Санкт-Петербургский Государственный Политехнический Университет

Институт информационных технологий и управления

Кафедра Компьютерных Систем и Программных Технологий

Отчет по лабораторной работе № 1

По дисциплине «Проектирование ОС и компонентов»

По теме «Создание приложения в виде демона, службы»

**Работу выполнила студентка группы № 13541/3**

Шаляпин Н.С. \_\_\_\_\_\_\_\_\_\_\_

**Работу принял преподаватель:**

Душутина Е.В. \_\_\_\_\_\_\_\_\_\_\_

.

Санкт-Петербург

2017

## Программа работы

Разработка компонентов прикладного уровня с заданной функциональностью,

реализация в виде: приложения, утилиты, демона, сервиса, службы.

Используются в качестве приложений: утилита, выводящая информацию о файле для Windows и утилита для Linux. Утилиты были разработаны в прошлом семестре.

Лабораторная работа выполнялась в ОС: Windows 7 и Ubuntu 14.04.

Windows

|  |
| --- |
| Имя узла: ADMIN-ПК  Название ОС: Microsoft Windows 7 Максимальная  Версия ОС: 6.1.7601 Service Pack 1 сборка 7601  Изготовитель ОС: Microsoft Corporation  Параметры ОС: Изолированная рабочая станция  Сборка ОС: Multiprocessor Free  Зарегистрированный владелец: Admin  Зарегистрированная организация:  Код продукта: 00426-OEM-8992662-00173  Дата установки: 02.04.2017, 14:53:54  Время загрузки системы: 27.05.2017, 21:26:27  Изготовитель системы: ASUSTeK Computer Inc.  Модель системы: K53SM  Тип системы: x64-based PC  Процессор(ы): Число процессоров - 1.  [01]: Intel64 Family 6 Model 42 Stepping 7 GenuineIntel ~2501 МГц  Версия BIOS: American Megatrends Inc. K53SM.214, 15.08.2012 |

Linux

|  |
| --- |
| nikita@nikita-K53SM:~$ uname -a  Linux nikita-K53SM 3.19.0-80-generic #88~14.04.1-Ubuntu SMP Fri Jan 13 14:54:07 UTC 2017 x86\_64 x86\_64 x86\_64 GNU/Linux |

1. **Создание службы в Windows**

Windows Service— приложения, автоматически запускаемые системой при запуске Windows и выполняющиеся вне зависимости от статуса пользователя. 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).

Реализуем собственный сервис:

