

Системное программирование

Лекция 6

Сервисы

План лекции

- Что такое служба или сервис?
- Особенности сервисов
- Порядок разработки Windows-сервисов
- Что такое процесс «демон»?
- Особенности «демонов»
- Порядок разработки Linux-«демонов»

Windows-сервисы

Сервис или **служба** – это процесс, который выполняет служебные функции. Сервисы являются аналогами резидентных программ, которые использовались в операционных системах, предшествующих операционной системе Windows NT

То есть сервис это такая программа, которая запускается при загрузке операционной системы или в процессе ее работы по специальной команде и заканчивает свою работу при завершении работы операционной системы или по специальной команде

НО! Не каждая программа запускаемая со стартом операционной системы является сервисом

Windows-сервисы

Обычно сервисы выполняют определенные служебные функции, необходимые для работы приложений или какого-то конкретного приложения. Примером сервиса может служить фоновый процесс, который обеспечивает доступ к базе данных – такие сервисы также называются **серверами**

Другой тип сервисов – это программы, обеспечивающие доступ к внешним устройствам, такие сервисы называются **драйверами**

Как сервис также может быть реализован процесс, отслеживающий работу некоторого приложения, такие сервисы также называются **мониторами**

Windows-сервисы

Характеристики сервисов:

- Работают только в фоновом режиме
- Не имеют собственного управляющего интерфейса (ни GUI, ни TUI)
- Управляются специальной программой ОС – менеджером служб
- Запускаются (останавливаются) со стартом (выключением) ОС, со входом (выходом) пользователя или по команде (от менеджера служб)
- Предназначены для предоставления услуг другим программам или ОС, а не пользователям

Windows-сервисы

Управляет работой сервисов специальная программа операционной системы, которая называется менеджер сервисов (Service Control Manager, SCM)

Функции, которые выполняет менеджер сервисов:

- Поддержка базы данных установленных сервисов
- Запуск сервисов при загрузке операционной системы или по запросу
- Перечисление установленных сервисов
- Поддержка информации о состоянии работающих сервисов
- Передача управляющих запросов работающим сервисам
- Блокировка и разблокирование базы данных сервисов

Windows-сервисы

SCM поддерживает базу данных установленных сервисов в реестре. База данных используется SCM и программами, которые добавляют, изменяют или настраивают сервисы

База данных находится под ключом

HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services

Этот раздел содержит подраздел для каждого установленного сервиса и сервиса драйвера. Название подраздела - это название службы, которое указывается в специальной функции WinAPI для их создания (но об этом чуть дальше)

Windows-сервисы

The image shows two overlapping Windows system windows. The background window is the Registry Editor, displaying the path `Computer\HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\AFD`. The foreground window is the Services console, showing a list of system services.

Registry Editor Data:

Name	Type	Data
(Default)	REG_SZ	(value not set)
BootFlags	REG_DWORD	0x00000001 (1)
Description	REG_SZ	@%systemroot%\system32\drivers\afd.sys,-1000
DisplayName	REG_SZ	@%systemroot%\system32\drivers\afd.sys,-1000
ErrorControl	REG_DWORD	0x00000001 (1)
Group	REG_SZ	PNP_TDI
ImagePath	REG_EXPAND_SZ	\SystemRoot\...
Start	REG_DWORD	0x00000001 (1)
Type	REG_DWORD	0x00000001 (1)

Services (Local) Data:

Name	Description	Status	Startup Type	Log On As
AppX Deployment Service (A...	Provides infr...	Running	Manual (Trigg...	Local System
ASP.NET State Service	Provides sup...		Manual	Network Se...
AssignedAccessManager Ser...	AssignedAcc...		Manual (Trigg...	Local System
Auto Time Zone Updater	Automaticall...		Disabled	Local Service
AVCTP service	This is Audio...	Running	Manual (Trigg...	Local Service
Background Intelligent Tran...	Transfers file...		Manual	Local System
Background Tasks Infrastruc...	Windows inf...	Running	Automatic	Local System
Base Filtering Engine	The Base Fil...	Running	Automatic	Local Service
BcastDVRUserService_472aa	This user ser...		Manual	Local System
BitLocker Drive Encryption S...	BDESVC hos...		Manual (Trigg...	Local System
Block Level Backup Engine S...	The WBENGL...		Manual	Local System
Bluetooth Audio Gateway Se...	Service supp...	Running	Manual (Trigg...	Local Service
Bluetooth Support Service	The Bluetoo...	Running	Manual (Trigg...	Local Service
BluetoothUserService_472aa	The Bluetoo...	Running	Manual (Trigg...	Local System
BranchCache	This service ...		Manual	Network Se...
Capability Access Manager S...	Provides faci...	Running	Manual	Local System
CaptureService_472aa	Enables opti...		Manual	Local System
cbdhsvc_472aa	This user ser...	Running	Manual	Local System
CDPUserService_472aa	This user ser...	Running	Automatic	Local System
Cellular Time	This service ...		Manual (Trigg...	Local Service
Certificate Propagation	Copies user ...		Manual (Trigg...	Local System
Client License Service (ClipSV...	Provides infr...		Manual (Trigg...	Local System
CNG Key Isolation	The CNG ke...	Running	Manual (Trigg...	Local System
COM Sample Service			Manual	Local System
COM+ Event System	Supports Sy...	Running	Automatic	Local Service
COM+ System Application	Manages th...		Manual	Local System
Computer Browser	Maintains a...		Manual (Trigg...	Local System
Connected Devices Platform	This service i...	Running	Automatic (De...	Local Service

Windows-сервисы

Информация о сервисе:

- `DependOnGroup` – группы порядка загрузки сервисов от которых зависит данный сервис
- `DependOnService` – сервисы от которых зависит данный сервис
- `Description` – описание данного сервиса
- `DisplayName` – отображаемое имя данного сервиса
- `ErrorControl` – уровень управления ошибками в рамках данного сервиса
- `FailureActions` – действия выполняемые при возникновении ошибки в данном сервисе

Windows-сервисы

Информация о сервисе:

- Group – группа порядка загрузки сервисов в которой состоит данный сервис
- ImagePath – путь к исполняемому файлу данного сервиса
- ObjectName – учетная запись от имени которой происходит запуск данного сервиса
- Start – тип запуска данного сервиса
- Tag – уникальный тег для данного сервиса в рамках группы порядка загрузки в которой он состоит
- Type – тип данного сервиса

Windows-сервисы

1C:Enterprise 8 Server Control Agent Service Properties (Local Com... X

General Log On Recovery Dependencies

Service name: 1C:Enterprise 8.3 Server Agent

Display name: 1C:Enterprise 8 Server Control Agent Service

Description: 1C:Enterprise 8 Server Control Agent Service

Path to executable: ImagePath
"C:\Program Files (x86)\1cv8\8.3.24.1691\bin\ragent.exe" -svc -agent -regpo

Startup type: Automatic Start

Service status: Running

Start Stop Pause Resume

You can specify the start parameters that apply when you start the service from here.

