Санкт-Петербургский политехнический университет Петра Великого Институт Информационных Технологий и Управления

Кафедра компьютерных систем и программных технологий

Отчёт по практической работе № 2

по предмету «Проектирование ОС и компонентов»

### Служюы операционных систем, дистрибуция приложений

Работу выполнил студент гр. 63501/3 Мартынов С. А. Работу принял преподаватель Душутина Е. В.

Санкт-Петербург 2016

**Содержание**

[Постановка задачи 3](#_TOC_250016)

[Введение 3](#_TOC_250015)

1. [Фоновые приложения в Linux 4](#_TOC_250014)
   1. [Понятие процесса и демона 4](#_TOC_250013)
   2. [Создание демона Linux 5](#_TOC_250012)
   3. [Работа с системным журналом 9](#_TOC_250011)
2. [Дистрибуция пакетов в Linux 12](#_TOC_250010)
   1. [Создание DEB/RPM/TGZ пакетов 12](#_TOC_250009)
   2. [Создание PKGBUILD 13](#_TOC_250008)
3. [Фоновые приложения в Windows 16](#_TOC_250007)
   1. [Службы Windows 16](#_TOC_250006)
   2. [Создание службы Windows с помощью программы Sc.exe 17](#_TOC_250005)
   3. [Создание службы Windows с помощью PowerShell 19](#_TOC_250004)
   4. [Работа с системным журналом Windows 20](#_TOC_250003)
4. [Дистрибуция пакетов в Windows 27](#_TOC_250002)

[Заключение 29](#_TOC_250001)

[Список литературы 30](#_TOC_250000)

# Постановка задачи

В рамках данной работы необходимо изучить процесс написания системных служб для ОС семейства Windows и демонов Linux, порядок их запуска и взаимодействия с системным журналом.

# Введение

В каждую минуту времени компьютер выполняет множество задач. Не все эти задачи происходят в графическом режиме и пользователь может их наблюдать. Многие вещи (служба печати, времени, индексации) скрыта, но продолжает работать. Это обеспечи- вает возможность компьютера в одно время быть сервером печати, сервером доступа к файлам и воспроизводить музыку. Эти службы (или демоны) обладают определенными особенностями, которые будут рассмотрены в данной работе.

Вторым важным вопросом является распространение кода программ. Исходный код позво- ляет всегда быть уверенным в том, что приложение делает то, что обещал разработчик, но на практике подобный подход не всегда возможет как с точки зрения защиты исходного кода коммерческих продуктов, так и по причине долгой сборки (к примеру браузер Mozilla Firefox может компилироваться почти 20 часов на среднем компьютере). Эта проблема ре- шается распространением бинарных пакетов, которые необходимо правильно организовать на компьютере пользователя.

# Фоновые приложения в Linux

## Понятие процесса и демона

В любой многозадачной системе одновременно может быть запущено много программ, то есть много процессов. В действительности в каждый момент времени выполняется только один процесс. Ядро (по средствам планировщика) выделяет каждому процессу небольшой квант времени и по истечении этого кванта передает управление следующему процессу. Кванты времени, выделяемые каждому процессу, на столько малы, что у пользователя создается иллюзия одновременного выполнения многих процессов. Для организации пе- реключения между процессами по истечении кванта времени, выполняется фиксация и сохранение в памяти состояния программы. Этот снимок содержит информацию о состоя- нии регистров центрального процессора на момент прерывания программы, указание на то, с какой команды возобновить исполнение программы (состояние счетчика команд), содержимое стека и подобные данные. Когда процесс снова получает в свое распоряжение ЦП, состояние регистров ЦП и стека восстанавливается из сделанного снимка и выполне- ние программы возобновляется в точности с того места, где она была остановлена. Такие же действия выполняются в тех случаях, когда какому-то процессу необходимо вызвать некоторую системную функцию (вызов ядра)[1].

Кроме организации переключения процессов, ядро в многозадачной системе отвечает за изоляцию процессов – два процесса не должны одновременно изменять какие-то данные одном участке памяти. Для этого каждому процессу выделяется свое виртуальное ад- ресное пространство. Его размер может даже превышать размер реальной оперативной памяти, что обеспечивается за счет применения страничной организации памяти и ме- ханизма свопинга. И физическая и виртуальная память организована в виде страниц – областей памяти фиксированного размера (обычно 4 Кбайта). Если страница долго не используется, ее содержимое переносится в область свопинга на жестком диске, а страница в оперативной памяти предоставляется в распоряжение другого процесса. Подсистема управления памятью поддерживает таблицу соответствия между страницами виртуальной памяти процессов и страницами физической памяти (включая страницы, перенесенные в область свопинга). В современных компьютерных системах эти механизмы реализуются на аппаратном уровне с помощью устройств управления памятью – Memory Management Unit (MMU). Если процесс обращается к странице виртуальной памяти, которая размещается в оперативной памяти, операция чтения или записи осуществляется немедленно. Если же страница в оперативной памяти отсутствует, генерируется аппаратное прерывание, в ответ на которое подсистема управления памятью определяет положение сохраненного содержимого страницы в области свопинга, считывает страницу в оперативную память,

корректирует таблицу отображения виртуальных адресов в физические, и сообщает про- цессу о необходимости повторить операцию. Все эти действия невидимы для приложения, которое работает с виртуальной памятью. При этом один процесс не может прочитать что-либо из памяти (или записать в нее) другого процесса без «разрешения» на то со стороны подсистемы управления памятью. При такой организации работы крах одного процесса никак не повлияет на другие выполняющиеся процессы и на всю систему в целом.

Среди всех процессов можно выделить несколько особых типов процессов.

Системные процессы являются частью ядра и всегда находятся в оперативной памяти. Такие процессы не имеют соответствующих им программ в виде исполняемых файлов и запускаются особым образом при инициализации ядра системы. Примерами системных процессов являются планировщик процессов, диспетчер свопинга, диспетчер буферного кэша, диспетчер памяти ядра. Такие процессы являются фактически потоками ядра.

Демоны отличаются от обычных процессов только тем, что они работают в интерактивном режиме. Если с обычным процессом всегда ассоциирован какой-то терминал или псевдо терминал, через который осуществляется взаимодействие процесса с пользователем, то де- мон такого терминала не имеет. Демоны обычно используются для выполнения сервисных функций, обслуживания запросов от других процессов, причем не обязательно выпол- няющихся на данном компьютере. Пользователь не может непосредственно управлять демонами, он может влиять на их работу, только посылая им какие-то задания, например, отправляя документ на печать.