|  |
| --- |
| #include "stdafx.h"  #include <windows.h>  #include <stdio.h>  #include "LibFullInfoFile.h"  #include <iostream>  #include <locale>  #include <winsvc.h>  #define \_CRT\_SECURE\_NO\_WARNINGS  #pragma warning(disable : 4996)  using namespace std;  typedef struct ARGS {  DWORD argc;  LPTSTR \*argv;  };  SERVICE\_STATUS\_HANDLE g\_StatusHandle = NULL;  SERVICE\_STATUS g\_ServiceStatus = { 0 };  HANDLE g\_ServiceStopEvent = INVALID\_HANDLE\_VALUE;  void WINAPI ServiceMain(DWORD argc, LPTSTR \*argv);  void WINAPI ServiceCtrlHandler(DWORD);  DWORD WINAPI ServiceWorkerThread(LPVOID lpParam);  #define SERVICE\_NAME \_T("My Chmod Service")  int \_tmain(int argc, TCHAR \*argv[])  {  SERVICE\_TABLE\_ENTRY ServiceTable[] =  {  { SERVICE\_NAME, (LPSERVICE\_MAIN\_FUNCTION)ServiceMain },  { NULL, NULL }  };  if (StartServiceCtrlDispatcher(ServiceTable) == FALSE)  {  return GetLastError();  }  return 0;  }  DWORD WINAPI ServiceWorkerThread(LPVOID lpParam)  {  ARGS \*pArgs = (ARGS\*)lpParam;  \_TCHAR\* fileName = L"D:\\getFileInfo.exe";  int i;  i = FullInfoFile::MyFullInfoFile::infoFile(fileName);  Sleep(3000);  return ERROR\_SUCCESS;  }  void WINAPI ServiceMain(DWORD argc, LPTSTR \*argv)  {  DWORD Status = E\_FAIL;  // Register our service control handler with the SCM  g\_StatusHandle = RegisterServiceCtrlHandler(SERVICE\_NAME, ServiceCtrlHandler);  if (g\_StatusHandle == NULL)  {  return;  }  // Tell the service controller we are starting  ZeroMemory(&g\_ServiceStatus, sizeof(g\_ServiceStatus));  g\_ServiceStatus.dwServiceType = SERVICE\_WIN32\_OWN\_PROCESS;  g\_ServiceStatus.dwControlsAccepted = 0;  g\_ServiceStatus.dwCurrentState = SERVICE\_START\_PENDING;  g\_ServiceStatus.dwWin32ExitCode = 0;  g\_ServiceStatus.dwServiceSpecificExitCode = 0;  g\_ServiceStatus.dwCheckPoint = 0;  if (SetServiceStatus(g\_StatusHandle, &g\_ServiceStatus) == FALSE)  {  OutputDebugString(\_T(  "My FullInfoFile Service: ServiceMain: SetServiceStatus returned error"));  }  // Perform tasks necessary to start the service here  // Create a service stop event to wait on later  g\_ServiceStopEvent = CreateEvent(NULL, TRUE, FALSE, NULL);  if (g\_ServiceStopEvent == NULL)  {  // Error creating event  // Tell service controller we are stopped and exit  g\_ServiceStatus.dwControlsAccepted = 0;  g\_ServiceStatus.dwCurrentState = SERVICE\_STOPPED;  g\_ServiceStatus.dwWin32ExitCode = GetLastError();  g\_ServiceStatus.dwCheckPoint = 1;  if (SetServiceStatus(g\_StatusHandle, &g\_ServiceStatus) == FALSE)  {  OutputDebugString(\_T(  "My FullInfoFile Service: ServiceMain: SetServiceStatus returned error"));  }  return;  }  // Tell the service controller we are started  g\_ServiceStatus.dwControlsAccepted = SERVICE\_ACCEPT\_STOP;  g\_ServiceStatus.dwCurrentState = SERVICE\_RUNNING;  g\_ServiceStatus.dwWin32ExitCode = 0;  g\_ServiceStatus.dwCheckPoint = 0;  if (SetServiceStatus(g\_StatusHandle, &g\_ServiceStatus) == FALSE)  {  OutputDebugString(\_T(  "My Chmod Service: ServiceMain: SetServiceStatus returned error"));  }  // Start a thread that will perform the main task of the service  ARGS args = {argc, argv};  HANDLE hThread = CreateThread(NULL, 0, ServiceWorkerThread, &args, 0, NULL);  // Wait until our worker thread exits signaling that the service needs to stop  WaitForSingleObject(hThread, INFINITE);  // Perform any cleanup tasks  CloseHandle(g\_ServiceStopEvent);  // Tell the service controller we are stopped  g\_ServiceStatus.dwControlsAccepted = 0;  g\_ServiceStatus.dwCurrentState = SERVICE\_STOPPED;  g\_ServiceStatus.dwWin32ExitCode = 0;  g\_ServiceStatus.dwCheckPoint = 3;  if (SetServiceStatus(g\_StatusHandle, &g\_ServiceStatus) == FALSE)  {  OutputDebugString(\_T(  "My FullInfoFile Service: ServiceMain: SetServiceStatus returned error"));  }  return;  }  void WINAPI ServiceCtrlHandler(DWORD CtrlCode)  {  switch (CtrlCode)  {  case SERVICE\_CONTROL\_STOP:  if (g\_ServiceStatus.dwCurrentState != SERVICE\_RUNNING)  break;  //Perform tasks necessary to stop the service here  g\_ServiceStatus.dwControlsAccepted = 0;  g\_ServiceStatus.dwCurrentState = SERVICE\_STOP\_PENDING;  g\_ServiceStatus.dwWin32ExitCode = 0;  g\_ServiceStatus.dwCheckPoint = 4;  if (SetServiceStatus(g\_StatusHandle, &g\_ServiceStatus) == FALSE)  {  OutputDebugString(\_T(  "My FullInfoFile Service: ServiceCtrlHandler: SetServiceStatus returned error"));  }  // This will signal the worker thread to start shutting down  SetEvent(g\_ServiceStopEvent);  break;  default:  break;  }  } |

Используются две глобальные переменные:

**SERVICE\_STATUS ServiceStatus;**   
**SERVICE\_STATUS\_HANDLE hStatus;**

Структура **SERVICE\_STATUS** используется для оповещения SCM текущего статуса сервиса.