Start parameters:

OK Cancel Apply

1C:Enterprise 8 Server Control Agent Service Properties (Local Com... X

General Log On Recovery Dependencies

Log on as:

☐ Local System account
☐ Allow service to interact with desktop

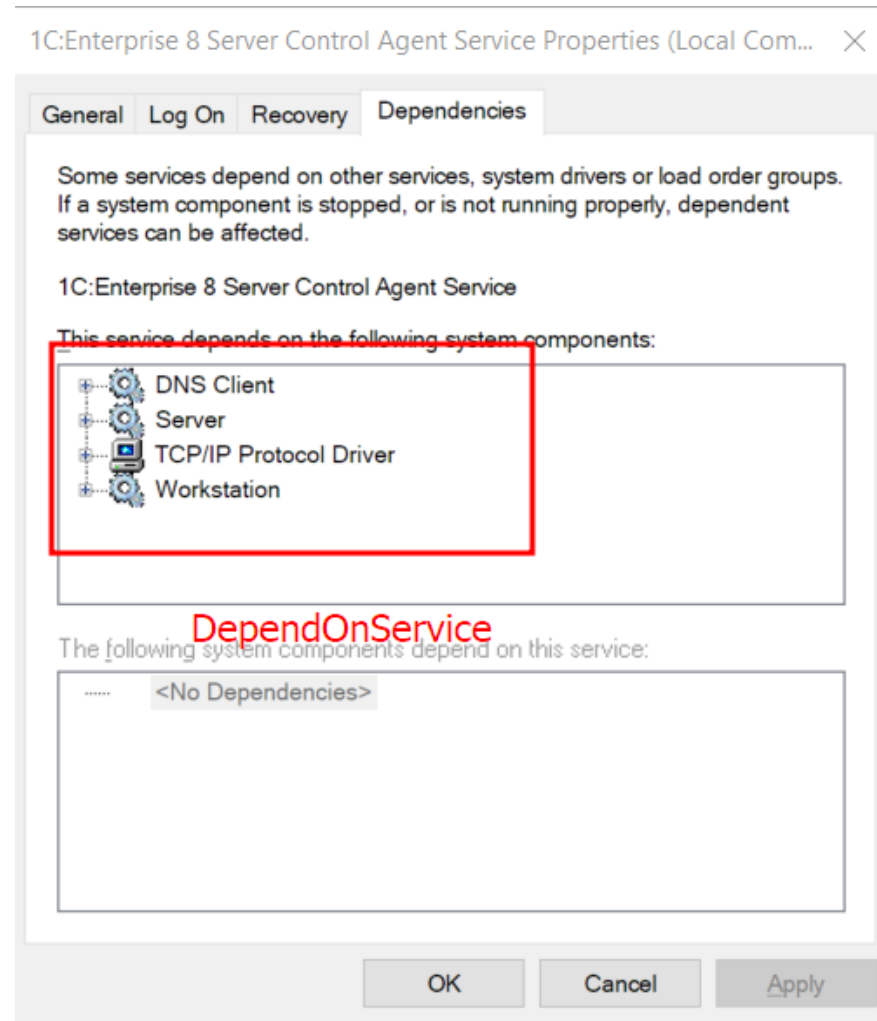
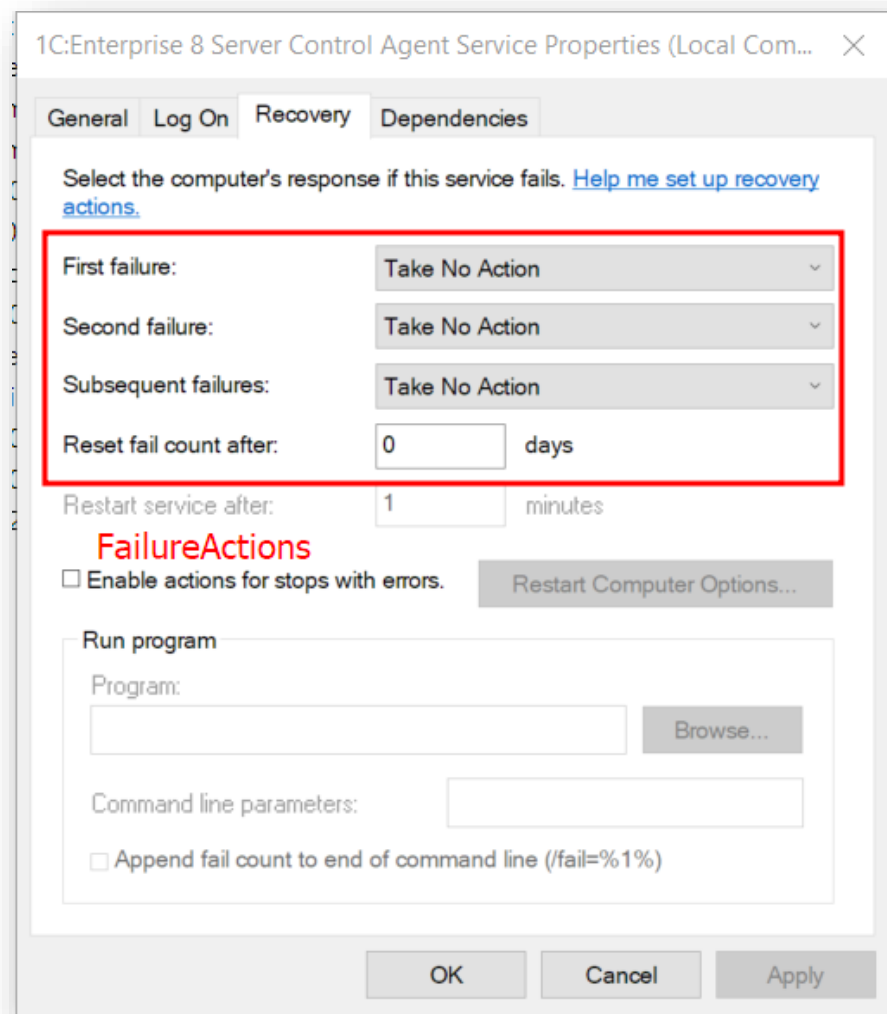
☒ This account: .\USR1CV8 ObjectName Browse...

