acm.mipt.ru

олимпиады по программированию на Физтехе

Файлы (системные функции)

Примеры использования системных функций для работы с файлами. Программа должна читать из текстового файла информацию и записывать ее в бинарный файл. Файловый дескриптор – целое число ≤ 0. Информация считывается из тестового файла. Имена обих файлов передаются как параметры командной строки.

Необходим инструментарий для получения информации о:

1. размере файла (в байтах)
2. владельце файла
3. том, является ли файл обычным файлом или директорией
4. правах доступа

и др.

Пример создания файла с заданными правами, запись в файл бинарной информации

Пример на языке C

```c
#include <unistd.h>
#include <stdio.h>
#include <string.h>
#include <sys/types.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <stdlib.h>
#include <errno.h>
#include <ctype.h>

// Описание структуры для записи в файл
typedef struct S{
   char key;
   float a,b;
}SomeStr;

int main(int argc, char **argv){
  int k;
// Дескриптор файла вывода
  int fd_out;
// Объявления файлового дескриптора
  FILE *f1;
//Если имена файлов не переданы, прервать программу
  if (argc<2){
    printf("Необходимо указать 2 имя файла\n");
    exit(1);
  }
// Открыть файл на чтение и запись только для владельца файла.
// Если файла нет - создать. Если есть - стереть все содержимое

  fd_out = open(argv[1],O_RDWR|O_CREAT|O_TRUNC,S_IRUSR|S_IWUSR);

// Если не удалось открыть файл, прервать программу
  if (fd_out < 1){
  perror("Файл не может быть открыт\n");
  exit(1);
  }
// Открываем текстовый файл на чтение. Права доступа определяются
// автоматически
  f1 = fopen(argv[2],"r");
  if(errno){
    perror("Файл не может быть открыт\n");
     exit(1);
   }
// Буфер структур для записи в бинарный файл
  SomeStr data[10];

  int  n,i = 0;
  float a,b;
  char z[100];
  char key;
  int c;

    int bf = 0;
// Чтение из тестового файла
// Пример записи в файл: f12 45w1 2, то есть  f 12 45; w 1 2
// fcanf возвращает количество прочитанных значений для переменных
// из списка
// Проверим, если удалось прочитать меньше трех,
// прерываем цикл (файл кончился)
   while(c=fscanf(f1,"%c%f%f",&key,&a,&b)){
        if(c<3) break;
// отладочная печать
      printf("key=%c a=%0.2f b=%0.2f\n",key,a,b);
      data[bf].key = key;
      data[bf].a = a;
      data[bf].b = b;
```

Пример на языке C++

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

#include <iostream>
#include <cstdlib>
#include <fstream>

using namespace std;

// Описание структуры для записи в файл
struct SomeStr{
   char key;
   float a,b;
};

int main(int argc, char **argv){
  int k;
// Дескриптор файла вывода
  int fd_out;
// объект-файл на ввод (i<nput>f<ile>stream).
// Все файлы - это потоки
  ifstream f1;

//Если имена файлов не переданы, прервать программу
  if (argc<2){
    printf("Необходимо указать 2 имя файла\n");
    exit(1);
  }
// Открыть файл на чтение и запись только для владельца ф
// Если файла нет - создать. Если есть - стереть все соде

  fd_out = open(argv[1],O_RDWR|O_CREAT|O_TRUNC,S_IRUSR|S_

// Если не удалось открыть файл, прервать программу
  if (fd_out < 1){
  perror("Файл не может быть открыт\n");
  exit(1);
  }

// Открываем файл - объект типа ifstream
// функция open
  f1.open(argv[2]);
// errno так же принимает нудные знвчения
  if(errno){
    perror("Файл не может быть открыт\n");
     exit(1);
   }
// Буфер структур для записи в бинарный файл
  SomeStr data[10];

  int  n,i = 0;
  float a,b;
  char key;
  int c;

    int bf = 0;
// Чтение из тестового файла
// Пример записи в файл: f12 45w1 2, то есть  f 12 45; w
// Если оператор ввода (>>) не может прочитать все значен
// переменныз из списка, цикл прервется
   while(f1 >> key >> a >> b){
```

