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

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 обеспечивает 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 позволяет 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 решается распространением бинарных пакетов, которые необходимо правильно организовать на компьютере пользователя.

1 Фоновые приложения в Linux

1.1 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 переключения между процессами по истечении кванта времени, выполняется фиксация и сохранение в памяти состояния программы. Этот снимок содержит информацию о состоянии 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 выполнение 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 адресное пространство. Его размер может даже превышать размер реальной оперативной памяти, что обеспечивается за счет применения страничной организации памяти и механизма 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 процессу 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 демон такого терминала не имеет. Демоны обычно используются для выполнения сервисных функций, обслуживания запросов от других процессов, причем не обязательно выполняющихся 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 процессов 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.

1.2 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)

```
#include <iostream>
#include <fstream>
#include <string>
#include <regex>

#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>

#include <syslog.h>
#include <signal.h>
#include <stdarg.h>

bool parse(char* ifname, long long* rx_bytes, long long* rx_packets,
           long long* tx_bytes, long long* tx_packets);

int main(int argc, char* argv[]) {
  openlog("netmonitor", LOG_PID | LOG_NDELAY | LOG_PERROR,
     LOG_DAEMON);
  if (argc < 2) {
    syslog(LOG_ERR, "Usage: %s interface_name", argv[0]);
    return 1;
  }

  int pid = fork();
  switch (pid) {
    case 0:
      umask(0);
      setsid();
      chdir("/");
      close(STDIN_FILENO);
      close(STDOUT_FILENO);
      close(STDERR_FILENO);

      {
        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          while (true) {
41            if (!parse(argv[1], &rx_bytes, &rx_packets, &tx_bytes, &
                tx_packets)) {
42              syslog(LOG_ERR, "Can't find such interface: %s", argv
                  [1]);
43              return 1;
44            }
45
46            syslog(LOG_INFO,
47                   "%s:\n\tReceive %lld  bytes (%lld packets)\n\
                      tTransmit %lld "
48                   " bytes (%lld packets)",
49                   argv[1], rx_bytes, rx_packets, tx_bytes,
                      tx_packets);
50
51            usleep(2000);
52          }
53        }
54
55      case -1:
56        syslog(LOG_ERR, "Fail: unable to fork");
57        closelog();
58        return 1;
59      default:
60        syslog(LOG_NOTICE, "OK: demon with pid %d is created\n", pid);
61        break;
62    }
63
64    closelog();
65    return 0;
66  }
67
68  bool parse(char* ifname, long long* rx_bytes, long long* rx_packets,
```

```
69              long long* tx_bytes, long long* tx_packets) {
70    std::string interface(ifname);
71    interface.append(":");
72    std::string buff;
73    std::ifstream netstat("/proc/net/dev");
74
75    while (std::getline(netstat, buff)) {
76      size_t shift = buff.find_first_not_of(' ');
77      if (buff.compare(shift, interface.length(), interface) == 0) {
78        std::regex rx(R"([^[:alpha:]][[:digit:]]+[^[:alpha:]])");
79        std::sregex_iterator pos(buff.cbegin(), buff.cend(), rx);
80
81        *rx_bytes = std::stoll(pos->str());
82        ++pos;
83        *rx_packets = std::stoll(pos->str());
84        std::advance(pos, 7);
85        *tx_bytes = std::stoll(pos->str());
86        ++pos;
87        *tx_packets = std::stoll(pos->str());
88
89        return true;
90      }
91    }
92    return false;
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