Password:

Confirm password:

OK Cancel Apply

Windows-сервисы



Windows-сервисы

Группы порядка загрузки – логически объединенный набор сервисов который определяет порядок загрузки сервисов входящих в него относительно остальных групп или сервисов

В целом порядок загрузки сервисом определяется следующим образом:

- 1) Порядок групп в списке групп порядка загрузки
HKEY_LOCAL_MACHINE\System\CurrentControlSet\Control\ServiceGroupOrder
- 2) Порядок загрузки сервисов в рамках своей группы
HKEY_LOCAL_MACHINE\System\CurrentControlSet\Control\GroupOrderList
- 3) Остальные зависимости каждого из сервиса

Windows-сервисы

Name	Type	Data
(Default)	REG_SZ	(value not set)
List	REG_MULTI_SZ	System Reserved EMS WdfLoadGroup Boot Bu

Edit Multi-String

Value name:
List

Value data:
Pointer Port
Keyboard Port
Pointer Class
Keyboard Class
Video Init
Video
Video Save
File System
Streams Drivers
NDIS Wrapper
COM Infrastructure
Event Log

OKCancel

\GroupOrderList		
Name	Type	Data
(Default)	REG_SZ	(value not set)
Base	REG_BINARY	10 00 00 00 0e 00 00 00 01 00 00 00 02
Boot Bus Extender	REG_BINARY	06 00 00 00 07 00 00 00 01 00 00 00 02
Core Security Extensi...	REG_BINARY	02 00 00 00 01 00 00 00 02 00 00 00
Cryptography	REG_BINARY	02 00 00 00 01 00 00 00 02 00 00 00
EMS	REG_BINARY	01 00 00 00 01 00 00 00
Event log	REG_BINARY	01 00 00 00 01 00 00 00
Extended Base	REG_BINARY	0e 00 00 00 01 00 00 00 02 00 00 00 04
ExtendedBase	REG_BINARY	02 00 00 00 01 00 00 00 02 00 00 00
Filter	REG_BINARY	01 00 00 00 01 00 00 00
FSFilter Activity Monit...	REG_BINARY	01 00 00 00 01 00 00 00
FSFilter Anti-Virus	REG_BINARY	01 00 00 00 01 00 00 00
FSFilter Bottom	REG_BINARY	01 00 00 00 01 00 00 00
FSFilter Cluster File Sy...	REG_BINARY	01 00 00 00 01 00 00 00
FSFilter Compression	REG_BINARY	01 00 00 00 01 00 00 00
FSFilter Content Scree...	REG_BINARY	01 00 00 00 01 00 00 00
FSFilter Continuous B...	REG_BINARY	01 00 00 00 01 00 00 00
FSFilter Copy Protecti...	REG_BINARY	01 00 00 00 01 00 00 00
FSFilter Encryption	REG_BINARY	01 00 00 00 01 00 00 00
FSFilter HSM	REG_BINARY	01 00 00 00 01 00 00 00
FSFilter Imaging	REG_BINARY	01 00 00 00 01 00 00 00
FSFilter Infrastructure	REG_BINARY	01 00 00 00 01 00 00 00
FSFilter Open File	REG_BINARY	01 00 00 00 01 00 00 00
FSFilter Physical Quot...	REG_BINARY	01 00 00 00 01 00 00 00

Windows-сервисы

Так как сервисы работают под управлением менеджера сервисов, то они должны удовлетворять определенным соглашениям, которые определяют интерфейс сервиса

Сам сервис может быть как консольным приложением, так и приложением с графическим интерфейсом. Это не имеет значения, т. к. сервисы могут взаимодействовать с пользователем только через рабочий стол или через окно сообщений

Поэтому обычно сервисы оформляются как консольные приложения

Windows-сервисы

Кроме того, каждый сервис должен содержать **две функции обратного вызова**, которые вызываются операционной системой

Функция обратного вызова – это функция которая передаётся другой функции в качестве аргумента

Одна из этих функций определяет точку входа сервиса, т. е., собственно, и является сервисом, а вторая – должна реагировать на управляющие сигналы от операционной системы

Windows-сервисы

Отсюда следует, что если консольное приложение определяет один сервис, то оно должно иметь следующую структуру:

```
// главная функция консольного приложения
int main(int  argc, char  *argv) { ... }

// точка входа сервиса
VOID WINAPI ServiceMain(DWORD  dwArgc, LPTSTR  *lpszArgv) { ... }

// обработчик запросов
VOID WINAPI ServiceCtrlHandler(DWORD  dwControl) { ... }
```

Windows-сервисы

Главной задачей функции **main** является запуск диспетчера сервиса, который является потоком и управляет этим сервисом

Диспетчер сервиса получает управляющие сигналы от менеджера сервисов по именованному каналу и передает эти запросы функции **ServiceCtrlHandler**, которая обрабатывает эти управляющие запросы

Если в приложении несколько сервисов, то для каждого сервиса запускается свой диспетчер и для каждого диспетчера определяется своя функция обработки управляющих запросов, которая выполняется в контексте соответствующего диспетчера сервисов

Windows-сервисы

Запуск диспетчеров сервисов выполняется при помощи вызова функции **StartServiceCtrlDispatcher**

```
// главная функция приложения
int main()
{
    // инициализируем структуру сервисов
    SERVICE_TABLE_ENTRY service_table[] =
    {
        [0]={service_name, ServiceMain},    // имя сервиса и функция сервиса
        [1]={ NULL, NULL }                  // больше сервисов нет
    };

    // запускаем диспетчер сервиса
    if (!StartServiceCtrlDispatcher(lpServiceStartTable: service_table))
    {
        out.open("C:\\ServiceFile.log");
        out << "Start service control dispatcher failed.";
        out.close();

        return 0;
    }
}
```

Windows-сервисы

Для подключения обработчика запросов к сервису используется функция [RegisterServiceCtrlHandler](#)

```
VOID WINAPI ServiceMain(DWORD dwArgc, LPTSTR *lpszArgv)
{
    // регистрируем обработчик управляющих команд для сервиса
    hServiceStatus = RegisterServiceCtrlHandler(
        lpServiceName: service_name,          // имя сервиса
        lpHandlerProc: ServiceCtrlHandler     // обработчик управляющих команд
    );
    if (!hServiceStatus)
    {
        out.open("C:\\ServiceFile.log");
        out << "Register service control handler failed.";
        out.close();

        return;
    }
}
```

Windows-сервисы

Функция, определяющая точку входа сервиса, должна иметь следующий прототип:

```
VOID WINAPI имя_точки_входа(DWORD dwArgc, LPTSTR *lpszArgv);
```

Если определяется только один сервис, то эта функция обычно называется **ServiceMain**, хотя возможны и другие, более подходящие по смыслу имена точек входа сервисов

