**Лабораторная работа № 11 (контролируемая управляемая самостоятельная работа)**

**Службы (сервисы)**

**(4 часа)**

**Цель работы:**изучить основы организации, функционирования и разработки служб ОС Windows; сформировать умения программирования простейших служб, их регистрации/удаления в SCM, управления их работой; закрепить навыки системного программирования в ОС Windows, разработки и отладки WinAPI-приложений.

**Содержание:** Понятие сервиса. Структура сервиса. Обработчик управляющих команд. Установка и удаление сервисов. Запуск и определение состояния сервиса. Управление сервисом.

Сервис или служба Windows – это неинтерактивный процесс, выполняющийся в отдельной сессии Windows и, как правило, исполняющий длительную работу.

Так как сервисы не являются приложениями, то их запуск, отслеживание работы и остановку выполняет непосредственно не пользователь. Сервисы работают под управлением специального компонента Windows – менеджера сервисов или сокращенно SCM (Service Control Manager, диспетчер управления сервисами).

SCM выполняет поддержку базы установленных сервисов, запускает их при загрузке ОС, обновляет информацию о состоянии работающих сервисов, выполняет передачу управляющих запросов сервисам на локальном или удаленном компьютере, выполняет перезапуск сервиса при возникновений в сервисе ошибок, а также другие функции, требующиеся для работы сервисов.

По типу создаваемого проекта для приложения-сервиса сервисы могут быть как консольными (чаще всего), так и GUI-приложениями.

Каждый сервис кроме главной функции приложения main() содержит 2 функции обратного вызова (CALLBACK):

-       Точку входа в сервис ServiceMain;

-       Обработчик управляющих сигналов от Windows ServiceCtrlHandler(Ex).

Точка входа сервиса может иметь произвольное имя, но обязательно соответствует следующему прототипу:

|  |  |
| --- | --- |
|  | VOID WINAPI ServiceMain (DWORD, LPTSTR\*); |

Обработчик управляющих сообщений от Windows будет рассмотрен ниже.

Разработку сервиса удобно выполнять в следующем порядке:

-       Создать и отладить консольное приложение, содержащее код, который должен выполняться в сервисе;

-       Создать новую точку входа main() для регистрации разработанной службы в SCM;

-       Заменить старую точку входа main() на функцию ServiceMain(), дополнив её регистрацией обработчика управляющих команд сервиса и кодом для обновления состояния сервиса у SCM;

 -       Опционально дополнить ServiceMain() возможностью регистрации событий в журнале событий Windows или кодом для протоколирования работы в обычный файл;

-       Создать функцию обработчика управляющих команд сервиса, реагирующую на команды SCM.

При запуске сервиса SCM запускает функцию main(). Задачи main() – это регистрация службы (или нескольких служб) в SCM и запуск для неё диспетчера сервиса. Запуск диспетчера сервиса выполняется с помощью функции StartServiceCtrlDistpatcher. Она должна быть вызвана в течение 30 секунд с момента запуска сервиса или SCM зафиксирует ошибку.

 BOOL StartServiceCtrlDistpatcher( CONST LPSERVICE\_TABLE\_ENTRY lpServiceTable);

В случае успеха функция возвращает ненулевое значение, иначе – FALSE. Как правило, это означает, что сервис уже работает или нет доступа к нужным записям реестра. При успешном выполнении SCM **не возвращает управление** до окончания работы всех служб, которые регистрируются этой функцией. Единственный параметр этой функции – это массив (таблица) из структур SERVICE\_TABLE\_ENTRY, каждая из которых описывает один сервис:

typedef struct \_SERVICE\_TABLE\_ENTRY {

     LPSTR lpServiceName;

     LPSERVICE\_MAIN\_FUNCTION     lpServiceProc;

}    SERVICE\_TABLE\_ENTRY, \*LPSERVICE\_TABLE\_ENTRY;

Последний элемент массива должен содержать структуру с парой пустых указателей как в приводимом ниже примере:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23 | void WINAPI ServiceMain(DWORD argc, TCHAR\* argv[]);   LPTSTR service\_name = TEXT ("SimpleDemoService");     int main(){        SERVICE\_TABLE\_ENTRY service\_table[] = { {service\_name, ServiceMain}, {NULL,NULL} };   if(!StartServiceCtrlDispatcher(service\_table))   {   // обработка ошибки   }   // Здесь окажемся только после завершения сервиса   return 0;   } |

Функция (функции)

|  |  |
| --- | --- |
| 1 | void ServiceMain(DWORD argc, TCHAR\* argv[] ); |

заданные в таблице диспетчеризации сервисов представляют точки входа в логические службы, содержащиеся в исполнимом файле. Каждая из них будет работать в собственном потоке, созданном SCM.

