УДК 378.147

Н.А. БОНДАРЕНКО, В.И. ШЕХОВЦОВА, Е.А.ЧАСОВСКАЯ

МОДЕЛИРОВАНИЕ РАБОТЫ ФАЙЛОВОЙ СИСТЕМЫ В ОПЕРАЦИОННОЙ СИСТЕМЕ WINDOWS

Проводится анализ основных принципов работы файловой системы операционной системы Windows. Строится объектная модель работы файловой системы. Разрабатываются классы – TFAT («Таблица размещения файлов») и TFST («Таблица свободного пространства»), которые используются для составления программы моделирования реакции файловой системы на запросы прикладных программ по добавлению и исключению файлов.

Введение

Известно, что основное назначение компьютеров заключается в обработке и хранении данных. Поэтому одним из самых главных объектов в нем есть файлы. Файлы отличаются один от другого именами, размерами, структурой и носителем. Для работы с файлами в современных языках программирования предусмотрены разнообразные операции – открытие, закрытие, чтение, запись, создание, исключение и др. Все эти операции выполняются через *файловую систему*, которая является составной частью операционной системы (ОС).

Исторически первой ОС для персональных компьютеров фирмы IBM (или совместимых с ними) была система MS DOS (дисковая операционная система фирмы Microsoft). Именно в рамках этой ОС была разработана файловая система, которая и до сих пор работает в современных компьютерах. Файловая система новой ОС фирмы Microsoft – Windows работает на тех же принципах и алгоритмах, что и в MS DOS (за исключением ОС Windows NT).

Современные файловые системы строятся на принципах быстродействия, удобства и безопасности данных. Они имеют специальные средства защиты файлов, предотвращения их повреждения, возобновления поврежденных файлов, форматирования и дефрагментации дисков.

Современные программисты должны хорошо понимать принципы работы файловой системы в целях ее эффективного использования для рациональной организации своих файлов и повышения быстродействия прикладных программ.

Эта работа посвящена моделированию основных принципов работы файловой системы в операционной системе Windows. Конечно, данная работа не может охватить все аспекты работы файловой системы. В то же время она может дать представление о том, как работает файловая система.

1. Анализ задачи и разработка объектной модели

1.1. Анализ основных принципов работы файловой системы

Как известно, с аппаратной точки зрения жесткий диск является магнитным диском, на поверхности которого сохраняются определенным образом намагниченные участки. Диск вращается на большой скорости над устройством считывания/записи (головкой), которое превращает электромагнитные импульсы от намагниченных участков в двоичные данные. Последние попадают в буфер данных и становятся доступными сначала для файловой системы, а через нее – для прикладной программы. Запись данных происходит в обратном направлении.

Наименьший такой участок (т.е. наименьшее количество данных, которые считываются или записываются на жесткий диск) называется *кластером*. Таким образом, с логической точки зрения жесткий диск можно рассматривать как последовательную совокупность кластеров. В зависимости от емкости диска и аппаратных особенностей размер кластера составляет от 512 байт до 2 Кбайт.

Каждый кластер имеет свой адрес, т.е. свой порядковый номер в последовательности кластеров. Таким образом, при вращении диска над устройством считывания/записи (головкой) последовательно проходят кластеры от первого до последнего.

Для осуществления операции считывания/записи адрес кластера предварительно размещается в регистр адреса. Когда над головкой проходит кластер именно с этим адресом (номером), происходит считывание/запись из буфера данных. При этом сам жесткий диск (его аппаратные средства) не «знает», где и какие файлы на нем размещены. Его функция – запись и считывание данных из кластеров.

Таким образом, главная задача файловой системы (ФС) заключается в обеспечении записи/чтения файлов на диск по запросам программ, т.е. в размещении файлов в кластерах. Например, при образовании нового файла с определенным именем, ФС отвела ему кластеры из 100-го по 500-й и записала в них соответствующие данные. При запросе на чтение файла с этим именем ФС должна обеспечить считывание данных именно из этих кластеров.