Главным демоном в системе является демон init[2]. Он является прародителем всех про- цессов в системе и имеет идентификатор 1. Выполнив задачи, поставленные в ему в файле inittab, демон init не завершает свою работу – он постоянно находится в памяти и отслеживает выполнение других процессов.

Прикладные процессы – это все остальные процессы, выполняющиеся в системе. Как правило, эти процессы порождаются в рамках сеанса работы пользователя. В каждом таком сеансе работы вначале запускается оболочка (командный интерпретатор) shell. Этот экземпляр оболочки называется login shell и завершение соответствующего процесса приводит к отключению пользователя от системы.

## Создание демона Linux

Для задачи демонизации будем использовать программу из предыдущей лабораторной работы. Она будет отслеживать состояние сетевого интерфейса и записывать результаты своей работы в системный журнал. Код демона представлен в листинге 1.

Листинг 1: Исходный код демона Linux (src/daemons/lin/main.cpp)

1 #i nc l ude <iostream>

2 #i nc l ude <fstream>

3 #i nc l ude <s tri ng >

4 #i nc l ude <regex>

5

6 #i nc l ude <uni s td . h>

7 #i nc l ude <sys / types . h>

8 #i nc l ude <sys / s ta t . h>

9

10 #i nc l ude <s y s l o g . h>

11 #i nc l ude <s i g n a l . h>

12 #i nc l ude <stdarg . h>

13

14 bool parse ( char *\** ifname , long long *\** rx\_bytes , long long *\** rx\_packets ,

15 long long *\** tx\_bytes , long long *\** tx\_packets ) ;

16

17 i n t main ( i n t argc , char *\** argv [ ] ) {

18 openlog ( " netmonitor " , LOG\_PID | LOG\_NDELAY | LOG\_PERROR, LOG\_DAEMON) ;

19 i f ( argc < 2 ) {

20 s y s l o g (LOG\_ERR, " Usage : %s interface\_name " , argv [ 0 ] ) ;

21 re turn 1 ;

22 }

23

24 i n t pid = f o rk ( ) ;

25 switch ( pid ) {

26 case 0 :

27 umask ( 0 ) ;

28 s e t s i d ( ) ;

29 c hdi r ( "/" ) ;

30 c l o s e (STDIN\_FILENO) ;

31 c l o s e (STDOUT\_FILENO) ;

32 c l o s e (STDERR\_FILENO) ;

33

34 {

35 long long rx\_bytes = 0 ;

|  |  |  |  |
| --- | --- | --- | --- |
| 36 | long | long | rx\_packets = 0 ; |
| 37 | long | long | tx\_bytes = 0 ; |
| 38 | long | long | tx\_packets = 0 ; |
| 39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68 | while ( true ) {  i f ( ! parse ( argv [ 1 ] , &rx tx\_packets ) ) {  s y s l o g (LOG\_ERR, "Can ’ [ 1 ] ) ;  re turn 1 ;  }  s y s l o g (LOG\_INFO,  "%s : \ n\ t Receiv tTransmit %  " bytes (% l l d argv [ 1 ] , rx\_b  tx\_packets )  us l e e p ( 2000 ) ;  }  }  case *−*1:  s y s l o g (LOG\_ERR, " Fai l : una c l o s e l o g ( ) ;  re turn 1 ; d e f a u l t :  s y s l o g (LOG\_NOTICE, "OK: dem break ;  }  c l o s e l o g ( ) ; re turn 0 ;  }  bool parse ( char *\** ifname , long lon | | |

\_bytes , &rx\_packets , &tx\_bytes , &

t f i n d such i n t e r f a c e : %s " , argv

e %l l d bytes (% l l d packets ) \n\ l l d "

packets ) " ,

ytes , rx\_packets , tx\_bytes ,

;

ble to f o rk " ) ;

on with pid %d i s c re ate d \n" , pid ) ;

g *\** rx\_bytes , long long *\** rx\_packets ,

69 long long *\** tx\_bytes , long long *\** tx\_packets ) {

70 std : : s t r i n g i n t e r f a c e ( i fname ) ;

71 i n t e r f a c e . append ( " : " ) ;

72 std : : s t r i n g buf f ;

73 std : : i f s tre a m n e ts ta t ( "/ proc / net/ dev" ) ;

74

75 while ( std : : g e t l i n e ( ne ts tat , buf f ) ) {

76 s i z e \_ t s h i f t = buf f . f ind\_ f i rst\_ not\_ of ( ’ ’ ) ;

77 i f ( buf f . compare ( s h i f t , i n t e r f a c e . l e ng th ( ) , i n t e r f a c e ) == 0 ) {

78 std : : regex rx (R" ( [ ^ [ : alpha : ] ] [ [ : d i g i t : ] ] + [ ^ [ : alpha : ] ] ) " ) ;

79 std : : s re g e x \_ i te ra to r pos ( buf f . cbegin ( ) , buf f . cend ( ) , rx ) ;

80

81 *\** rx\_bytes = std : : s t o l l ( pos*−*>s t r ( ) ) ;

82 ++pos ;

83 *\** rx\_packets = std : : s t o l l ( pos*−*>s t r ( ) ) ;

84 std : : advance ( pos , 7 ) ;

85 *\** tx\_bytes = std : : s t o l l ( pos*−*>s t r ( ) ) ;

86 ++pos ;

87 *\** tx\_packets = std : : s t o l l ( pos*−*>s t r ( ) ) ;

88

89 re turn true ;

90 }

91 }

92 re turn f a l s e ;

93 }

Логика работы самого приложения не изменилась, изменился только способ запуска. После проверки аргументов (стр. 19), приложение выполняет операцию fork() (стр. 24), после чего "родительская"часть спокойно завершает свою работу (стр. 59), а "дочернее"выполняет ряд операций, характерных для демона.

Для начала в процессе потомка нужно разрешить выставлять все биты прав на создаваемые файлы, это избавляет от проблемы с правами доступа (стр. 27). Потом создаётся новый сеанс, чтобы не зависеть от родителя (стр. 28). Далее осуществляется переход в корень диска, если этого не сделать, то могут быть проблемы, к примеру с размонтированием дисков (стр. 29). И в конце происходит закрытие дескрипторов ввода/вывода/ошибок, так как демону они не понадобятся (стр. 30-32).

После запуска демона, убедиться в его работоспособности можно так

enp2s0

user@host$ ps aux | grep netmonitor

sam 5776 31.0 0.0 13528 180 ?