 После запуска ServiceMain должен выполнить следующие действия:

-       Немедленно зарегистрировать для сервиса обработчик управляющих команд от SCM при помощи RegisterServiceCtrlHandlerEx();

-       Установить для сервиса стартующее состояние SERVICE\_START\_PENDING функцией SetServiceStatus();

-       Провести локальную инициализацию сервиса;

-       Установить рабочее состояние сервиса SERVICE\_RUNNING при помощи SetServiceStatus();

-       Приступить к выполнению работы сервиса, учитывая состояния сервиса, которые могут изменяться обработчиком;

-       После перехода в состояние SERVICE\_STOPPED выполнить освобождение захваченных ресурсов.

Функция обработчика управляющих команд от SCM имеет следующий вид:

SERVICE\_STATUS\_HANDLE RegisterServiceCtrlHandlerEx(

LPCTSTR lpServiceName,

LPHANDLER\_FUNCTION\_EX lpHandlerProc,

LPVOID lpContext

);

В случае успешного завершения возвращается дескриптор состояния сервиса, в случае неудачи – 0. Функция должна быть вызвана в начале ServiceMain() и более нигде не повторяться.

Её параметры:

lpServiceName - указывает на строку с именем сервиса;

lpHandlerProc – адрес функции обработчика управляющих команд от SCM;

lpContext – определенные пользователем данные, передаваемые обработчику.

После регистрации обработчика управляющих сообщений нужно перевести службу в состояние SERVICE\_START\_PENDING. Для этого и для других изменений состояния сервиса используется функция:

BOOL SetServiceStatus(

SERVICE\_STATUS\_HANDLE hServiceStatus,

     LPSERVICE\_STATUS lpServiceStatus

);

Её параметр hServiceStatus должен содержать дескриптор состояния сервиса SERVICE\_STATUS\_HANDLE, полученный функцией RegisterServiceCtrlHandlerEx(), а lpServiceStatus должен указывать на экземпляр структуры типа SERVICE\_STATUS, которая содержит информацию о состоянии сервиса:

typedef struct \_SERVICE\_STATUS {

     DWORD dwServiceType;

     DWORD dwCurrentState;

     DWORD dwControlsAccepted;

     DWORD dwWin32ExitCode;

     DWORD dwServiceSpecificExitCode;

     DWORD dwCheckPoint;

     DWORD dwWaitHint;

}    SERVICE\_STATUS, \*LPSERVICE\_STATUS;

Здесь:

dwServiceType – тип службы. Одно из следующих значений:

SERVICE\_WIN32\_OWN\_PROCESS – сервис как самостоятельный процесс со своими ресурсами;

SERVICE\_WIN32\_SHARE\_PROCESS – сервис разделяет процесс с другими сервисами (общие ресурсы);

SERVICE\_KERNEL\_DRIVER – драйвер устройства;

SERVICE\_FILE\_SYSTEM\_DRIVER – драйвер файловой системы.

1 и 2 типы могут быть установлены вместе с SERVICE\_INTERACTIVE\_PROCESS – сервис может взаимодействовать с рабочим столом.

Параметр dwCurrentState может задавать следующие состояния сервиса:

SERVICE\_START\_PENDING – сервис стартует;

SERVICE\_RUNNING – сервис исполняется;

SERVICE\_STOP\_PENDING – сервис останавливается, но  еще не завершил свою работу;

SERVICE\_STOPPED – сервис не выполняется;

SERVICE\_PAUSE\_PENDING – идёт переход в состояние приостановки;

SERVICE\_PAUSED – на паузе;

SERVICE\_CONTINUE\_PENDING – сервис возобновляет работу после приостановки.

