8. Управление файлами в ОС GNU/Linux

Разделы:

- Файлы и файловые системы
- Файловые операции

Основные цели использования файлов

- Долговременное и надежное хранение информации (достигается за счет использования ЗУ, не зависящих от питания, определяется средствами защиты доступа к файлам и общей организацией программного кода ОС)
- Совместное использование информации (Файлы обеспечивают естественный и легкий способ разделения информации между программами и их пользователями за счет наличия понятного символьного имени и постоянства хранимой информации и расположения файла)

Компоненты файловой системы

- Совокупность всех файлов во внешней памяти
- Наборы структур данных для управления файлами (каталоги, дескрипторы, таблицы распределения свободного и занятого пространства на диске)
- Комплекс программных средств, реализующих различные операции над файлами

Основные функции ФС

- именование файлов;
- программный интерфейс для прикладных программ;
- отображение логической модели ФС на физическую организацию хранилища данных;
- устойчивость ФС к сбоям электропитания, ошибкам программных и аппаратных средств;
- организацию совместного доступа к файлу нескольких процессов;
- защиту файлов одного пользователя от несанкционированного доступа другого и т.д.

Типы файлов

- Обычные (регулярные)
- Каталоги
- Специальные файлы устройст
- Конвейеры
- Сокеты
- Символьные связи и ссылки

Типы имен файлов

- В иерархически организованных ФС обычно используются три типа имен файлов: простые, составные и относительные
- Простое (короткое) символьное имя идентифицирует файл в пределах одного каталога
- Полное имя представляет собой цепочку простых имен всех каталогов, через которые проходит путь от корня до данного файла
- Полное имя является составным
- Относительное имя файла определяется через понятие «*текущий каталог*»

Примеры имен

- Короткое имя:

```
file1
```

- Полное имя:

```
1/2/3/file1
```

- Относительное имя:

```
./file1
../directory/file2
```

Файловые операции

- Для выполнения некоторых файловых операций есть несколько системных вызовов, выглядящих как обычные функции языка программирования высокого уровня
- При выполнении этих операций участвуют **файловые дескрипторы**
- Чтобы использовать эти функции в программах, необходимо подключить заголовочные файлы *fcntl.h*, *sys/types.h*, *sys/stat.h*, *unistd.h*
- Чтобы открыть файл и получить дескриптор для работы с ним, необходимо вызвать функцию *open ()*

```
#include <fcntl.h>
int open(char* file   // имя файла
        , int oflag  // способ открытия файла
        , ...        // необязательный третий аргумент
);
```

Файловые операции

- *O_RDONLY* – файл открывается только для чтения;
- *O_WRONLY* – файл доступен только для записи;
- *O_RDWR* – файл открывается и для чтения, и для записи;
- *O_TRUNC* – приводит к очистке существующего файла;
- *O_APPEND* – приводит к открытию файла в режиме добавления;
- *O_CREAT* – означает создание нового файла. Если файл уже существует, он будет открыт;
- *O_EXCL* – при использовании совместно с *O_CREAT* откажется открывать существующий файл.

Файловые операции

```
#include <fcntl.h>
#include <stdio.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>

int main(int argc, char* argv[])
{
  /* The path at which to create the new file.  */
  char* path = argv[1];
  /* The permissions for the new file.  */
  mode_t mode = S_IRUSR | S_IWUSR | S_IRGRP | S_IWGRP | S_IROTH;

  /* Create the file.  */
  int fd = open(path, O_WRONLY | O_EXCL | O_CREAT, mode);
  if (-1 == fd)
  {
    /* An error occurred.  Print an error message and bail.  */
    perror("open");
    return 1;
  }
  return 0;
}
```

Файловые операции

- Можно воспользоваться функцией создания файла

```
#include <unistd.h>
int creat(char* file        // символьное имя файла
          , mode_t mode     // режим создания файла
);
```

- По окончании работы с файлом его необходимо закрыть

```
#include <unistd.h>
int close(int fd      // дескриптор открытого файла
);
```

Файловые операции

- Для записи данных предназначена функция *write()*

```
#include <unistd.h>
ssize_t write(int fd         // дескриптор файла
              , void* Buf    // адрес буфера, в котором хранятся данные для записи
              , size_t BytesToWrite // сколько байтов записать
);
```

- Природа данных, которые записываются посредством функции *write()*, ей неинтересна
- Она работает с байтовыми последовательностями

