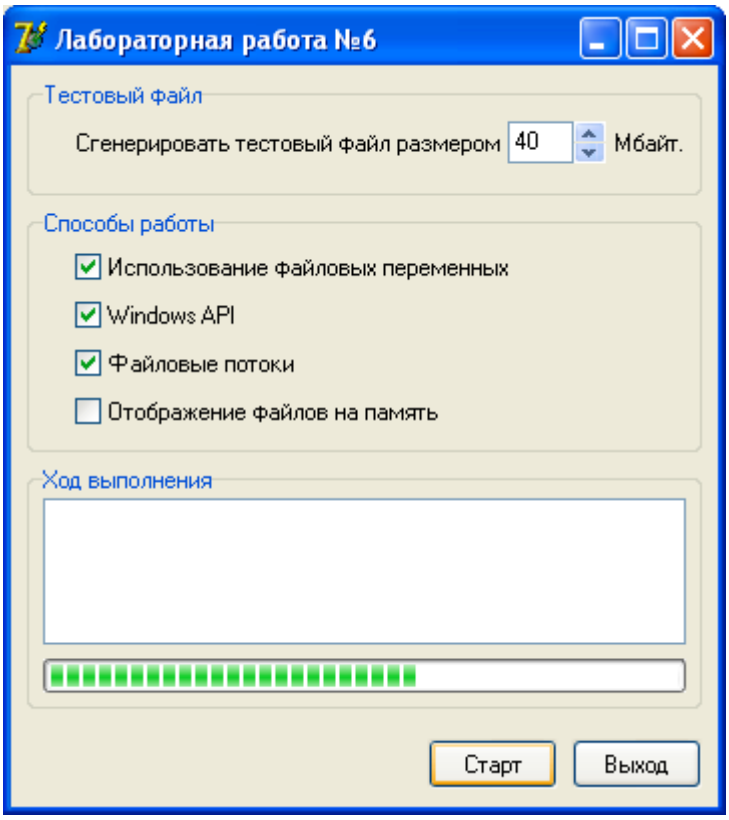
Лабораторная работа №4

Работа с файлами

Задание:

Разработать приложение в среде Delphi, реализующие работу с файлами 4-мя способами: через файловые переменные, с использованием функций Windows API, с использование файловых потоков и с помощью файлов, отображаемых на память. Для всех способов доступа использовать один и тат же файл, одинаковый размер блока данных. Определить время, затрачиваемое программой на обработку файла (результат вывести в поле мемо). Ход обработки файла отобразить с помощью элемента Progress Bar. При нажатии на кнопку старт последовательно выполняются только отмеченные пользователем тесты.

Почему полученные результаты могут меняться? Предложите способ корректного измерения результатов и реализуйте его (подсказка: посмотреть функцию GetThreadTimes).



Отображение файлов на память:

```
function TfrmFileTest.CopyByMapping:Integer;
type
  TBuf=array[0..1024-1] of Byte;
  PBuf=^TBuf;
var
  i,SrcSize:ULONG;
  hSrc,hDst:THandle;
  ptrIn,ptrOut:Pointer;
  hMapFileIn,hMapFileOut:THandle;
  BPI,BPO:PBuf;
  Start,Finish:TDateTime;
Begin
  if FileExists(DestinationFileName) then DeleteFile(DestinationFileName);
  Start:=GetTime;
```

```
  hSrc:=CreateFile(PAnsiChar(SourceFileName),GENERIC_READ,FILE_SHARE_READ
      ,nil,OPEN_EXISTING ,FILE_ATTRIBUTE_NORMAL,0);
  hDst:=CreateFile(PAnsiChar(DestinationFileName),GENERIC_WRITE or
GENERIC_READ,FILE_SHARE_WRITE ,nil,CREATE_NEW    ,FILE_ATTRIBUTE_NORMAL,0);
  SrcSize:=GetFileSize(hSrc,nil);
  prbStatus.Max:=SrcSize div 1024;
  hMapFileIn:=CreateFileMapping(hSrc,nil,PAGE_READONLY,0,0,nil);
  hMapFileOut:=CreateFileMapping(hDst,nil,PAGE_READWRITE,0 ,SrcSize,nil);
  ptrIn:=MapViewOfFile(hMapFileIn,FILE_MAP_READ,0,0,0);
  ptrOut:=MapViewOfFile(hMapFileOut,FILE_MAP_WRITE,0,0,SrcSize);

  BPI:=PBuf(ptrIn);
  BPO:=PBuf(ptrOut);
  for i:=1 to SrcSize div 1024 do
  begin
    BPO^:=BPI^;
    Inc(BPO);
    inc(BPI);
    prbStatus.Position:=i;
    Application.ProcessMessages;
  end;

  UnmapViewOfFile(ptrIn);
  UnmapViewOfFile(ptrOut);
  CloseHandle(hMapFileIn);
  CloseHandle(hMapFileOut);
  CloseHandle(hSrc);
  CloseHandle(hDst);

  Finish:=GetTime;
  CopyByMapping:=MilliSecondsBetween(Start,Finish);

  prbStatus.Position:=0;
end;
```

