Санкт-Петербургский политехнический университет Петра Великого
Кафедра компьютерных систем и программных технологий

Отчет по дисциплине
«Проектирование ОС и их компонентов»

Сервисы/Службы/Демоны (C/C++) под Windows/Linux

Работу выполнил студент группы №: 13541/3 Чеботарёв М. М.
Работу принял преподаватель: _____ Душутина Е. В.

Санкт-Петербург
2017 г.

1. Используемая система и версия ядра

а) Windows

```
Тип системы:      Windows 7 Ultimate Compact (2009) Service Pack 1. x64.
```

б) Linux

```
michael@michael-LIFEBOOK-AH531:~$ lsb_release  -a
No LSB modules are available.
Distributor ID:   Ubuntu
Description:      Ubuntu 16.04.1 LTS
Release:          16.04
Codename:         xenial


michael@michael-LIFEBOOK-AH531:~$ cat /proc/version
Linux version 4.4.0-38-generic (buildd@lgw01-58) (gcc version 5.4.0 20160609
(Ubuntu 5.4.0-6ubuntu1~16.04.2) ) #57-Ubuntu SMP Tue Sep 6 15:42:33 UTC 2016
```

2. Создание сервиса на основе утилиты в Windows

2.1. Модификация и дополнение кода Win-утилиты

Для полного погружения в тему, и осознания, как создается, работает, конфигурируется и управляется сервис, одной программы сервиса недостаточно, поэтому работа включает в себя 3 программы:

- **myServiceUSB.exe** – название разработанного сервиса;
- **SvcConfig.exe** – программа для задания конфигурации сервиса (программно);
- **SvcControl.exe** – программа для управления сервисом (программно).

За основу взята статья с официального сайта Microsoft[1].

myServiceUSB.exe. Сервис для контроля подключенных устройств по USB. Файл со списком подключенных USB-устройств находится по пути «C:\usbService\DevicesInfo.txt». Хабы и хост-контроллеры в список не включаются. Программа поддерживается команду **install**, производящую установку (подробнее в пунтке 2.2).

Листинг №2. myServiceUSB.cpp

```
// libs for service
#include <windows.h>
#include <tchar.h>
#include <strsafe.h>
#include "sample.h"

// libs for USB function
#include <iomanip>
#include <clocale>
#include <iostream>
#include "myWinUsb.h"
```

```
using namespace ::std;

#pragma comment(lib, "advapi32.lib")

#define SVCNAME TEXT("myService")

const int REAL_MAX_PATH = 1024;
SERVICE_STATUS          gSvcStatus;
SERVICE_STATUS_HANDLE   gSvcStatusHandle;
HANDLE                  ghSvcStopEvent = NULL;

VOID SvcInstall(void);
VOID WINAPI SvcCtrlHandler(DWORD);
VOID WINAPI SvcMain(DWORD, LPTSTR *);
VOID ReportSvcStatus(DWORD, DWORD, DWORD);
VOID SvcInit(DWORD, LPTSTR *);
VOID SvcReportEvent(LPTSTR);


void __cdecl _tmain(int argc, TCHAR *argv[]) {
	// If command-line parameter is "install", install the service.
	// Otherwise, the service is probably being started by the SCM.

	if (lstrcmpi(argv[1], TEXT("install")) == 0)
	{
		SvcInstall();
		return;
	}

	// TO_DO: Add any additional services for the process to this table.
	SERVICE_TABLE_ENTRY DispatchTable[] =
	{
		{ SVCNAME, (LPSERVICE_MAIN_FUNCTION)SvcMain },
		{ NULL, NULL }
	};

	// This call returns when the service has stopped.
	// The process should simply terminate when the call returns.
	if (!StartServiceCtrlDispatcher(DispatchTable))
	{
		SvcReportEvent(TEXT("StartServiceCtrlDispatcher"));
	}
}

VOID SvcInstall() {
	SC_HANDLE schSCManager;
	SC_HANDLE schService;
	WCHAR szPath[REAL_MAX_PATH];

	if (!GetModuleFileName(GetModuleHandle(NULL), szPath, REAL_MAX_PATH)) {
		printf("Cannot install service (%d)\n", GetLastError());
		return;
	}
```

```
	// Get a handle to the SCM database.
	schSCManager = OpenSCManager(
		NULL,                    // local computer
		NULL,                    // ServicesActive database
		SC_MANAGER_ALL_ACCESS);  // full access rights

	if (NULL == schSCManager)
	{
		printf("OpenSCManager failed (%d)\n", GetLastError());
		return;
	}

	// Create the service
	schService = CreateService(
		schSCManager,              // SCM database
		SVCNAME,                   // name of service
		SVCNAME,                   // service name to display
		SERVICE_ALL_ACCESS,        // desired access
		SERVICE_WIN32_OWN_PROCESS, // service type
		SERVICE_DEMAND_START,      // start type
		SERVICE_ERROR_NORMAL,      // error control type
		szPath,                    // path to service's binary
		NULL,                      // no load ordering group
		NULL,                      // no tag identifier
		NULL,                      // no dependencies
		NULL,                      // LocalSystem account
		NULL);                     // no password

	if (schService == NULL) {
		printf("CreateService failed (%d)\n", GetLastError());
		CloseServiceHandle(schSCManager);
		return;
	}
	else printf("Service installed successfully\n");

	CloseServiceHandle(schService);
	CloseServiceHandle(schSCManager);
}


VOID WINAPI SvcMain(DWORD dwArgc, LPTSTR *lpszArgv) {
	// Register the handler function for the service
	gSvcStatusHandle = RegisterServiceCtrlHandler(
		SVCNAME,
		SvcCtrlHandler);

	if (!gSvcStatusHandle)
	{
		SvcReportEvent(TEXT("RegisterServiceCtrlHandler"));
		return;
	}

	// These SERVICE_STATUS members remain as set here
```

```
	gSvcStatus.dwServiceType = SERVICE_WIN32_OWN_PROCESS;
	gSvcStatus.dwServiceSpecificExitCode = 0;

	// Report initial status to the SCM
	ReportSvcStatus(SERVICE_START_PENDING, NO_ERROR, 3000);

	// Perform service-specific initialization and work.
	SvcInit(dwArgc, lpszArgv);
}


VOID SvcInit(DWORD dwArgc, LPTSTR *lpszArgv) {
	// Create an event. The control handler function, SvcCtrlHandler,
	// signals this event when it receives the stop control code.
	ghSvcStopEvent = CreateEvent(
		NULL,    // default security attributes
		TRUE,    // manual reset event
		FALSE,   // not signaled
		NULL);   // no name

	if (ghSvcStopEvent == NULL)
	{
		ReportSvcStatus(SERVICE_STOPPED, NO_ERROR, 0);
		return;
	}

	// Report running status when initialization is complete.
	ReportSvcStatus(SERVICE_RUNNING, NO_ERROR, 0);

	// TO_DO: Perform work until service stops.
	DEVICE_HUB_LIST ask;
	int countOfDevicesOld = -1, countOfDevices = 0;
	while (1)
	{
		// check signal from Service Manager every 1 second
		if (WaitForSingleObject(ghSvcStopEvent, 1000) == WAIT_OBJECT_0) {
			ReportSvcStatus(SERVICE_STOPPED, NO_ERROR, 0);
			return;
		}
		// get info about number of hubs and devices
		getCountInfo(&ask);

		// compare count of devices
		if (ask.countOfDevices + ask.countOfDevices == countOfDevicesOld)
			continue;
		countOfDevicesOld = ask.countOfDevices + ask.countOfDevices;

		// print inf about all external USB devices
		printDevicesInfo("C:\\usbService\\DevicesInfo.txt");
	}
}
```

```
...
```

SvcControl.exe. Программа для выполнения контроля над сервисом (листинг 2) позволяет выполнять три действия:

- запускать/останавливать сервис (команды start/stop);
- изменять права доступа к нему (команда dacl).

Синтаксис вызовов: **SvcControl.exe** [command] [service_name]

Листинг №2. SvcControl.cpp

```
#include <windows.h>
#include <tchar.h>
#include <strsafe.h>
#include <aclapi.h>
#include <stdio.h>

#pragma comment(lib, "advapi32.lib")

TCHAR szCommand[10];
TCHAR szSvcName[80];

SC_HANDLE schSCManager;
SC_HANDLE schService;

VOID __stdcall DisplayUsage(void);
VOID __stdcall DoStartSvc(void);
VOID __stdcall DoUpdateSvcDacl(void);
VOID __stdcall DoStopSvc(void);
BOOL __stdcall StopDependentServices(void);

void _tmain(int argc, TCHAR *argv[])
{
	printf("\n");
	if (argc != 3)
	{
		printf("ERROR: Incorrect number of arguments\n\n");
		DisplayUsage();
		return;
	}

	StringCchCopy(szCommand, 10, argv[1]);
	StringCchCopy(szSvcName, 80, argv[2]);

	if (lstrcmpi(szCommand, TEXT("start")) == 0)
		DoStartSvc();
	else if (lstrcmpi(szCommand, TEXT("dacl")) == 0)
		DoUpdateSvcDacl();
	else if (lstrcmpi(szCommand, TEXT("stop")) == 0)
		DoStopSvc();
	else
	{
		_tprintf(TEXT("Unknown command (%s)\n\n"), szCommand);
```

```
            DisplayUsage();
      }
}

...
```

SvcConfig.exe. Программа для задания конфигурации сервиса (листинг 3) позволяет выполнят следующие действия:

- получать конфигурацию сервиса (команда **query**);
- устанавливать значение описания для сервиса (команда **describe**);
- включать (в режиме «в ручную»)/отключать сервис из автозапуска (команды **enable/disable**);
- удалять сервис из реестра, из списка Менеджера Сервисов (команда **delete**).