Параметр dwControlsAccepted указывает коды управляющих команд, которые сервис будет воспринимать и обрабатывать:

SERVICE\_ACCEPT\_STOP – сервис принимает SERVICE\_CONTROL\_STOP

SERVICE\_ACCEPT\_PAUSE\_CONTINUE

SERVICE\_ACCEPT\_SHUTDOWN

SERVICE\_ACCEPT\_PARAMCHANGE

SERVICE\_ACCEPT\_NETBINDCHANGE (SERVICE\_CONTROL\_NETBINDADD, REMOVE, ENABLE, DISABLE)

SERVICE\_ACCEPT\_HARDWAREPROFILECHANGE

SERVICE\_ACCEPT\_POWEREVENT

SERVICE\_ACCEPT\_SESSIONCHANGE

*Каждый обработчик может получить SERVICE\_CONTROL\_INTERROGATE – немедленно обновить статус сервиса.*

Параметр dwWin32ExitCode может принимать:

NO\_ERROR – в процессе выполнения и при нормальном завершении - ошибки нет;

ERROR\_SERVICE\_SPECIFIC\_ERROR – ошибка во время запуска или остановки сервиса задается параметром dwServiceSpecificExitCode.

dwCheckPoint – значение, которое сервис должен периодически увеличивать на 1 сообщая о прогрессе во время инициализации или переходе из состояния в состояние. Если состояние не отслеживается или переход менее 30 секунд – может быть равен 0.

dwWaitHint - интервал в миллисекундах между вызовами SetServiceStatus() в течение которого сервис переходит из состояния в состояние. Если в течение dwWaitHint сервис не перешел в другое состояние и не изменил dwCheckPoint, то SCM считает, что в сервисе произошла ошибка.

После инициализации и запуска сервиса он может переходить к работе, выставив статус SERVICE\_RUNNING.

В качестве обработчика управляющих команд сервиса выступает функция зарегистрированная RegisterServiceCtrlHandlerEx() и имеющая следующий прототип:

DWORD WINAPI ServiceCtrlHandlerEx (

 DWORD dwControl,

 DWORD dwEventType,

 LPVOID lpEventData,

 LPVOID lpContext);

Здесь:

dwControl – задает управляющую команду, поступившую от SCM:

SERVICE\_CONTROL\_STOP

SERVICE\_CONTROL\_PAUSE

SERVICE\_CONTROL\_CONTINUE

SERVICE\_CONTROL\_INTERROGATE

SERVICE\_CONTROL\_SHUTDOWN

SERVICE\_CONTROL\_PARAMCHANGE

SERVICE\_CONTROL\_NETBINDADD и т.д. + коды от 128 до 255, определенные пользователем.

dwEventType – обычно 0, не нулевые значения используются для управления устройствами;

lpEventData – определяет дополнительную информацию для устройств;

lpContext – пользовательские данные, переданные через RegisterServiceCtrlHandlerEx() SCM для идентификации сервиса.

Обработчик должен изменить состояние сервиса в течение 30 секунд. Для обработки SERVICE\_CONTROL\_SHUTDOWN сервису отводится 20 секунд.

После компиляции, перед использованием сервис нужно **установить**. Для установки и удаления сервисов используются специальные функции WinAPI. После установки информация о сервисах хранится в реестре по пути:

HKEY\_LOCAL\_MACHINE\System\CurrentControlSet\Services

Управлять работой сервисов можно как при помощи приложений ( Control Panel | Administrative Tools | Services = Панель управления | Администрирование | Службы или команды sc) так и программно.

Последовательность установки сервиса (программно):

-       Открыть SCM;

-       Создать сервис под управлением SCM;

-       Запустить сервис.

Все эти действия выполняются не с сервисом, а с SCM. Для регистрации службы, как правило, используется отдельный процесс.

Рассмотрим алгоритм по шагам. Открытие SCM выполняется функцией:

SC\_HANDLE OpenSCManager(

     LPCTSTR lpMachineName,

     LPCTSTR lpDatabaseName,

     DWORD dwDesiredAccess);

В случае успеха она возвращает дескриптор SCM, иначе - NULL. Параметры вызова:

lpMachineName - имя компьютера с ведущими «\\.» или NULL, если доступ к SCM получается на локальном компьютере;

lpDatabaseName – обычно NULL;

dwDesiredAccess – обычно SC\_MANAGER\_ALL\_ACCESS (полные права доступа к базе сервисов).