Если же в приложении определяется несколько сервисов, то естественно каждый из них должен иметь свое имя. Эта функция содержит два параметра, которые аналогичны параметрам функции **main** консольного приложения

Windows-сервисы

Как уже было сказано, главной задачей функции **main** является запуск диспетчера сервиса для каждого из сервисов. Для запуска диспетчера используется функция **StartServiceCtrlDispatcher**, которая должна быть вызвана в течение **30 секунд** с момента запуска программы **main**

Если в течение этого промежутка времени функция **StartServiceCtrlDispatcher** вызвана не будет, то последующий вызов этой функции закончится **неудачей**

Поэтому всю необходимую инициализацию сервиса нужно делать в самом сервисе, т. е. в теле функции **ServiceMain**. Если же необходимо выполнить инициализацию глобальных переменных, то для этой цели лучше запустить из функции **main** отдельный поток

Windows-сервисы

Функция **ServiceMain** должна выполнить следующую последовательность действий:

1. Немедленно запустить обработчик управляющих команд от менеджера сервисов, вызвав функцию **RegisterServiceCtrlHandler**
2. Установить стартующее состояние сервиса SERVICE_START_PENDING посредством вызова функции **SetServiceStatus**
3. Провести локальную инициализацию сервиса
4. Установить рабочее состояние сервиса SERVICE_RUNNING посредством вызова функции **SetServiceStatus**
5. Выполнять работу сервиса, учитывая состояния сервиса, которые могут изменяться обработчиком управляющих команд от менеджера сервисов
6. После перехода в состояние останова SERVICE_STOPPED выполнить освобождение захваченных ресурсов и закончить работу

Windows-сервисы

```
// инициализируем структуру состояния сервиса
service_status.dwServiceType = SERVICE_WIN32_OWN_PROCESS;
service_status.dwCurrentState = SERVICE_START_PENDING;
service_status.dwControlsAccepted = SERVICE_ACCEPT_STOP |
| | | | | | | | | | | | | | | | SERVICE_ACCEPT_SHUTDOWN;
service_status.dwWin32ExitCode = ERROR_SERVICE_SPECIFIC_ERROR;
service_status.dwServiceSpecificExitCode = 0;
service_status.dwCheckPoint = 0;
service_status.dwWaitHint = 5000;
```

```
// устанавливаем состояние сервиса
if (!SetServiceStatus(hServiceStatus, lpServiceStatus: &service_status))
{
    out.open("C:\\ServiceFile.log");
    out << "Set service status 'SERVICE_START_PENDING' failed."
    out.close();

    return;
}
```

```
// определяем сервис как работающий
service_status.dwCurrentState = SERVICE_RUNNING;
// нет ошибок
service_status.dwWin32ExitCode = NO_ERROR;
// устанавливаем новое состояние сервиса
if (!SetServiceStatus(hServiceStatus, lpServiceStatus: &service_status))
{
    out.open("C:\\ServiceFile.log");
    out << "Set service status 'START_PENDING' failed.";
    out.close();
    return;
}
```


Windows-сервисы

Параметр *lpServiceStatus* функции **SetServiceStatus** должен указывать на структуру типа SERVICE_STATUS, которая содержит информацию о состоянии сервиса. Эта структура имеет следующий формат:

```
typedef struct _SERVICE_STATUS {
    DWORD dwServiceType;           // тип сервиса
    DWORD dwCurrentState;          // текущее состояние сервиса
    DWORD dwControlsAccepted;       // допускаемое управление
    DWORD dwWin32ExitCode;          // код возврата для Win32
    DWORD dwServiceSpecificExitCode; // специфический код возврата
                                    // сервиса
    DWORD dwCheckPoint;             // контрольная точка
    DWORD dwWaitHint;               // период ожидания команды управления
} SERVICE_STATUS, *LPSERVICE_STATUS;
```

Windows-сервисы

Поле ***dwServiceType*** содержит тип сервиса и может принимать следующие значения:

- **SERVICE_WIN32_OWN_PROCESS** – сервис является самостоятельным процессом
- **SERVICE_WIN32_SHARE_PROCESS** – сервис разделяет процесс с другими сервисами
- **SERVICE_KERNEL_DRIVER** – сервис является драйвером устройства
- **SERVICE_FILE_SYSTEM_DRIVER** – сервис является драйвером файловой системы
- **SERVICE_USER_OWN_PROCESS** – сервис является самостоятельным процессом запускаемым для определенного пользователя
- **SERVICE_USER_SHARE_PROCESS** – сервис разделяет процесс с другими сервисами запускаемыми для определенного пользователя

Кроме того, первые два флага могут быть установлены совместно с флагом **SERVICE_INTERACTIVE_PROCESS** – сервис может взаимодействовать с рабочим столом

Windows-сервисы

Поле ***dwCurrentState*** содержит текущее состояние сервиса. Это поле может принимать одно из следующих значений:

- **SERVICE_STOPPED** – сервис остановлен
- **SERVICE_START_PENDING** – сервис стартует
- **SERVICE_STOP_PENDING** – сервис останавливается
- **SERVICE_RUNNING** – сервис работает
- **SERVICE_CONTINUE_PENDING** – сервис переходит в рабочее состояние
- **SERVICE_PAUSE_PENDING** – сервис переходит в состояние ожидания
- **SERVICE_PAUSED** – сервис находится в состоянии ожидания

Windows-сервисы

Поле ***dwControlsAccepted*** содержит коды управляющих команд, которые могут быть переданы обработчику этих команд, определенному в приложении. В этом поле может быть установлена любая комбинация следующих флагов:

- **SERVICE_ACCEPT_STOP** – можно остановить сервис
- **SERVICE_ACCEPT_PAUSE_CONTINUE** – можно приостановить и возобновить сервис
- **SERVICE_ACCEPT_SHUTDOWN** – сервис информируется о выключении системы
- **SERVICE_ACCEPT_PRESHUTDOWN** – сервис может выполнить действия перед выключением системы
- **SERVICE_ACCEPT_PARAMCHANGE** - сервис может вновь прочитать свои стартовые параметры без перезагрузки
- **SERVICE_ACCEPT_NETBINDCHANGE** - сервис является сетевой компонентой

Windows-сервисы

Поле ***dwServiceSpecificExitCode*** содержит код возврата из сервиса, этот код действителен только в том случае, если в поле ***dwWin32ExitCode*** установлено значение `ERROR_SERVICE_SPECIFIC_ERROR`

Поле ***dwCheckPoint*** содержит значение, которое сервис должен периодически увеличивать на единицу, сообщая о продвижении своей работы во время инициализации и длительных переходов из состояния в состояние

Это значение может использоваться программой пользователя, которая отслеживает работу сервиса. Если это значение не используется пользовательской программой и переход из состояния в состояние занимает менее 30 секунд, то оно может быть установлено в 0