Синтаксис вызовов: **SvcConfig.exe** [command] [service_name].

Листинг №3. SvcConfig.cpp

```
#include <windows.h>
#include <tchar.h>
#include <strsafe.h>
#include <stdio.h>

#pragma comment(lib, "advapi32.lib")

TCHAR szCommand[10];
TCHAR szSvcName[80];

VOID __stdcall DisplayUsage(void);
VOID __stdcall DoQuerySvc(void);
VOID __stdcall DoUpdateSvcDesc(void);
VOID __stdcall DoDisableSvc(void);
VOID __stdcall DoEnableSvc(void);
VOID __stdcall DoDeleteSvc(void);

void __cdecl _tmain(int argc, TCHAR *argv[]) {
      printf("\n");
      if (argc != 3) {
            printf("ERROR:\tIncorrect number of arguments\n\n");
            DisplayUsage();
            return;
      }

      StringCchCopy(szCommand, 10, argv[1]);
      StringCchCopy(szSvcName, 80, argv[2]);

      if (lstrcmpi(szCommand, TEXT("query")) == 0)
            DoQuerySvc();
      else if (lstrcmpi(szCommand, TEXT("describe")) == 0)
            DoUpdateSvcDesc();
      else if (lstrcmpi(szCommand, TEXT("disable")) == 0)
```

```
            DoDisableSvc();
      else if (lstrcmpi(szCommand, TEXT("enable")) == 0)
            DoEnableSvc();
      else if (lstrcmpi(szCommand, TEXT("delete")) == 0)
            DoDeleteSvc();
      else {
            _tprintf(TEXT("Unknown command (%s)\n\n"), szCommand);
            DisplayUsage();
      }
}

...
```

Листинг №4. Sample.cpp

```
MessageIdTypedef=DWORD

SeverityNames=(Success=0x0:STATUS_SEVERITY_SUCCESS
    Informational=0x1:STATUS_SEVERITY_INFORMATIONAL
    Warning=0x2:STATUS_SEVERITY_WARNING
    Error=0x3:STATUS_SEVERITY_ERROR
    )


FacilityNames=(System=0x0:FACILITY_SYSTEM
    Runtime=0x2:FACILITY_RUNTIME
    Stubs=0x3:FACILITY_STUBS
    Io=0x4:FACILITY_IO_ERROR_CODE
)

LanguageNames=(English=0x409:MSG00409)

; // The following are message definitions.

MessageId=0x1
Severity=Error
Facility=Runtime
SymbolicName=SVC_ERROR
Language=English
An error has occurred (%2).
.

; // A message file must end with a period on its own line
; // followed by a blank line.
```

2.2. Сборка сервиса

Следующие шаги помогут подготовить все исполняемые файлы и записи в реестре системы:

1. Сначала необходимо собрать динамическую библиотеку (DLL) на основе sample.mc. Для этого выполняем следующие команды:

```
mc -U sample.mc
rc -r sample.rc
link -dll -noentry -out:sample.dll sample.res
```

Примечание: mc – message compiler; rc – resources compiler; link – линковщик, необходимый для создания DLL. Все перечисленные утилиты присутствуют в системе.

Примеры возможных путей (искать через «пуск»):

```
C:\Microsoft Visual Studio\Shared\14.0\VC\bin\link.exe;
C:\Program Files (x86)\Windows Kits\10\bin\x64\mc.exe;
C:\Program Files (x86)\Windows Kits\8.1\bin\x64\rc.exe;
```

2. Скомпилировать `Svc.exe, SvcConfig.exe, и SvcControl.exe` из Svc.cpp, SvcConfig.cpp SvcControl.cpp соответственно;

3. Исполнить следующие команды, для добавления записи сервиса (myService) в реестр системы:

а) добавление параметра "**`EventMessageFile`**", хранящего путь к **`sample.dll`** (тип **`REG_DWORD):`**

```
REG ADD
HKLM\SYSTEM\CurrentControlSet\services\eventLog\Application\mySer
vice /v EventMessageFile /t REG_SZ /d "C:\sample.dll"
```

б) добавление параметра "**`TypesSupported`**" – маска событий (тип **`REG_SZ)`**, 0x07 – реагировать на все события:

```
REG ADD
HKLM\SYSTEM\CurrentControlSet\services\eventLog\Application\mySer
vice /v TypesSupported /t REG_DWORD /d 0x00000007
```

Примечание: сначала убедиться, что "`sample.dll`" присутствует по указанным путям. В результате в системном реестре должна добавиться запись о сервисе с двумя полями (см. рис..1).

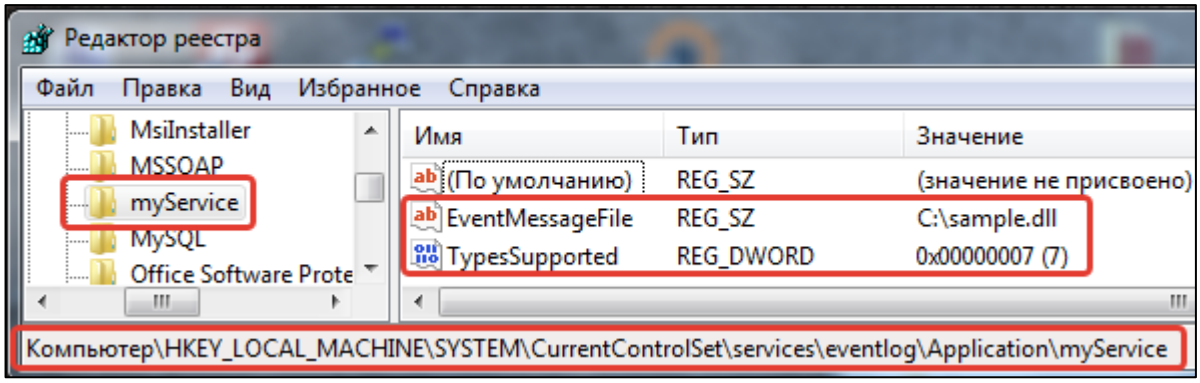


Рисунок 1. Системный реестр. Выделены красным: **сервис**, его **поля** и **путь**

2.3. Тестирование работы

1. Установка

Выполним запуск программы с параметром install: **myService.exe install**

Успех	D:\> myService.exe install Service installed **successfully**
Потрачено	D:\ > myService.exe install **Cannot install service (126)**

Примечание: сначала все же проверить правильность предыдущих шагов, а затем проверить возможное решение **(см. источник[2]).** Так же возможна установка с помощью утилиты **sc** (service control): **“`sc create SampleService binpath= c:\myService.exe`”**

В случае успеха в менеджере служб должна добавиться новая служба – **myService** (см. рис.2).

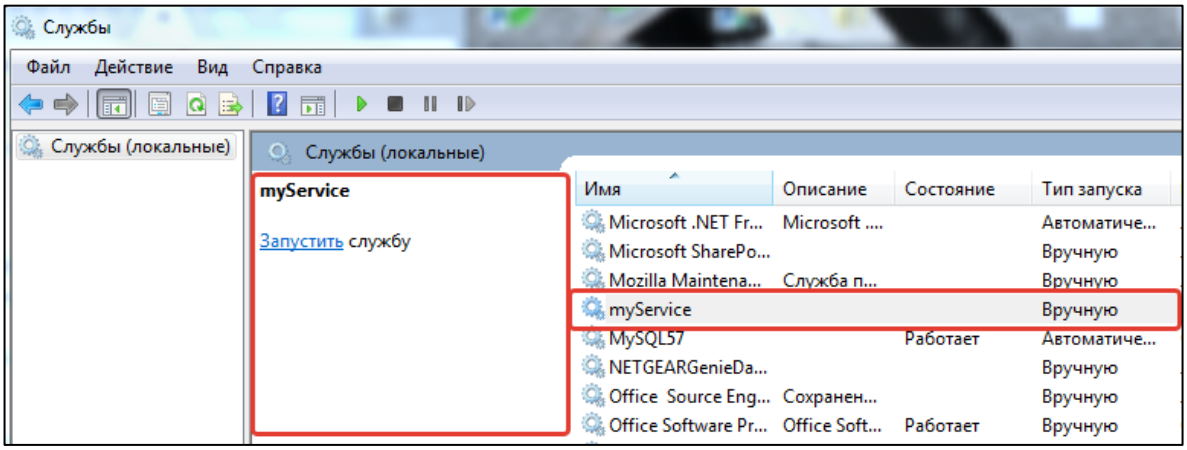


Рисунок 2. Добавленный сервис в списке установленных служб

2. Запуск службы

Теперь запустим сервис с помощью второй разработанной программы (**`svcControl.exe`**), с флагом **`start.`** В результате исполнения в мы видим:

```
D:\ > svcControl.exe start myService
Service start pending...
Service started successfully.
```

А в списке установленных сервисов так же изменилось состояние сервиса:

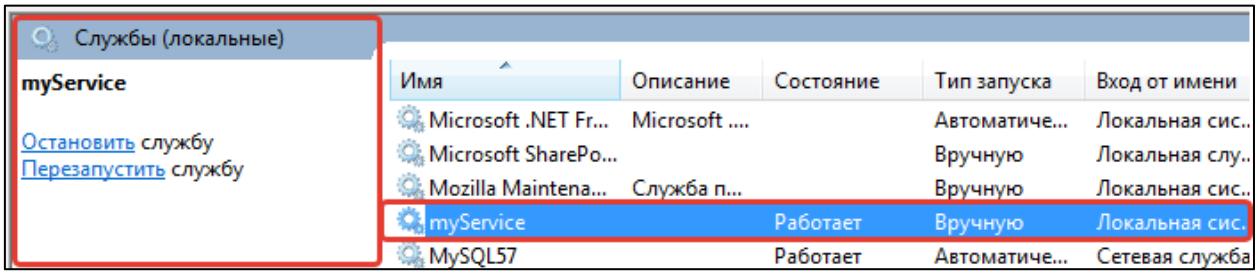


Рисунок 3. Сервис работает

Код, прописанный в теле функции SeviceMain(), начал исполняться сервис менеджером. В случае возникновения ошибок сервис запишет в лог название функции, в которой произошла ошибка, а так же ее код.