```
user@host$ ps aux | grep netmonitor
sam       5776 31.0  0.0  13528   180 ?        Ss   21:25   0:13 ./netmonitor enp2s0
user@host$
```

1.3 Работа с системным журналом

Функция системного журналирования (т.н. "логи"или логирование) – это основной источник информации о работе системы и ошибках. Журналирование может осуществляться на локальной системе, а так же сообщения журналирования могут пересылаться на удаленную 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 системному 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 использовать 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 пользователем;
- **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 21:28:29 spb netmonitor[5776]: message repeated 3043 times: [
       enp2s0:#012#011Receive 1535932378  bytes (1111392 packets)
     #012#011Transmit 202177932  bytes (700394 packets)]
2  Apr 23 21:28:29 spb netmonitor[5776]: enp2s0:#012#011Receive
     1535932378  bytes (1111392 packets)#012#011Transmit 202178497
     bytes (700395 packets)
3  Apr 23 21:28:29 spb netmonitor[5776]: message repeated 25 times: [
     enp2s0:#012#011Receive 1535932378  bytes (1111392 packets)
     #012#011Transmit 202178497  bytes (700395 packets)]
4  Apr 23 21:28:29 spb netmonitor[5776]: enp2s0:#012#011Receive
     1535932444  bytes (1111393 packets)#012#011Transmit 202178497
     bytes (700395 packets)
5  Apr 23 21:28:29 spb netmonitor[5776]: message repeated 126 times: [
     enp2s0:#012#011Receive 1535932444  bytes (1111393 packets)
     #012#011Transmit 202178497  bytes (700395 packets)]
6  Apr 23 21:28:29 spb netmonitor[5776]: enp2s0:#012#011Receive
     1535932738  bytes (1111394 packets)#012#011Transmit 202178497
     bytes (700395 packets)
7  Apr 23 21:28:29 spb netmonitor[5776]: message repeated 12 times: [
     enp2s0:#012#011Receive 1535932738  bytes (1111394 packets)
     #012#011Transmit 202178497  bytes (700395 packets)]
8  Apr 23 21:28:29 spb netmonitor[5776]: enp2s0:#012#011Receive
     1535932738  bytes (1111394 packets)#012#011Transmit 202178563
     bytes (700396 packets)
9  Apr 23 21:28:29 spb netmonitor[5776]: message repeated 12 times: [
     enp2s0:#012#011Receive 1535932738  bytes (1111394 packets)
     #012#011Transmit 202178563  bytes (700396 packets)]
10 Apr 23 21:28:29 spb netmonitor[5776]: enp2s0:#012#011Receive
     1535933590  bytes (1111395 packets)#012#011Transmit 202178629
     bytes (700397 packets)
```

2 Дистрибуция пакетов в Linux

Программное обеспечение в ОС Ubuntu Linux распространяется в так называемых deb-пакетах. Обычно при установке программы из репозитория система автоматически скачивает 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 дублирования 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 поставится 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, если 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.

2.1 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 выполняется 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 следующим 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.

2.2 Создание PKGBUILD

Пользовательский репозиторий Arch Linux (Arch User Repository, AUR) – это поддерживаемое 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 использует 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, разделенных 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.

3 Фоновые приложения в Windows

3.1 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», который 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.