Ss 21:25 0:13 ./netmonitor

user@host$

## Работа с системным журналом

Функция системного журналирования (т.н. "логи"или логирование) – это основной источ- ник информации о работе системы и ошибках. Журналирование может осуществляться на локальной системе, а так же сообщения журналирования могут пересылаться на удален- ную систему. Журналирование осуществляется при помощи демона syslogd или rsyslogd. Журнал обычно получает входную информацию при помощи сокета /dev/log (локально) или с udp-порта 514 (с удаленных машин)[3].

Соединение с журналом было установлено в строке 18. Первый параметр сообщил систем- ному журналу имя приложения, которое будет использоваться при ведении записей, а два оставшихся поля состоят из флагов флагов[2].

Предпоследнее поле (option) принимает дизъюнкцию следующих значений:

* + - **LOG\_CONS** написать сообщение об ошибке прямо на консоли, если была ошибка при записи данных в системный журнал;
    - **LOG\_NDELAY** устанавливать соединение немедленно (обычно оно устанавливается только при поступлении первого сообщения);
    - **LOG\_NOWAIT** не ожидает дочерние процессы которые могут быть созданы во время отправки этого сообщения
    - **LOG\_ODELAY**обратно от LOG\_NDELAY; открытие соединения откладывается до вызова syslog().
    - **LOG\_PERROR** посылать сообщение еще и в поток stderr;
    - **LOG\_PID** добавлять к каждому сообщению идентификатор

Последнее поле (facility) используется для указания типа программы, записывающей сообщения и принимает дизъюнкцию следующих значений:

* + - **LOG\_AUTH** сообщения о безопасности/авторизации (РЕКОМЕНДУЕТСЯ исполь- зовать вместо него LOG\_AUTHPRIV).
    - **LOG\_AUTHPRIV** сообщения о безопасности/авторизации (частные);
    - **LOG\_CRON** демон часов (cron и at);
    - **LOG\_DAEMON** другие системные демоны;
    - **LOG\_KERN** сообщения ядра;
    - **LOG\_LOCAL0 до LOG\_LOCAL7** зарезервированы для определения пользова- телем;
    - **LOG\_LOG\_LPR** подсистема принтера;
    - **LOG\_MAIL** почтовая подсистема;
    - **LOG\_NEWS** подсистема новостей USENET;
    - **LOG\_SYSLOG** сообщения, генерируемые syslogd;
    - **LOG\_USER** (по умолчанию) – общие сообщения на уровне пользователя;
    - **LOG\_UUCP** – подсистема UUCP

При записи сообщения, можно указать его тип (критичность) для последующей фильтрации (показывать сообщения не ниже определённого уровня). Это используется в строках 20, 42, 46, 56, 60.

Уровень важности сообщения по понижению:

* + - **LOG\_EMERG** система остановлена;
    - **LOG\_ALERT** требуется немедленное вмешательство;
    - **LOG\_CRIT** критические условия;
    - **LOG\_ERR** ошибки;
    - **LOG\_WARNING** предупреждения;
    - **LOG\_NOTICE** важные рабочие условия;
    - **LOG\_INFO** информационные сообщения;
    - **LOG\_DEBUG** сообщения об отладке.

В строке 64 соединение с системным логом закрывается.

Записи системного лога попадают в файл /var/log/syslog. В листинге 2 показан вывод (без форматирования) некоторых (10 последних) строк этого файла. Важно отметить, что когда системный журнал получает повторяющиеся события (т.е. состояние счётчиков на сетевой карте не успело измениться), он делает пометку о повторе, вместо прямого дублирования.

Листинг 2: Системный журнал Linux

1 Apr 23 2 1 : 2 8 : 2 9 spb netmonitor [ 5 7 7 6 ] : message repeated 3043 times : [ enp2 s 0 :#012#011 Receive 1535932378 bytes ( 1111392 packets )

#012#011 Transmit 202177932 bytes ( 700394 packets ) ]

2 Apr 23 2 1 : 2 8 : 2 9 spb netmonitor [ 5 7 7 6 ] : enp2 s 0 :#012#011 Receive

1535932378 bytes ( 1111392 packets )#012#011 Transmit 202178497 bytes ( 700395 packets )

3 Apr 23 2 1 : 2 8 : 2 9 spb netmonitor [ 5 7 7 6 ] : message repeated 25 times : [ enp2 s 0 :#012#011 Receive 1535932378 bytes ( 1111392 packets )

#012#011 Transmit 202178497 bytes ( 700395 packets ) ]

4 Apr 23 2 1 : 2 8 : 2 9 spb netmonitor [ 5 7 7 6 ] : enp2 s 0 :#012#011 Receive

1535932444 bytes ( 1111393 packets )#012#011 Transmit 202178497 bytes ( 700395 packets )

5 Apr 23 2 1 : 2 8 : 2 9 spb netmonitor [ 5 7 7 6 ] : message repeated 126 times : [ enp2 s 0 :#012#011 Receive 1535932444 bytes ( 1111393 packets )

#012#011 Transmit 202178497 bytes ( 700395 packets ) ]

6 Apr 23 2 1 : 2 8 : 2 9 spb netmonitor [ 5 7 7 6 ] : enp2 s 0 :#012#011 Receive

1535932738 bytes ( 1111394 packets )#012#011 Transmit 202178497 bytes ( 700395 packets )

7 Apr 23 2 1 : 2 8 : 2 9 spb netmonitor [ 5 7 7 6 ] : message repeated 12 times : [ enp2 s 0 :#012#011 Receive 1535932738 bytes ( 1111394 packets )

#012#011 Transmit 202178497 bytes ( 700395 packets ) ]

8 Apr 23 2 1 : 2 8 : 2 9 spb netmonitor [ 5 7 7 6 ] : enp2 s 0 :#012#011 Receive

1535932738 bytes ( 1111394 packets )#012#011 Transmit 202178563 bytes ( 700396 packets )

9 Apr 23 2 1 : 2 8 : 2 9 spb netmonitor [ 5 7 7 6 ] : message repeated 12 times : [ enp2 s 0 :#012#011 Receive 1535932738 bytes ( 1111394 packets )

#012#011 Transmit 202178563 bytes ( 700396 packets ) ]

10 Apr 23 2 1 : 2 8 : 2 9 spb netmonitor [ 5 7 7 6 ] : enp2 s 0 :#012#011 Receive

1535933590 bytes ( 1111395 packets )#012#011 Transmit 202178629 bytes ( 700397 packets )