Эта задача решается созданием специальной таблицы в составе файловой системы, а именно – так называемой таблицы размещения файлов, или FAT (от англ. File Allocation Table). Каждая строка этой таблицы содержит информацию об имени файла, номер кластера, с которого начинается файл, количество кластеров, занимающих файл (длина файла), и дополнительную информацию (дата образования файла, дата последней модификации, владелец файла и пр.). Приблизительный вид структуры показан в табл. 1.

Таблица 1. Приблизительный вид структуры FAT

Имя файла	Номер кластера, с которого начинается файл	Количество кластеров, которые занимает файл	Дополнительная информация

Но при этом появляются некоторые проблемы. Во-первых, при такой организации FAT каждый новый файл должен начинаться с первого незанятого (свободного) кластера и занимать непрерывный участок кластеров. После исключения любого файла участок, который он занимал, больше не может быть занятым. Во-вторых, если не использовать дисковое пространство, которое освобождается после исключения файлов, диск через очень короткое время будет переполнен. Дело в том, что сама операционная система в процессе работы постоянно записывает, считывает и изымает временные системные файлы независимо от пользователя. Например, операции копирования в буфер (Ctrl+C) и вставки из буфера (Ctrl+V) в действительности происходят через запись соответствующих данных на диск, которые изымаются после закрытия прикладной программы. То же происходит и у многих современных программ, например, MS Word, Excel, PowerPoint, Photoshop и др.

Поэтому проблема использования дискового пространства, которое освобождается после исключения файлов, является очень острой. Она может быть решена за счет другой структуры FAT, а также за счет введения в состав файловой системы новой таблицы – таблицы свободного пространства, или FST (от англ. Free Space Table).

Новая структура FAT также содержит имя файла и дополнительную информацию. Но вместо номера кластера, с которого начинается файл и их количества (т.е. информации об одном непрерывном участке) FAT будет содержать информацию о совокупности участков, в которых будет размещен файл (табл. 2).

Таблица 2. Новая структура FAT

Имя файла	Дополнительная информация	(N1, L1)	. . .	(Nk, Lk)

Здесь (N1, L1) – номер кластера, с которого начинается первый участок и количество кластеров, который он занимает; (Nk, Lk) – номер кластера, с которого начинается к-й (последний) участок, и количество кластеров, который он занимает.

Таким образом, файлы размещаются не в одном непрерывном участке кластеров, а в совокупности таких участков.

Информация о свободных участках кластеров содержится в таблице свободного пространства (FST), которая имеет простую структуру:

(A1, L1)
. . .
(An, Ln)

где (A1, L1) – номер и длина первого свободного участка кластеров;

(An, Ln) – номер и длина последнего свободного участка кластеров.

Взаимодействие между FAT и FST происходит таким способом. После запроса программы на размещение какого-либо файла определенной длины ФС записывает в FAT его имя и дополнительную информацию. Дальше она последовательно берет из FST информацию о свободных участках кластеров и записывает ее в FAT, пока их совокупная длина не будет равняться длине файла. При этом информация об участках, которые отведены под файл, изымается из FST.

При запросе на исключение файла информация о его размещении возвращается к FST. При этом смежные участки объединяются, а сама FST сортируеться по увеличению номеров кластеров. Соответствующая строка изымается из FAT.

Такой подход позволяет рационально использовать дисковое пространство, но и имеет определенный недостаток – снижение скорости при работе с файлами, поскольку теперь данные считываються/записываються не одним непрерывным блоком, а несколькими фрагментами, которые разбросаны по отдельным участкам дискового пространства. Поэтому данная ситуация получила название *фрагментации* диска. Чтобы ускорить работу с файлами, необходимо периодически проводить операцию *дефрагментации*, при которой файловая система физически перемещает файлы так, чтобы каждый из них занимал один непрерывный участок, а свободное пространство также было одним участком.

1.2. Разработка объектной модели

Разработка объектной модели заключается в описании классов (их свойств и методов), которые будут использованы при созданнии программы моделирования работы файловой системы.

На основании проведенного в п. 1.1 анализа работы файловой системы нами предложены два новых класса – FAT и FST. Рассмотрим их детальнее.