```
      bf++;
// если буфер полон, записываем его в бинарный файл
        if(bf==10){
        printf("bf:=%d пишем\n",bf);
        write(fd_out,&data,sizeof(SomeStr)*10);
        bf = 0;
       }
   }
// Если буфер не полон, тоже нужно записать в файл.
      bf--;
      if(bf > 0 && bf <9){
        printf("bf:=%d после пишем\n",bf);
        write(fd_out,data,sizeof(SomeStr)*(bf));
      }

// Закрываем текстовый файл
  fclose(f1);

// Закрываем бинарный файл
   close(fd_out);

return 0;
}
```

```
// отладочная печать
      cout << key << ' ' << a << ' ' << b << endl;
      data[bf].key = key;
      data[bf].a = a;
      data[bf].b = b;
      bf++;
// если буфер полон, записываем его в бинарный файл
        if(bf == 10){
        cout << "bf:=" << bf << " пишем\n";
        write(fd_out, data, sizeof(SomeStr) * 10 );
        bf = 0;
       }
   }
// Если буфер не полон, тоже нужно записать в файл.
      bf--;
      if(bf > 0 && bf <9){
        cout << "bf:=" << bf << " после пишем\n";
        write(fd_out, data, sizeof(SomeStr)*(bf));
      }

// Закрываем текстовый файл
   f1.close();

// Закрываем бинарный файл
   close(fd_out);

return 0;
}
```

Но, для того чтобы получить ИНСТРУМЕНТ, лучше написать класс на C++.

Сначала определим какие функции этого инструмента будут нужны. Для этого пишем **интерфейс** класса. В нем определяются все атрибуты (объекты-переменные) и функции, которые будут использоваться для объектоа этого класса.

Файл с описанием всех констант, объявлением новых типов данных, классов, структур, и интерфейсов функций – это заголовочный файл. Имеет расширение **.h**

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

#include <iostream>
#include <cstdlib>

using namespace std;


class SystemFile{
// системная структура для информации о пользователе
  struct passwd *result;
// системная структура для информации о файле
  struct stat sb;
// файловый дескрипор
  int id;
// для прав доступа к файлу
  char mod;

public:
// Конструктор. Вызывается всегда при создании объекта
  SystemFile();
// Деструктор вызывается всегда, когда должен быть
// удален объект: при завершении функции, в которой он локальный
// или при вызове delete
   ~SystemFile();
// закрытие файла
  bool closef();
// открытие файла по его имени в каталоге
  unsigned char openf(char*);
// получить имя владельца файла
  string getUser();
// получить атрибуты доступа файла
  mode_t getMode();
// получить имя владельца файла (в байтах)
  long getSize();
// запись в файл. Передается уаказатель
// на память для записи и размер (в байтах)
  bool writef(void*, long);
// Чтение из файла. В память по указателю
// указаного размера. Память предварительно
// должна быть ВЫДЕЛЕНА
  bool readf(void*, long);

// печать общей информации о файле
  void about();
// получить количество записей в файле,
// если известен размер записи
  long numbersOfRecord(long);

// найти f1 и заменить запись на f2 (полное соответствие)
// только для бинарного файла
  void replace(void* f1, void* f2, size_t sizeRec);

// найти f1 и удалить (полное соответствие)
// только для бинарного файла
// для удаления использовать функцию truncate
```

```
// или ftruncate
  void deletef(void* f1,  size_t sizeRec);

// Копировать текущий файл в файл с именем name
// Если файла name нет, то мы его создаем (O_CREATE|O_EXCL|O_WRONLY)
// При ошибки существования файла, просто открываем его с флагами:
// O_TRANC|O_WRONLY

   void copy(char * name);
};
```