# Дистрибуция пакетов в Linux

Программное обеспечение в ОС Ubuntu Linux распространяется в так называемых deb- пакетах. Обычно при установке программы из репозитория система автоматически ска- чивает и устанавливает deb-пакеты. Главной причиной использовать этот путь является автоматическое разрешение зависимостей. Программу можно установить, только если уже установлены пакеты, от которых она зависит. Такая схема позволяет избежать дублиро- вания данных в пакетах (например, если несколько программ зависят от одной и той же библиотеки, то не придётся пихать эту библиотеку в пакет каждой программы – она поста- вится один раз отдельным пакетом). В отличие от, например, Slackware или Windows, в Ubuntu зависимости разрешаются пакетным менеджером (Synaptic, apt, Центр приложений, apt-get, aptitude) – он автоматически установит зависимости из репозитория. Зависимости придётся устанавливать вручную, если нужный репозиторий не подключен, недоступен, ес- ли нужного пакета нет в репозитории, если вы ставите пакеты без использования пакетного менеджера (используете Gdebi или dpkg), если вы устанавливаете программу не из пакета (компилируете из исходников, запускаете установочный run/sh скрипт). Операционные системы на базе Debian распространяют пакеты deb, на базе RedHat – rpm.

## Создание DEB/RPM/TGZ пакетов

CheckInstall – это удобная утилита, позволяющая создавать бинарные пакеты для Linux из исходного кода приложения. После компиляции программного обеспечения checkinstall может автоматически сгенерировать Slackware-, RPM- или Debian-совместимый пакет, который впоследствии может быть полностью удалён через соответствующий менеджер пакетов. Эта возможность является предпочтительной при установке любых пакетов[4].

### Установка программы checkinstall

Установка пакета checkinstall не должна вызвать особых сложностей. В операционных системах, использующих DEB пакеты, установка производится командой:

user@host$ sudo apt-get install checkinstall

В операционной системе, использующей RPM пакеты, установка пакета checkinstall выпол- няется командой:

user@host$ sudo rpm -i checkinstall

Если такой пакет в Вашей ОС не обнаружен, то следует посетить домашнюю страницу проекта и скачать требуемую версию для Вашего дистрибутива:

<http://checkinstall.izto.org/download.php>

### Компилирование исходников

Далее следует перейти в каталог с программой и провести её компиляцию.

Программа, которая была рассмотрена в предыдущем разделе может быть собрана следу- ющим образом.

user@host$ g++ --std=c++14 main.cpp -o netmonitor

### Создание DEB-пакета из исходного кода

Программа checkinstall создает и устанавливает пакет для основных ОС. Тип пакета (DEB или RPM) checkinstall определяет сам. Для жесткого указания типа создаваемого пакета используем команду checkinstall с ключами:

Создает и устанавливает RPM пакет

user@host$ sudo checkinstall -R

Создает и устанавливает DEB пакет

user@host$ sudo checkinstall -D

Создает и устанавливает TGZ пакет (дистрибутивы: Slackware, Zenwalk, DeepStyle, Vektorlinux, Mops)

user@host$ sudo checkinstall -S

Далее следует ответить на несколько вопросов. По умолчанию все ответы на задаваемые вопросы подходят в большинстве случаев, поэтому везде нажимаем Enter.

## Создание PKGBUILD

Пользовательский репозиторий Arch Linux (Arch User Repository, AUR) – это поддер- живаемое сообществом хранилище ПО для пользователей Arch. Он содержит описания пакетов (файлы PKGBUILD), которые позволят скомпилировать пакет из исходников с помощью makepkg и затем установить его, используя pacman. В AUR пользователи могут добавлять свои собственные сборки пакетов (PKGBUILD и другие необходимые файлы). Сообществу предоставлена возможность голосовать за эти пакеты или против них. Если пакет становится популярным, распространяется под подходящей лицензией и может быть

собран без дополнительных сложностей, то, вероятно, он будет перенесен в репозиторий community (непосредственно доступный при помощи утилит pacman и abs)[4].

Файл PKGBUILD по сути напоминает Makefile, и требует установки значений следующих переменных в зависимости от пакета:

* + - pkgname – название пакета. Можно использовать только строчные английские буквы. Значение этой переменной большой роли не играет, но может помочь, если установить сюда имя рабочей директории, или, например, имя файла с исходным кодом (\*.tar.gz), который требуется загрузить
    - pkgver – версия пакета. Эта переменная может содержать буквы, цифры, знаки препинания, но не может содержать дефисов. Содержимое этой переменной зависит от метода присвоения версий (major.minor.bugfix, major.date, и т.д.) который исполь- зует программа. Чтобы следующие шаги были наиболее эффективными и лёгкими, рекомендуется включить номер версии в имя файла с исходным кодом.
    - pkgrel – число, которое нужно увеличивать каждый раз после новой сборки пакета. При первой сборке пакета значение pkgrel должно быть установлено в "1". Цель этой переменной состоит в том, чтобы различать разные сборки пакета одной и той же версии.
    - pkgdesc – краткое описание пакета, обычно не более 76 символов.
    - arch – список архитектур, где может быть использован данный PKGBUILD (обычно это "i686").
    - url – адрес веб-сайта программы, где заинтересовавшиеся могут получить более подробную информацию о программе.
    - license – тип лицензии (может быть ’unknown’).
    - depends – список пакетов, разделенный пробелами, которые должны быть установлены до использования пакета. Во избежании проблем, имена пакетов заключаются в апострофы (’), а весь массив в скобки. Используя математическое "больше или равно можно указать минимальную допустимую версию пакета-зависимости.
    - makedepends – список пакетов, которые потребуются для сборки пакета, но которые не нужны для его использования.
    - provides – список пакетов, необходимость в которых пропадает, так как собираемый пакет выполняет, по крайней мере, похожие функции.
    - conflicts – список пакетов, которые, если установлены, могут создать проблемы во время использования собираемого пакета.
* replaces – список пакетов, которые заменит собираемый пакет.
* source – список файлов, которые потребуются во время сборки пакета. Здесь должна быть ссылка на архив с исходным кодом программы (в большинстве случаев такая ссылка представляет из себя HTTP или FTP ссылку, заключённую в кавычки).
* md5sums – список контрольных сумм для файлов из предыдущей переменной, разде- ленных пробелами и заключённых в апострофы. Как только станут доступны все файлы из списка source, md5 суммы файлов будут автоматически сгенерированы и проверены на соответствие с этим списком.