Использование Windows API:

```
function TfrmFileTest.CopyByWindowsAPI: integer;
var
  hSrc,hDst:THandle;
  FileSize:ULONG;
  Buf:array[0..1024-1]of Byte;
  i:Integer;
  Start,Finish:TDateTime;
begin

  if FileExists(DestinationFileName) then DeleteFile(DestinationFileName);
  Start:=GetTime;
  hSrc:=CreateFile(PAnsiChar(SourceFileName),GENERIC_READ,FILE_SHARE_READ
      ,nil,OPEN_EXISTING ,FILE_ATTRIBUTE_NORMAL,0);

hDst:=CreateFile(PAnsiChar(DestinationFileName),GENERIC_WRITE,FILE_SHARE_WRITE
      ,nil,CREATE_NEW    ,FILE_ATTRIBUTE_NORMAL,0);
  FileSize:=GetFileSize(hSrc,nil);
  prbStatus.Max:=FileSize div 1024;

  for I:=1 to FileSize div 1024 do
  begin
    // вставить операторы для чтения из файла источника и записи в файл
приемник
    prbStatus.Position:=i;
    Application.ProcessMessages;
  end;
```

```
  CloseHandle(hSrc);
  CloseHandle(hDst);

  Finish:=GetTime;
  CopyByWindowsAPI:=MilliSecondsBetween(Start,Finish);

  prbStatus.Position:=0;

end;
```

Использование файловых потоков:

```
(Реализовать самостоятельно)
```

Использование файловых переменных:

```
(Реализовать самостоятельно)
```

Генерация тестового файла:

```
procedure TfrmFileTest.CreateTestFile;
var i,FileLen:integer;
  F:TFileStream;
  Buf:array[0..1023]of byte;
begin
    FileLen:= StrToInt(txtFileSize.Text);
    prbStatus.Max:=FileLen*1024;
    F:=TFileStream.Create(SourceFileName, fmCreate);
    for i:=1 to FileLen*1024 do
    begin
      F.Write(Buf,1024);
      prbStatus.Position:=i;
      Application.ProcessMessages;
    end;
    F.Free;
end;
```

Среда Delphi предоставляет возможность выбрать один из четырех вариантов работы с файлами:

- использование традиционного набора функций работы с файлами, унаследованного от Turbo Pascal;
- использование функций ввода/вывода из Windows API;
- использование потоков (TStream и его потомки);
- использование отображаемых файлов.

Использование файловых переменных

При организации операций файлового ввода/вывода в приложении большое значение имеет, какого рода информация содержится в файле. Чаше всего это строки, но встречаются двоичные данные или структурированная информация, например массивы или записи.

Естественно, что сведения о типе хранящихся в файле данных важно изначально задать. Для этого используются специальные файловые переменные, определяющие тип файла. Они делятся на нетипизированные и типизированные.

Перед началом работы с любым файлом необходимо описать файловую переменную, соответствующую типу данных этого файла. В дальнейшем эта переменная используется при обращении к файлу.

В Delphi имеется возможность создавать нетипизированные файлы. Для их обозначения используется ключевое слово file:

```
var UntypedFile: file;
```

Такие файловые переменные используются для организации быстрого и эффективного ввода/вывода безотносительно к типу данных. При этом подразумевается, что данные читаются

или записываются в виде двоичного массива. Для этого применяются специальные процедуры блочного чтения и записи (см. ниже).