Windows-сервисы

Поле ***dwWaitHint*** содержит интервал времени в миллисекундах, в течение которого сервис переходит из состояния в состояние перед вызовом функции установки состояния **SetServiceStatus**

Если в течение этого интервала не произошло изменение состояния сервиса в поле **dwServiceState** или не изменилось значение поля ***dwCheckPoint***, то менеджер сервисов считает, что в сервисе произошла ошибка

Windows-сервисы

Функция, определяющая обработчик управляющих запросов, должна иметь следующий прототип:

```
VOID WINAPI имя_обработчика_запросов(DWORD dwControl);
```

Если определяется только один сервис, то эта функция обычно называется **ServiceCtrlHandler**

Если же в приложении определяется несколько сервисов, то естественно, что обработчик запросов для каждого сервиса должен иметь свое имя. Эта функция содержит только один параметр, который содержит код управляющего сигнала

Так как обработчик запросов вызывается диспетчером сервиса, то и коды управляющих сигналов он получает от своего диспетчера

Windows-сервисы

Возможны следующие основные управляющие коды:

- `SERVICE_CONTROL_STOP` – остановить сервис
- `SERVICE_CONTROL_PAUSE` – приостановить сервис
- `SERVICE_CONTROL_CONTINUE` – возобновить сервис
- `SERVICE_CONTROL_INTERROGATE` – обновить состояние сервиса
- `SERVICE_CONTROL_SHUTDOWN` – закончить работу сервиса

Windows-сервисы

Но обработчик должен обрабатывать только те команды, которые допускаются в поле ***dwControlsAccepted*** структуры типа **SERVICE_STATUS**, рассмотренной ранее

По соглашению обработчик всегда получает сигнал с кодом **SERVICE_CONTROL_INTERROGATE**, по которому он должен немедленно обновить состояние сервиса

Для обновления состояния сервиса используется функция **SetServiceStatus**, которая была рассмотрена в предыдущем разделе. Кроме того, обработчик может получать коды, определенные пользователем. Для кодов пользователя зарезервирован диапазон от 128 до 255

Windows-сервисы

Вызывается обработчик управляющих запросов диспетчером сервиса и, следовательно, выполняется в его контексте

Обработчик должен изменить состояние сервиса в течение 30 секунд, в противном случае диспетчер сервисов считает, что произошла ошибка

Если для изменения состояния сервиса требуется более продолжительный интервал времени, то для этой цели нужно запустить отдельный поток

Для обработки кода **SERVICE_CONTROL_SHUTDOWN** сервису отводится 20 секунд, в течение которых он должен освободить захваченные им ресурсы

Windows-сервисы

```
VOID WINAPI ServiceCtrlHandler(DWORD dwControl)
{
    switch(dwControl)
    {
        case SERVICE_CONTROL_STOP:        // остановить сервис
            // записываем конечное значение счетчика
            out << "Count = " << nCount << endl;
            out << "The service is finished." << endl << flush;
            // закрываем файл
            out.close();

            // устанавливаем состояние остановки
            service_status.dwCurrentState = SERVICE_STOPPED;
            // изменить состояние сервиса
            SetServiceStatus(hServiceStatus, lpServiceStatus: &service_status);
            break;

        case SERVICE_CONTROL_SHUTDOWN:    // завершить сервис
            service_status.dwCurrentState = SERVICE_STOPPED;
            // изменить состояние сервиса
```

Windows-сервисы

Для того чтобы менеджер сервисов знал о существовании определенного сервиса – его нужно установить

Сервис может запускаться как операционной системой при загрузке, так и **программно** – из приложения

Если сервис больше не нужен, то его нужно удалить из базы данных операционной системы

Для работы с сервисами в операционных системах Windows предназначены специальные функции из **WinAPI Service**

Управлять работой сервисов можно также и через панель управления

Windows-сервисы

Функции **WinAPI** для программного взаимодействия с сервисами:

- **OpenSCManager** - установка связи с менеджером сервисов и открытие доступа к базе данных сервисов
- **CreateService** – установка сервиса в базу данных
- **OpenService** – открытие уже установленного в базу данных сервиса
- **StartService** – запуск сервиса
- **ControlService** – управление сервисом (отправка управляющей команды)
- **DeleteService** – удаление сервиса из базы данных
- **QueryServiceStatusEx** – определение состояния сервиса

Для успешной сборки проекта использующего данные функции необходимо подключить библиотеку **Advapi32.dll (-ladvapi32)**

Windows-сервисы


















Также существует консольная утилита для работы с сервисами **sc.exe**

Основные команды:

- **sc create** – установка сервиса в базу данных
- **sc start** – запуск сервиса
- **sc control** – управление сервисом (отправка управляющей команды)
- **sc stop** – остановка сервиса
- **sc delete** – удаление сервиса из базы данных
- **sc query** – определение состояния сервиса

Windows-сервисы

```
└─Δ sc create DemoService BinPath="C:\Program Files\DemoService\DemoService.exe"  
[SC] CreateService SUCCESS
```

	CoreMessaging	Manages co...	Running	Automatic	Local Service
	Credential Manager	Provides sec...	Running	Manual	Local System
	CredentialEnrollmentManag...	Credential E...		Manual	Local System
	Cryptographic Services	Provides thr...	Running	Automatic	Network Se...
	Data Sharing Service	Provides dat...	Running	Manual (Trigg...	Local System
	Data Usage	Network dat...	Running	Automatic	Local Service
	DCOM Server Process Launc...	The DCOML...	Running	Automatic	Local System
	Declared Configuration(DC) ...	Process Decl...		Manual (Trigg...	Local System
	Delivery Optimization	Performs co...	Running	Automatic (De...	Network Se...
	DemoService			Manual	Local System
	Device Association Service	Enables pairi...	Running	Manual (Trigg...	Local System
	Device Install Service	Enables a co...		Manual (Trigg...	Local System
	Device Management Enroll...	Performs De...		Manual	Local System
	Device Management Wireles...	Routes Wirel...		Manual (Trigg...	Local System
	Device Setup Manager	Enables the ...		Manual (Trigg...	Local System
	DeviceAssociationBrokerSvc...	Enables app...		Manual	Local System
	DevicePickerUserSvc_571b...	This user ser...		Manual	Local System