3. Добавление описания

Для того чтобы добавить описание нашего сервиса (в менеджере сервисов), следует использовать третью программу **`svcconfig`** с параметром **`describe`**. В результате исполнения мы видим:

```
D:\> svcConfig.exe describe myService
Service description updated successfully.
```

А текстовое описание сервиса изменилось на "This is a test description", текст жестко задан внутри функции **`DoUpdateSvcDesc().`**

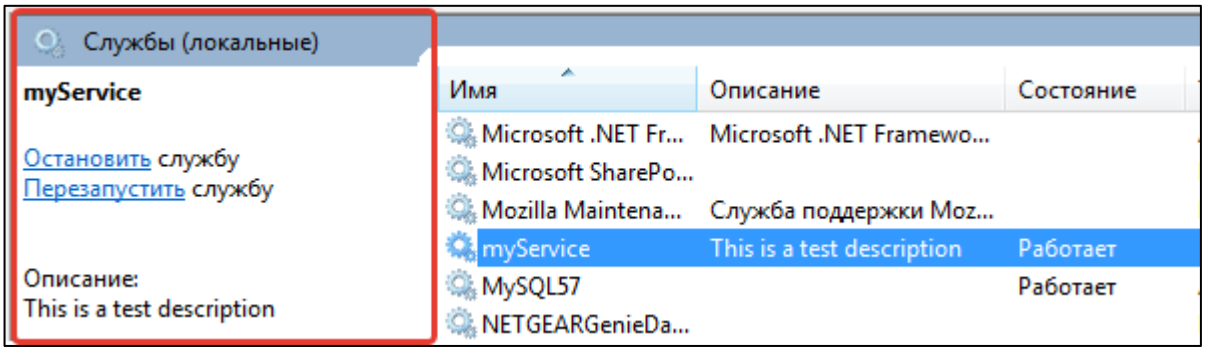


Рисунок 4. Менеджер служб. Текстовое описание myService изменилось

4. Получение информации о конфигурации сервиса

Выполним следующую команду: **svcconfig.exe query SvcName.** В результате в консоль будет выведена вся доступная информация о созданном сервисе (то есть конфигурация сервиса).

Результат:

```
D:\> svcConfig.cpp.exe query myService
myService configuration:
  Type: 0x10
  Start Type: 0x3
  Error Control: 0x1
  Binary path: D:\mySerice.exe
  Account: LocalSystem
  Description: This is a test description
```

5. Редактирование DACL (discretionary access control list)

Access Control Lists (ACL) - это список контроля доступа, состоящий из записей управления доступом (ACE, Access Control Entries), каждая запись содержит только одно указание (SID, security identifier) на доверенное лицо (**trustee**), то есть на пользователя, группу или сессию. Так же каждая запись содержит права доступа, запрещающие, разрешающие или зарезервированные за данной записью управления доступом.

Discretionary Access Control List (DACL) – это список доверенных лиц (**trustee**), которым позволено использовать данные объект (службу). Если DACL пуст – доступ к объекту имеют все.

Для редактирования списка доверенный лиц используем следующий вызов:

```
svcControl.exe dacl myService
```

Результат:

```
D:\>svcControl.exe dacl myService
Service DACL updated successfully
```

Примечание: в коде программы прописаны следующие правила доступа для 1го пользователя michael: SERVICE_START | SERVICE_STOP | READ_CONTROL | DELETE. Теперь такие права установлены только за пользователем michael, другие пользователи не имеют прав абсолютно.

6. Исключение/Включение сервиса (disable/enable)

Для того, чтобы исключить сервис используем команду: svcConfig.exe disable myService.

Результат исполнения (логи):

```
D:\> svcConfig.exe disable myService

Service disabled successfully.
```

В результате выполнения команды сервис был удален из автозапуска, но продолжает выполнять свою задачу (см. рис. 5).

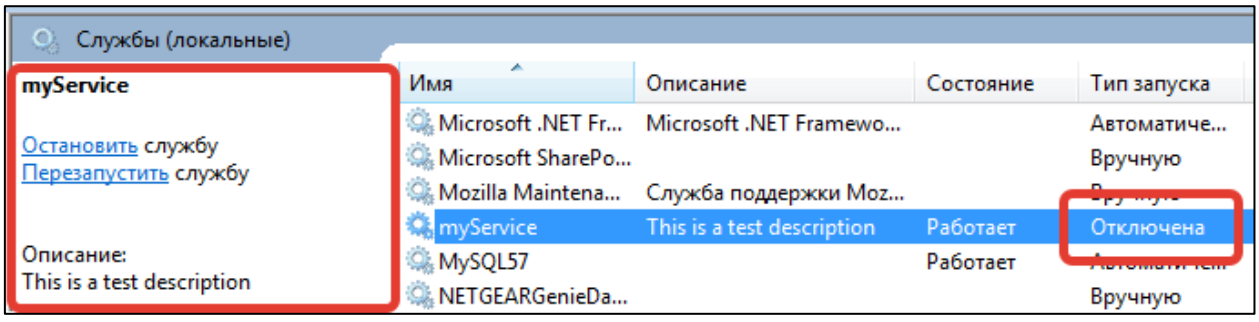


Рисунок 5. Менеджер служб. Служба работает, но удалена из автозапуска

Для включения сервиса в список автозагрузки выполним: svcConfig.exe enable myService.

Результат исполнения (логи):

```
D:\> svcConfig.exe enable myService

Service enabled successfully.
```

Теперь сервис запускается в ручную (см. рис. 6).

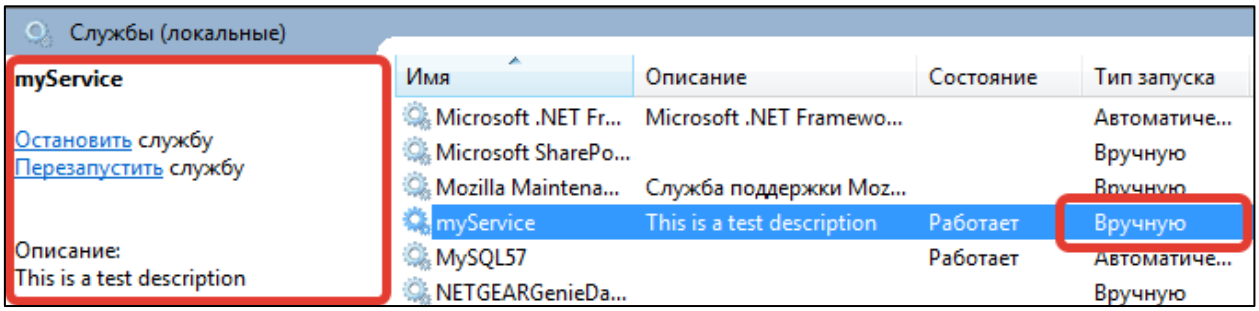


Рисунок 6. Менеджер служб. Служба работает, и запускается вручную

Примечание: для того, чтобы изменить способ запуска сервис используется функция **`ChangeServiceConfig(),`** которая позволяет изменять конфигурацию сервиса, в том числе: способ запуска сервиса, путь в бинарному файлу, имя сервиса, пароль для управления и др. Ниже приведен пример вызова функции, устанавливающий способ запуска сервиса **`SERVICE_DISABLED.`**

```
   ChangeServiceConfig(
      schService,         // handle of service
      SERVICE_NO_CHANGE, // service type: no change
      SERVICE_DISABLED,  // service start type
      SERVICE_NO_CHANGE, // error control: no change
      NULL,              // binary path: no change
      NULL,              // load order group: no change
      NULL,              // tag ID: no change
      NULL,              // dependencies: no change
      NULL,              // account name: no change
      NULL,              // password: no change
      NULL));            // display name: no change
```

7. Остановка и удаление сервиса

Для **остановки** выполним команду: `svcControl.exe stop myService`
для **удаления:** `svcconfig.exe delete myService`

В результате:

```
D:\> svcControl.exe stop myService

Service stopped successfully
D:\>
D:\> svcConfig.exe delete myService

Service deleted successfully
```

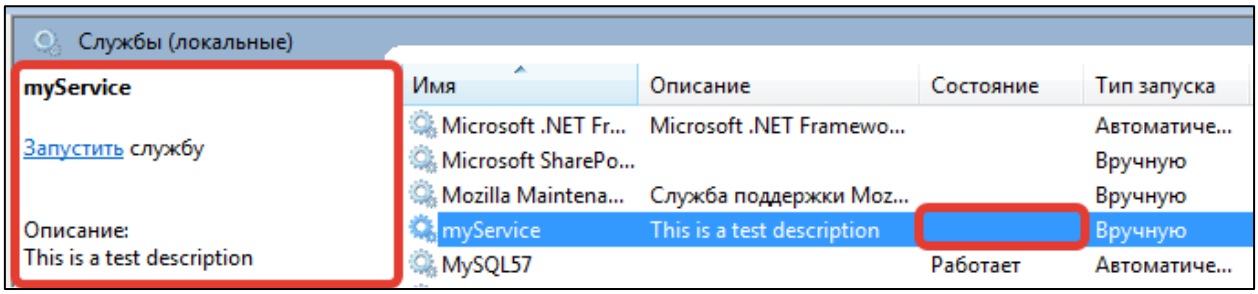


Рисунок 7. Менеджер служб. Сервис остановлен

Примечание: если при попытке остановить сервис возникли ошибки – см. логи менеджера служб.