3.2 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 прикладного 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 структуры 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 службами и изменить любой параметр, используя командную строку. Данное средство предоставляет 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 s
```

По умолчанию создается служба типа WIN32_SHARE_PROCESS с типом запуска SERVICE_DEMA
Она не имеет никаких зависимостей и выполняется в контексте безопасности LocalSystem.

Результат добавления приложения в список сервисов показан на рисунке 1.

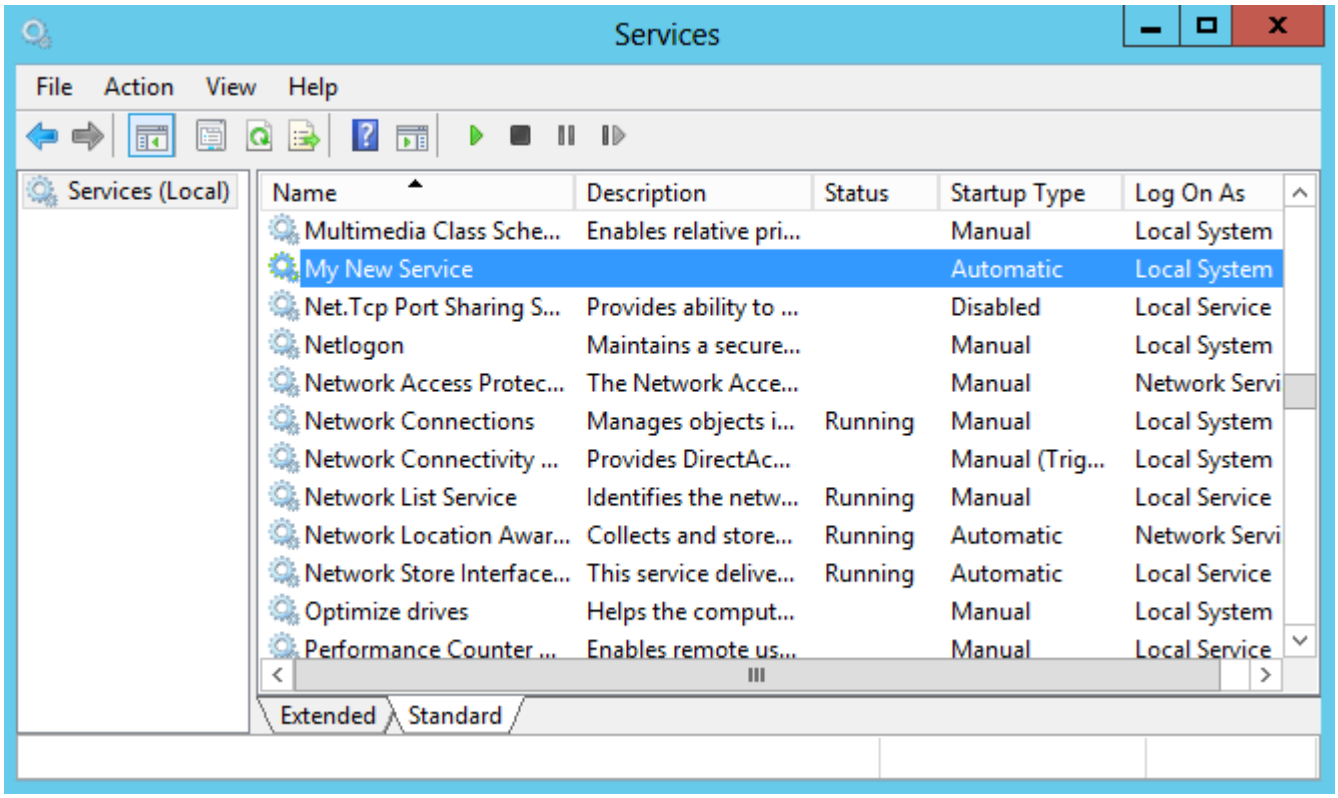

Рис. 1: Добавление сервиса из приложения в windows

3.3 Создание службы Windows с помощью PowerShell

Возможности управления системой из консоли в последних версиях Windows были значительно 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.

3.4 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)

```
;#ifndef _EVENT_LOG_MESSAGE_FILE_H_
;#define _EVENT_LOG_MESSAGE_FILE_H_

MessageIdTypeDef=DWORD


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

LanguageNames=(EnglishUS=0x401:MSG00401
               Neutral=0x0000:MSG00000
               )

MessageId=0x0    SymbolicName=MSG_INFO_1
Severity=Informational
Facility=Application
Language=Neutral
%1
.

MessageId=0x1    SymbolicName=MSG_WARNING_1
Severity=Warning
Facility=Application
Language=Neutral
%1
.

MessageId=0x2    SymbolicName=MSG_ERROR_1
Severity=Error
```

```
Facility=Application
Language=Neutral
%1
.

MessageId=0x3     SymbolicName=MSG_SUCCESS_1
Severity=Success
Facility=Application
Language=Neutral
%1
.