Класс FAT. Свойства класса: Count – количество занятых строк FAT. Режим доступа – только чтение. Это свойство автоматически изменяется при добавлении и изъятии файлов. Поэтому режим доступа к этому свойству – только чтение.

Методы класса:

1. AddFile(FN:string;Len:integer) – добавление файла к FAT. Эта процедура имеет два параметра: FN – имя файла и Len – длина файла (количество кластеров, которые занимает файл).

2. DeleteFile(FN:string) – исключение файла. Эта функция имеет один параметр FN – имя файла, который должен быть изъят из FAT. Функция возвращает список участков, которые занимал файл.

3. GetFileInfo(i:integer; out FN:string; out C:chain) – получение информации о файле из FAT. Эта процедура имеет один входной параметр – номер файла в FAT. Исходные параметры: FN – имя файла и C – список участков, которые занимает файл.

Класс FST. Свойства класса: Count – количество занятых строк FST. Режим доступа – только чтение. Это свойство автоматически изменяется при добавлении и изъятии участков из FST. Поэтому режим доступа к этому свойству – только чтение.

Методы класса:

1. GetFreeSpace(L:integer) – выделение участков под файл длиной L. Эта функция возвращает значение в виде списка участков, в которых будет размещен файл. При этом данные участки изымаются из FST.

2. AddFreeSpace(C:space) – возвращение свободных участков к FST. Эта процедура имеет один параметр C – список участков, которые были высвобождены при изъятии файла. Она соответствующим образом коррегирует FST (сортирует по адресу и совмещает смежные участки).

3. GetFreeSpaceInfo(i:integer; out addr:integer; out len:integer) – получение информации об *i-м* свободном участке в FST. Исходные параметры этой процедуры: addr – адрес *i-го* свободного участка; len – длина этого участка.

4. GetTotalFree – получение информации о количестве свободного пространства на диске. Эта функция не имеет входных параметров и возвращает общее количество свободных кластеров на диске.

Такие основные свойства и методы, которые дают возможность для разработки программы моделирования работы файловой системы.

Таким образом, на основании анализа файловой системы разработана объектная модель, с помощью которой можно разработать программу моделирования ее работы. В этой модели предложены классы FAT и FST, определены их свойства и методы.

2. Программная реализация объектной модели

В соответствии с объектной моделью было выполнено программирование свойств и методов классов FAT и FST средствами языка Delphi 7. Описание и реализация этих классов содержится в одном модуле.

Таким образом, выполнена программная реализация классов TFAT и TFST, с помощью которых будет разработана программа моделирования работы файловой системы.

Соответствующий программный код приведен в Приложении 1.

3. Разработка программы моделирования файловой системы

3.1. Разработка программы моделирования

Программа моделирования работы файловой системы состоит из формы. Содержание моделирования заключается в том, что с помощью полей «Имя файла» и «Длина» будем задавать произвольное имя файла и его длину. Кнопка «ПРИБАВИТЬ ФАЙЛ» моделирует реакцию файловой системы на запрос по образованию нового файла, а именно:

1. Выполняется сравнение длины файла и общего свободного пространства на диске.
2. Из таблицы FST выбираются свободные участки кластеров для размещения файла.
3. Выбранные участки изымаются из FST и добавляются к FAT с соответствующей коррекцией этих таблиц.

Все изменения в этих таблицах отображаются в полях FAT и FST.

Аналогичным способом отслеживается реакция файловой системы на запрос по исключению файлов.

В приложении 2 приводится текст программы моделирования файловой системы на языке Delphi 7.

3.2. Результаты моделирования

В начале моделирования работы файловой системы условно принято, что дисковое пространство состоит из свободных 16000 кластеров, номер первого кластера равняется 1, таблица FAT пустая. Для улучшения отслеживания действий файловой системы будем считать, что дополнительная информация о файлах отсутствует.

После запуска программы моделирования были отображены начальные состояния FAT и FST в соответствии с этими предположениями.

Здесь запись (1 16000) означает участок, который начинается из кластера №1 и имеет длину 16000 кластеров.