# Фоновые приложения в Windows

## Службы Windows

Служба (сервис от англ. service) - это программы, которые автоматически запускаются системой при загрузке Windows и выполняются в любом случае, вне зависимости от действий пользователя.

В большинстве случаев службам запрещено взаимодействие с консолью или рабочим столом пользователей (как локальных, так и удалённых), однако для некоторых сервисов возможно исключение — взаимодействие с консолью (сессией с номером 0, в которой зарегистрирован пользователь локально или при запуске службы mstsc с ключом /console).

Существует четыре режима для сервисов:

* + - запрещён к запуску;
    - ручной запуск (по запросу);
    - автоматический запуск при загрузке компьютера;
    - обязательный сервис (автоматический запуск и невозможность (для пользователя) остановить сервис).

Windows предлагает программу Service Control Manager, с её помощью можно управлять созданием, удалением, запуском и остановкой служб. Приложение, имеющее статус сервиса, должно быть написано таким образом, чтобы оно могло принимать сообщения от Service Control Manager. Затем, одним или несколькими вызовами API, имя службы и другие атрибуты, такие, как его описание, регистрируются в Service Control Manager.

Список служб находится в ветке реестра HKEY\_LOCAL\_MACHINE\SYSTEM\CurrentControlSet\S Значения параметра «Start» имеют тип «REG\_DWORD» и могут принимать значения:

«0», «1», «2», «3» и «4» (когда служба не запускается, то есть запуск данной службы запрещен)[5].

Сервисы Windows по умолчанию запускаются от имени пользователя «LocalSystem», кото- рый обладает полными правами в системе (превосходящими права даже учётной записи Administrator). Рабочим каталогом будет системный каталог Windows (обычно C:\WINNT или C:\WINDOWS), а каталог для хранения временных файлов будет C:\WINNT\TEMP.

Поскольку это не настоящий пользователь, а «виртуальный», появляются некоторые трудности, когда приложению необходимо сохранить данные, относящиеся к пользователю (user-specific data), поскольку не существует папки этого пользователя.

Важно также то, что в случае если служба работает от имени локального пользователя (реальный пользователь созданный для служебных целей) если пароль такого пользователя изменён, сервис не будет запускаться до тех пор, пока пароль для сервиса тоже не будет изменен.

## Создание службы Windows с помощью программы Sc.exe

Этот способ является рекомендованным корпорацией Microsoft[6].

Для создания служб Windows можно использовать программу Sc.exe, включенную в пакет ресурсов Resource Kit, которая реализует вызовы ко всем функциям интерфейса приклад- ного программирования (API) управления службами Windows. Настроить параметры для этих функций можно, задав их в командной строке. С помощью средства Sc.exe имеется возможность запросить состояние службы и получить значения, хранящиеся в полях струк- туры состояний. SC позволяет задавать имя удаленного компьютера, что дает возможность вызвать функции интерфейса API службы и посмотреть структуры состояния службы на удаленном компьютере.

Кроме того, Sc.exe позволяет вызвать любую функцию интерфейса API управления служ- бами и изменить любой параметр, используя командную строку. Данное средство предо- ставляет удобный способ создания и изменения записей службы в реестре и в базе данных диспетчера служб. Для настройки службы нет необходимости вручную создавать записи в реестре и затем перезагружать компьютер, чтобы обеспечить обновление базы данных диспетчером служб.

Программа Sc.exe использует следующий синтаксис:

sc [Servername] Command Servicename

Команда **sc create** создает запись службы в реестре и в базе данных диспетчера служб. Синтаксис

sc [Servername] create Servicename [Optionname=Optionvalue...

Параметры могут быть следующими:

* + - Servername – необязательный параметр. Задает имя удаленного сервера, на котором будут запускаться команды.
    - Command – задает команду sc. Команды могут быть следующие:
      * Config – изменяет конфигурацию службы (постоянные параметры).
      * Continue – посылает службе запрос Continue.
      * Control – посылает службе запрос Control.
      * Create – создает службу (добавляет ее в реестр).
      * Delete – удаляет службу (из реестра).
      * EnumDepend – перечисляет зависимости служб.
      * GetDisplayName – указывает отображаемое имя службы.
      * GetKeyName – указывает имя раздела службы.
      * Interrogate – посылает службе запрос Interrogate.
      * Pause – посылает службе запрос Pause.
      * qc – запрашивает конфигурацию службы.
      * Query – запрашивает состояние службы или указывает состояние по типам служб.
      * Start – запускает службу.
      * Stop – посылает службе запрос Stop.
    - Servicename – указывает имя, присвоенное разделу службы в реестре.
    - Optionname – служит для указания имен и значений дополнительных параметров.
    - Optionvalue – задает значение параметра, которому присвоено имя параметром

«Optionname».

Для выполнения ряда команд необходимо иметь права администратора. Следовательно, необходимо обладать правами администратора на компьютере, на котором создается служба.

Запустим netmonitor в качестве сервиса

Sc create MyService binPath=C:\netmonitor.exe DisplayName=*′′*My New Service*′′* type=own

По умолчанию создается служба типа WIN32\_SHARE\_PROCESS с типом запуска SERVICE\_DEM Она не имеет никаких зависимостей и выполняется в контексте безопасности LocalSystem.

Результат добавления приложения в список сервисов показан на рисунке 1.

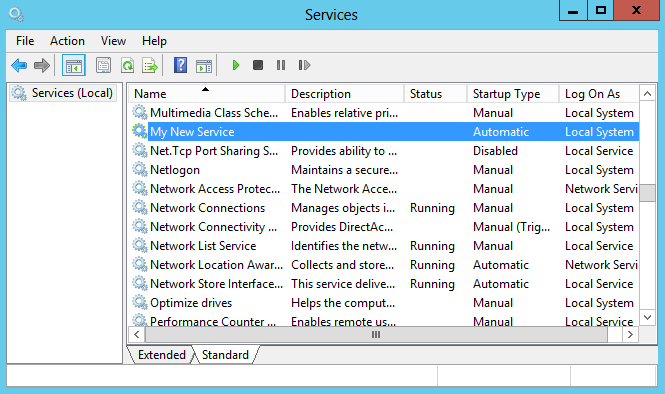


Рис. 1: Добавление сервиса из приложения в windows

## Создание службы Windows с помощью PowerShell

Возможности управления системой из консоли в последних версиях Windows были значи- тельно расширены. В том числе стало доступно и управление службами Windows. Создать новую службу можно с помощью командлета New-Service. Создадим такой же сервис, как и в предыдущем примере, только добавим к нему описание (Description):

New-Service -Name MyService -BinaryPathName C:\netmonitor.exe‘

-DisplayName *′′*My New Service*′′* -Description *′′*Very Important Service !!!*′′*

Изменить параметры службы можно командлетом Set-Service:

Set-Service -Name MyService -Description *′′*Not Very Important Service*′′* -StartupType Ma

PowerShell имеет примерно такой же функционал как и Sc.exe. Его особенностью является добавление описаний, но он не имеет простого способа удаления сервисов.