Типизированные файлы обеспечивают ввод/вывод с учетом конкретного типа данных. Для их объявления используется ключевое слово `file of`, к которому добавляется конкретный тип данных. Например, для работы с файлом, содержащим набор байтов, файловая переменная объявляется так:

```
var ByteFile: file of byte;
```

При этом можно использовать любые типы фиксированного размера, за исключением указателей. Разрешается применять структурные типы, если их составные части удовлетворяют названному выше ограничению. Например, можно создать файловую переменную для записи:

```
type Country = record
    Name: String;
    Capital: String;
    Population: LongInt;
    Square: LongInt;
end;
var CountryFile: file of Country;
```

Для работы с текстовыми файлами используется специальная файловая переменная TextFile или Text:

```
var F: TextFile;
```

Теперь рассмотрим две самые распространенные операции, выполняемые при работе с файлами. Это чтение и запись. Для их осуществления применяются специальные функции файлового ввода/вывода.

Итак, для выполнения операции чтения или записи необходимо произвести следующие действия:

1. Объявить файловую переменную необходимого типа.
2. При помощи функции AssignFile связать эту переменную с требуемым файлом.
3. Открыть файл при помощи функций Append, Reset, Rewrite.
4. Выполнить операции чтения или записи. При этом, в зависимости от сложности задачи и структуры данных, может использоваться целый ряд вспомогательных функций.
5. Закрыть файл при помощи функции CloseFile.

В качестве примера рассмотрим небольшой фрагмент исходного кода.

```
var F:TextFile; S:string;
begin
  if OpenDlg.Execute then
    AssignFiie(F, OpenDlg.FileName)
  else Exit;
  Reset(F);
  while Not EOF(F) do
  begin
    Readln(F, S) ;
    Memo.Lines.Add(S);
  end;
  CloseFile(F);
end;
```

Если в диалоге открытия файла OpenDlg был выбран файл, то его имя связывается с файловой переменной F при помощи процедуры AssignFile. В качестве имени файла рекомендуется всегда передавать полное имя файла (включая его маршрут). Как раз в таком виде возвращают результат выбора файла диалоги работы с файлами TOpenDialog, TOpenPictureDiaiog. Затем при помощи процедуры Reset этот файл открывается для чтения и записи.

В цикле выполняется чтение из файла текстовых строк и запись их в компонент TMemo. Процедура Readin осуществляет чтение текущей строки файла и переходит на следующую строку. Цикл выполняется, пока функция EOF не сообщит о достижении конца файла.

После завершения чтения файл закрывается.

Такой же исходный код можно использовать и для записи данных в файл. Необходимо только заменить процедуру чтения на процедуру записи.

Теперь остановимся подробнее на назначении используемых для файлового ввода/вывода функций.

Открытие файла может осуществляться тремя процедурами — в зависимости от типа его дальнейшего использования.

procedure Reset(var F: File [; RecSize: Word]);	открывает существующий файл для чтения и записи, текущая позиция устанавливается на первой строке файла.
procedure Append(var F: Text);	открывает файл для записи информации после его последней строки, текущая позиция устанавливается на конец файла.
procedure Rewrite(var F: File [; RecSize: Word]);	создает новый файл и открывает его, текущая позиция устанавливается в начало файла. Если файл с таким именем уже существует, то он перезаписывается. Переменная RecSize используется только при работе с нетипизированными файлами и определяет размер одной записи для операции передачи данных. Если этот параметр опущен, то по умолчанию RecSize равно 128 байт.

Чтение данных из типизированных и текстовых файлов выполняют процедуры Read и Readln. Для записи используются процедуры в файл write и writeln.

procedure Read([var F: Text;] V1 [, V2,...,Vn]);
 	
procedure Read(F, V1 [, V2,...,Vn]);
	При одном вызове процедуры можно читать данные в произвольное число переменных. Естественно, что тип переменных должен совпадать с типом файла. При чтении в очередную переменную читается ровно столько байтов из файла, сколько занимает тип данных. В следующую переменную читается столько же байтов, расположенных следом. После выполнения процедуры текущая позиция устанавливается на первом непрочитанном байте. Аналогично работают несколько процедур Read для одной переменной, выполненных подряд.

procedure Readln([var F: Text;] VI [, V2,...,Vn]);	считывает одну строку текстового файла и устанавливает текущую позицию на следующей строке. Если использовать процедуру без переменных v1. .vn, то она просто передвигает текущую позицию на очередную строку файла.
procedure Write([var F: Text;] PI [, P2,..., Pn]) ; procedure Writeln([var F: Text;] PI [, P2,...,Pn]);	Параметры P1, P2, ..., Pn могут быть одним из целых или вещественных типов, одним из строковых типов или логическим типом. Но у них есть возможность дополнительного форматирования при выводе.