Дальше была выполнена проверка реакции файловой системы на добавление и исключение файлов. В соответствующие поля было введено произвольное имя файла «ABC» и длина 100 и нажата кнопка «ПРИБАВИТЬ ФАЙЛ». Получен такой результат.

1. В FAT появилась строка с именем файла и участком, какой он будет занимать.
2. Файлу отведен один участок в соответствии с его длиной.
3. Свободное пространство сократилось и теперь начинается с 101-го кластера и составляет 15900 кластеров (т.е. 1600–100=15900).

Дальше аналогичным способом были прибавлены еще три файла разной длины.

Был проверен запрос на исключение первого файла («ABC»).

Для этого в списке FAT был выделен файл «ABC» и нажата кнопка «ИЗЪЯТЬ ФАЙЛ». Таким образом, полученный результат отвечает ожидаемому.

Дальше был изъят «Файл_3». После исключения этого файла в программе моделирования также получен ожидаемый результат.

Таким образом, после исключения файлов начинается фрагментация свободного пространства.

Было также проверено добавление файла в этих условиях. Прибавлен файл «XXX» длиной 450 кластеров. Свободные участки должны быть занятыми частями нового файла.

Результат, полученный в программе моделирования, полностью отвечает ожиданиям.

Таким образом, разработанная программа моделирования работы файловой системы работает в соответствии с ожиданием и может быть использована в учебных целях.

С этой целью в программе для быстрого заполнения диска предусмотрена кнопка «ТЕСТ», которая моделирует автоматическое добавление 20 файлов с именами «FILE_1» ... «FILE_20» определенной длины. Дальше в произвольном порядке было изъято несколько файлов и прибавлен файл «Aaa» длиной 5000 кластеров. Таким образом, разработана программа моделирования файловой системы в операционной системе Windows. Показана реакция файловой системы на запросы прикладных программ на добавление и исключение файлов. Полученные результаты подтверждают правильность работы этой программы.

Заключение

Результаты выполненной работы следующие:

1. Проведен анализ основных принципов работы файловой системы операционной системы Windows.
2. На основании этого анализа построена объектная модель работы файловой системы.
3. Средствами языка Delphi 7 разработаны классы – TFAT («Таблица размещения файлов») и TFST («Таблица свободного пространства»), которые были использованы для разработки программы моделирования.
4. Проведено тестирование разработанной программы с использованием новых классов на примере моделирования реакции файловой системы на запросы прикладных программ по добавлению и исключению файлов.
5. Полученные результаты демонстрируют правильность работы новых классов и программы моделирования.
6. Данная разработка может быть полезной при преподавании дисциплин, связанных с вычислительной техникой, системным и прикладным программированием.

Список литературы: **1.** *Культин Н.* Программирование в Delphi 5. 2000. Спб.: Питер. 464 с. **2.** *Бондаренко М. А.* Основи інформаційних технологій та програмування, Х.: ФОП Павленко О.Г.. 2010. 600 с.

Поступила в редколлегию 06.02.2010

Бондаренко Николай Андреевич, канд. техн. наук, профессор кафедры информатики и компьютерных технологий Украинской инженерно-педагогической академии. Научные интересы: проектирование технических систем. Адрес: Украина, 61145, Харьков, ул. Университетская, 16, тел. 773-79-17.

Шеховцова Виктория Ивановна, ассистент кафедри информатики и компьютерных технологий Украинской инженерно-педагогической академии. Научные интересы: проектирование информационных систем. Адрес: Украина, 61145, Харьков, ул. Университетская, 16, тел. 773-79-17.