## Работа с системным журналом Windows

Взаимодействие с системным журналом в Windows несколько сложнее, чем в Linux. Для начала требуется создать манифест (mc-файл) с описанием сообщений (листинг 2)[7].

Листинг 3: mc-файл с описанием сообщений(src/daemons/win/eventlog.mc)

1 ;# i f n d e f \_EVENT\_LOG\_MESSAGE\_FILE\_H\_

2 ;# d e f i n e \_EVENT\_LOG\_MESSAGE\_FILE\_H\_

3

4 MessageIdTypeDef=DWORD

5

6

7 Severity Names=( Success=0x0 : STATUS\_SEVERITY\_SUCCESS

8 Inf o rmati o nal=0x1 :STATUS\_SEVERITY\_INFORMATIONAL

9 Warning=0x2 :STATUS\_SEVERITY\_WARNING

10 Error=0x3 :STATUS\_SEVERITY\_ERROR

11 )

12

13 LanguageNames=(EnglishUS=0x401 : MSG00401

14 Neutral=0x0000 : MSG00000

15 )

16

17 Message Id=0x0 SymbolicName=MSG\_INFO\_1

18 S e v e r i ty=Inf o rmati o nal

19 F a c i l i t y=Appl i c ati o n

20 Language=Neutral

21 %1

22 .

23

24 Message Id=0x1 SymbolicName=MSG\_WARNING\_1

25 S e v e r i ty=Warning

26 F a c i l i t y=Appl i c ati o n

27 Language=Neutral

28 %1

29 .

30

31 Message Id=0x2 SymbolicName=MSG\_ERROR\_1

32 S e v e r i ty=Error

33

F a c i l i t y=Appl i c ati o n

Language=Neutral

%1

.

Message Id=0x3 SymbolicName=MSG\_SUCCESS\_1

S e v e r i ty=Success

F a c i l i t y=Appl i c ati o n Language=Neutral

%1

.

;# e n d i f

34

35

36

37

38

39

40

41

42

43

44

45

46

Следующие две команды генерируют ресурсный файл и хедер для общения с этим ресурс- ным файлом.

mc.exe -A -b -c -h . -r resources eventlog.mc

rc.exe -foresources/eventlog.res resources/eventlog.rc

После этого остаётся добавить eventlog.res при линковке бинарника, а eventlog.h подключить к основному модулю программы. Использование API системного лога Windows показано в листинге 3.

Листинг 4: Исходный код службы Windows, демонстрирующий API системного журнала (src/daemons/win/main.cpp)

#i nc l ude <iostream>

#i nc l ude <windows . h>

#i nc l ude <i ph l p api . h>

#i nc l ude " e ve ntl o g . h"

bool Get I f Table (PMIB\_IFTABLE*\** m\_pTable) ;

void install\_ event\_ l og\_ source ( const std : : s t r i n g& a\_name) {

const std : : s t r i n g key\_path (

"SYSTEM\\ Current Control Set \\ S e r v i c e s \\" " EventLog \\ Appl i c ati o n \\" +

1

2

3

4

5

6

7

8

9

10

11

12

13 a\_name) ;

14

15 HKEY key ;

16

17 DWORD l a s t\_ e r ro r =

18 RegCreateKeyEx (HKEY\_LOCAL\_MACHINE, key\_path . c\_str ( ) , 0 , 0 ,

19 REG\_OPTION\_NON\_VOLATILE, KEY\_SET\_VALUE, 0 , &key

, 0 ) ;

20

21 i f (ERROR\_SUCCESS == l a s t\_ e r ro r ) {

22

23 UINT max\_path = 512 *\** s i z e o f (WCHAR) ;

24 HMODULE hModule = GetModuleHandleW (NULL) ;

25 WCHAR exe\_path [ max\_path ] ;

26 GetModuleFileNameW ( hModule , exe\_path , max\_path) ;

27

28 //BYTE exe\_path [ ] = "C: \\ path \\ to \\ your \\ a p p l i c a t i o n . exe ";

29 DWORD l a s t\_ e r ro r ;

30 const DWORD types\_supported =

31 EVENTLOG\_ERROR\_TYPE | EVENTLOG\_WARNING\_TYPE | EVENTLOG\_INFORMATION\_TYPE;

32

33 l a s t\_ e r ro r = RegSetValueEx ( key , " EventMessage File " , 0 , REG\_SZ, exe\_path ,

34 s i z e o f ( exe\_path ) ) ;

35

36 i f (ERROR\_SUCCESS == l a s t\_ e r ro r ) {

37 l a s t\_ e r ro r =

38 RegSetValueEx ( key , " TypesSupported" , 0 , REG\_DWORD,

39 (LPBYTE)&types\_supported , s i z e o f ( types\_supported ) ) ;

40 }

41

42 i f (ERROR\_SUCCESS != l a s t\_ e r ro r ) {

43 std : : c e r r << " Fai l e d to i n s t a l l so urc e val ue s : " << l a s t\_ e r ro r

<< "\n" ;

44 }

45

46 RegCloseKey ( key ) ;

47 } e l s e {

48 std : : c e r r << " Fai l e d to i n s t a l l so urc e : " << l a s t\_ e r ro r << "\n" ;

49 }

50 }

51

52 void log\_event\_log\_message ( const std : : s t r i n g& a\_msg , const WORD a\_type ,

53 const std : : s t r i n g& a\_name) {

54 DWORD event\_id ;

55

56 switch ( a\_type ) {

57 case EVENTLOG\_ERROR\_TYPE:

58 event\_id = MSG\_ERROR\_1;

59 break ;

60 case EVENTLOG\_WARNING\_TYPE:

61 event\_id = MSG\_WARNING\_1;

62 break ;

63 case EVENTLOG\_INFORMATION\_TYPE:

64 event\_id = MSG\_INFO\_1;