**SERVICE\_TABLE\_ENTRY** это структура, которая описывает точку входа для сервис-менеджера, в данном случае вход будет происходить через функцию **ServiceMain**. Функция **StartServiceCtrlDispatcher** связывает сервис с **SCM** (Service Control Manager).

Логика функции ServiceMain. Сначала регистрируем функцию, которая будет обрабатывать управляющие запросы от SCM, например, запрос на остановку. Регистрация производиться при помощи функции **RegisterServiceCtrlHandler**. Для изменения статуса сервиса используется функция **SetServiceStatus**. Функция по обработке запросов **ServiceCtrlHandler** вызывается каждый раз, как SCM шлет запросы на изменения состояния сервиса. В основном ее используют для описания корректной завершении работа сервиса. Так как сервисы не имеют доступ к терминалу, выходная информация будет записываться в файл.

После того, как сервис написан, необходимо его установить. Далее представлены команды установки, запуска и удаления службы:

|  |
| --- |
| C:\Windows\system32>sc create "My FullInfoFile Service" binPath= "C:\Users\Admin \Documents\Visual Studio 2013\Projects\Service\Debug\Service.exe"  [SC] CreateService: успех  C:\Windows\system32>sc start "My FullInfoFile Service" "D:\"  Имя\_службы: My FullInfoFile Service  Тип : 10 WIN32\_OWN\_PROCESS  Состояние : 4 RUNNING  (STOPPABLE, NOT\_PAUSABLE, IGNORES\_SHUTDOWN)  Код\_выхода\_Win32 : 0 (0x0)  Код\_выхода\_службы : 0 (0x0)  Контрольная\_точка : 0x0  Ожидание : 0x0  ID\_процесса : 936  Флаги :  C:\Windows\system32>sc delete "My FullInfoFile Service"  [SC] DeleteService: успех |

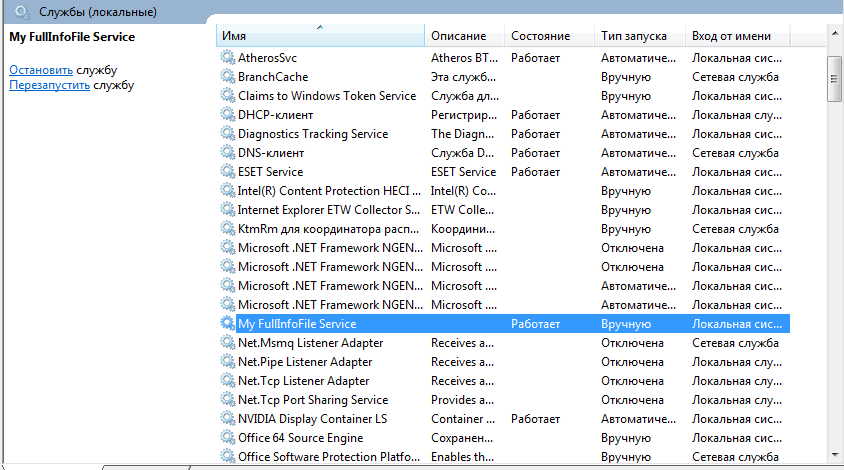


Рис. 1. Службы

После установки сервиса в Службах появляется новый сервис My FullInfoFile Service. После запуска службы в файл results.txt были добавлены строки, служба работает корректно:

|  |
| --- |
| File Index : 295211048576  Number Of Links : 1  File Type: FILE\_TYPE\_DISK  The final path is: \\?\D:\getFileInfo.exe  The full path name is: D:\getFileInfo.exe  File attributes:  The attribute FILE\_ATTRIBUTE\_ARCHIVE is establis  File size= 77404656 bytes  File creation time: 02/06/2017 18:23  File last access time: 02/06/2017 18:23  File last write time: 09/14/2016 18:04  FILE\_ENCRYPTABLE  Section: .text  =======================  Virtual size: 0xf244  Raw size: 0xf400  Virtual address: 0x1000  Raw address: 0x400  Characteristics: R X  Section: .itext  =======================  Virtual size: 0xf64  Raw size: 0x1000  Virtual address: 0x11000  Raw address: 0xf800  Characteristics: R X  Section: .data  =======================  Virtual size: 0xc88  Raw size: 0xe00  Virtual address: 0x12000  Raw address: 0x10800  Characteristics: R W  Section: .bss  =======================  Virtual size: 0x56bc  Raw size: 0  Virtual address: 0x13000  Raw address: 0x11600  Characteristics: R W  Section: .idata  =======================  Virtual size: 0xe04  Raw size: 0x1000  Virtual address: 0x19000  Raw address: 0x11600  Characteristics: R W  Section: .tls  =======================  Virtual size: 0x8  Raw size: 0  Virtual address: 0x1a000  Raw address: 0x12600  Characteristics: R W  Section: .rdata  =======================  Virtual size: 0x18  Raw size: 0x200  Virtual address: 0x1b000  Raw address: 0x12600  Characteristics: R  Section: .rsrc  =======================  Virtual size: 0xb200  Raw size: 0xb200  Virtual address: 0x1c000  Raw address: 0x12800  Characteristics: R  SCS\_32BIT\_BINARY  Для продолжения нажмите любую клавишу . . . |

Также просмотреть все сервисы можно из Visual Studio, открыв Server Explorer (View -> Server Explorer).

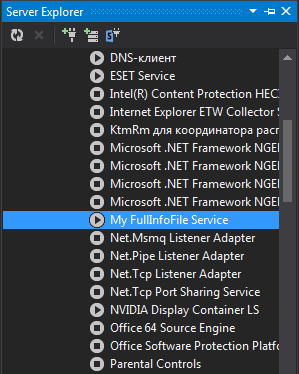


Рис. 2. My Chmod Service в Server Explorer

1. **Создание демона в Linux**

Демон — компьютерная программа в системах класса UNIX, запускаемая самой системой и работающая в фоновом режиме без прямого взаимодействия с пользователем. Исходный код демона (main.c), сохраняющего информацию о файле, который передан в качестве аргумента при запуске:

|  |
| --- |
| #include <unistd.h>  #include <stdio.h>  #include <sys/stat.h>  #include <sys/types.h>  #include <time.h>  #include "functions.h"  #include <stdlib.h>  #include <string.h>  #include <errno.h>  #include <fcntl.h>  #include <syslog.h>  #include <sys/vfs.h>  char\* getTime() { //функция получение текущего времени  time\_t now;  struct tm \*ptr;  static char tbuf[64];  bzero(tbuf,64);  time(&now);  ptr = localtime(&now);  strftime(tbuf,64, "%Y-%m-%d %H:%M:%S", ptr);  return tbuf;  }  int writeLog(char msg[1024]) { //функция логирования  FILE \* pLog;  pLog = fopen("/home/nikita/Документы/OS/Lab2demon/daemon/daemon.log", "a");  if(pLog == NULL) {  return 1;  }  char str[1110];  bzero(str, 1110);  strcpy(str, getTime());  strcat(str, "\n==========================\n");  strcat(str, msg);  strcat(str, "\n==========================\n");  fputs(str, pLog);  fclose(pLog);  return 0;  }  int Daemon(int argc, char\* argv[]) { //бесконечный цикл демона  char \*log;  struct stat fStat;  while(1) {  if(argc != 2 || stat(argv[1],&fStat) < 0)  {  writeLog("Cycle terminated. Invalid arguments!");  return 1;  }  log = printInfo(fStat);  writeLog(log);  sleep(60);  }  return 0;  }  int main(int argc, char\* argv[]) {  writeLog("Daemon Start");  pid\_t parpid, sid;  parpid = fork(); // создание дочернего процесса  if(parpid < 0) {  exit(1);  } else if(parpid != 0) {  exit(0);  }  sid = setsid(); //создание нового сеанса, не зависящего от родителя  if(sid < 0) {  exit(1);  }  if((chdir("/")) < 0) { //переход к корню диска  exit(1);  }  //закрытие дескрипторов ввода/вывода/ошибок  close(STDIN\_FILENO);  close(STDOUT\_FILENO);  close(STDERR\_FILENO);  return Daemon(argc, argv);  } |