Часовская Елена Александровна, магистрант Украинской инженерно-педагогической академии. Научные интересы: проектирование информационных систем. Адрес: Украина, 61145, Харьков, ул. Університетська, 16, тел. 773-79-17.

```
unit CLASSFST;
interface
uses
 Windows, Messages, SysUtils, Variants, Classes,
Graphics, Controls, Forms
 Dialogs, StdCtrls, DateUtils;
type space = record
 addr:integer; // Адрес кластера
 len:integer; // Длина участка
end;
type chain=record
 ASP:array[1..100] of space; // Цепочка участков
 Len:integer;
end;
type fal=record // Запись в таблице файлов
 FileName:string; // Имя файла
 Alloc:chain; // Размещение файла
end;

 { Определение класса «Таблица свободного
пространства»}
type
TFST = class
private
 { Внутренние переменные}
 ifSTCount: integer; // Для хранения количества
цепочек в таблице
 aFST: array [1..1000] of space; // Таблица сво-
бодного пространства

 { Процедуры и функции доступа к свойствам}
 function GetCount: integer; // Только чтение
public
 { Свойства класса}
 { Количество цепочек}
 property Count: integer read GetCount;

 { Методы класса}
 { Запрос на свободное пространство}
 function GetFreeSpace(L:integer):chain;
 { Возвращение свободного пространства к таб-
лице}
 procedure AddFreeSpace(C:space);
 procedure GetFreeSpaceInfo(i:integer; out
addr:integer; out len:integer);
 function GetTotalFree():integer;
end;
 { Определение класса «Таблица размещения
файлов»}
type
TFAT = class
private
 { Внутренние переменные}
 ifATCount: integer; // Для хранения количества
файлов в таблице
 aFAT: array [1..1000] of fal; // Таблица разме-
щения файлов

 { Процедуры и функции доступа к свойствам}
 function GetCount: integer; // Только чтение
public
 { Свойства класса}
 { Количество файлов}
 property Count: integer read GetCount;

 { Методы класса}
 { Исключение файла}
 function DeleteFile(FN:string):chain;
 { Добавление нового файла}
 procedure AddFile(FN:string;C:chain);
 { Получение информации о размещении файла}
 procedure GetFileInfo(i:integer; out FN:string; out
C:chain);
end;

implementation
 { Реализация свойств класса FST}
 { Чтение количества цепочек}
function TFST.GetCount: integer;
begin
 Result:= ifSTCount;
end;

 { Реализация методов класса FST}
 { Запрос на свободное пространство}
function TFST.GetFreeSpace(L:integer):chain;
var
 C:chain;
 i,k,sum:integer;
 Rest:integer;
begin
 Rest:=L;
 // Проверка количества свободного простран-
ства
 sum:=0;
 for i:=1 to ifSTCount do sum:=sum+aFST[i].len;
 if Rest>sum then
 begin
 ShowMessage("НЕТ СВОБОДНОГО ПРО-
СТРАНСТВА ДЛЯ РАЗМЕЩЕНИЯ ФАЙЛА");
 C.Len:=0;
 Result:=C;
 exit;
 end;
 for i:=1 to ifSTCount do
 begin
 if Rest>=aFST[i].len then // Свободная область
меньше за остальной файл
 begin
 C.Len:=i; // Адрес участка
 C.ASP[i].addr:= aFST[i].addr; // Длина участка
 C.ASP[i].len:= aFST[i].len;
 Rest:=Rest-aFST[i].len; // Остальной файл
 aFST[i].addr:=0; // Признак занятости участка
 aFst[i].len:=0;
 end
 else // Свободная область большая за остальной
файл
 begin
 C.Len:=i;
 C.ASP[i].addr:= aFST[i].addr;
 C.ASP[i].len:=rest;
 // Корегування FST
 aFST[i].addr:=aFST[i].addr+Rest; // Новый адрес
участка
 aFST[i].len:=aFST[i].len-Rest; // Новая длина уча-
стка
 Rest:=0; // Файл размещен
 break;
 end;
 end;
 // Исключение занятых участков
 sum:=0; // Подсчет занятых участков
 i:=1;
```

```
      while и <= ifSTCount do
      begin