8. Service Control (sc.exe) [4]

Установить и удалить сервис так же возможно с помощью системной утилиты sc.exe:

```
sc create myService binpath= c:\SampleService.exe
sc delete myService
```

9. Оценка работы сервиса

Сервис работает по следующему принципу: каждую секунду происходит опрос реестра системы на количество подключенных usb hub`ов и usb устройств. Если количество подключенных устройств отличается от количества устройств, выявленных 1 секунду назад – происходит опрос всех подключенных по USB шине устройств, а информация заносится в файл «C:\usbService\DevicesInfo.txt».

Результат исполнения:

Стартовый набор устройств	После удаления мышки, клавиатуры и внешней аудиокарты
---------- Device Type: DeviceInfo type of device idVendor: 0x41e iProduct: 0x2 bcdDevice: 0x1001 iManufacturer: 0x1 DeviceDriverName: 0065A568 ---------- Device Type: DeviceInfo type of device idVendor: 0x1005 iProduct: 0x2 bcdDevice: 0x100 iManufacturer: 0x1 DeviceDriverName: 0065A6E8 ---------- Device Type: DeviceInfo type of device	---------- ---------- ---------- ---------- Device Type: DeviceInfo type of device DeviceClass: LibUsbDevices DeviceDesc: Intel(R) Centrino(R) Wireless Bluetooth(R) 3.0 + High Speed Adapter DeviceId: USB\VID_8086&PID_0189\6&34BF07AE&0&4 HwId: USB\VID_8086&PID_0189&REV_6919 Service: libusb0 idVendor: 0x8086 iProduct: 0x0 bcdDevice: 0x6919 iManufacturer: 0x0 DeviceDriverName: 0065A748

DeviceClass: USB
DeviceDesc: Составное USB устройство
DeviceId: USB\VID_09DA&PID_054F\6&34BF07AE&0&3
HwId: USB\VID_09DA&PID_054F&REV_0274
Service: usbccgp
idVendor: 0x9da
iProduct: 0x2
bcdDevice: 0x274
iManufacturer: 0x1
DeviceDriverName: 0065A628

Device Type: DeviceInfo type of device
DeviceClass: LibUsbDevices
DeviceDesc: Intel(R) Centrino(R) Wireless Bluetooth(R) 3.0 + High Speed Adapter
DeviceId: USB\VID_8086&PID_0189\6&34BF07AE&0&4
HwId: USB\VID_8086&PID_0189&REV_6919
Service: libusb0
idVendor: 0x8086
iProduct: 0x0
bcdDevice: 0x6919
iManufacturer: 0x0
DeviceDriverName: 0065A748

Device Type: DeviceInfo type of device
DeviceClass: USB
DeviceDesc: Запоминающее устройство для USB
DeviceId: USB\VID_0BDA&PID_0138\20090516388200000
HwId: USB\VID_0BDA&PID_0138&REV_3882
Service: USBSTOR
idVendor: 0xbda
iProduct: 0x2
bcdDevice: 0x3882
iManufacturer: 0x1
DeviceDriverName: 0065A688

Device Type: DeviceInfo type of device
DeviceClass: USB
DeviceDesc: Составное USB устройство
DeviceId: USB\VID_04F2&PID_B213\SN0001
HwId:

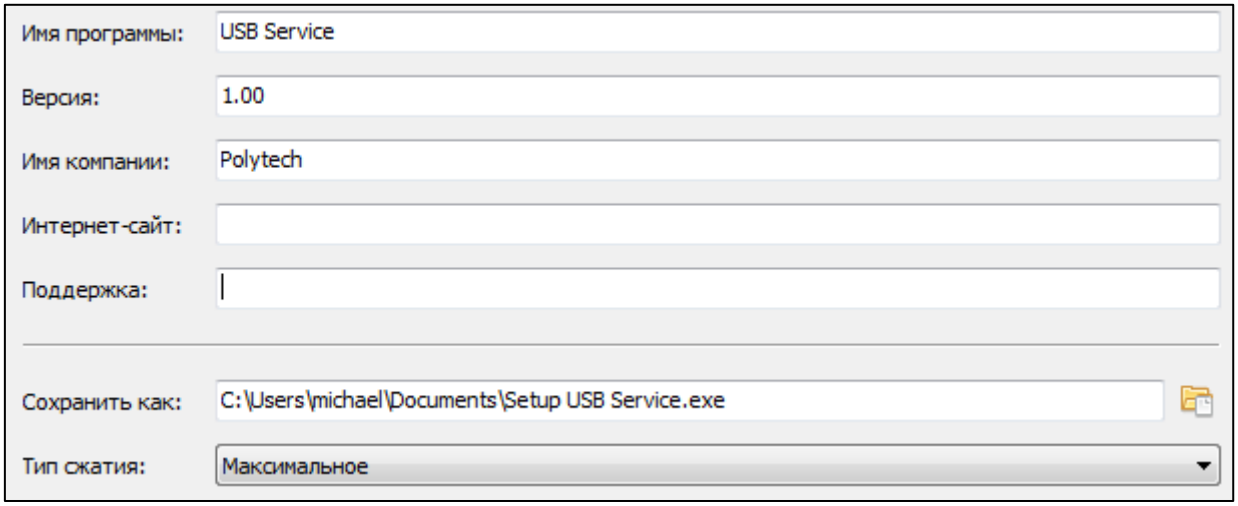
Device Type: DeviceInfo type of device
DeviceClass: USB
DeviceDesc: Запоминающее устройство для USB
DeviceId: USB\VID_0BDA&PID_0138\20090516388200000
HwId: USB\VID_0BDA&PID_0138&REV_3882
Service: USBSTOR
idVendor: 0xbda
iProduct: 0x2
bcdDevice: 0x3882
iManufacturer: 0x1
DeviceDriverName: 0065A688

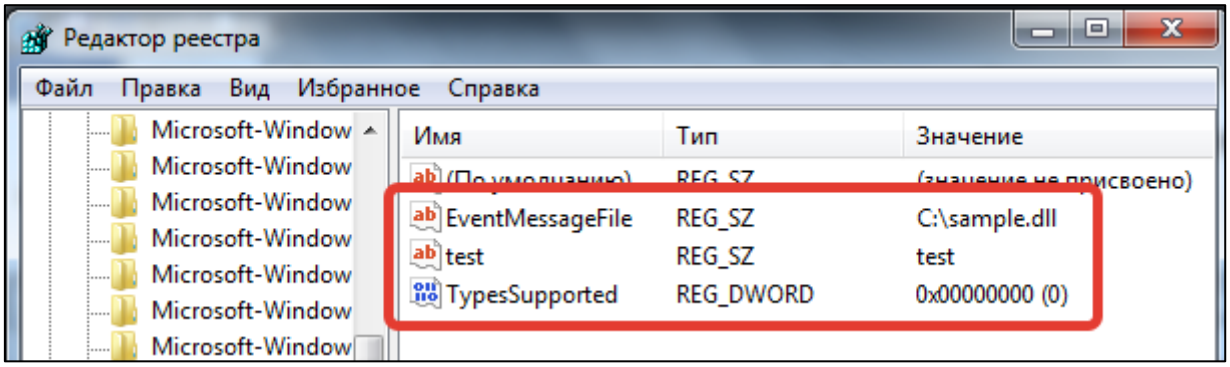
Device Type: DeviceInfo type of device
DeviceClass: USB
DeviceDesc: Составное USB устройство
DeviceId: USB\VID_04F2&PID_B213\SN0001
HwId: USB\VID_04F2&PID_B213&REV_3761
Service: usbccgp
idVendor: 0x4f2
iProduct: 0x1
bcdDevice: 0x3761
iManufacturer: 0x2
DeviceDriverName: 0065A5C8