;#endif
```

Следующие две команды генерируют ресурсный файл и хедер для общения с этим ресурсным 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)

```
#include <iostream>
#include <windows.h>
#include <iphlpapi.h>

#include "eventlog.h"

bool GetIfTable(PMIB_IFTABLE* m_pTable);

void install_event_log_source(const std::string& a_name) {
  const std::string key_path(
      "SYSTEM\\CurrentControlSet\\Services\\"
      "EventLog\\Application\\" +
```

```
13        a_name);
14
15    HKEY key;
16
17    DWORD last_error =
18        RegCreateKeyEx(HKEY_LOCAL_MACHINE, key_path.c_str(), 0, 0,
19                       REG_OPTION_NON_VOLATILE, KEY_SET_VALUE, 0, &key
                           , 0);
20
21    if (ERROR_SUCCESS == last_error) {
22
23      UINT max_path = 512 * sizeof(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\\application.exe";
29      DWORD last_error;
30      const DWORD types_supported =
31          EVENTLOG_ERROR_TYPE | EVENTLOG_WARNING_TYPE |
              EVENTLOG_INFORMATION_TYPE;
32
33      last_error = RegSetValueEx(key, "EventMessageFile", 0, REG_SZ,
            exe_path,
34                                 sizeof(exe_path));
35
36      if (ERROR_SUCCESS == last_error) {
37        last_error =
38            RegSetValueEx(key, "TypesSupported", 0, REG_DWORD,
39                          (LPBYTE)&types_supported, sizeof(
                              types_supported));
40      }
41
42      if (ERROR_SUCCESS != last_error) {
43        std::cerr << "Failed to install source values: " << last_error
              << "\n";
44      }
```

```
45
46      RegCloseKey(key);
47    } else {
48      std::cerr << "Failed to install source: " << last_error << "\n";
49    }
50  }
51
52  void log_event_log_message(const std::string& a_msg, const WORD
        a_type,
53                             const std::string& 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      default:
67        std::cerr << "Unrecognised type: " << a_type << "\n";
68        event_id = MSG_INFO_1;
69        break;
70    }
71
72    HANDLE h_event_log = RegisterEventSource(0, a_name.c_str());
73
74    if (0 == h_event_log) {
75      std::cerr << "Failed open source '" << a_name << "': " <<
            GetLastError()
76                << "\n";
77    } else {
78      LPCTSTR message = a_msg.c_str();
79
```

```
80      if (FALSE ==
81          ReportEvent(h_event_log, a_type, 0, event_id, 0, 1, 0, &
                message, 0)) {
82        std::cerr << "Failed to write message: " << GetLastError() <<
              "\n";
83      }
84
85      DeregisterEventSource(h_event_log);
86    }
87  }
88
89  void uninstall_event_log_source(const std::string& a_name) {
90    const std::string key_path(
91        "SYSTEM\\CurrentControlSet\\Services\\"
92        "EventLog\\Application\\" +
93        a_name);
94
95    DWORD last_error = RegDeleteKey(HKEY_LOCAL_MACHINE, key_path.c_str
          ());
96
97    if (ERROR_SUCCESS != last_error) {
98      std::cerr << "Failed to uninstall source: " << last_error << "\n
            ";
99    }
100 }
101
102 int main(int argc, char* argv[]) {
103   PMIB_IFTABLE m_pTable = NULL;
104   const std::string event_log_source_name("netmonitor");
105   install_event_log_source(event_log_source_name);
106   log_event_log_message("Netmonitor start", EVENTLOG_WARNING_TYPE,
107                         event_log_source_name);
108
109   if (GetIfTable(&m_pTable) == false) {
110     return 1;
111   }
112
```