      if aFST[i].len=0 then // Участок занят
      begin
      sum:=sum+1; // Подсчет занятых участков
         for k:=i to ifSTCount-1 do
aFST[k]:=aFST[k+1];
      end
      else i:=i+1;
      end;
       ifSTCount:=ifSTCount-sum; // Количество
записей в FST
      result:=C;

     end;

      { Возвращение свободного пространства в
таблицу}
     procedure TFST.AddFreeSpace(C:space);
     var
      i,k,sum:integer;
      SP:space;
     begin
       ifSTCount:=ifSTCount+1;
      aFST[ifSTCount].addr:=C.addr;
       aFST[ifSTCount].len:=C.len;
      // Сортировка FST по адресу
      SP.addr:=aFST[1].addr;
      SP.len:=aFST[1].len;
      for i:=1 to ifSTCount-1 do
      begin
      SP.addr:=aFST[i].addr;
      SP.len:=aFST[i].len;
      for k:=i+1 to ifSTCount do
      begin
      if aFST[k].addr<SP.addr then
      begin
      SP.addr:=aFST[k].addr;
      SP.len:=aFST[k].len;
      aFST[k]:=aFST[i];
      aFST[i]:=SP;
      end;
      end;
      end;
      // Объединение смежных участков
      sum:=0;
      for i:=1 to ifSTCount-1 do
      begin
       if aFST[i].addr+aFST[i].len=aFST[i+1].addr
then // Смежные участки
      begin
      aFST[i].len:=aFST[i].len+aFST[i+1].len; // Но-
вая длина участка
      sum:=sum+1;
      // Исключение смежных участков
         for k:=i+1 to ifSTCount do
aFST[k]:=aFST[k+1];
      end;

      end;
       ifSTCount:=ifSTCount-sum;

     end;
     procedure TFST.GetFreeSpaceInfo(i:integer; out
addr:integer; out len:integer);
     begin
      if (i<=0) or (i>ifSTCount) then
      begin
      addr:=0;
      len:=0;
      Showmessage("ОШИБКА ДОСТУПА К FST");
      exit;
      end;
      addr:=aFST[i].addr;
      len:=aFST[i].len;
     end;
     function TFST.GetTotalFree():integer;
     var
      i,sum:integer;
     begin
      sum:=0;
      for i:=1 to ifSTCount do sum:=sum+aFST[i].len;
      result:=sum;
     end;

     //—

      { Реализация свойств класса FAT}
      { Чтение количества файлов}
     function TFAT.GetCount: integer;
     begin
      Result:= ifATCount;
     end;

      { Реализация методов класса FAT}
      { Исключение файла}
     function TFAT.DeleteFile(FN:string):chain;
     var
      и, к: integer;
      C:chain;
     begin
     // Showmessage("DeleteFile");
      // Поиск файла в FAT
      k:=0;
      for i:=1 to ifATCount do
      begin
      if FN=aFAT[i].FileName then
      begin
      k:=i;
      break;
      end;
      end;
      if k=0 then
      begin
      ShowMessage("ФАЙЛ "+FN+" НЕ НАЙДЕН');
      C.Len:=0;
      result:=C;
      exit;
      end;
      result:=aFAT[k].Alloc;
      // Корегування FAT
      if k=ifATCount then
      begin
      ifATCount:=ifATCount-1;
      exit;
      end;
      for i:=k to ifATCount do
      begin
       aFAT[i]:=aFAT[i+1];
      end;
      ifATCount:=ifATCount-1;

     end;
```

```
{ Добавление нового файла}
procedure TFAT.AddFile(FN:string;C:chain);
var
 i:integer;
begin
 for i:=1 to ifATCount do
 begin
 if aFAT[i].FileName=FN then
 begin
 ShowMessage("ФАЙЛ "+FN+" УЖЕ СУЩЕ-
СТВУЕТ');
 exit;
 end;
 end;
 ifATCount:=ifATCount+1;
 aFAT[ifATCount].FileName:=FN;
 aFAT[ifATCount].Alloc:=C;
end;
procedure TFAT.GetFileInfo(i:integer; out
FN:string; out C:chain);
begin
 if (i<=0) or (i>ifATCount) then
 begin
 FN:=';
 Showmessage("ОШИБКА ДОСТУПА К FAT");
 exit;
 end;
 FN:=aFAT[i].FileName;
 C:= aFAT[i].Alloc;
end;

end.
```