Для контроля за текущей позицией в файле применяются две основные функции. Функция EOF(F) возвращает значение True, если достигнут конец файла. Функция EOLN(F) аналогично сигнализирует о достижении конца строки. Естественно, в качестве параметра в функции необходимо передавать файловую переменную.

Процедура procedure Seek(var F; N: Longint); обеспечивает смещение текущей позиции на N элементов. Размер одного элемента в байтах зависит от типа данных файла (от типизированной переменной).

Рассмотрим теперь режим блочного ввода/вывода данных между файлом и областью адресного пространства (буфером). Этот режим отличается значительной скоростью передачи данных, причем скорость пропорциональна размеру одного передаваемого блока — чем больше блок, тем больше скорость.

Для реализации этого режима необходимо использовать только нетипизированные файловые переменные. Размер блока определяется в процедуре открытия файла (Reset, Rewrite). Непосредственно для выполнения операций используются процедуры BlockRead и BlockWrite.

Процедура

procedure BlockRead(var F:File; var Buf; Count:Integer [; var AmtTransferred:Integer]); выполняет запись блока из файла в буфер. Параметр F ссылается на нетипизированную файловую переменную, связанную с нужным файлом.

Параметр Buf определяет любую переменную (число, строку, массив, структуру), в которую читаются байты из файла. Параметр Count содержит число считываемых блоков. Наконец, необязательный параметр AmtTransferred возвращает число реально считанных блоков.

При использовании блочного чтения или записи размер блока необходимо выбирать таким образом, чтобы он был кратен размеру одного значения того типа, который хранится в файле. Например, если в файле хранятся значения типа Double (8 байт), то размер блока может быть равен 8, 16, 24, 32 и т. д. Фрагмент исходного кода блочного чтения при этом выглядит следующим образом:

```
procedure TForm2.Button1Click(Sender: TObject);
 var F: File;
  DoubleArray: array [0..255] of Double;
  Transfered: Integer;
begin
  if OpenDlg.Execute then AssignFile(F, OpenDlg.FileName) else Exit;
  Reset(F, 64);
  BlockRead(F, DoubleArray, 32, Transferee!);
  CloseFile(F);
  ShowMessage('Ñ÷èòàíî '+IntToStr(Transfered)+' áëîêîâ');
 end;
end;
```

Как видно из примера, размер блока установлен в процедуре Reset и кратен размеру элемента массива DoubleArray, в который считываются данные. В переменной Transfered возвращается число считанных блоков. Если размер файла меньше заданного в процедуре BlockRead числа блоков, ошибка не возникает, а в переменной Transfered передается число реально считанных блоков.

Процедура

procedure BlockWrite(**var** f: File; **var** Buf; Count: Integer [; **var** AmtTransferred: Integer]); используется аналогично.

Таблица 1. Процедуры и функции для работы с файлом

Объявление	Описание
ChangeFileExt	Функция позволяет изменить расширение файла. При этом сам файл не переименовывается.
ChDir	Процедура изменяет текущий каталог на другой, путь к которому задается параметром
CloseFile	Вызов процедуры разрывает связь между файловой переменной и файлом на диске.
DeleteFile	Функция производит удаление файла с диска и возвращает значение False, если файл удалить не удалось или файл не существует
ExtractFileExt	Функция возвращает расширение файла
ExtractFileName	Извлекает имя и расширение файла, содержащегося в параметре FileName
ExtractFilePath	Функция возвращает полный путь к файлу
Erase	Удаляет файл, связанный с файловой переменной
FileSearch	Данная процедура производит поиск в каталогах DirList