Назовем этот файл с интерфейсом **sysfile.h.**

В отдельном файле опишем реализацию всех объявленных в файле sfile.h. Это будет отдельный файл file.cpp.

```
#include "systfile.h"
// Конструктор. Инициализируем атрибуты.
SystemFile::SystemFile(){
  result = 0; //указатель на структуру 0
  id = -1; // если файл не открыт, дескриптор -1

};

// Деструктор. При удаленни объкта типа SystemFile
// файл должен быть закрыт.
SystemFile::~SystemFile(){
    if (id > -1){
       close(id);
       id = -1;
    }
};

//функция класса для открытия файла
unsigned char SystemFile::openf(char* name)
{
   unsigned char er;
// Связывание дескриптора id с именем файла
  id = open(name,O_RDWR|O_CREAT|O_EXCL,0775);
  er = errno;
//Проверка открылся ли файл
    if (errno){
      if (errno == EEXIST){
       id = open(name,O_RDWR);
       er = 0;
      }else{
       return er;
      }
   }
// заполнение структуры stat информацией о файле
   fstat(id, &sb);
// получение имени пользователя по uid
   result = getpwuid(sb.st_uid);
   return er;
};
// запись в файл всего что есть в буфере (побайтно)
bool SystemFile::writef(void* buf, long size){
    long skolko = 0;
// системный вызов
    skolko = write(id, buf, size);
    if (skolko == size)
      return 1;
    else
      return 0;
};

// закрыть файл
bool SystemFile::closef(){
     if (id>-1){
       close(id);
       id = -1;
       return 1;
     }
    return 0;
};

// печать информации о файле
void SystemFile::about(){
    printf("user: %s\n",result->pw_name);
   switch (sb.st_mode & S_IFMT) {
        case S_IFBLK:  printf("block device\n");
                       break;
        case S_IFCHR:  printf("character device\n");
                   break;
        case S_IFDIR:  printf("directory\n");
                       break;
       case S_IFIFO:  printf("FIFO/pipe\n");
                      break;
       case S_IFLNK:  printf("symlink\n");
                      break;
       case S_IFREG:  printf("regular file\n");
                  break;
       case S_IFSOCK: printf("socket\n");
                      break;
       default:       printf("unknown?\n");
   }

// Для получения той же информации можно воспользоваться макросами:
// S_ISREG(m) -обычный файл?, S_ISDIR(m) -  каталог? и др.

   printf("I-node number:  %ld\n", (long) sb.st_ino);

   printf("Mode:           %lo (octal)\n",(unsigned long) sb.st_mode);

   printf("Link count:    %ld\n", (long) sb.st_nlink);
```

```
   printf("Ownership:  UID=%ld   GID=%ld\n",(long) sb.st_uid, (long) sb.st_gid);
   printf("File size:  %lld bytes\n",(long long) sb.st_size);
   printf("Last status change:  %s", ctime(&sb.st_ctime));
   printf("Last file access: %s", ctime(&sb.st_atime));
   printf("Last file modification:   %s",ctime(&sb.st_mtime));


};
```

Пример файла для тестирования функций (C++) testF.cpp.

```
// включиить заголовочный файл
#include "sysfile.h"

int main(int argc, char** argv){
// создается объект типа  SystemFile
// работает коструктор
  SystemFile file;
// функция openf() возвращает значение errno,
// полученное в процессе создания или открытия файла
  if(file.openf(argv[1])){
     perror("не открывается никак");
     exit(1);
  }
//  печать информации о файле
  file.about();
// объект C++ string
  string s;
// получить строку с консоли
  cin>>s;
// c_str() - функция string возвращает указатель на
// C-строку (массив символов с окончанием '\0'
// length() - возвращает размер строки в символах
  file.writef((void*)s.c_str(),s.length()*sizeof(char));
// закрытие файлаint n
  file.closef();

}
```

Компиляция этого проекта

```
>g++ testF.cpp file.cpp -o testf
```

Задачи

Задача 1.

Реализовать все описанные в интерфейсе класса **SystemFile**, но еще не реализованные функции. Проверить их работу

Задача 2.

Дан интерфейс класса для работы с каталогами.

Написать функции класса.