```
113   // Обход списка сетевых интерфейсов
114   std::string buff;
115   for (UINT i = 0; i < m_pTable->dwNumEntries; i++) {
116     MIB_IFROW Row = m_pTable->table[i];
117     char szDescr[MAXLEN_IFDESCR];
118     memcpy(szDescr, Row.bDescr, Row.dwDescrLen);
119     szDescr[Row.dwDescrLen] = 0;
120
121     // Вывод собранной информации
122
123     buff.append(szDescr);
124     buff.append(":\n");
125     buff.append("\tReceived: ");
126     buff.append(Row.dwInOctets);
127     buff.append(", Sent: ");
128     buff.append(Row.dwOutOctets);
129     buff.append("\n\n");
130     log_event_log_message(buff.c_str(), EVENTLOG_INFORMATION_TYPE,
131                           event_log_source_name);
132     buff.clear();
133   }
134
135   // Завершение работы
136   log_event_log_message("Netmonitor end", EVENTLOG_WARNING_TYPE,
137                         event_log_source_name);
138   delete (m_pTable);
139   char a = getchar();
140   return 0;
141 }
142
143 bool GetIfTable(PMIB_IFTABLE* m_pTable) {
144   // Тип указателя на функцию GetIfTable
145   typedef DWORD(_stdcall * TGetIfTable)(
146       MIB_IFTABLE * pIfTable,   // Буфер таблицы интерфейсов
147       ULONG * pdwSize,          // Размер буфера
148       BOOL bOrder);             // Сортировать таблицу?
149
```

```
150   // Пытаемся подгрузить iphlpapi.dll
151   HINSTANCE iphlpapi;
152   iphlpapi = LoadLibrary(L"iphlpapi.dll");
153   if (!iphlpapi) {
154     log_event_log_message("iphlpapi.dll not supported",
          EVENTLOG_ERROR_TYPE,
155                           event_log_source_name);
156     return false;
157   }
158
159   // Получаем адрес функции
160   TGetIfTable pGetIfTable;
161   pGetIfTable = (TGetIfTable)GetProcAddress(iphlpapi, "GetIfTable");
162
163   // Получили требуемый размер буфера
164   DWORD m_dwAdapters = 0;
165   pGetIfTable(*m_pTable, &m_dwAdapters, TRUE);
166
167   *m_pTable = new MIB_IFTABLE[m_dwAdapters];
168   if (pGetIfTable(*m_pTable, &m_dwAdapters, TRUE) != ERROR_SUCCESS)
        {
169     log_event_log_message("Error while GetIfTable",
          EVENTLOG_ERROR_TYPE,
170                           event_log_source_name);
171     delete *m_pTable;
172     return false;
173   }
174
175   return true;
176 }
```

4 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 разрабатываемой программы netmonitor представлен в листинге 5.

Листинг 5: Скрипт генерации установочного файла

```
; Script generated by the Inno Setup Script Wizard.
; SEE THE DOCUMENTATION FOR DETAILS ON CREATING INNO SETUP SCRIPT FILES!

#define MyAppName "NetMonitor"
#define MyAppVersion "1.0"
#define MyAppPublisher "Semen Martynov"
#define MyAppURL "https://cloudtips.org"
#define MyAppExeName "netmonitor.exe"

[Setup]
; NOTE: The value of AppId uniquely identifies this application.
; Do not use the same AppId value in installers for other applications.
; (To generate a new GUID, click Tools | Generate GUID inside the IDE.)
AppId={{55125EB5-4278-440E-82D9-638FD219F5F5}
AppName={#MyAppName}
AppVersion={#MyAppVersion}
;AppVerName={#MyAppName} {#MyAppVersion}
AppPublisher={#MyAppPublisher}
AppPublisherURL={#MyAppURL}
AppSupportURL={#MyAppURL}
AppUpdatesURL={#MyAppURL}
DefaultDirName={pf}\{#MyAppName}
```