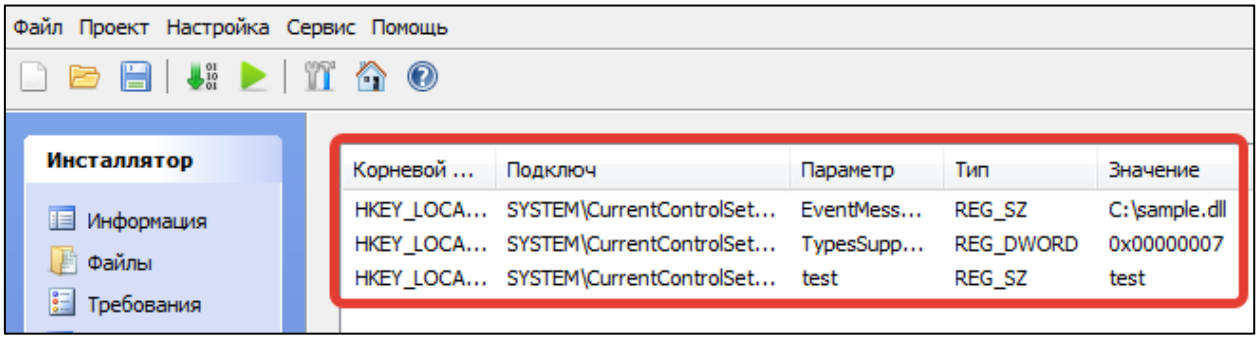

```
USB\VID_04F2&PID_B213&REV_3761
Service:              usbccgp
idVendor:             0x4f2
iProduct:             0x1
bcdDevice:            0x3761
iManufacturer:        0x2
DeviceDriverName:     0065A5C8

----------------------------------------
----------------------------------------
----------------------------------------
----------------------------------------
```

2.4. Smart Install Maker 5.04

Конфигурирование инсталлятора







3. Создание сервиса на основе утилиты в Linux

3.1. Описание большинства возможностей system в частности при работе с сервисом

Примечание: демон (daemon), сервис или служба – это процесс, зарегистрированный в менеджере системы (systemd, system V), и запускаемый в фоновом режиме. Каждый демон имеет файл конфигурации, по которому системный менеджер определяет, как запускать, останавливать и вообще управлять данной службой.

systemctl status предоставляет список всех служб,с указанием состояний, как «родных» (native) для systemd, так и классических SysV/LSB служб, поддерживаемых в целях совместимости. Если нужно для конкретной службы, стоит указать имя службы:

```
michael@michael-LIFEBOOK-AH531:~$ systemctl status windows.mount
● windows.mount - /windows
   Loaded: loaded (/etc/fstab; bad; vendor preset: enabled)
   Active: active (mounted) since Сб 2017-04-08 13:58:18 MSK; 45min ago
    Where: /windows
     What: /dev/sda1
     Docs: man:fstab(5)
           man:systemd-fstab-generator(8)
  Process: 510 ExecMount=/bin/mount /dev/disk/by-uuid/7AC860AAC86065FB /windows -t
ntfs -o defaults,umask=007,gid=46 (code=ex
   CGroup: /system.slice/windows.mount
           └─534 /sbin/mount.ntfs /dev/sda1 /windows -o rw,umask=007,gid=46

апр 08 13:58:18 michael-LIFEBOOK-AH531 systemd[1]: Mounting /windows...
апр 08 13:58:18 michael-LIFEBOOK-AH531 systemd[1]: Mounted /windows.
апр 08 13:58:18 michael-LIFEBOOK-AH531 ntfs-3g[534]: Version 2015.3.14AR.1 integrated
FUSE 28
апр 08 13:58:18 michael-LIFEBOOK-AH531 ntfs-3g[534]: Mounted /dev/sda1 (Read-Write,
label "", NTFS 3.1)
апр 08 13:58:18 michael-LIFEBOOK-AH531 ntfs-3g[534]: Cmdline options:
rw,umask=007,gid=46
апр 08 13:58:18 michael-LIFEBOOK-AH531 ntfs-3g[534]: Mount options:
rw,allow_other,nonempty,relatime,default_permissions,fsna
апр 08 13:58:18 michael-LIFEBOOK-AH531 ntfs-3g[534]: Global ownership and permissions
enforced, configuration type 7
апр 08 13:58:18 michael-LIFEBOOK-AH531 ntfs-3g[534]: Warning : using problematic
uid==0 and gid!=0
```

3.1.1. Контрольные группы и прочее

В Linux множество процессов, и понять какой кем был запущен и как что починить, если оно сломалось тяжело. Довольно часто процессы вызывают другие (например при старте Cron испольняет список команд, прописанных в crontab), отследить иерархию тяжело. Одной из отличительных черт systemd является то, запуская новый процесс, systemd помещает его в отдельную контрольную группу с соответствующим именем. Когда какой-либо процесс порождает потомка, этот потомок автоматически включается в ту же группу, что и родитель. При этом, что очень важно, непривилегированные процессы не могут изменить свое положение в этой иерархии. Таким образом, контрольные группы позволяют точно установить происхождение конкретного процесса, вне зависимости от того, сколько раз он форкался и переименовывал себя — имя его контрольной группы невозможно спрятать или изменить. Кроме того, при штатном завершении родительской службы, будут заверше ны и все порожденные ею процессы, как бы они ни пытались

сбежать. С systemd уже невозможна ситуация, когда после остановки web-сервера, некорректно форкнувшийся CGI-процесс продолжает исполняться вплоть до последних секунд работы системы.

Для того, **чтобы увидеть иерархию контрольных групп**, следует использовать команду systemd-cgls

```
michael@michael-LIFEBOOK-AH531:~$ systemd-cgls
Control group /:
-.slice
├─init.scope
│ └─1 /sbin/init splash
├─system.slice
│ ├─avahi-daemon.service
│ │ ├─1444 avahi-daemon: running [michael-LIFEBOOK-AH531.local
│ │ └─1446 avahi-daemon: chroot helpe
│ ├─windows.mount
│ │ └─534 /sbin/mount.ntfs /dev/sda1 /windows -o rw,umask=007,gid=46
```

Примечание: systemd занимается отслеживанием и группировкой не только процессов, относящихся к системным службам, но и процессов, запущенных в рамках пользовательских сеансов (/user/lennart/1).

Стоит особо отметить, что использование контрольных групп не только упрощает процесс уничтожения форк-бомб, но и значительно уменьшает ущерб от работающей форк-бомбы. Так как systemd автоматически помещает каждую службу и каждый пользовательский сеанс в свою контрольную группу по ресурсу процессорного времени, запуск форк-бомбы одним пользователем или службой не создаст значительных проблем с отзывчивостью системы у других пользователей и служб. Таким образом, в качестве основной угрозы форк-бомбардировки остаются лишь возможности исчерпания памяти и идентификаторов процессов (PID). Впрочем, и их тоже можно легко устранить: достаточно задать соответствующие лимиты в конфигурационном файле службы (см. systemd.exec(5) и systemd.resource-control(5)).

В чем разницамежду systemctl kill и systemctl stop?

Отличие состоит в том, что kill просто отправляет сигнал заданному процессу, в то время как stop действует по «официально» определенному методу, вызывая команду, определенную в параметре ExecStop конфигурации службы.

Три способа выключения демона (на примере демона ntpd.service)

1. systemctl stop ntpd.service. Простая остановка службы. Оставляет возможность запустить сервис аналогичной stop, командой start (в ручную). Сервис может быть перезапущен автоматически при загрузке системы, при поступлении запроса через сокет или системную шину, при срабатывании таймера, при подключении соответствующего оборудования и т.д.

2. systemctl disable ntpd.service. Позволяет отключить службу, то есть отсоединить ее от всех триггеров активации. В результате служба уже не будет автоматически запускаться ни при загрузке системы, ни при обращении к сокету или адресу на шине, ни при подключении оборудования, и т.д. Но при этом сохраняется возможность «ручного» запуска службы (командой systemctl start). Важно отметить, что служба продолжает работу до выключения ОС, либо до получения команды **systemctl stop.**

***Примечание*:** отключение службы является перманентной мерой, и действует вплоть до явной отмены соответствующей командой. Перезагрузка системы не отменяет отключения службы.

`3.ln -s /dev/null /etc/systemd/system/ntpd.service` уничтожение ссылки.

Как отменить произведенные изменения, и реанимировать демона?

systemctl **start** отменяет действия systemctl **stop**
systemctl **enable** отменяет действие systemctl **disable**
а **rm** отменяет действие **ln**.

3.1.2. Новые конфигурационные файлы

/etc/hostname: имя хоста для данной системы. Одна из наиболее простых и важных системных настроек. В разных дистрибутивах оно настраивалось по-разному: Fedora использовала /etc/sysconfig/network, OpenSUSE — /etc/HOSTNAME, Debian — etc/hostname. Мы остановились на варианте, предложенном Debian. **/etc/vconsole.conf:** конфигурация раскладки клавиатуры и шрифта для консоли.

/etc/locale.conf: конфигурация общесистемной локали. /etc/modules-load.d/*.conf: каталог 35 для перечисления модулей ядра, которые нужно принудительно подгрузить при загрузке (впрочем, необходимость в этом возникает достаточно редко).

/etc/sysctl.d/*.conf: каталог для задания параметров ядра (sysctl). Дополняет классический конфигурационный файл /etc/sysctl.conf.

/etc/tmpfiles.d/*.conf: каталог для управления настройками временных файлов (systemd обеспечивает создание, очистку и удаление временных файлов и каталогов, как во время загрузки, так и во время работы системы).