	файла Name. Если в процессе выполнения FileSearch обнаруживается искомое имя файла, то функция возвращает в строке типа string полный путь к найденному файлу. Если файл не найден, то возвращается пустая строка
FileSetAttr	Присваивает файлу с именем FileName атрибуты Attr. Функция возвращает 0, если присвоение атрибутов прошло успешно. В противном случае возвращается код ошибки
FilePos	Возвращает текущую позицию файла. Функция используется для нетекстовых файлов. Перед вызовом FilePos файл должен быть открыт
MkDir	Процедура создает новый каталог
Rename	Процедура изменяет имя файла, связанного с файловой переменной F. Переменная NewName является строкой типа string или PChar (если включена поддержка расширенного синтаксиса)
RmDir	Процедура удаляет пустой каталог, путь к которому задается в строке S. Если указанный каталог не существует или он не пустой, то возникает сообщение об ошибке ввода/вывода

Потоки

Потоки - очень удачное средство для унификации ввода/вывода для различных носителей. Потоки представляют собой специальные объекты-наследники абстрактного класса TStream. Сам TStream "умеет" открываться, читать, писать, изменять текущее положение и закрываться. Поскольку для разных носителей эти вещи происходят по-разному, конкретные аспекты реализованы в его потомках. Наиболее часто используются потоки для работы с файлами на диске и памятью.

Базовые классы TStream и THandleStream

В основе иерархии классов потоков лежит класс Tstream. Он обеспечивает выполнение основных операций потока безотносительно к реальному носителю информации. Основными из них являются чтение и запись данных. Класс TStream порожден непосредственно от класса TObject.

Таблица 2. Свойства и методы класса TStream

Объявление	Описание
property Position: Longint;	Определяет текущую позицию в потоке
property Size: Longint;	Определяет размер потока в байтах
function CopyFrom(Source: TStream; Count: Longint): Longint;	Копирует из потока Source Count байты, начиная с текущей позиции. Возвращает число скопированных байтов
function Read(var Buffer; Count: Longint) : Longint; virtual; abstract;	Абстрактный класс, перекрываемый в наследниках. Считывает из потока Count байты в буфер Buffer. Возвращает число скопированных байтов
procedure Read3uffer (var Buffer; Count: Longint) ;	Считывает из потока Count байты в буфер Buffer. Возвращает число скопированных байтов
function Seek (Off set: Longint; Origin: Word): Longint; virtual; abstract;	Абстрактный класс, перекрываемый в наследниках. Смещает текущую позицию в реальном носителе данных на Offset байтов в зависимости от условия Origin (см. ниже)
function Write (const Buffer; Count: Longint): Longint; virtual;	Абстрактный класс, перекрываемый в наследниках. Записывает в поток Count байты из буфера Buffer. Возвращает число скопированных байтов

abstract;	
procedure WriteBuffer (const Buffer; Count: Longint);	Записывает в поток Count байты из буфера Buffer. Возвращает число скопированных байтов

Итак, в основе операций считывания и записи данных в потоке лежат методы Read и Write. Именно они вызываются для реального выполнения операции внутри методов ReadBuffer и WriteBuffer. Так как класс TStream является абстрактным, то методы Read и Write также являются абстрактными. В классах-наследниках они перекрываются, обеспечивая работу с конкретным физическим носителем данных.

Метод Seek используется для изменения текущей позиции в потоке. "Точка отсчета" позиции зависит от значения параметра Origin:

- soFromBeginning — смещение должно быть положительным и отсчитывается от начата потока;
- soFromCurrent — смещение относительно текущей позиции в потоке;
- soFromEnd — смещение должно быть отрицательным и отсчитывается от конца потока.

Класс THandleStream инкапсулирует поток, связанный с физическим носителем данных через дескриптор.

Для создания потока используется конструктор

constructor Create(AHandle: Integer);

в параметре которого передается дескриптор. Впоследствии доступ к дескриптору осуществляется через свойство:

property Handle: Integer;

Класс TFileStream

Класс TFileStream позволяет создать поток для работы с файлами. При этом поток работает с файлом без учета типа хранящихся в нем данных.

Полное имя файла задается в параметре FileName при создании потока:

constructor Create(const FileName: string; Mode: Word);

Параметр Mode определяет режим работы с файлом. Он составляется из флагов режима открытия:

- fmCreate - файл создается;
- fmOpenRead - файл открывается для чтения;
- fmopenwrite - файл открывается для записи;
- fmOpenReadWrite — файл открывается для чтения и записи.