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

#include <iostream>
#include <cstdlib>

using namespace std;
class Catalog{
// указатель на системную файловую
// структуру для записей в каталоге (список файлов)
   struct dirent* dirrec;
// дескриптор каталога
   DIR *dir;
   unsigned int numbersf;
// массив имен файлов
   string names[100];
public:
// конструкторы
   Catalog();
   Catalog(char *);
// деструктор
   ~Catalog();
// открыть каталог
   unsigned char dopen(char*);
// получить имя файла из списка (по номеру)
   string getFile(int);
// проверить есть ли файл в каталоге
   int isExFile(char*);
// получить атрибуты доступа к файлу, если он есть
// -1, если файла не, остальное как в mod_t
   int getMode(char* );
// печать содержимого файла, если он есть
// 0 - OK, 1 - нет доступа, -1 нет файла
   int catF(char *);
// удалить файл
   int delF(char*);
// закрыть каталог
   bool dclose();
```

```
// получить количество файлов.
   int getNumbers();
// Напечать список файлов с указание их атрибутов, размера и владельца,
//  а также времени создания
   void list();
};
```

Реализовать все функции для работы с каталогом.

Задача 3

Задача решается двумя студентами.

В файл **users** пишется следующая информация:

```
struct User{
  char name[20]; // имя пользователя в системе
   char passwd[25]; // пароль для Вашей программы
// Для получения uid по имени воспользоваться функцией getpwnam
  uid_t; muid// uid  пользователя в системе
  gid_t ngid; // guid  пользователя в системе
  ino_t inod ; // inod каталога, к которому имеет доступ пользователь
  char del; // метка для последующего удаления, если 1 - помечен для удаления, 0 - нет

};
```

Создается "темный каталог", который закрыт на исполнение для всех, кроме его создателя. В этом каталоге есть несколько других каталогов, у которых регулируется доступ для группы и "всех остальных". Каждому пользователю доступен только один каталог. Его **inod** указывается в структуре **User**. Если нужен доступ к файлам из разных каталогов, организуются жесткие ссылки.

Первая программа обеспечивает добаление и удаление пользователей. Удаление пользователей происходит по конкретному запросу "удалить данного пользователя". Так же при каждом обращении к файлу программа проверяет во всех записях метку **del**. Если **del = 1**, эта запись удаляется.

Вторая программ работает непосредственно с "темным" каталогом, предоставляя пользователю следующие возможности: просмотр каталога (**inod**), чтение содержимого заданного файла из этого каталога. При этом программа должна учитывать системные права доступа к каталогу для группы и всех остальных (проверяет к какой группе принадлежит данный пользователь). В начале работы программа запрашивает имя пользователя и пароль, если не совпадают, пользователь не может работать с программой. Все действия пользователей записываются в файл **work_log** в следующем виде:

```
struct LockInf{
  uid_t; muid// uid  пользователя в системе
  char time[20]; // время обращения к программе этого пользователя в формате YY-MM-DD HH:MM:SS
  char answer; // результат выполнения запроса: 0 - не верный логин-пароль, 1 - запрос выполнен, 2 - отказ в доступе

};
```

Если в течении 5 минут пользователь получает **answer = 0** больше 2-х раз подряд, его запись в файле **users** помечается для удаления.

Задача Блокировка.

Рассмотрим задачу блокировки файла и пример на языке С

При попытке одновременного доступа к файлу необходимо организовать разделяемый доступ к файлу. Для этого должна быть установленна блокировка либо на весь файл, либо на его часть.

Примеры двух программ. Первая (**f1.c**) записывает информацию в бинарный файл и ждет нажатия клавиши, а вторая (**f2.c**) - сразу пишет в файл. Сначала запусаем **f1**

f1.c

Файл **block1.c.**

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

int main(int argc, char** argv){
// открыть файл с информацией для записи
 FILE *f = fopen(argv[1], "r");
// структура для установки блокировки
  struct flock lock;
// заполним структуру 0
  memset(&lock, '\0', sizeof(lock));

//  Установка блокировки для записи
// эксклюзивная блокировка (только этот процесс
// может манипулировать с файлом)
  lock.l_type = F_WRLCK;
  lock.l_start = 0L;   // с самого начала файла
  lock.l_whence = 0;// до конца
  lock.l_len = 0L;
  lock.l_pid = getpid(); // установка pid процесса

// откроем файл для блокировки и записи (должен существовать)
   int fb = open(argv[2],O_RDWR|O_APPEND);
// установим моду для обязательной блокировки
  mode_t rw_mode = S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH;
  rw_mode |= S_ISGID; /* добавить бит обязательной блокировки */

  if (fchmod(fb, rw_mode) < 0) {
   perror("cannot change mode\n");
   close(fb);
   return 1;
  }
// прочитаем строчки из файла с информацией
```

f2.c

Файл **block2.c.**

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

int main(int argc, char** argv){
// открыть файл с информацией для записи
 FILE *f = fopen(argv[1], "r");
// структура для установки блокировки
  struct flock lock;
// заполним структуру 0
  memset(&lock, '\0', sizeof(lock));