65 break ;

66 d e f a u l t :

67 std : : c e r r << " Unrecognised type : " << a\_type << "\n" ;

68 event\_id = MSG\_INFO\_1;

69 break ;

70 }

71

72 HANDLE h\_event\_log = Register Event Source ( 0 , a\_name . c\_str ( ) ) ;

73

74 i f ( 0 == h\_event\_log ) {

75 std : : c e r r << " Fai l e d open so urc e ’ " << a\_name << " ’ : " << GetLastError ( )

76 << "\n" ;

77 } e l s e {

78 LPCTSTR message = a\_msg . c\_str ( ) ;

79

80 i f (FALSE ==

81 ReportEvent ( h\_event\_log , a\_type , 0 , event\_id , 0 , 1 , 0 , & message , 0 ) ) {

82 std : : c e r r << " Fai l e d to w r i te message : " << GetLastError ( ) << "\n" ;

83 }

84

85 Deregister Event Source ( h\_event\_log ) ;

86 }

87 }

88

89 void uninstall\_ event\_ log\_ source ( const std : : s t r i n g& a\_name) {

90 const std : : s t r i n g key\_path (

91 "SYSTEM\\ Current Control Set \\ S e r v i c e s \\"

92 " EventLog \\ Appl i c ati o n \\" +

93 a\_name) ;

94

95 DWORD l a s t\_ e r ro r = RegDeleteKey (HKEY\_LOCAL\_MACHINE, key\_path . c\_str ( ) ) ;

96

97 i f (ERROR\_SUCCESS != l a s t\_ e r ro r ) {

98 std : : c e r r << " Fai l e d to u n i n s t a l l sourc e : " << l a s t\_ e r ro r << "\n " ;

99 }

100 }

101

102 i n t main ( i n t argc , char *\** argv [ ] ) {

103 PMIB\_IFTABLE m\_pTable = NULL;

104 const std : : s t r i n g event\_log\_source\_name ( " netmonitor " ) ;

105 install\_ event\_ l og\_ source ( event\_log\_source\_name ) ;

106 log\_event\_log\_message ( " Netmonitor s t a r t " , EVENTLOG\_WARNING\_TYPE,

107 event\_log\_source\_name ) ;

108

109 i f ( Get I f Table(&m\_pTable) == f a l s e ) {

110 re turn 1 ;

111 }

112

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 113 |  | // Обход списка с е те вых интерфейс ов | | |
| 114 |  | std : : s t r i n g buf f ; | | |
| 115 |  | f o r (UINT i = 0 ; i < m\_pTable*−*>dwNumEntries ; i ++) { | | |
| 116 |  | MIB\_IFROW Row = m\_pTable*−*>t ab l e [ i ] ; | | |
| 117 |  | char sz Descr [ MAXLEN\_IFDESCR ] ; | | |
| 118 |  | memcpy( sz Descr , Row. bDescr , Row. dwDescrLen ) ; | | |
| 119 |  | sz Descr [ Row. dwDescrLen ] = 0 ; | | |
| 120 |  |  | | |
| 121 |  | // Выв од с обранной информации | | |
| 122 |  |  | | |
| 123 |  | buf f . append ( sz Descr ) ; | | |
| 124 |  | buf f . append ( " : \ n" ) ; | | |
| 125 |  | buf f . append ( "\ t Received : " ) ; | | |
| 126 |  | buf f . append (Row. dw InOctets ) ; | | |
| 127 |  | buf f . append ( " , Sent : " ) ; | | |
| 128 |  | buf f . append (Row. dwOutOctets ) ; | | |
| 129 |  | buf f . append ( "\n\n" ) ; | | |
| 130 |  | log\_event\_log\_message ( buf f . c\_str ( ) , EVENTLOG\_INFORMATION\_TYPE, | | |
| 131 |  | event\_log\_source\_name ) ; | | |
| 132 |  | buf f . c l e a r ( ) ; | | |
| 133 |  | } | | |
| 134 |  |  | | |
| 135 |  | // Завершение работы | | |
| 136 |  | log\_event\_log\_message ( " Netmonitor end" , EVENTLOG\_WARNING\_TYPE, | | |
| 137 |  | event\_log\_source\_name ) ; | | |
| 138 |  | d e l e te ( m\_pTable) ; | | |
| 139 |  | char a = getchar ( ) ; | | |
| 140 |  | re turn 0 ; | | |
| 141 | } |  | | |
| 142 |  |  | | |
| 143  144  145 | bool Get I f Table (PMIB\_IFTABLE*\** m\_pTable) {  // Тип указателя на функцию Get I f Table typedef DWORD( \_ s tdc a l l *\** TGetIf Table ) ( | | | |
| 146 | MIB\_IFTABLE *\** p If Table , | | // | Буфер таблицы интерфейс ов |
| 147 | ULONG *\** pdwSize , | | // | Размер буфера |
| 148 | BOOL bOrder ) ; | | // | Сортировать таблицу? |
| 149 |  | |  |  |

подгрузить i p hl p api . d l l h l papi ;

|  |  |  |
| --- | --- | --- |
| 150  151  152  153  154  155  156  157  158  159  160  161  162  163  164  165  166  167  168  169  170  171  172 |  | // Пыта емся HINSTANCE i p  i p hl pap i = L i f ( ! i ph l pap log\_event\_  EVENTLO  re turn f a l  }  // Получаем TGetIf Table p GetIf Table  // Получили DWORD m\_dw p GetIf Table (  *\**m\_pTable =  i f ( p GetIf Ta  {  log\_event\_ EVENTLO  d e l e te *\**m re turn f a l |
| 173 |  | } |
| 174 |  |  |
| 175 |  | re turn true ; |
| 176 | } |  |

oad Library (L" i p hl p ap i . d l l " ) ; i ) {

log\_message ( " i p hl pap i . d l l not supported " , G\_ERROR\_TYPE,

event\_log\_source\_name ) ;

s e ;

адре с функции p GetIf Table ;

= ( TGetIf Table ) GetProcAddress ( i phl papi , " Get I f Table " ) ;

тре буемый размер буфера Adapters = 0 ;

*\** m\_pTable , &m\_dwAdapters , TRUE) ;

new MIB\_IFTABLE[ m\_dwAdapters ] ;

ble ( *\** m\_pTable , &m\_dwAdapters , TRUE) != ERROR\_SUCCESS)

log\_message ( " Error while Get I f Table " , G\_ERROR\_TYPE,

event\_log\_source\_name ) ;

\_pTable ; s e ;

# Дистрибуция пакетов в Windows

В мире Windows распространение программ осуществляется при помощи инсталляционных пакетов.