И флагов режима совместного использования:

- fmShareExciusive - файл недоступен для открытия другими приложениями;
- fmShareDenyWrite - другие приложения могут читать данные из файла;
- fmShareDenyRead - другие приложения могут писать данные в файл;
- fmShareDenyNone - другие приложения могут производить с файлом любые операции.

Для чтения и записи из потока используются методы Read и Write, унаследованные от класса THandleStream:

```
procedure TForm2.Button1Click(Sender: TObject);
var Stream1, Stream2: TFileStream;
  IntBuf: array[0..9] of Integer;
begin
  if Not OpenDlg.Execute then Exit;
  try
    Stream1:=TFileStream.Create(OpenDlg.FileName, fmOpenRead);
    Stream1.ReadBuffer(IntBuf, SizeOf(IntBuf));
    try
      Stream2:=TFileStream.Create('TextFile.tmp', fmOpenWrite);
      Stream2.Seek(0, soFromEnd);
```

```
      Stream2.WriteBuffer(IntBuf, SizeOf(IntBuf));
    finally
      Stream2.Free;
    end;
  finally
    Stream1.Free;
  end;
end;
```

Обратите внимание, что в данном фрагменте кода функция seek используется для записи данных в конец файлового потока.

При необходимости копирования одного файла в другой целиком используется метод CopyFrom, унаследованный от класса TStream:

```
procedure TForm2.Button1Click(Sender: TObject);
 var Stream1, Stream2: TFileStream;
begin
 if Not OpenDlg.Execute then Exit;
 try
  Stream1:= TFileStream.Create(OpenDlg.FileName, fmOpenRead);
  Stream2:= TFileStream.Create('Sample.tmp', fmOpenWrite);
  Stream2.Seek(0, soFromEnd);
  Stream2.CopyFrom(Stream1, Stream1.Size);
 finally
  Streaml.Free;
  Stream2.Free;
 end;
end;
```

Обратите внимание, что в данном случае идя определения размера передаваемого потока необходимо использовать свойство stream, size, которое дает реальный объем данных, содержащихся в потоке. Функция sizeof (stream) в этом случае даст размер объекта потока, и не более того.

Класс TMemoryStream

Класс TMemoryStream обеспечивает сохранение данных в адресном пространстве. При этом методы доступа к этим данным остаются теми же, что и при работе с файловыми потоками. Это позволяет использовать адресное пространство для хранения промежуточных результатов работы приложения, а также при помощи стандартных методов осуществлять обмен данными между памятью и другими физическими носителями.

Свойство

```
property Memory: Pointer;
```

определяет область памяти, отведенную для хранения данных потока. Изменение размера отведенной памяти осуществляется методом

```
procedure SetSize(NewSize: Longint); override;
```

Для очистки памяти потока используется метод

```
procedure Clear;
```

Чтение/запись данных в память выполняется привычными методами Read и Write.

Также запись данных в память может осуществляться методами:

```
procedure LoadFromFile(const FileName: string); — из файла;
procedure LoadFromStream(Stream: TStream) ; — из другого потока.
```

Дополнительно можно использовать методы записи данных в файл или поток:

```
procedure SaveToFile(const FileName: string);
procedure SaveToStream(Stream: TStream);
```

Использование отображаемых файлов

Создание и использование объектов файлового отображения осуществляется посредством функций Windows API. Этих функций три: CreateFileMapping, MapViewOfFile, UnMapViewOfFile.

Отображаемый файл создается операционной системой при вызове функции CreateFileMapping. Этот объект поддерживает соответствие между содержимым файла и адресным пространством процесса, использующего этот файл. Функция CreateFileMapping имеет шесть параметров:

```
function CreateFileMapping(hFile: THandle; lpFileMappingAttributes: PSecurityAttributes;
flProtect, dwMaximumSizeHigh, dwMaximumSizeLow: DWORD; lpName: PChar): THandle;
```

Первый параметр имеет тип THandle. Он должен соответствовать дескриптору уже открытого при помощи функции CreateFile файла. Второй параметр — указатель на запись типа TSecurityAttributes. При отсутствии требований к защите данных в Windows NT значение этого параметра всегда равно nil. Третий параметр имеет тип DWORD. Он определяет атрибут защиты. Если при помощи отображаемого файла вы планируете совместное использование данных, третьему параметру следует присвоить значение PAGE_READWRITE.