```
23 DisableProgramGroupPage=yes
24 OutputBaseFilename=setup
25 Compression=lzma
26 SolidCompression=yes
27
28 [Languages]
29 Name: "english"; MessagesFile: "compiler:Default.isl"
30
31 [Tasks]
32 Name: "desktopicon"; Description: "{cm:CreateDesktopIcon}";
      GroupDescription: "{cm:AdditionalIcons}"; Flags: unchecked
33
34 [Files]
35 Source: "C:\Program Files (x86)\NetMonitor\netmonitor.exe"; DestDir:
       "{app}"; Flags: ignoreversion
36 Source: "C:\Users\user\Documents\Visual Studio 2015\Projects\
      netmonitor\x64\Release\netmonitor.exe"; DestDir: "{app}"; Flags:
      ignoreversion
37 ; NOTE: Don't use "Flags: ignoreversion" on any shared system files
38
39 [Icons]
40 Name: "{commonprograms}\{#MyAppName}"; Filename: "{app}\{#
      MyAppExeName}"
41 Name: "{commondesktop}\{#MyAppName}"; Filename: "{app}\{#
      MyAppExeName}"; Tasks: desktopicon
42
43 [Run]
44 Filename: "{app}\{#MyAppExeName}"; Description: "{cm:LaunchProgram
      ,{#StringChange(MyAppName, '&', '&&')}}"; Flags: nowait
      postinstall skipifsilent
```

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).

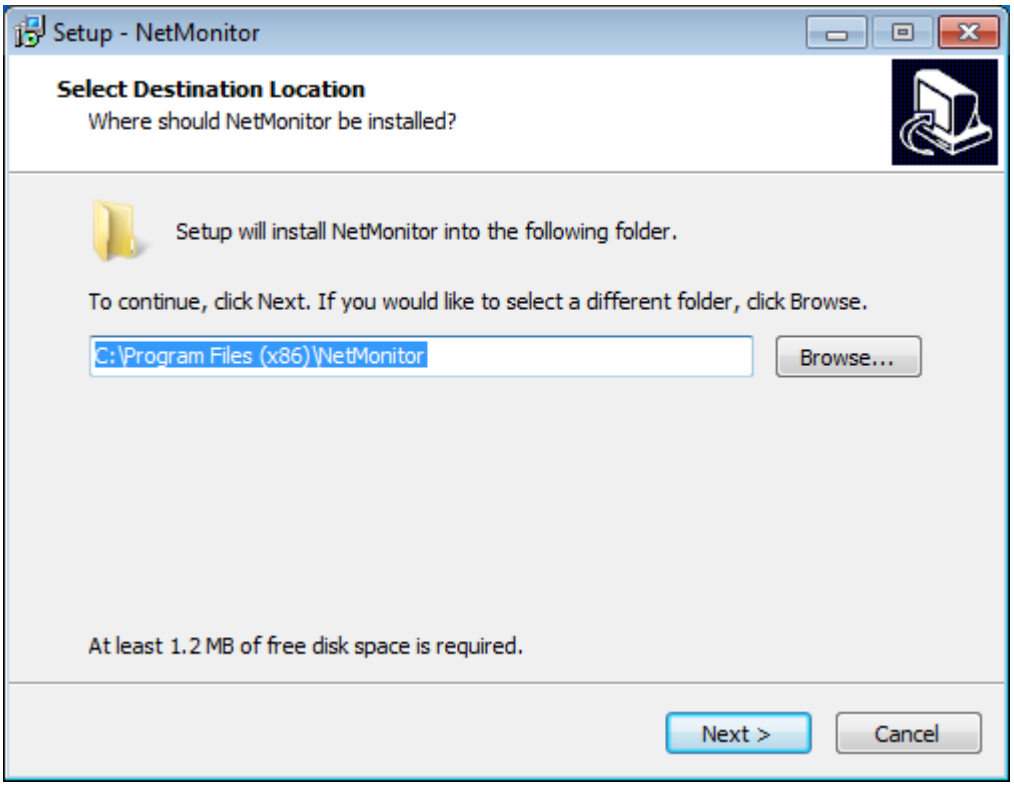

Рис. 2: Интерфейс установки приложения netmonitor

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 благодаря 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 помощью программы Sc.exe. https://support.microsoft.com/ru-ru/kb/251192 [Дата обращения 25 апреля 2016]

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