//  Установка блокировки для записи
// эксклюзивная блокировка (только этот процесс
// может манипулировать с файлом)
// Читать файл могут сразу несколько процессов.
// Если установлена блокировка для записи, никакой процесс
// установить блокировку на этот файл
// Если установлена блокирока для чтения, то другой процесс
// установить блокировку для чтения
// Блокировку для записи в этом случае установить не получи
  lock.l_type = F_WRLCK;
  lock.l_start = 0L;   // с самого начала файла
  lock.l_whence = 0;// до конца
  lock.l_len = 0L;
  lock.l_pid = getpid(); // установка pid процесса

// откроем файл для блокировки и записи (должен существоват
   int fb = open(argv[2],O_RDWR|O_APPEND);
// установим моду для обязательной блокировки
  mode_t rw_mode = S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH;
  rw_mode |= S_ISGID; /* добавить бит обязательной блокиров

  if (fchmod(fb, rw_mode) < 0) {
```

```
 char bif[100];
 fscanf(f, "%s", bif);

 int k;
// попробуем заблокировать файл.
// F_SETLKW - флаг означает, что если уже блокировку
//  установить нельзя, процесс
// будет ждать пока не получит сигнал о
// разблокировнии файла
if (fcntl(fb, F_SETLKW, &lock) < 0){
   perror("Block: ");
   exit(1);
}
// "поймал мыша - ешь не спеша"
// блокировка установлена, никакой другой процесс не может
// обращаться к файлу, пока ее не снимут
  write(fb,bif,strlen(bif));

// организуем ожидание для иллюстрации процесса
  int d;
  printf("нажать на клавишу\n");
  d = getchar();

// снимем блокировку
//  изменили значение поля
  lock.l_type = F_UNLCK;
// еще раз вызывает функцию
k = fcntl(fb,F_SETLKW,&lock);
// сообщение о снятии блокировки
printf("k: %d ok\n",k);
// закрыли блокируемый файл.
close(fb);
// fclose(f);
return 0;
}
```

```
   perror("cannot change mode\n");
   close(fb);
   return 1;
  }
// прочитаем строчки из файла с информацией
 char bif[100];
 fscanf(f, "%s", bif);

 int k;
// попробуем заблокировать файл.
// F_SETLKW - флаг означает, что если уже блокировку
//  установить нельзя, процесс
// будет ждать пока не получит сигнал о
// разблокировнии файла
if (fcntl(fb, F_SETLKW, &lock) < 0){
   perror("Block: ");
   exit(1);
}
// "поймал мыша - ешь не спеша"
// блокировка установлена, никакой другой процесс не может
// обращаться к файлу, пока ее не снимут
  write(fb,bif,strlen(bif));

// организуем ожидание для иллюстрации процесса
  printf("будем ждать\n");

// снимем блокировку
//  изменили значение поля
  lock.l_type = F_UNLCK;
// еще раз вызывает функцию
   k = fcntl(fb,F_SETLKW,&lock);
// сообщение о снятии блокировки
   printf("k: %d ok\n",k);
// закрыли блокируемый файл.
   close(fb);
   fclose(f);
return 0;
}
```

Задача 4

Добавить в класс **SystemFile** две функции **int lock()** и **int unlock()**. Если удалось заблокировать или разблокировать файл, возвращается 0, если нет 1.

Проверить их работу.

Задача 5

Так как и к файлу **users**, и к файлу **work_log** могут одновременно обращаться несколько процессов, предусмотреть блокировку файлов во время записи и чтения.

-- TatyanaOvsyannikova2011 - 12 Oct 2018