Четвертый и пятый параметры также имеют тип DWORD. Когда выполняется функция CreateFileMapping, значение типа DWORD четвертого параметра сдвигается влево на четыре байта и затем объединяется со значением пятого параметра посредством операции and. Проще говоря, значения объединяются в одно 64-разрядное число, равное объему памяти, выделяемой объекту файлового отображения из страничного файла операционной системы. Поскольку вы вряд ли попытаетесь осуществить выделение более чем 4 Гбайт данных, то значение четвертого параметра всегда должно быть равно нулю. Используемый затем пятый параметр должен показывать, сколько памяти в байтах необходимо зарезервировать в качестве совместной. Если вы хотите отобразить весь файл, четвертый и пятый параметры должны быть равны нулю.

Шестой параметр имеет тип PChar и представляет собой имя объекта файлового отображения.

Функция CreateFileMapping возвращает значение типа THandle. В случае успешного завершения возвращаемое функцией значение представляет собой дескриптор созданного объекта файлового отображения. В случае возникновения какой-либо ошибки возвращаемое значение будет равно 0.

Следующая задача - спроецировать данные файла в адресное пространство нашего процесса. Этой цели служит функция MapviewOfFile. Функция MapViewOfFile имеет пять параметров:

```
function MapViewOfFile(hFileMappingObject: THandle; dwDesiredAccess: DWORD;
dwFileOffsetHigh, dwFileOffsetLow, dwNumberOfBytesToMap: DWORD): Pointer;
```

Первый параметр имеет тип THandle. Его значением должен быть дескриптор созданного объекта файлового отображения - тот, который возвращает функция createFileMapping. Второй параметр определяет режим доступа к файлу: FILE_MAP_WRITE, FILE_MAP_READ или FILE_MAP_ALL_ACCESS.

Третий и четвертый параметры также имеют тип DWORD. Это - смещение отображаемого участка относительно начала файла в байтах. В нашем случае эти параметры должны быть установлены в нуль, поскольку значение, которое мы даем пятому (последнему) параметру функции MapViewOfFile, также равно нулю.

Пятый (и последний) параметр функции MapViewOfFile, как и предыдущие параметры, имеет тип DWORD. Он используется для определения (в байтах) количества данных объекта файлового отображения, которые надо отобразить в процесс (сделать доступными для вас). Для достижения наших целей это значение должно быть установлено в нуль, что означает автоматическое отображение в процессе всех данных, выделенных перед этим функцией CreateFileMapping.

Значение, возвращаемое функцией MapViewOfFile, имеет тип "указатель".

Если функция отработала успешно, то она вернет начальный адрес данных объекта файлового отображения.

Следующий фрагмент кода демонстрирует вызов функции MapViewOfFile:

```
procedure TForm2.Button1Click(Sender: TObject);
var hMappedFile: THandle; pSharedBuf: PChar;
begin
```

```
      hMappedFile := CreateFileMapping(fHandle, nil, PAGE_READWRITE, 0, 0,
'SharedBlock');
      if (hMappedFile = 0) then
        ShowMessage('Mapping error!')
      else
        begin
          pSharedBuf:= MapViewOfFile(hMappedFile, FILE_MAP_ALL_ACCESS, 0, 0, 0) ;
            if (pSharedBuf = nil) then
              ShowMessage ('MapView error');
        end;
    end;
    end;
```

После того как получен указатель pSharedBuf, вы можете работать со своим файлом как с обычной областью памяти, не заботясь о вводе, выводе, позиционировании и т. п. Все эти проблемы берет на себя файловая система. Последние две функции, имеющие отношение к объекту файлового отображения, называются UnMapViewOfFile и CloseHandle. Функция UnMapViewOfFile делает то, что подразумевает ее название. Она прекращает отображение в адресное пространство процесса того файла, который перед этим был отображен. При помощи функции MapViewOfFile. Функция CloseHandle закрывает дескриптор объекта файлового отображения, возвращаемый функцией CreateFileMapping. Функция UnMapViewOfFile должна вызываться перед функцией CloseHandle.