Windows-сервисы

```
└─┐ sc query demoservice
```

```
SERVICE_NAME: demoservice
```

```
        TYPE               : 10  WIN32_OWN_PROCESS
```

```
        STATE               : 1  STOPPED
```

```
        WIN32_EXIT_CODE      : 1077 (0x435)
```

```
        SERVICE_EXIT_CODE   : 0   (0x0)
```

```
        CHECKPOINT          : 0x0
```

```
        WAIT_HINT           : 0x0
```

```
└─┐ sc start demoservice
```

```
SERVICE_NAME: demoservice
```

```
        TYPE               : 10  WIN32_OWN_PROCESS
```

```
        STATE               : 4  RUNNING
```

```
(STOPPABLE, NOT_PAUSABLE, ACCEPTS_SHUTDOWN)
```

```
        WIN32_EXIT_CODE      : 0   (0x0)
```














```
        SERVICE_EXIT_CODE   : 0   (0x0)
```

```
        CHECKPOINT          : 0x0
```

```
        WAIT_HINT           : 0x0
```

```
        PID                 : 19720
```

```
        FLAGS                :
```

	Data Sharing Service	Provides dat...	Running	Manual (Trigg...	Local System
	Data Usage	Network dat...	Running	Automatic	Local Service
	DCOM Server Process Launc...	The DCOML...	Running	Automatic	Local System
	Declared Configuration(DC) ...	Process Decl...		Manual (Trigg...	Local System
	Delivery Optimization	Performs co...	Running	Automatic (De...	Network Se...
	DemoService		Running	Manual	Local System
	Device Association Service	Enables pairi...	Running	Manual (Trigg...	Local System
	Device Install Service	Enables a co...		Manual (Trigg...	Local System
	Device Management Enroll...	Performs De...		Manual	Local System
	Device Management Wireles...	Routes Wirel...		Manual (Trigg...	Local System
	Device Setup Manager	Enables the ...		Manual (Trigg...	Local System
	DeviceAssociationBrokerSvc...	Enables app...		Manual	Local System
	DevicePickerUserSvc 571be	This user ser		Manual	Local System

Windows-сервисы

```
└─△ sc stop demoservice
```

```
SERVICE_NAME: demoservice
        TYPE               : 10  WIN32_OWN_PROCESS
        STATE                : 1  STOPPED
        WIN32_EXIT_CODE       : 0   (0x0)
        SERVICE_EXIT_CODE    : 0   (0x0)
        CHECKPOINT           : 0x0
        WAIT_HINT             : 0x0
```

```
└─△ sc delete demoservice
[SC] DeleteService SUCCESS
```

	Data Sharing Service	Provides dat...	Running	Manual (Trigg...	Local System
	Data Usage	Network dat...	Running	Automatic	Local Service
	DCOM Server Process Launc...	The DCOML...	Running	Automatic	Local System
	Declared Configuration(DC) ...	Process Decl...		Manual (Trigg...	Local System
	Delivery Optimization	Performs co...	Running	Automatic (De...	Network Se...
	Device Association Service	Enables pairi...	Running	Manual (Trigg...	Local System
	Device Install Service	Enables a co...		Manual (Trigg...	Local System
	Device Management Enroll...	Performs De...		Manual	Local System
	Device Management Wireles...	Routes Wirel...		Manual (Trigg...	Local System
	Device Setup Manager	Enables the ...		Manual (Trigg...	Local System
	DeviceAssociationBrokerSvc...	Enables app...		Manual	Local System
	DevicePickerUserSvc_571be...	This user ser...		Manual	Local System

Linux-демон

Демон (англ. **daemon**) – это процесс, обладающий следующими свойствами:

- Имеет длинный жизненный цикл. Часто демоны создаются во время загрузки системы и работают до момента ее выключения
- Выполняется в фоновом режиме и не имеет контролирующего терминала

Последняя особенность гарантирует, что ядро не сможет генерировать для такого процесса никаких сигналов, связанных с терминалом или управлением заданиями (таких как SIGINT, SIGTSTP и SIGHUP)

Названия демонов принято заканчивать буквой **d** (хотя это не является обязательным правилом)

Linux-демон

Для того чтобы стать демоном, программа должна выполнить следующие шаги:

1. Сделать вызов **fork**, после которого родитель завершается, а потомок продолжает работать (в результате этого демон становится потомком процесса `init`)

Этот шаг делается по двум следующим причинам:

- Исходя из того, что демон был запущен в командной строке, завершение родителя будет обнаружено командной оболочкой, которая вслед за этим выведет новое приглашение и позволит потомку выполняться в фоновом режиме
- Потомок гарантированно не станет лидером группы процессов, поскольку он наследует PGID от своего родителя и получает свой уникальный идентификатор, который отличается от унаследованного PGID. Это необходимо для успешного выполнения следующего шага

Linux-демон

Для того чтобы стать демоном, программа должна выполнить следующие шаги:

2. Дочерний процесс вызывает **setsid**, чтобы начать новую сессию и разорвать любые связи с контролирующим терминалом

3. Если после этого демон больше не открывает никаких терминальных устройств, мы можем не волноваться о том, что он восстановит соединение с контролирующим терминалом. В противном случае нам необходимо 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.
- Есть более простой вариант: после **setsid** можно 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

Linux-демон

Для того чтобы стать демоном, программа должна выполнить следующие шаги:

4. Очистить атрибут **umask** процесса, чтобы файлы и каталоги, созданные демоном, имели запрашиваемые права доступа

5. Поменять текущий рабочий каталог процесса (обычно на корневой – /)

Это необходимо, поскольку демон обычно выполняется вплоть до выключения системы. Если файловая система, на которой находится его текущий рабочий каталог, не является корневой, она не может быть отключена. Как вариант, в качестве рабочего каталога демон может задействовать то место, где он выполняет свою работу, или воспользоваться значением в конфигурационном файле; главное, чтобы файловая система, в которой находится этот каталог, никогда не нуждалась в отключении

Linux-демон

Для того чтобы стать демоном, программа должна выполнить следующие шаги:

6. Закрывать все открытые файловые дескрипторы, которые демон унаследовал от своего родителя (возможно, некоторые из них необходимо оставить открытыми, поэтому данный шаг является необязательным и может быть откорректирован)

Это делается по целому ряду причин. Поскольку демон потерял свой контролируемый терминал и работает в фоновом режиме, ему больше не нужно хранить дескрипторы с номерами 0, 1 и 2 (если они ссылаются на терминал)

Кроме того, мы не можем отключить файловую систему, на которой долгоживущий демон удерживает открытыми какие-либо файлы. И, следуя обычным правилам, мы должны закрывать неиспользуемые файловые дескрипторы, поскольку их число ограничено

Linux-демон

Для того чтобы стать демоном, программа должна выполнить следующие шаги:

7. Закрыв дескрипторы с номерами 0, 1 и 2, демон обычно перенаправляет их в предварительно открытый файл **/dev/null**, используя вызов **dup2** (или похожий)

Это делается по двум причинам:

- Это позволяет избежать ошибки при вызове библиотечных функций, которые выполняют операции ввода/вывода с этими дескрипторами
- Это исключает возможность повторного открытия демоном файлов с помощью дескрипторов 1 или 2, так как библиотечные функции, которые записывают в них данные, ожидают, что эти дескрипторы указывают на потоки **stdout** (стандартный вывод) и **stderr** (стандартный вывод ошибок)

Linux-демон

```
root@kali:~/ds# ./daemon_SIGHUP
root@kali:~/ds# ps -ef
```

UID	PID	PPID	C	STIME	TTY	TIME	CMD
root	1	0	0	01:50	hvc0	00:00:00	/init
root	6	1	0	01:50	hvc0	00:00:00	plan9 --control-socket 5 --log-level 4 --server-fd 6 --p
root	672	1	0	04:48	?	00:00:00	/init
root	673	672	0	04:48	?	00:00:00	/init
pasha	674	673	0	04:48	pts/0	00:00:00	-bash
root	688	1	0	04:48	?	00:00:00	/init
root	689	688	0	04:48	?	00:00:00	/init
pasha	690	689	0	04:48	pts/2	00:00:00	-bash
root	1028	673	0	05:18	?	00:00:00	/usr/sbin/testd
pasha	1045	673	0	05:24	?	00:00:00	./daemon_SIGHUP
pasha	1046	674	0	05:24	pts/0	00:00:00	ps -ef

```
root@kali:~/ds# cat /tmp/ds.log
```

2024-11-22 00:52:39: Opened log file
2024-11-22 00:52:54: Main: 1
2024-11-22 00:53:09: Main: 2
2024-11-22 00:53:24: Main: 3
2024-11-22 00:53:39: Main: 4
2024-11-22 00:53:54: Main: 5
2024-11-22 00:54:09: Main: 6
2024-11-22 00:54:24: Main: 7
2024-11-22 00:54:39: Main: 8

Linux-демон

```
root@kali:~/ds# cat /tmp/ds.conf  
CHANGED  
root@kali:~/ds# kill -HUP 1045
```

```
2024-11-22 00:54:40: Closing log file  
2024-11-22 00:54:40: Opened log file  
2024-11-22 00:54:40: Read config file: CHANGED  
2024-11-22 05:24:26: Opened log file  
2024-11-22 05:24:26: Read config file: CHANGED
```

Linux-демон

Рекомендации:

Как уже отмечалось выше, процесс-демон обычно завершается во время выключения системы. Для многих стандартных демонов предусмотрены специальные скрипты, которые выполняются, когда система завершает работу

Остальные демоны просто получают сигнал **SIGTERM**, который при выключении компьютера отправляется процессом **init** всем своим потомкам. По умолчанию этот сигнал приводит к завершению процесса. Если демону перед этим необходимо освободить какие-либо ресурсы, он должен делать это в обработчике данного сигнала

Linux-демон

Рекомендации:

Эту процедуру следует выполнять как можно быстрее, поскольку через 5 секунд после **SIGTERM** процесс **init** отправляет сигнал **SIGKILL** (это вовсе не означает, что у демона есть 5 секунд процессорного времени на освобождение ресурсов; **init** шлет эти сигналы всем процессам в системе одновременно, поэтому процедуру очистки в этот момент может выполнять каждый из них)

Linux-демон

Рекомендации:

Так как демоны имеют длинный жизненный цикл, нам следует особенно тщательно следить не только за потенциальными утечками памяти, но и за файловыми дескрипторами (когда приложению не удастся закрыть все файловые дескрипторы, которые оно открыло). Для временного исправления подобных ошибок демон приходится перезапускать заново

Часто демону необходимо убедиться в том, что **только один его экземпляр активен** в любой заданный момент времени

Linux-демон

Рекомендации:

Обычно это достигается следующим образом: демон создает файл в стандартном каталоге и применяет к нему блокировку для записи. Он удерживает ее на протяжении всего своего существования и удаляет прямо перед завершением. Если попытаться запустить другой экземпляр того же демона, то он не сможет получить блокировку для соответствующего файла и автоматически завершится, понимая: один его экземпляр уже выполняется в системе

Linux-демон

Для работы с демонами в Linux также как и в Windows существует менеджер сервисов: **init** (считается устаревшим) или **systemd** (является более новым)

Стоит заметить, что ранее описанный алгоритм «демонизации» приложения выполняется менеджерами самостоятельно! Нет необходимости в коде вашего будущего демона совершать описанные манипуляции!

Они также позволяют настроить автозапуск сервисов вместе с запуском операционной системы

В рамках подсистемы Linux для Windows можно встретить оба варианта, например, Debian использует **init**, а Ubuntu - **systemd**

Linux-демон

При использовании менеджера **init**:

- Каталог с конфигурационными файлами сервиса должен располагаться по пути **/etc/<имя демона>**
- Каталог приложения расположить по пути **/sbin/<имя демона>**
- Файлы со значениями по умолчанию для скрипта
- Скрипт с настройками управления сервисом должны располагаться в каталоге **/etc/init.d/<имя демона>**
- Файлы журналов должны находиться в каталоге **/var/log**

Первый и последний пункты не являются требованиями, а скорее общепринятыми практиками при создании системных сервисов

Linux-демон

INIT скрипт может иметь следующие состояния:

- **start** – служит командой для запуска сервиса
- **stop** – выполняет остановку сервиса
- **restart** – перезапуск сервиса, а по факту – остановка и затем запуск сервиса
- **reload** – перезагрузка сервиса, т.е. команда для повторного считывания конфигурации без перезапуска или остановки сервиса
- **force-reload** – перезагрузка конфигурации (перечитать конфиг), если сервис поддерживает это, в противном случае – выполнит перезапуск сервиса
- **status** – покажет состояние сервиса

Также существует консольная утилита для работы с сервисами **service** (**systemctl** в случае **systemd**)

Linux-демон

Добавление сервиса на уровне системы с помощью Initd (auto_daemon.c в примерах):

1. Компилируем приложение будущего демона
2. Располагаем получившийся бинарный файл в одной из папок **/sbin** или **/usr/sbin**

```
bash@LAPTOP-NCM9418:~$ ls /usr/sbin
accessdb          dpkg-reconfigure  halt              nologin           switch_root
addgnupghome      dump2fs           hwlock           ownership         systemctl
addgroup          e2fsck            iconvconfig      pam-auth-update   tarcat
add-shell         e2fsck            icuunpack        pam_getenv        tz
adduser           e2imgn           ifdown           pam_namespace_helper  telinit
agetty            e2label          ifquery          pam_timestamp_check  testd
applygnupgdefaults  e2mmpstatus      ifup             paperconfig       tipc
arpd              e2scrub          init             pivot_root        tune2fs
```