Для регистрации службы у SCM вызывается:

SC\_HANDLE CreateService(

     SC\_HANDLE hSCManager,

     LPCTSTR lpServiceName,

     LPCTSTR lpDisplayName,

     DWORD dwDesiredAccess,

     DWORD dwServiceType,

     DWORD dwStartType,

     DWORD dwErrorControl,

     LPCTSTR lpBinaryPathName,

     LPCTSTR lpLoadOrderGroup,

     LPDWORD lpdwTagId,

     LPCTSTR lpDependencies

     LPCTSTR lpServiceStartName,

     LPCTSTR lpPassword);

где,

hSCManager – дескриптор SCM полученный от OpenSCManager();

lpServiceName – имя сервиса, определенное в вызове StartServiceCtrlDispatcher();

lpDisplayName – имя, отображаемое в реестре и утилите Services Панели управления;

dwDesiredAccess – SERVICE\_ALL\_ACCESS или комбинация GENERIC\_-прав;

dwServiceType – тип службы, вида SERVICE\_WIN32\_OWN\_PROCESS и т.д. (см. выше);

dwStartType – способ запуска сервиса:

* SERVICE\_DEMAND\_START – запуск по требованию;
* SERVICE\_BOOT\_START – запуск драйвера при начальной загрузке;
* SERVICE\_SYSTEM\_START – запуск драйвера при загрузке системы;
* SERVICE\_AUTO\_START – запуск сервиса при старте системы автоматически;

dwErrorControl,

lpBinaryPathName – имя исполняемого файла службы (без .exe)

lpLoadOrderGroup,

lpdwTagId,

lpDependencies – зависимости сервиса

lpServiceStartName – учетная запись для старта

lpPassword – пароль для старта

Для закрытия дескрипторов сервисов SC\_HANDLE следует вызвать CloseServiceHandle().

К прочим вызовам для управления сервисом относятся:

ChangeServiceConfig(), ChangeServiceConfig2() - изменение параметров уже существующей службы;

OpenService() - получение дескриптора сервиса по имени;

DeleteService() – удаление сервиса из реестра.

Для запуска сервиса (ServiceMain()) требуется указать дескриптор сервиса, полученный от CreateService() и параметры argc и argv ожидаемые сервисом:

BOOL StartService(

     SC\_HANDLE hService,

     DWORD argc,

     LPTSTR argv[]);

Для управления службой следует передать SCM требуемый управляющий код:

BOOL ControlService (

     SC\_HANDLE hService,

     DWORD dwControlCode

     LPSERVICE\_STATUS lpServiceStatus

);

Здесь, dwControlCode – одно из значений:

 SERVICE\_CONTROL\_STOP

 SERVICE\_CONTROL\_PAUSE

 SERVICE\_CONTROL\_CONTINUE

 SERVICE\_CONTROL\_INTERROGATE

 SERVICE\_CONTROL\_SHUTDOWN

или 128-255 пользовательские;

lpServiceStatus – указатель на структуру SERVICE\_STATUS, получающую текущее состояние сервиса.

Опрос состояния службы (получение структуры SERVICE\_STATUS) можно выполнить при помощи функции:

BOOL QueryServiceStatus (

     SC\_HANDLE hService,

     LPSERVICE\_STATUS lpServiceStatus

);

Обычно службы выполняются без уведомлений для пользователя. При этом действия сервиса удобно протоколировать в текстовый файл или файл журнала событий Windows. Для этого используются следующие функции:

RegisterEventSource ()– получить дескриптор файла журнала;

ReportEvent() – внести запись в журнал;

DeregisterEventSource() – закрыть дескриптор файла журнала.

**Задание 1.** Создайте сетевой сервис, взяв за основу третью часть лабораторной работы 7. Разработанная служба должна позволять клиентам найти сервер в сети и подключиться к нему. По окончании игры служба должна продолжать работу и ожидать подключения новых клиентов.

**Литература:**

1. Джеффри Рихтер, Джейсон Кларк, Программирование серверных приложений для Microsoft Windows 2000. Мастер-класс./Пер. с англ. - СПб.: Питер; М.:Издательско-торговый дом "Русская Редакция", 2001. (глава 3-4, сс.60-132).

2. Побегайло А.П., Системное программирование в Windows. - СПб.: БХВ-Петербург, 2006. (глава 33-34, сс.605-653).

3. Services: https://docs.microsoft.com/en-us/windows/win32/services/services (включая раздел Service Functions: https://docs.microsoft.com/en-us/windows/win32/services/service-functions).