Файловые операции

```
#include <fcntl.h>
#include <stdio.h>
#include <string.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <time.h>
#include <unistd.h>

char* GetTimestamp()
{
  time_t now = time(NULL);
  return asctime(localtime(&now));
}

int main(int argc, char* argv[])
{
  char* filename = argv[1];
  char* timestamp = GetTimestamp();
  int fd = open(filename, O_WRONLY | O_CREAT | O_APPEND, 0666);
  size_t length = strlen(timestamp);
  write(fd, timestamp, length);
  close(fd);
  return 0;
}
```

Файловые операции

```
#include <assert.h>
#include <unistd.h>

ssize_t WriteAll(int fd, const void* buffer, size_t count)
{
  size_t leftToWrite = count;
  while (leftToWrite > 0)
  {
    size_t written = write(fd, buffer, count);
    if (-1 == written)
      /* An error occurred; fail.  */
      return -1;
    else
      /* Keep count of how much more we need to write.  */
      leftToWrite -= written;
  }
  /* We should have written no more than COUNT bytes!  */
  assert(leftToWrite == 0);

  /* The number of bytes written is exactly COUNT.  */
  return count;
}
```

Файловые операции

- Чтение осуществляется функцией *read()*

```
#include <unistd.h>
ssize_t read(int fd                    // дескриптор файла
             , void* Buf               // адрес буфера, в который
   читаются данные
             , size_t BytesToRead // сколько байтов прочитать
);
```

- Природа данных, которые считываются из файла посредством функции *read()*, ей неинтересна
- Она работает с байтовыми последовательностями

Файловые операции

```
#include <fcntl.h>
#include <stdio.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>
int main(int argc, char* argv[])
{
  unsigned char buffer[16];
  size_t offset = 0;
  size_t bytesRead;
  int i;
  int fd = open(argv[1], O_RDONLY);
  do
  {
    bytesRead = read(fd, buffer, sizeof (buffer));
    printf ("0x%06x : ", offset);
    for (i = 0; i < bytes_read; ++i)
      printf("%02x ", buffer[i]);
    printf("\n");
    offset += bytesRead;
  }
  while (bytesRead == sizeof (buffer));
  close(fd);
  return 0;
}
```

Файловые операции

- Для выполнения перемещения внутри файла - функция *lseek()*

```
#include <unistd.h>
off_t lseek(int fd                  // Дескриптор открытого файла
          , off_t bytesToMove       // Количество байтов, на которые надо перейти
          , int whence              // С какой позиции перемещаться
);
// Пример использования
off_t pos = lseek(fd, 0, SEEK_CUR);
```

- Если *whence == SEEK_SET*, то второй аргумент - смещение от начала файла
- Если *whence == SEEK_CUR*, то смещение производится от текущей позиции в файле
- Если *whence == SEEK_END*, то второй аргумент - смещение от конца файла

Файловые операции

```
#include <fcntl.h>
#include <stdlib.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>

int main (int argc, char* argv[])
{
  int zero = 0;
  const int megabyte = 1024 * 1024;

  char* filename = argv[1];
  size_t length = (size_t) atoi(argv[2]) * megabyte;

  int fd = open(filename, O_WRONLY | O_CREAT | O_EXCL, 0666);
  lseek(fd, length - 1, SEEK_SET);
  write(fd, &zero, 1);
  close(fd);
  return 0;
}
```

Файловые операции

- Информацию о файле можно получить с помощью функций *stat()* и *fstat()*

```
#include <sys/stat.h>

int stat(char* filename             // символьное имя файла
        , struct stat* stat_info // указатель на дескриптор информации о файле
);

int fstat(int fd // дескриптор файла
         , struct stat* stat_info // указатель на дескриптор информации о файле
);
```

Файловые операции

- Полезные поля структуры *stat*:
  - *st_mode* – код доступа к файлу.
  - *st_uid* и *st_gid* – идентификаторы пользователя и группы.
  - *st_size* – размер файла в байтах.
  - *st_atime* – время последнего обращения к файлу.
  - *st_mtime* – время последней модификации файла.

