задачи для службы обратимся к её свойствам, указанных в базе данных SCM. Там есть специальный параметр, который отвечает за способ запуска службы. Возможные его значения: вручную, автоматически и отключена. Таким образом, для организации автозагрузки службы достаточно изменить тип запуска на автоматически. Это можно сделать с помощью различных утилит, например, sc или Services.exe, или с помощью функции на С/С++ ChangeServiceConfig2. В результате, во время запуска ОС SCM пройдёт по всем службам в своей базе данных и запустит те, у который тип запуска стоит автоматически.

Реализация этой задачи для демонов оказалась более трудной, чем для служб, но не невозможной. Можно, например, написать специальный скрипт для инициализации демона, который будет исполняться при запуске системы.

**4.5 Передача параметров службам и демонам**

Передача параметров службам и демонам при запуске схожа. Они задаются, как параметры командной строки и обрабатываются также. Отличие лишь в том, что службы запускают немного иначе. Например, в случае утилиты sc параметры задаются через арг1, арг2 и так далее.

**4.6 Зависимость между службами и демонами**

Из-за подхода к реализации служб организация зависимости между ними сводится лишь к изменению соответствующего параметра в свойствах этих служб. Зависимая служба должна добавить в свой список зависимостей ту службу, от которой она зависит и SCM дальше обо всём позаботиться сам. При запуске некоторой службы SCM пройдёт по списку её зависимостей и в случае, если какая-то служба из списка не запущена, SCM запустит её. При попытке завершения службы SCM просматривает список служб, которые зависят от неё. В случае, если в этом списке найдётся служба, которая ещё работает, SCM не даст завершиться исходной службе, но предоставит список тех служб, которые мешают её завершению и предложит вариант завершения их всех, то есть и исходной службы и всех работающих зависимых от неё служб.

Изменить список зависимостей службы можно, например, с помощью утилиты sc, или программно с помощью функции С/С++ ChangeServiceConfig2.

В Linux такой поддержки нет и зависимость создавать приходится самому. Как вариант, можно использовать pid-файлы, которые создают демоны при инициализации. Перед завершением инициализации демон просматривает pid-файлы тех демонов, от которых он зависит, и проверяет работают ли они сейчас. Если какой-то демон не запущен, то происходит его запуск. При завершении демона он просматривает pid-файлы демонов, которые зависят от него, и все запущенные из них завершают свою работу.

**4.7 Обработка сигналов службами и демонами**

При создании служб необходимо обязательно реализовывать обработку некоторый управляющих кодов. Замечу лишь то, что кроме стандартных кодов можно обрабатывать и коды, определённые разработчиком, что даёт некоторые дополнительные возможности при реализации приложений-служб.

В Linux для достижения чего-то подобного я использовал механизмы обработки сигналов процессов. Мной в качестве был написан свой обработчик для сигнала SIGTERM для более хорошего завершения моего демона.

**4.8** **Возможность удалённого администрирования**

Windows предоставляет возможность подключаться удалённо к SCM другого компьютера, что означает, что мы можем с лёгкостью управлять службами на другим компьютерах, соединённых в единую сеть, если имеем соответствующие права доступа. В Linux подобного инструмента мной обнаружено не было.

**ЗАКЛЮЧЕНИЕ**

В ходе данного курсового проекта:

- были получены новые знания по следующим темам: разработка служб в ОС Windows и способы управления ими, разработка демонов в ОС Unix-Linux и возможные решения для организации управления ими;

- также при изучении литературы и разработке практической части были получены дополнительные знания по темам: порты завершения ввода-вывода в Windows, организация и использование именованных каналов в ОС Windows и ОC Unix-Linux, работа со скриптами в Linux, механизмы обработки сигналов в ОС Unix-Linux.

- были проанализированы и описаны общие черты и особенности исследуемой области в рассматриваемых системах;

- был подготовлен пример приложения, демонстрирующий создание и работу серверных приложений в обеих из рассматриваемых систем.

Можно сформулировать общие черты подходов к созданию серверных приложений в Windows и Unix-Linux. Демоны и службы работают в фоновом режиме и не нуждаются в пользовательском интерфейсе.