Файл functions.c:

|  |
| --- |
| #include "functions.h"  /\* Return a character indicating the type of file described by  file mode BITS:  '-' regular file  'b' block special file  'c' character special file  'C' high performance ("contiguous data") file  'd' directory  'D' door  'l' symbolic link  'm' multiplexed file (7th edition Unix; obsolete)  'n' network special file (HP-UX)  'p' fifo (named pipe)  'P' port  's' socket  'w' whiteout (4.4BSD)  '?' some other file type \*/  static char  ftypelet (mode\_t bits)  {  /\* These are the most common, so test for them first. \*/  if (S\_ISREG (bits))  return '-';  if (S\_ISDIR (bits))  return 'd';  /\* Other letters standardized by POSIX 1003.1-2004. \*/  if (S\_ISBLK (bits))  return 'b';  if (S\_ISCHR (bits))  return 'c';  if (S\_ISLNK (bits))  return 'l';  if (S\_ISFIFO (bits))  return 'p';  /\* Other file types (though not letters) standardized by POSIX. \*/  if (S\_ISSOCK (bits))  return 's';  /\* Nonstandard file types. \*/  /\*if (S\_ISCTG (bits))  return 'C';  if (S\_ISDOOR (bits))  return 'D';  if (S\_ISMPB (bits) || S\_ISMPC (bits) || S\_ISMPX (bits))  return 'm';  if (S\_ISNWK (bits))  return 'n';  if (S\_ISPORT (bits))  return 'P';  if (S\_ISWHT (bits))  return 'w';\*/  return '?';  }  /\* Like filemodestring, but rely only on MODE. \*/  void  strmode (mode\_t mode, char \*str)  {  str[0] = ftypelet (mode);  str[1] = mode & S\_IRUSR ? 'r' : '-';  str[2] = mode & S\_IWUSR ? 'w' : '-';  str[3] = (mode & S\_ISUID  ? (mode & S\_IXUSR ? 's' : 'S')  : (mode & S\_IXUSR ? 'x' : '-'));  str[4] = mode & S\_IRGRP ? 'r' : '-';  str[5] = mode & S\_IWGRP ? 'w' : '-';  str[6] = (mode & S\_ISGID  ? (mode & S\_IXGRP ? 's' : 'S')  : (mode & S\_IXGRP ? 'x' : '-'));  str[7] = mode & S\_IROTH ? 'r' : '-';  str[8] = mode & S\_IWOTH ? 'w' : '-';  str[9] = (mode & S\_ISVTX  ? (mode & S\_IXOTH ? 't' : 'T')  : (mode & S\_IXOTH ? 'x' : '-'));  str[10] = ' ';  str[11] = '\0';  }  char\* printFileType(struct stat fStat)  {  static char str[24];  bzero(str, 24);  switch (fStat.st\_mode & S\_IFMT) {  case S\_IFBLK:  sprintf(str, "блочное устройство\n");  break;  case S\_IFCHR:  sprintf(str, "символьное устройство\n");  break;  case S\_IFDIR:  sprintf(str, "директория\n");  break;  case S\_IFIFO:  sprintf(str, "FIFO/pipe\n");  break;  case S\_IFLNK:  sprintf(str, "символьная ссылка\n");  break;  case S\_IFREG:  sprintf(str, "обычный файл\n");  break;  case S\_IFSOCK:  sprintf(str, "socket\n");  break;  default:  sprintf(str, "?\n");  break;  }  return str;  }  char\* printInfo(struct stat fStat)  {  static char str[1024];  bzero(str, 1024);  sprintf(str + strlen(str), "Размер файла: \n\t%d байт\n",fStat.st\_size);  sprintf(str + strlen(str), "Число жёстких ссылок: \n\t%d\n",fStat.st\_nlink);  sprintf(str + strlen(str), "inode файла: \n\t%d\n",fStat.st\_ino);  sprintf(str + strlen(str), "ID устройства, содержащего файл: \n\t%d\n",fStat.st\_dev);  sprintf(str + strlen(str), "ID пользователя владельца: \n\t%d\n",fStat.st\_uid);  sprintf(str + strlen(str), "ID группы владельца: \n\t%d\n",fStat.st\_gid);  sprintf(str + strlen(str), "ID устройства (если специальный файл): \t%d\n",fStat.st\_rdev);  sprintf(str + strlen(str), "Размер блока для I/O: \n\t%d\n",fStat.st\_blksize);  sprintf(str + strlen(str), "Число выделенных блоков размером 512 байт:\n\t%d\n",fStat.st\_blocks);  char perms[12];  strmode (fStat.st\_mode, perms);  sprintf(str + strlen(str), "Права доступа к файлу:\n\t%s\n", perms);  sprintf(str + strlen(str), "Тип файла:\n\t");  sprintf(str + strlen(str), "%s", printFileType(fStat));  struct tm \* timeinfo;  sprintf(str + strlen(str), "Последнее изменение статуса:\n\t%s", ctime(&fStat.st\_ctime));  sprintf(str + strlen(str), "Последний доступ к файлу:\n\t%s", ctime(&fStat.st\_atime));  sprintf(str + strlen(str), "Последния модификация файла:\n\t%s", ctime(&fStat.st\_mtime));  return str;  } |