/etc/binfmt.d/*.conf: каталог для регистрации дополнительных бинарных форматов (например, форматов Java, Mono, WINE).

/etc/os-release: стандарт для файла, обеспечивающего идентификацию дистрибутива и его версии. Сейчас различные дистрибутивы используют для этого разные файлы (например, /etc/fedora-release в Fedora), и поэтому для решения такой простой задачи, как вывод имени дистрибутива, необходимо использовать базу данных, содержащую перечень возможных названий файлов. Проект LSB попытался создать такой инструмент — lsb_release — однако реализация столь простой функции через скрипт на Python'е является не самым оптимальным решением. Чтобы исправить сложившуюся ситуацию, мы решили перейти к единому простому формату представления этой информации.

/etc/machine-id: файл с идентификатором данного компьютера (перекрывает аналогичный идентификатор D-Bus). Гарантируется, что в любой системе, использующей systemd, этот файл будет существовать и содержать корректную информацию (если его нет, он автоматически создается при загрузке). Мы вынесли этот файл из-под эгиды D-Bus, чтобы упростить решение множества задач, требующих наличия уникального и постоянного идентификатора компьютера.
/etc/machine-info: новый конфигурационный файл, хранящий информации о полном (описательном) имени хоста (например, «Компьютер Леннарта») и значке, которым он будет обозначаться в графических оболочках, работающих с сетью (раньше этот значок мог определяться, например, файлом /etc/favicon.png). Данный конфигурационный файл обслуживается демоном systemd-hostnamed.

3.2. Секции конфигурационного файла

NAME_OF_DAEMON.service – файл конфигурации демона; Данный файл используется systemd для управления и наблюдения за демоном NAME_OF_DAEMON.

Набор параметров, который может быть использован при конфигурации сервисов, сокетов, точек монтирования и устройств подкачки (swap). Набор параметров определяет, каким обрзом производить подготовку среды к запуску процесса.

Примечание**: основных сущностей четыре**: systemd.service, systemd.socket, systemd.swap и systemd.mount.

Автоматические зависимости

Некоторые параметры, такие как WorkingDirectory= or RootDirectory=, устанавливаются автоматически, в зависимости от Requires= and After=.

3.2.1. [UNIT] СЕКЦИЯ

Содержит общую информацию о модуле, не зависящую от его конкретного типа.

Примечание: типы модулей: сервис, сокет, точка монтирования и что-то еще.

Description=

Поле, а точнее строка, которая должна описывает информацию о модуле. Должно быть кратко и лаконично, чтобы конечный пользователь понимал, с чем имеет дело. Хороший пример: "Apache2 Web Server", плохой пример: "high-performance light-weight HTTP server" или “Apache2”

Documentation=

Список источников документации. Элементы разделяются через пробелы и могут быть следующих типов "http://", "https://", "file:", "info:", "man:".

Requires=

Зависимости модуля. Обязательные требования для стабильной работы. Например демон№1 в данном поле указывает демон№2, причем демон№2 не указан ни в поле After, ни в поле Before. Тогда при запуска демона№1 **одновременно** с ним стартует демон№2 (такой момент стоит учитывать).

Requisite=

Аналогично полю Requires=, однако если модуль, указанный в данном поле уже не был запущен, то его запуск производиться не будет, а вся попытка запуска (транзакция) текущего модуля будет считать провальной.

Wants=

Аналогично Requisite= (опция с более слабыми требованиями), но при если модуль не получиться запустить, то прерывание всей операции не наступит. Так же смотри WantedBy=, RequiredBy=.

PartOf=

Опция, которая указывает, что текущий модуль являет частью чего-то более крупного, другого модуля. Если systemd остановит или перезагрузит данный сервис, то тоже будет выполнено для модуля, указанного в данном поле. Однако связь односторонняя, перезагрузив или остановив модуль, указанный в этой опции, текущий модуль продолжит работу.

Conflicts=

Список тех модулей, что мешают свободно дышать нашему. При попытке запустить модуль, будет остановлен конфликтующий модуль, и наоборот.

Before=, After=

Список модулей, требований, которые должны быть запущены ДО и ПОСЛЕ запуска текущего модуля. Данное поле независимо от поля Requires=, и ему подобных. Если вызвать команду отключается всех модулей, то оно производится задом наперёд.

AllowIsolate=

флаг. Указывает можно ли использовать команду **systemctl isolate** для изменения уровня изоляции модуля. По умолчанию false.

SourcePath=

Путь к конфигурационному файлу, который будет создан под данную систему на основе исходного конфига. Это, прежде всего, полезно для реализации инструментов генератора, которые преобразовывают конфигурацию из внешнего формата конфигурационного файла в собственные файлы модуля.

3.2.2. [INSTALL] СЕКЦИЯ

Секция, содержащая информацию о.. о установке, кэп.

Alias=

Список с именами модулей, разделенных через запятую, что обязаны быть установлены. Каждый из алиасов обязан иметь имя, префексом которого является имя текущего модуля. Опция не действует на точки монтирования (mount & automount), slice, swap. Вместо перечисления через пробел возможно использование этой же опции несколько раз. В момент установки, команда **systemctl enable** создаст символьные ссылки на модули с указанными в поле именами

WantedBy=, RequiredBy=

Список с именами модулей, разделенных через запятую. Опцию можно использовать несколько раз. Допустим где-то есть mainModule и текущий модуль curModule. У mainModule должно быть поле Wants или Required, в котором указано имя curModule, а у текущего модуля должно соответственно быть поле WantedBy= или RequiredBy=. Тогда, когда будет запущен mainModule – текущий будет запускать автоматически.

Also=

Список модулей, который будут установлен/удалены автоматически при установке текущего модуля.

3.3. [SERVICE] СЕКЦИЯ

3.3.1. Systemd.kill [5]

KillMode = [control-group, process, mixed, none]

***control-group* –** отключить все процессы из данной группы процессов **(process group)**, состоящей из его прямого родителя (и других предков), братьев и сестер, а также детей (и прочих потомков).

Примечание не по теме: процессы из одной группы можгут послать сигнал сразу всей своей группе за один системный вызов.

***process* –** только главный процесс;

***mixed* –** главному процессу посылается сигнал SIGTERM, затем всем оставшимся процессам из данной группы посылается сигнал SIGKILL. В целом, главный процесс (main) должен сам позаботить о родственных процессах, иначе они будут экстренно завершены.

***None* –** ни один процесс не будет завершен. Только команда STOP убьет процесс. Процессы оставшиеся в данной группе продолжают исполнение.

Отключение сервиса происходит по сигналу SIGTERM. Затем, если установлен флаг SendSIGHUP=yes, посылает сигнал SIGHUP. Затем через TimeoutStopSec (опция описана ниже) посылается сигнал SendSIGKILL, опция SendSIGKILL=yes.

KillSignal= [SIGTERM, SIGKILL или другой сигнал]

По умолчанию посылается сигнал SIGTERM, обычно за ним следует сигнал SIGKILL (конечно, если опция SendSIGKILL=1 включена). Интересно, что сам systemd сразу посылает сигнал SIGCONT, который возобновляет работу процесса, чтобы убедиться, что все процессы были успешно остановлены.

SendSIGHUP= [yes, no]

Флаг указывает, посылать или нет сигнал **SendSIGHUP** после сигнала **KillSignal=.** Если опция включена (yes), то сразу после **KillSignal** будет послан сигнал SIGHUP. Данную опцию удобно использовать с shell+аналоги программами: позволяет понять, что соединение разорвано. По умолчанию [no].

SendSIGKILL= [yes, no]

Флаг, указывающий следует ли посылать нетерминирующийся сигнал SendSIGKILL, после стандартного сигнала **KillSignal.** Если опция включена, то сигнал SIGKILL будет послан спустя TimeoutStopSec секунд. По умолчанию опция включена [yes].

Type= [simple, forking, oneshot, dbus, notify or idle]

simple – по умолчанию, если не определены Type и BusName, но определен параметр ExecStar. Ожидается, что процесс, сконфигурированный с ExecStart =, является основным процессом службы. Все средства межпроцессного взаимодействия должны быть опеделны, до того, как процесс демон будет запущен (сокеты могут быть определены через socket activation в systemd).

forking – традиционное поведение при создании демона в UNIX. ExecStart = вызывает процесс, который производит необходимые для денома настройки (IPC и др.), вызывает дочерний процесс с помощью **fork**(), а сам завершается. Созданный таким образом

процесс уже является демоном. Рекомендуется использовать опцию PIDFile=, чтобы systemd мог идентифицировать главный процесс демона (их ведь может быть много).

oneshot – тип похож на simple, однако, ожидается, что процесс должен выйти, прежде чем systemd запустит последующие модули. RemainAfterExit= особенно полезен для этого типа службы. Устанавливается по умолчанию, если ни Type, ни ExecStart= не определены.

dbus – тип так же похожий на simple. Тип устанавливается по умолчанию, если BusName= задано. Systemd запустит процесс, после того как демону будет присвоено указанное в BusName= имя на Dbus шине. Службы с этой опцией, сконфигурированной неявно, получают зависимости от dbus.socket модуля.

Notify - тип так же похожий на simple. После того как демон завершил свой запуск, он должен послать сообщение через **sd_notify**(3) или другим способом. Systemd продолжит загрузку осташихся модулей данного демона, только после получения сообщения. Если выбран этот тип, то опция NotifyAccess= должна быть предоставлять демону возможность использовать нотификационные сообщения. Если эта опция при данном типе будет задана как none или пропущена, то она будет насильно установлена в main.