Приложение 2. Программа моделирования файловой системы

```
unit Unit1;
interface
uses
 Windows, Messages, SysUtils, Variants,
Classes, Graphics, Controls, Forms
 Dialogs, CLASSFST, StdCtrls;

type
 TForm1 = class(TForm)
 Edit1: TEdit;
 Label1: TLabel;
 Edit2: TEdit;
 Label2: TLabel;
 Button1: TButton;
 Button2: TButton;
 ListBox1: TListBox;
 Label3: TLabel;
 Label4: TLabel;
 ListBox2: TListBox;
 Button3: TButton;
 Label5: TLabel;
 procedure FormCreate(Sender: TObject);
 procedure Button1Click(Sender: TObject);
 procedure Button2Click(Sender: TObject);
 procedure SHOWFAT;
 procedure SHOWFST;
 procedure Button3Click(Sender: TObject);
 procedure ListBox1Click(Sender: TObject);
 private
 { Private declarations }
 public
 { Public declarations }
 end;
var
 Form1: TForm1;
 FST:TFST;
 FAT:TFAT;
 C:chain;
 SP:space;
implementation
{$R *.dfm}
procedure TForm1.FormCreate(Sender:
TObject);
begin
 { Образование таблиц}
 FST:=TFST.Create;
 FAT:=TFAT.Create;
 { Начальное образование свободного простран-
ства}
 SP.addr:=1; // Адрес свободного участка
 SP.len:=16000; // Длина участка
 FST.AddFreeSpace(SP);
 SHOWFAT;
 SHOWFST;
end;

procedure TForm1.Button1Click(Sender:
TObject);
// Добавление к файлу
var
 L:integer;
begin
 L:=StrToInt(edit2.text);
 C:=FST.GetFreeSpace(L);
 if C.Len=0 then exit;
 FAT.AddFile(edit1.text, C);
 SHOWFAT;
 SHOWFST;
end;

procedure TForm1.Button2Click(Sender:
TObject);
// Исключение файла
var
 C:chain;
 SP:space;
 и: integer;
begin
 C:=FAT.DeleteFile(edit1.Text);
 for i:=1 to C.Len do
 begin
 SP:=C.ASP[i];
 FST.AddFreeSpace(SP);
 end;
 SHOWFAT;
 SHOWFST;
end;

procedure TForm1.ShowFAT;
// Отображение состояния FAT
var
 i,k:integer;
 FN, S:string;
 C:chain;
 SP:space;
```

```
 T:TObject;
begin
 listBox1.Clear;
 for i:=1 to FAT.Count do
 begin
 S:='';
 FAT.GetFileInfo(i,FN,C);
 for k:=1 to C.Len do
 begin
 s:=s+'('+inttostr(C.ASP[k].addr)+' ';
 s:=s+inttostr(C.ASP[k].len)+') ';
 end;
 S:=FN+' '+s;
 listbox1.AddItem(S,T);
 end;
end;

procedure TForm1.ShowFST;
// Отображение состояния FST
var
 i,k:integer;
 addr,len:integer;
 S:string;
 T:TObject;
begin
 listBox2.Clear;
 for i:=1 to FST.Count do
 begin
 S:='';
 FST.GetFreeSpaceInfo(i,addr,len);
 S:='('+IntToStr(addr)+' '+IntToStr(len)+') ';
 listbox2.AddItem(S,T);
 end;
                    label15.Caption:='РАЗМ:
'+IntToStr(FST.GetTotalFree);
end;


procedure TForm1.Button3Click(Sender: TObject);
// Автоматическое тестирование
var
 i:integer;
begin
 for i:=1 to 20 do
 begin
 C:=FST.GetFreeSpace(i*50);
 if C.Len=0 then exit;
 FAT.AddFile('FILE_'+IntToStr(i), C);
 end;
 SHOWFAT;
 SHOWFST;
end;

procedure TForm1.ListBox1Click(Sender: TObject);
var
 s:string;
 k:integer;
 C:chain;
begin
 k:=ListBox1.ItemIndex+1;
 FAT.GetFileInfo(к,s,C);
 edit1.Text:=s;
end;
end.
```