Файловые операции

- Следующие макросы, проверяют поле *st_mode*, и определяют тип файла:
  - *S_ISBLK* – блочное устройство.
  - *S_ISCHR* – символьное устройство.
  - *S_ISDIR* – каталог.
  - *S_ISFIFO* – именованный конвейер.
  - *S_ISLNK* – символическая ссылка.
  - *S_ISREG* – обычный файл.
  - *S_ISSOCK* – сокет.
- В поле *st_dev* структура *stat* содержится младший и старший номера устройства, на котором расположен файл
- В поле *st_info* содержится номер индексного дескриптора файла, определяющий местоположение файла в ФС

Файловые операции

```
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>

char* ReadFile(const char* filename, size_t* length)
{
  int fd;
  struct stat fileInfo;
  char* buffer;
  fd = open(filename, O_RDONLY);
  fstat(fd, &fileInfo);
  *length = fileInfo.st_size;
  if (0 == S_ISREG(file_info.st_mode))
  {
    close(fd);
    return NULL;
  }

  buffer = (char*) malloc(*length);
  read(fd, buffer, *length);
  close (fd);
  return buffer;
}
```

Файловые операции

- Функция *writev()* записывает в файл несколько несвязанных буферов
- Это **векторная запись**
- Нужно создать структуру, задающей начало и конец каждого буфера и представляющей собой массив элементов типа *struct iovec*
- Каждый из них описывает одну область памяти
- В поле *iov_base* задается адрес начала области, а в поле *iov_len* – ее длина
- Антипод - функция **векторного чтения** *readv()*

Файловые операции

```
#include <fcntl.h>
#include <stdlib.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <sys/uio.h>
#include <unistd.h>
int main(int argc, char* argv[])
{
  int fd;
  struct iovec* vec;
  struct iovec* vecNext;
  int i;
  char newline = '\n';
  char* filename = argv[1];
  argc -= 2;
  argv += 2;
  vec = (struct iovec*) malloc(2 * argc * sizeof (struct iovec));
  vecNext = vec;
  ...
```

Файловые операции

```
  ...
  for (i = 0; i < argc; ++i)
  {
    vecNext->iov_base = argv[i];
    vecNext->iov_len = strlen (argv[i]);
    ++vecNext;
    vecNext->iov_base = &newline;
    vecNext->iov_len = 1;
    ++vecNext;
  }
  fd = open(filename, O_WRONLY | O_CREAT);
  writev(fd, vec, 2 * argc);
  close(fd);
  free(vec);
  return 0;
}
```

Файловые операции

```
#include <unistd.h>
char* getcwd(char* buf    // буфер для заполнения полным именем текущего каталога
             , size_t size // размер буфера
);

#include <sys/stat.h>
#include <sys/types.h>
int chdir(char* path // символьное имя каталога
);

int rmdir(char* path // символьное имя каталога
);

int mkdir(char* path // символьное имя создаваемого каталога
, mode_t mode
);
```

Файловые операции

```
#include <dirent.h>
#include <sys/types.h>

DIR* opendir(char* path   // символьное имя каталога
);

struct dirent* readdir (DIR* dir_p // указатель на структуру каталога
);

int closedir (DIR* dir_p // указатель на структуру каталога
);
```

Файловые операции

```
#include <assert.h>
#include <dirent.h>
#include <stdio.h>
#include <string.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>

const char* GetFileType(const char* path)
{
  struct stat st;
  lstat(path, &st);
  if (S_ISLNK(st.st_mode))
    return "symbolic link";
  else if (S_ISDIR(st.st_mode))
    return "directory";
  else if (S_ISCHR(st.st_mode))
    return "character device";
  else if (S_ISBLK(st.st_mode))
    return "block device";
  else if (S_ISFIFO(st.st_mode))
    return "fifo";
  else if (S_ISSOCK(st.st_mode))
    return "socket";
  else if (S_ISREG(st.st_mode))
    return "regular file";
  else
    /* Unexpected.  Each entry should be one of the types above.  */
    assert(0);
}
...
```

Файловые операции

```
...
int main(int argc, char* argv[])
{
  char* dirPath;
  DIR* dir;
  struct dirent* entry;
  char entryPath[PATH_MAX + 1];
  size_t pathLen;
  if (argc >= 2)
    dirPath = argv[1];
  else
    dir_path = ".";
  strncpy(entryPath, dirPath, sizeof (entryPath));
  pathLen = strlen(dirPath);
  if (entryPath[pathLen - 1] != '/')
  {
    entryPath[pathLen] = '/';
    entryPath[pathLen + 1] = '\0';
    ++pathLen;
  }
  dir = opendir(dirPath);
  while ((entry = readdir(dir)) != NULL)
  {
    const char* type;
    strncpy(entryPath + pathLen, entry->d_name,
         sizeof (entryPath) - pathLen);
    type = GetFileType(entryPath);
    printf("%-18s: %s\n", type, entryPath);
  }
  closedir(dir);
  return 0;
}
```