Последовательность работы демона:

**функция main**

* 1. создание дочернего процесса;
  2. создание для него нового сеанса, отключение от терминала;
  3. переход к корню диска;
  4. закрытие дескрипторов ввода/вывода/ошибок;
  5. вызов функции Daemon, состоящей из бесконечного цикла демона, цикл повторяется каждую минуту;

**в цикле:**

* + 1. вызов функции char\* printInfo(struct stat fStat) содержащей основную функциональность работы демона, возвращает строку результата;
    2. int writeLog(char msg[1024]) – запись строки результата в файл логирования "/home/nikita/labs/2 semester/lab1/daemon/daemon.log" с указание времени;
    3. char\* getTime() - получение текущего времени.

В результате в файле логирования "/home/user/workspace/sem2/demon/daemon.log" отображена следующая информация, каждую минуту в файл будут добавляться новые строки.

|  |
| --- |
| 2017-05-03 00:09:56  ==========================  Daemon Start  ==========================  2017-05-03 00:09:56  ==========================  Размер файла:  797 байт  Число жёстких ссылок:  1  inode файла:  793075  ID устройства, содержащего файл:  2055  ID пользователя владельца:  1000  ID группы владельца:  1000  ID устройства (если специальный файл): 0  Размер блока для I/O:  4096  Число выделенных блоков размером 512 байт:  8  Права доступа к файлу:  -rw-rw-r--  Тип файла:  обычный файл  Последнее изменение статуса:  Tue May 2 23:35:15 2017  Последний доступ к файлу:  Tue May 2 23:35:15 2017  Последния модификация файла:  Tue May 2 23:35:15 2017  ==========================  2017-05-03 00:10:56  ==========================  Размер файла:  797 байт  Число жёстких ссылок:  1  inode файла:  793075  ID устройства, содержащего файл:  2055  ID пользователя владельца:  1000  ID группы владельца:  1000  ID устройства (если специальный файл): 0  Размер блока для I/O:  4096  Число выделенных блоков размером 512 байт:  8  Права доступа к файлу:  -rw-rw-r--  Тип файла:  обычный файл  Последнее изменение статуса:  Tue May 2 23:35:15 2017  Последний доступ к файлу:  Tue May 2 23:35:15 2017  Последния модификация файла:  Tue May 2 23:35:15 2017  ========================== |

Для того чтобы остановить работу демон, используем команду kill, указав id демона. После этого, работа демона завершена, файл логирования больше не обновляется.

|  |
| --- |
| nikita@nikita-K53SM:~/labs/2 semester/lab1/daemon$ ps -aux | grep infofile  nikita 10484 0.0 0.0 4360 92 ? Ss 00:09 0:00 ./infofile /home/nikita/labs/2 semester/lab1/daemon/temp.txt  nikita 10513 0.0 0.0 21316 1028 pts/2 S+ 00:10 0:00 grep --color=auto infofile  nikita@nikita-K53SM:~/labs/2 semester/lab1/daemon$ kill 10484  nikita@nikita-K53SM:~/labs/2 semester/lab1/daemon$ ps -aux | grep infofile  nikita 10738 0.0 0.0 21316 924 pts/2 S+ 00:11 0:00 grep --color=auto infofile |

**Вывод**

Были изучены основы создания служб в Windows и демонов в Linux. Служба и демон были созданы на основе утилит, разработанных в прошлом семестре. Служба была создана на основе утилиты chmod для Windows. Был создан демон на основе утилиты для Linux, выводящей информацию о файле, демон сохраняет информацию в файл "/home/nikita/labs/2 semester/lab1/daemon/daemon.log".

**Список использованных источников**

1. <https://habrahabr.ru/post/71533/>
2. <https://msdn.microsoft.com/ru-ru/library/zt39148a(v=vs.110).aspx>