Примечание: данные тип демона не работает, если он совместно используется с флагом PrivateNetwork=**yes**.

idle – почти точная копия simple: отличие в том, что бинарник демона будет выполнен в последнюю очередь, после того, как все процессы будут обслужены. Тип эффективен только при считывании вывода консоли, в остальных случаях лучше использовать simple и установленный time-out в N секунд.

RemainAfterExit= [yes, no]

Флаг, определяющий нужно ли службу считать активной, даже когда все ее процессы (видимо процессы ее контрольной группы) отключены. По умолчанию no.

GuessMainPID= [yes, no]

Флаг указывает systemd пытаться ли угадать PID службы, если его не удается определить достоверно. Данный флаг имеет значение, если Type=forking и **PIDFile=** не определен, т.к. для всех остальных типов служб PID демона известен достоверно. Алгоритм “угадывания” может прийти к не правильному заключению, если демон состоит из нескольких процессов. Если PID основного процесса не может быть определен, обнаружение отказов и автоматический перезапуск службы будут работать не надежно. По умолчанию yes.

PIDFile=

Параметр содержит абсолютный путь к PID файлу демона. Особенно рекомендуется использовать данный параметр для сервисов, тип которых **forking**. systemd считывает PID главного процесса, после запуска сервиса. systemd ничего не пишет в данный файл, но если после команды отключения сервис продолжает выполнять, файл по указанному пути будет удален.

BusName=

Параметр актуален для служб с Type= dbus. Параметр указывает имя демона, по которому к нему можно обратиться на Dbus-шине.

ExecStart=

Команда запуска сервиса с заданными аргументами. Например:

ExecStart=/home/root/myDaemon/mainProc “Hello” 1 0x01.

Для всех типов, кроме Type=oneshot должно быть задано как минимум 1 значение (способов запуска может быть несколько, несколько одинаковых строк). Для oneshot это поле может оставаться пустым. Если поле не задано, то обязательно должен быть установлен флаг RemainAfterExit=yes.

Каждая команда, указывающая способ запуска сервиса должна содержать абсолютный путь к исполняемому файлу. Если абсолютный путь имеет префикс:

- “@”. Первый аргумент, который будет передан исполняемому процессу "argv[0]" будет являтся вторый аргументом в строке ExecStart=. Если данный не префикс не указывать, то первым аргументом для исполняемого процесса будет его абсолютный путь.
- “-”. если, при попытке нормальной остановки сервиса получен код ошибки – ошибка будет проигнорирована, а завершение работы демона будет считаться успешным.
- “+”. Сервис будет запущен в привилегированном режиме, проще говоря демон будет запущен с правами “бога” (то есть root`a).

Все три префикса могут использовать совместно друг с другом и в разном порядке.

Если в файле задано несколько возможных способов запуска демона, выполнение оных будет производится в последовательном порядке, но если в строке запуска не указан префикс “-”, и произойдет ошибка при попытке запуска – нижестоящие команды запуска так и не будут исполнены, а попытка запуска демона будет считаться неуспешной.

Для демоном с типом forking, процесс, что запускается данной командой, будет считаться главным.

ExecStartPre=, ExecStartPost=

Синтаксис команд аналогичный ExecStart: команд может быть несколько, если хотя бы одна из ExecStartPre или ExecStartPost провалиться – запуск сервиса будет считаться не успешным, будет вызвана команда, указанная в ExecStopPost, а команда останова сервиса (ExecStop) будет пропущена. Все команды исполняются последовательно, при условии, что до этого все было ОК (или использовался перфикс “-”): сначала ExecStartPre, затем ExecStart и в завершении ExecStartPost.

Для всех вызовов доступны перечисленные выше 3 префикса “@”, “+” и ”-”. Так же стоит отменить, что вызывать длительные процессы (например, логирование) в **ExecStartPre** не имеет смысла, так как pre/post-процессы убиваются.

ExecReload=

Команда для перезагрузки сервиса. Поддерживает множественные вызовы, по той же схеме, что и ExecStart=, так же поддерживает префиксы “+,-,@”

Если известен PID главного процесса, то перезагрузку сервиса можно выполнить комнадой /bin/kill -HUP $MAINPID. Однако это ассинхронный рестарт сервиса, настоятельно рекомендуется установить с помощью **ExecReload=** команду, которая не

только инициировала перезагрузку конфигурации демона (способ приведенный выше), но также и синхронно ожидала его, чтобы завершиться.

ExecStop=

Команда используется для остановки сервиса. Возможно множественное объявление, такое же как и для ExecStart=. После запуска команды, все процессы, относящиеся к указанному демону завершаются в соответствии с установленным KillMode=. Если опция не установлена, то демон (все его процессы) будут убиты сигналом KillSignal=.

Команда должна быть синхронной: то есть после вызова, она должна дождаться того, чтобы сервис остановился. Если команда, указанная в **ExecStop**, завершится сразу, то оставшимся в живых процессам демона будет послан сигнал **SIGKILL**, что повлечет не корректное завершение работы демона.

Демон может быть остановлен этой комнадой, только если он был успешно запущен. Если демон не запускался, или запускался, но не удачно, будет использована команда ExecStopPost.

Вызов этой комнады предполагает, что сервис работает в штатном режиме и готов ее воспринять. Для того, чтобы завершить *красиво* остановку сервиса, и *подчистить* за собой, следует использовать команду ExecStopPost=.

ExecStopPost=

Синтаксис похож на **ExecStart:** команда будет выполняется в случаях, когда демон пытался запуститься, но не удачно, когда приходит повторная команда останова и когд команда **ExecStop=** не помогла или не задана.

Основное применение данного вызова: обработка не корректного запуска сервиса. Команда должна “наводить порядок”, после того как сервис “завалился” в середине загрузки и оставил после себя неприменимый контекст, кароче говоря сервис “наследил”. Считается, что на момент исполнения **ExecStopPost=** все процессы демона остановлены и связываться с ними нет смысла.

Все команды, будут вызывать в контексте, содержащем переменные $SERVICE_RESULT, $EXIT_CODE и $EXIT_STATUS, отвечающие за код завершения работы сервиса, код результата работы и статус сервиса.

RestartSec=

Параметр указывает, какое время необходимо подождать до, того как перезапустить демона командой Restart=. Единицы измерения - секунды, по умолчанию 100ms, однако можно задать и в таком формате: “5min 20s".

TimeoutStartSec=

Параметр отвечает за промежуток времени, который необходимо подождать, перед тем как считать, что сервис не запустился и послать ему сигнал останова. Задается в секундах или в текстовом формате "5min 20s" (аналогично RestartSec=). Для того, чтобы отключить таймер в качестве параметр стоит использовать infinity. Для все типов демонов опция по

умолчани равна DefaultTimeoutStartSec, кроме типа oneshot – для опция по умолчанию отключена.

TimeoutStopSec=

Время, ожидаемое после команды останова: сначала сервису дается команда SIGTERM, и если он не ответит в течении заданномго времени, то ему посылает нетерминируемый сигнал SIGKILL. Задается аналогично предыдущим двум командам.

TimeoutSec=

Опция позволяет задать TimeoutStartSec= и TimeoutStopSec= одновременно.

RuntimeMaxSec=

Задает максимальное время, которое сервис может работ. Спустя указанное время сервис останавливается. Значение по умолчанию - infinty. Для демонов с типом oneshot данная опция не имеет никакого значения.

Пробуем запустить свой сервис

Скопировали конфиг

```
michael@michael-LIFEBOOK-AH531:~/Desktop/SP-8-semester/lab2-LinuxDaemon$ sudo cp ~/Desktop/usbDevWatcher.service /etc/systemd/system/
```

Добавили в список демонов

```
michael@michael-LIFEBOOK-AH531:~/Desktop/SP-8-semester/lab2-LinuxDaemon$ systemctl enable usbDevWatcher.service
Created symlink from /etc/systemd/system/multi-user.target.wants/usbDevWatcher.service to /etc/systemd/system/usbDevWatcher.service.
```

Попробовали запустить, но вылетела ошибка

```
michael@michael-LIFEBOOK-AH531:~/Desktop/SP-8-semester/lab2-LinuxDaemon$ systemctl start usbDevWatcher
Failed to start usbDevWatcher.service: Unit usbDevWatcher.service is not loaded properly: Invalid argument.
See system logs and 'systemctl status usbDevWatcher.service' for details.
michael@michael-LIFEBOOK-AH531:~/Desktop/SP-8-semester/lab2-LinuxDaemon$ systemctl status usbDevWatcher
● usbDevWatcher.service - USB devices watcher
   Loaded: error (Reason: Invalid argument)
   Active: inactive (dead)

апр 17 21:03:05 michael-LIFEBOOK-AH531 systemd[1]: [/etc/systemd/system/usbDevWa
апр 17 21:03:05 michael-LIFEBOOK-AH531 systemd[1]: [/etc/systemd/system/usbDevWa
апр 17 21:03:05 michael-LIFEBOOK-AH531 systemd[1]: usbDevWatcher.service: Servic
lines 1-7/7 (END)
```

Ошибка:

“[/etc/systemd/system/usbDevWatcher.service:11] Executable path is not absolute,...”