Inno Setup – система создания инсталляторов для Windows программ с открытым исходным кодом. Впервые выпущенный в 1997 году, отличается функциональности и стабильности. Кроме того, обладает интерфейсом, к которому привыкли многие пользователи.

Inno Setup графическим интерфейсом, который (по средствам мастера) позволяет создать скрипт, на основании которого генерируется установочный пакет. Скрипт для разрабаты- ваемой программы netmonitor представлен в листинге 5.

Листинг 5: Скрипт генерации установочного файла

1 ; S c r i p t generated by the Inno Setup S c r i p t Wizard .

2 ; SEE THE DOCUMENTATION FOR DETAILS ON CREATING INNO SETUP SCRIPT FILES !

3

4 #d e f i n e MyAppName " NetMonitor"

5 #d e f i n e MyAppVersion " 1 . 0 "

6 #d e f i n e MyAppPublisher "Semen Martynov"

7 #d e f i n e MyAppURL " https : // c l o u d t i p s . org "

8 #d e f i n e MyAppExeName " netmonitor . exe "

9

10 [ Setup ]

11 ; NOTE: The value o f AppId uniquely i d e n t i f i e s t h i s a p p l i c a t i o n .

12 ; Do not use the same AppId value in i n s t a l l e r s f o r other a p p l i c a t i o n s .

13 ; (To g e ne rate a new GUID, c l i c k Tools | Generate GUID i n s i d e the IDE . )

14 AppId={{55125EB5*−*4278*−*440E*−*82D9*−*638FD219F5F5}

15 AppName={#MyAppName}

16 AppVersion={#MyAppVersion}

17 ; AppVerName={#MyAppName} {#MyAppVersion}

18 AppPublisher={#MyAppPublisher}

19 AppPublisherURL={#MyAppURL}

20 AppSupportURL={#MyAppURL}

21 AppUpdatesURL={#MyAppURL}

22 DefaultDirName={pf }\{#MyAppName}

23 DisableProgramGroupPage=yes

24 OutputBaseFilename=setup

25 Compression=lzma

26 Solid Compression=yes

27

28 [ Languages ]

29 Name : " e n g l i s h "; Messages File : " compiler : Def ault . i s l "

30

31 [ Tasks ]

32 Name : " desktopicon "; D e s c r i pti o n : "{cm : Create Desktop Icon } ";

Group Description : "{cm : A ddi t i o nal I c o ns } "; Flags : unchecked

33

34 [ F i l e s ]

35 Source : "C: \ Program F i l e s ( x86 ) \ NetMonitor\ netmonitor . exe "; DestDir : "{ app } "; Flags : i g n o r e v e r s i o n

36 Source : "C: \ Users \ user \Documents\ Visual Studio 2015\ Pro j e c ts \

netmonitor \ x64 \ Release \ netmonitor . exe "; DestDir : "{ app } "; Flags : i g n o r e v e r s i o n

37 ; NOTE: Don ’ t use " Flags : i g n o r e v e r s i o n " on any shared system f i l e s

38

39 [ Ico ns ]

40 Name : "{ commonprograms}\{#MyAppName} "; Filename : "{ app}\{# MyAppExeName}"

41 Name : "{ commondesktop}\{#MyAppName} "; Filename : "{ app}\{# MyAppExeName} "; Tasks : desktopicon

42

43 [ Run ]

44 Filename : "{ app}\{#MyAppExeName} "; D e s c r i pti o n : "{cm : LaunchProgram

,{# String Change (MyAppName, ’& ’ , ’&& ’) }}"; Flags : nowait p o s t i n s t a l l s k i p i f s i l e n t

Получив установочный пакет, можно его распространять на других Windows-системах. Установка также происходит при помощи графического мастера и не должна вызывать сложности у пользователя (Рисунок 2).

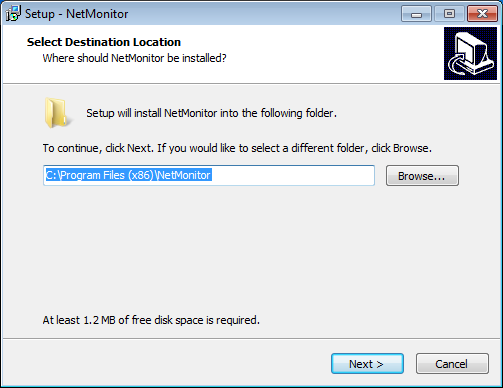


Рис. 2: Интерфейс установки приложения netmonitor

# Заключение

Программы, работающие в фоновом режиме широко распространены как в Windows так и в Linux. Для создания сервисов Windows можно использовать встроенные средства, которые значительно упрощают жизнь администратора (раньше этот процесс был много сложнее), в то время как в Linux демон должен пройти определённое количество обязательных шагов (форк, переход в корень файловой системы, закрытие стандартных дескрипторов...). С точки зрения использования системного журнала для логирования программы, то в Linux это реализованно на много проще (всего три posix вызова), чем в Windows (где требуется создавать файл манифеста).

Дистрибуция программ производится одинаково легко как в Windows так и в Linux бла- годаря утилитам с удобным графическим/текстовым интерфейсом. Стоит отметить, что установочный пакет Windows как правило самодостаточный, в то время как в Linux широко распространена политика зависимостей, для решения которых во время установки может потребоваться доступ к интернету и репозиторию основных пакетов.

# Список литературы

[1] Лав Р. Linux. Системное программирование. 2-е изд. – СПб.: Питер, 2014 – 448 стр. [2] Собел М. Г. Linux. Администрирование и системное программирование. – СПб.: Питер

, 2011 – 880 стр.

[3] Иванов Н. Программирование в Linux. 2-е изд. – СПб.: БХВ-Петербург, 2012 – 400 стр. [4] Уорд Б. Внутреннее устройство Linux. – СПб.: Питер, 2016 – 384 стр.

[5] Петцольд Ч. Программирование для Microsoft Windows 8. – СПб.: Питер, 2014 – 1008 стр.

[6] Microsoft Software Developer Network, ст. kb251192. Создание службы Windows с помо- щью программы Sc.exe. https://support.microsoft.com/ru-ru/kb/251192 [Дата обращения 25 апреля 2016]

[7] Харт Дж. Системное программирование в среде Windows. – M.: 2005