Файловые операции

```
#include <unistd.h>
int rename(char* old_path   // старое имя файла
           , char* new_path // новое имя файла
);

int remove(char* path // имя файла для удаления
);

int unlink(char* path   // имя файла
);
```

Файловые операции

```
#include <errno.h>
#include <stdio.h>
#include <unistd.h>

int main(int argc, char* argv[])
{
  char targetPath[256];
  char* linkPath = argv[1];

  int len = readlink(linkPath, targetPath, sizeof (targetPath) - 1);

  if (-1 == len)
  {
    if (errno == EINVAL)
      fprintf(stderr, "%s is not a symbolic link\n", linkPath);
    else
      perror("readlink");
    return 1;
  }
  else
  {
    targetPath[len] = '\0';
    printf("%s\n", targetPath);
    return 0;
  }
}
```

Файловые операции

- Функция *access()* определяет, имеет ли вызвавший ее процесс право доступа к указанному файлу
- Она принимает два аргумента: имя проверяемого файла и битовое объединение флагов *R_OK*, *W_OK* и *X_OK*
- В качестве второго аргумента можно передавать *F_OK*
- В этом случае проверяется только существование файла

Файловые операции

```
#include <errno.h>
#include <stdio.h>
#include <unistd.h>
int main(int argc, char* argv[])
{
  char* path = argv[1];
  int rval;

  rval = access(path, F_OK);
  if (0 == rval)
    printf("%s exists\n", path);
  else
  {
    if (errno == ENOENT)
      printf("%s does not exist\n", path);
    else if (errno == EACCES)
      printf("%s is not accessible\n", path);
    return 0;
  }
  rval = access(path, R_OK);
  if (0 == rval)
    printf("%s is readable\n", path);
  else
    printf("%s is not readable (access denied)\n", path);
  rval = access(path, W_OK);
  if (0 == rval)
    printf("%s is writable\n", path);
  else if (errno == EACCES)
    printf("%s is not writable (access denied)\n", path);
  else if (errno == EROFS)
    printf("%s is not writable (read-only filesystem)\n", path);
  return 0;
}
```

Файловые операции

- Функция *fcntl()* – это точка доступа к нескольким особенным файловым операциям
- Первый аргумент – дескриптор файла, второй – код операции
- Для некоторых операций используется третий аргумент
- Перед блокированием файла необходимо создать и проинициализировать структуру типа *flock*

Файловые операции

- В поле *l_type* следует указать константу *F_RDLCK* для блокировки чтения и *F_WRLCK* – для блокировки записи
- Затем вызывается функция *fcntl()*, которой передается дескриптор файла, код операции *F_SETLCLW* и указатель на структуру типа *flock*
- Если такая блокировка была сделана другим процессом, то функция *fcntl()* перейдет в режим ожидания, пока «чужая» блокировка не будет снята

Файловые операции

```
#include <fcntl.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>
int main(int argc, char* argv[])
{
  char* file = argv[1];
  int fd;
  struct flock lock;
  printf ("opening %s\n", file);
  fd = open(file, O_WRONLY);
  printf ("locking\n");
  memset(&lock, 0, sizeof(lock));
  lock.l_type = F_WRLCK;
  fcntl(fd, F_SETLKW, &lock);
  printf ("locked; hit enter to unlock... ");
  getchar();
  printf ("unlocking\n");
  lock.l_type = F_UNLCK;
  fcntl(fd, F_SETLKW, &lock);
  close (fd);
  return 0;
}
```

See also

- Робачевский, А. Операционная система Unix, 2 изд./ А.Робачевский, С.Немнюгин, О.Стесик. – СПб.: БХВ-Петербург, 2010. – 656 с.
- Инструменты Linux для Windows-программистов - http://rus-linux.net/nlib.php?name=/MyLDP/BOOKS/Linux-tools/index.html
- Лав, Р. Linux. Системное программирование/ Р.Лав. – СПб.: Питер, 2008. – 416 с.
- Файловая система NTFS - http://www.ixbt.com/storage/ntfs.html
- Volume and File Structure of Disk Cartridges for Information Interchange - http://www.ecma-international.org/publications/files/ECMA-ST
- Second Extended File System - http://www.nongnu.org/ext2-doc/
- НИЗКОУРОВНЕВЫЙ ВВОД-ВЫВОД - http://www.opennet.ru/docs/RUS/zlp/005.html
- Network File System Version 4 (nfsv4) - http://datatracker.ietf.org/wg/nfsv4/charter/