Внесенные исправления:

1. ExecStart. Исправил путь к исполняемому файлу: пришлось убрать префикс “+”, позволяющий запускать демон с правами root`a. Не нашел ни одного примера использования хоть какого-нибудь префикса в systemd. Похоже на битую фичу, которая не реализована.

2. ExecStopPost. Команда удаления файла-отчета (“rm /home/…/Note.txt”) перенесена в отдельный shell-скрипт – rmNote.sh.

Итоговый конфиг файл:

```
[Unit]
Description=USB devices watcher
After=network.target

[Install]
WantedBy=multi-user.target

[Service]
Type=simple

ExecStart="/home/michael/usbDevWatcher/main.bin"
RuntimeMaxSec=3600

Restart=always
RestartSec=5
```

```
KillMode=process
KillSignal=SIGTERM
SendSIGKILL=yes
ExecStopPost=/home/michael/usbDevWatcher/rmNote.sh
TimeoutStopSec=5
```

Файл /home/michael/usbDevWatcher/rmNote.sh

```
#!/bin/bash
$(rm /home/michael/usbDevWatcher/Note.txt 2>/dev/null)
```

Перезагружаем systemd-daemon

```
michael@michael-LIFEBOOK-AH531:$ sudo systemctl daemon-reload
michael@michael-LIFEBOOK-AH531:$ systemctl status usbDevWatcher
● usbDevWatcher.service - USB devices watcher
   Loaded: loaded (/etc/systemd/system/usbDevWatcher.service; enabled; vendor preset: enabled)
   Active: inactive (dead)

апр 17 21:03:05 michael-LIFEBOOK-AH531 systemd[1]:
[/etc/systemd/system/usbDevWatcher.service:11] Executable path is not absolute, ignori
…
/* список старых ошибок убран */

michael@michael-LIFEBOOK-AH531:$ systemctl start usbDevWatcher
michael@michael-LIFEBOOK-AH531:$ systemctl status usbDevWatcher
● usbDevWatcher.service - USB devices watcher
   Loaded: loaded (/etc/systemd/system/usbDevWatcher.service; enabled; vendor preset: enabled)
   Active: active (running) since Пн 2017-04-17 21:36:28 MSK; 1s ago
 Main PID: 11423 (main.bin)
   CGroup: /system.slice/usbDevWatcher.service
           └─11423 /home/michael/usbDevWatcher/main.bin

апр 17 21:36:28 michael-LIFEBOOK-AH531 systemd[1]: Started USB devices watcher.
```

Теперь проверим качество работы:

На момент запуска демона в системе обнаружено 9 устройст

```
Count of all USB devices: 9
Information about all devices
---------------------------------------
bDeviceClass:    224
idVendor:        0x8086
idProduct:       0x189
iProduct:        0
```

```
iManufacturer:   0
bNumConfigs:     0
bLength:         12
bDescriptorType: 1
bcdUSB:          200
bDeviceSubClass: 1
bDeviceProtocol: 1
bMaxPacketSize0: 40
...
```

Теперь удалим флешку, мышь+клавиатура и внешнюю аудиокарту:

```
Count of all USB devices: 6
Information about all devices
---------------------------------------
bDeviceClass:    224
idVendor:        0x8086
idProduct:       0x189
iProduct:        0
iManufacturer:   0
bNumConfigs:     0
bLength:         12
bDescriptorType: 1
bcdUSB:          200
bDeviceSubClass: 1
bDeviceProtocol: 1
bMaxPacketSize0: 40

---------------------------------------
bDeviceClass:    9
idVendor:        0x8087
```

Пробуем остановить демон, и проверить, что скрипт удаления файла-отчета действительно сработает.

```
michael@michael-LIFEBOOK-AH531:~/Desktop/SP-8-semester/lab2-LinuxDaemon$ systemctl
stop usbDevWatcher
Warning: usbDevWatcher.service changed on disk. Run 'systemctl daemon-reload' to reload units.
```

Итог: файл не удалился. Решение проблемы: сделать файл исполняемым.

```
michael@michael-LIFEBOOK-AH531:~/usbDevWatcher$ ls -l
total 24
-rwxrwxr-x 1 michael michael 14808 апр 17 20:14 main.bin
-rw-rw-r-- 1 michael michael  2027 апр 17 21:56 Note.txt
-rw-rw-r-- 1 michael michael    68 апр 17 21:35 rmNote.sh
michael@michael-LIFEBOOK-AH531:~/usbDevWatcher$ chmod +x rmNote.sh
```

```
michael@michael-LIFEBOOK-AH531:~/usbDevWatcher$ ls -l
total 24
-rwxrwxr-x 1 michael michael 14808 апр 17 20:14 main.bin
-rw-rw-r-- 1 michael michael  2027 апр 17 21:56 Note.txt
-rwxrwxr-x 1 michael michael    68 апр 17 21:35 rmNote.sh
michael@michael-LIFEBOOK-AH531:~/usbDevWatcher$ ./rmNote.sh
```

Обнаружена другая, более забавная проблема: демон создает и удаляет файл! Вот что "отловлено" с помощью команды "sudo systemctl status usbDevWatcher":

```
michael@michael-LIFEBOOK-AH531:~/Desktop/SP-8-semester/lab2-LinuxDaemon$ sudo
systemctl status usbDevWatcher
● usbDevWatcher.service - USB devices watcher
   Loaded: loaded (/etc/systemd/system/usbDevWatcher.service; enabled; vendor preset: enabled)
   Active: deactivating (stop-sigterm) (Result: timeout) since Пн 2017-04-17 21:59:52 MSK;
3ms ago
  Process: 13693 ExecStopPost=/home/michael/usbDevWatcher/rmNote.sh (code=exited,
status=0/SUCCESS)
 Main PID: 13716 (main.bin)
   CGroup: /system.slice/usbDevWatcher.service
           ‣ 13716 [main.bin]

апр 17 21:59:47 michael-LIFEBOOK-AH531 systemd[1]: Started USB devices watcher.
апр 17 21:59:52 michael-LIFEBOOK-AH531 systemd[1]: usbDevWatcher.service: Service
reached runtime time limit. Stopping.
michael@michael-LIFEBOOK-AH531:$ sudo systemctl status usbDevWatcher
● usbDevWatcher.service - USB devices watcher
   Loaded: loaded (/etc/systemd/system/usbDevWatcher.service; enabled; vendor preset: enabled)
   Active: activating (auto-restart) (Result: timeout) since Пн 2017-04-17 21:59:52 MSK; 760ms
ago
  Process: 13751 ExecStopPost=/home/michael/usbDevWatcher/rmNote.sh (code=exited,
status=0/SUCCESS)
  Process: 13716 ExecStart=/home/michael/usbDevWatcher/main.bin (code=killed, signal=TERM)
 Main PID: 13716 (code=killed, signal=TERM)

апр 17 21:59:52 michael-LIFEBOOK-AH531 systemd[1]: usbDevWatcher.service: Failed with
result 'timeout'.
```

Итог: задал слишком маленький параметр RuntimeMaxSec=3600. Меняем параметр и используем команды для обновления конфигурации и перезагрузки сервиса:

```
michael@michael-LIFEBOOK-AH531:$ sudo systemctl stop usbDevWatcher
Warning: usbDevWatcher.service changed on disk. Run 'systemctl daemon-reload' to reload units.
michael@michael-LIFEBOOK-AH531:$ sudo systemctl daemon-reload
michael@michael-LIFEBOOK-AH531:$ sudo systemctl start usbDevWatcher
michael@michael-LIFEBOOK-AH531:$ sudo systemctl status usbDevWatcher
● usbDevWatcher.service - USB devices watcher
   Loaded: loaded (/etc/systemd/system/usbDevWatcher.service; enabled; vendor preset: enabled)
```

```
   Active: active (running) since Пн 2017-04-17 22:14:22 MSK; 1min 40s ago
 Main PID: 15085 (main.bin)
   CGroup: /system.slice/usbDevWatcher.service
           └─15085 /home/michael/usbDevWatcher/main.bin

апр 17 22:14:22 michael-LIFEBOOK-AH531 systemd[1]: Started USB devices watcher.
```

Итог: сервис успешно включился и работает.

ИСТОЧНИКИ

1. The Complete Service Sample , https://msdn.microsoft.com/en-us/library/windows/desktop/bb540476(v=vs.85).aspx

2. Сообщение об ошибке при запуске службы установщика модулей Windows (TrustedInstaller): "Системная ошибка 126: не удалось найти конкретный модуль"

https://support.microsoft.com/ru-ru/help/959077/error-message-when-you-start-the-windows-modules-installer-service-trustedinstaller-system-error-126-the-specific-module-could-not-be-found

3. Creating a Resource DLL (*.mc) , https://msdn.microsoft.com/en-us/library/ms853727.aspx

4. Создание своего Windows Service. https://habrahabr.ru/post/71533/

5. Конфигурация процедуры остановки процесса systemd.kill

http://manpages.ubuntu.com/manpages/zesty/man5/systemd.kill.5.html

6*. About types of targets

https://www.freedesktop.org/software/systemd/man/systemd.special.html

7*. About systemd during 5 minutes

https://habrahabr.ru/company/centosadmin/blog/255845/

8*. Systemd.unit — Unit configuration

https://www.freedesktop.org/software/systemd/man/systemd.unit.html