Linux-демон

Добавление сервиса на уровне системы с помощью Initd (auto_daemon.c в примерах):

3. Располагаем скрипт запуска **<имя демона>** в каталоге **/etc/init.d/** (меняем ему права через **chmod** на **755**)

```
bash@LAPTOP-NCM9419:~$ ls /etc/init.d/  
cron dbus hwclock.sh kmod networking procps sudo testd udev x11-common  
bash@LAPTOP-NCM9419:~$
```

4.(Опционально) Создаём каталог **/etc/<имя демона>** в котором располагаем конфигурационные файлы которые будут использоваться самим сервисом

5.(Справочно) Файлы логов обычно располагают в каталоге **/var/log**

Linux-демон

Добавление сервиса на уровне системы с помощью Initd (auto_daemon.c в примерах):

6.Проверяем с помощью команды **service**, что система распознала скрипт инициализации сервиса

```
patch@LAPTOP-HC7H9419:~$ service --status-all
[ - ] cron
[ - ] dbus
[ ? ] hwclock.sh
[ ? ] kmod
[ ? ] networking
[ - ] procps
[ - ] sudo
[ + ] testd
[ + ] udev
[ - ] x11-common
```

Linux-демон

Добавление сервиса на уровне системы с помощью Initd (auto_daemon.c в примерах):

7. Если скрипт написан корректно, то будет доступно управление сервисом через команду **service**

```
pasha@LAPTOP-HCMN9419:~$ sudo service testd start
Starting testd daemon: testd.
pasha@LAPTOP-HCMN9419:~$ ps -ef
UID          PID    PPID  C STIME TTY          TIME CMD
root           1        0  0  14:12 ?           00:00:00 /init
root           6         1  0  14:12 ?           00:00:00 plan9 --control-soc
root           9         1  0  14:12 ?           00:00:00 /init
root          10         9  0  14:12 ?           00:00:00 /init
pasha         11        10  0  14:12 pts/0       00:00:00 -bash
root          137        10  0  14:29 ?           00:00:00 /usr/sbin/testd
pasha         138        11  0  14:29 pts/0       00:00:00 ps -ef
pasha@LAPTOP-HCMN9419:~$ sudo service testd status
testd is running.
pasha@LAPTOP-HCMN9419:~$ sudo service testd reload
Reloading testd daemon: testd.
pasha@LAPTOP-HCMN9419:~$ sudo service testd stop
Stopping testd daemon: testd.
pasha@LAPTOP-HCMN9419:~$
```

Linux-демон

Добавление сервиса на уровне системы с помощью Initd (auto_daemon.c в примерах):

8. Для автозапуска вместе с системой надо перейти в каталог **/etc/init.d** и выполнить следующую команду:

```
/etc/init.d$ sudo update-rc.d testd defaults
```

В рамках ОС Debian с использованием WSL2 данная команда не приводит к какому-либо видимому результату

В системах с менеджером служб **systemd** в режиме совместимости с **initd**, данная последовательность действий также является полностью рабочей!

Linux-демон

Добавление сервиса на уровне системы с помощью Systemd:

1. Компилируем приложение будущего сервиса
2. Располагаем получившийся бинарный файл в одной из папок **/sbin** или **/usr/sbin**

```
pasha@LAPTOP-HCM49418:~$ ls /usr/sbin
acccssdb          dphg-reconfigure  halt              nologin           switch_root
addgnupghome     dmpe2fs           hwlock           ownership         systemctl
addgroup         e2fsck            iconvconfig      pam-auth-update   tarcat
add-shell        e2fsck            icuunpack        pam_getenv        tz
adduser          e2imgn            ifdown           pam_namespace_helper telinit
agetty           e2label           ifquery          pam_timestamp_check textd
applygnupgdefaults e2mmpstatus      ifup             paperconfig       tipc
arpd             e2scrub           init             pivot_root        tune2fs
```

Linux-демон

Добавление сервиса на уровне системы с помощью Systemd:

3. Располагаем файл описания **<имя демона>.service** в каталоге **/etc/systemd/system**

```
nasha@LAPTOP-NCHM9419:~$ ls /etc/systemd/system
cloud-config.target.wants      final.target.wants            sysinit.target.wants
cloud-final.service.wants      getty.target.wants            syslog.service
cloud-init.target.wants        multi-user.target.wants       testd.service
dbus-org.freedesktop.resolve1.service  paths.target.wants            timers.target.wants
dbus-org.freedesktop.timedate1.service  sockets.target.wants
```

4.(Опционально) Создаём каталог **/etc/<имя демона>** в котором располагаем конфигурационные файлы которые будут использоваться самим демоном

5.(Справочно) Файлы логов обычно располагают в каталоге **/var/log**

Linux-демон

Добавление сервиса на уровне системы с помощью Systemd

6.Проверяем с помощью команды **systemctl**, что система распознала описание сервиса

```
pasha@LAPTOP-HCMN9419:~$ systemctl list-unit-files t*
UNIT FILE          STATE    PRESET
testd.service      disabled enabled
time-set.target    static   -
time-sync.target   static   -
timers.target       static   -

4 unit files listed.
```


Linux-демон

Добавление сервиса на уровне системы с помощью Systemd (auto_daemon.c в примерах):

7. Если файл с описанием написан корректно, то будет доступно управление сервисом через команду **systemctl**

```
pasha@LAPTOP-HCMN9419:~$ sudo systemctl start testd
pasha@LAPTOP-HCMN9419:~$ sudo systemctl status testd
● testd.service - Example daemon
   Loaded: loaded (/etc/systemd/system/testd.service; disabled; preset: enabled)
   Active: active (running) since Mon 2024-11-25 15:32:16 +03; 2s ago
 Main PID: 1006 (testd)
    Tasks: 1 (limit: 9123)
  Memory: 168.0K ()
   CGroup: /system.slice/testd.service
           └─1006 /usr/sbin/testd

Nov 25 15:32:16 LAPTOP-HCMN9419 systemd[1]: Starting testd.service - Example daemon.
Nov 25 15:32:16 LAPTOP-HCMN9419 systemd[1]: Started testd.service - Example daemon.
pasha@LAPTOP-HCMN9419:~$ sudo systemctl stop testd
```

Linux-демон

Добавление сервиса на уровне системы с помощью Systemd (auto_daemon.c в примерах):

8. Для автозапуска сервиса вместе с системой **systemctl** имеет следующую команду

```
razli@LAPTOP-HCMN9419:~$ sudo systemctl enable testd  
Created symlink /etc/systemd/system/multi-user.target.wants/
```

Системное программирование

Лекция 6

Сервисы