В Unix системах создание демона – более простая задача, нежели в Windows. Реализация демонов предоставляет больше свободы. В то же время сложность реализации в Windows окупается предоставлением мощного инструмента управления службами, который позволяет без особых затруднений запускать, останавливать, приостанавливать службы. Также службы легко делаются автозапускаемыми, легко организуется зависимость между службами и управление службой с удалённого компьютера. Для достижения подобного Linux необходимо реализовывать эти инструменты разработчику вручную.**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. Таненбаум Э., М. ван Стеен. Распределённые системы. Принципы и парадигмы, СПб, 2003 г.
2. Рихтер Дж., Кларк Дж. Д., Программирование серверных приложений для Microsoft Windows 2000, СПб, 2001г.
3. Лав Р., Linux. Системное программирование. 2-ое изд., СПб, 2014г.
4. Вахалия Ю. Unix изнутри - 2003 г.
5. Стивенс У.Р., Феннер Б., Рудофф Э. М., UNIX: разработка сетевых приложений. 3-е изд., СПб., 2007г.

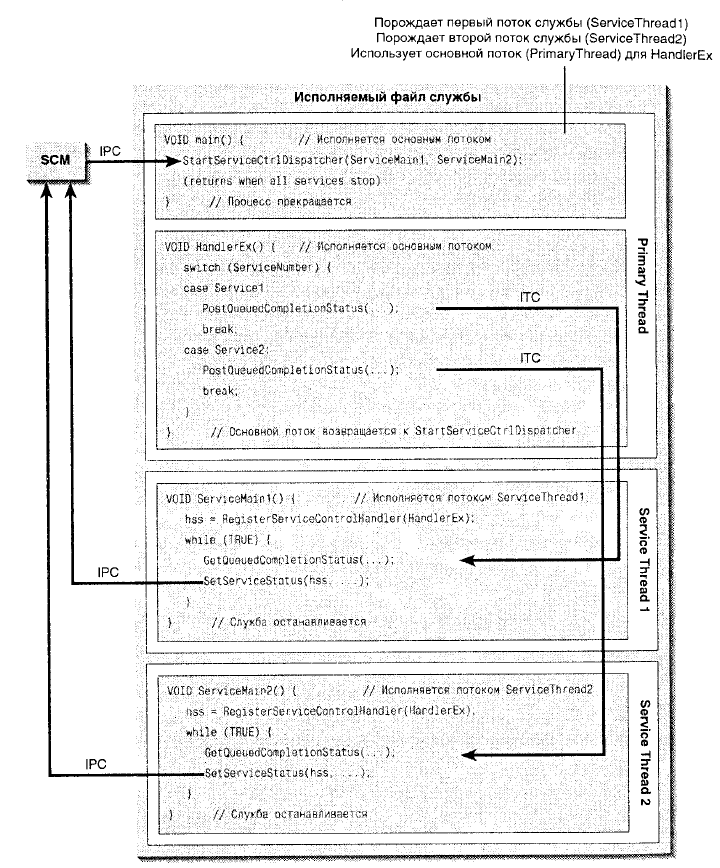
Интернет ресуры:

1. <https://www.freedesktop.org/software/systemd/man/daemon.html>
2. <http://man7.org/linux/man-pages/man3/daemon.3.html>

**ПРИЛОЖЕНИЯ**

**ПРИЛОЖЕНИЕ А**

Структура приложений-служб в Windows



**ПРИЛОЖЕНИЕ Б**

Фрагмент кода программы запуска в Linux

***// создаем потомка***

**pid = fork();**

***if* (pid == -1) { *// если не удалось запустить потомка***

***// выведем на экран ошибку и её описание***

**printf("Start Daemon Error: %s\n", strerror(errno));**

***return* -1;**

**} *else* *if* (!pid) { *// если это потомок***

***// разрешаем выставлять все биты прав на создаваемые файлы***

**umask(0);**

***// создаём новый сеанс, чтобы не зависеть от родителя***

**setsid();**

***// переходим в корень диска***

**chdir("/");**

***// закрываем дискрипторы***

**close(STDIN\_FILENO);**

**close(STDOUT\_FILENO);**

**close(STDERR\_FILENO);**

***// Данная функция будет осуществлять слежение за процессом***

**status = MonitorProc();**

***return* status;**

**} *else* { *// если это родитель***

***return* 0;**

**}**