ОБРАЩЕНИЕ К ФУНКЦИЯМ ОПЕРАЦИОННОЙ СИСТЕМЫ

Большинство библиотечных процедур и функций является своеобразным интерфейсом между языковыми средствами Турбо-Паскаля и функциями операционной системы.

Следует учесть, что единственным механизмом обращения к функциям операционной системы является <u>инициация программного прерывания</u>.

ПРЕРЫВАНИЕ – ЭТО ОСОБОЕ СОСТОЯНИЕ ВЫЧИСЛИТЕЛЬНОГО ПРОЦЕССА.

- ***В момент прерывания нарушается нормальный порядок выполнения команд программы, и управление передается специальной процедуре, которая входит в состав MS DOS и называется процедурой обработки прерывания.***
- ***Каждое прерывание характеризуется в рамках MS DOS порядковым номером и связано со своей процедурой обработки.***
- ***В архитектуре центрального процессора ЭВМ предусмотрены прерывания двух типов: аппаратные и программные.***
- ***Аппаратные прерывания создаются схемами контроля и управления ЭВМ и сигнализируют операционной системе о переходе какого-либо устройства в новое состояние или о возникновении неисправности.***

Прерывания MS-DOS.

Всего в MS DOS имеется около 40 программных прерываний.

- каждое из которых может активизировать одну или несколько функций MS DOS.

Одно из прерываний - с номером 33 ($21) обеспечивает доступ к 85 функциям.

- а всего в MS DOS имеется более 200 разнообразных функций.

Модуль Dos

- *Модуль Dos содержит ряд подпрограмм для работы с файлами и доступа к средствам операционной системы.*
- *Ни одна из программ модуля Dos не определена в стандартном Паскале, поэтому они помещены в отдельный модуль.*

Процедуры и функции модуля Dos.

- **Модуль DOS**
  - Функции управления операционной средой.
  - Процедуры управления процессами.
  - Процедуры обслуживания прерываний.
  - Процедуры работы с датой и временем.
  - Процедуры и функции проверки состояния диска.
  - Процедуры и функции работы с файлами.
  - Процедуры и функции различного назначения.

Функции управления операционной средой.

- *Программа, написанная на языке Турбо Паскаль 7.0, имеет возможность получать от MS-DOS информацию об окружении (environment).*

- *Каждое описание в области окружения представляет собой строку вида:* **Имя = Значение**.
  - **Имя** — это имя переменной, **Значение** — текстовая строка.

- *Переменные окружения предназначены для хранения определенной системной информации, которая необходима различным прикладным программам, использующим ее в своих целях.*

В Турбо Паскале имеется три функции для работы с переменными окружения:

FUNCTION EnvCount : INTEGER;

FUNCTION EnvStr(index:INTEGER) : STRING;

FUNCTION GetEnv(EnWar: STRING) : STRING;

Функции управления операционной средой.

Функция **EnvCount**	• не имеет параметров и возвращает общее число переменных окружения.
Функция **EnvStr**	• имеет один параметр и возвращает строку, содержащую имя и значение переменной, которая соответствует значению указанного индекса. • *индекс самой первой переменной — 1.* • *если указанный индекс меньше 1 или больше* **EnvCount,** *функция* **EnvStr** *возвращает пустую строку.*

Пример.

```
{Использование функций EnvCount и EnvStr}
USES Dos;
VAR
    I:    INTEGER;
BEGIN
    FOR I    := 1 TO EnvCount DO WriteLn(i,'    ',EnvStr(I));
    ReadLn;
END.
```

Результат работы данной программы может выглядеть примерно следующим образом:

```
COMSPEC=C:\COMMAND.COM
PATH=C:\NU;C:\;D:\TOOLS;C:\NC;C:\WINDOWS;
SYMANTEC=C:\SYMANTEC
NU=C:\NU
```

Функции управления операционной средой.

Функция
GetEnv

- позволяет по имени переменной окружения получить ее значение.
- *при вызове функции указывается один параметр — имя переменной окружения.*

Например, используя оператор
WriteLn(GetEnv('PATH'))
можно получить на экране следующее:

C:\;C:\NC;\C:\ WINDOWS;D:\TOOLS;

Процедуры управления процессами.

Используя специальные средства языка Турбо Паскаль, можно организовать вызов из программы любой другой программы, которую называют программой-потомком.

Важно помнить, что для того, чтобы программа-потомок успешно загрузилась в память и начала выполняться, необходимо обеспечить выделение ей требуемого объема памяти.

- *Так как программа, которая выполняется в данный момент, по умолчанию захватывает всю свободную динамическую память (кучу) системы, то для загрузки программы-потомка просто нет места.*
- *Для того чтобы выделить для программы-потомка достаточное количество свободного места, необходимо в начале программы указать с помощью директивы компилятора **$M** минимальный размер выделяемой программе памяти, а если в программе не используется динамическая память, то вовсе не выделять ее.*

- ***Директива {$M 16384,0,655360} устанавливается для программ по умолчанию.***
  - *Здесь максимальный размер кучи равен всей доступной памяти.*
  - *Поэтому вызов программы-потомка невозможен.*
- ***Но можно задать и такую директиву {$M 1024,0,0}.***
  - *В этом случае динамическая память вообще не выделяется.*
  - *Теперь можно вызывать программу-потомка.*

Процедуры управления процессами.

Вызов программы-потомка производится с помощью процедуры:

Exec(Path,CmdLine : STRING);

Path — это полный или сокращенный путь к исполняемому файлу и его имя

CmdLine — параметр, в котором можно передать вызываемой программе командную строку.

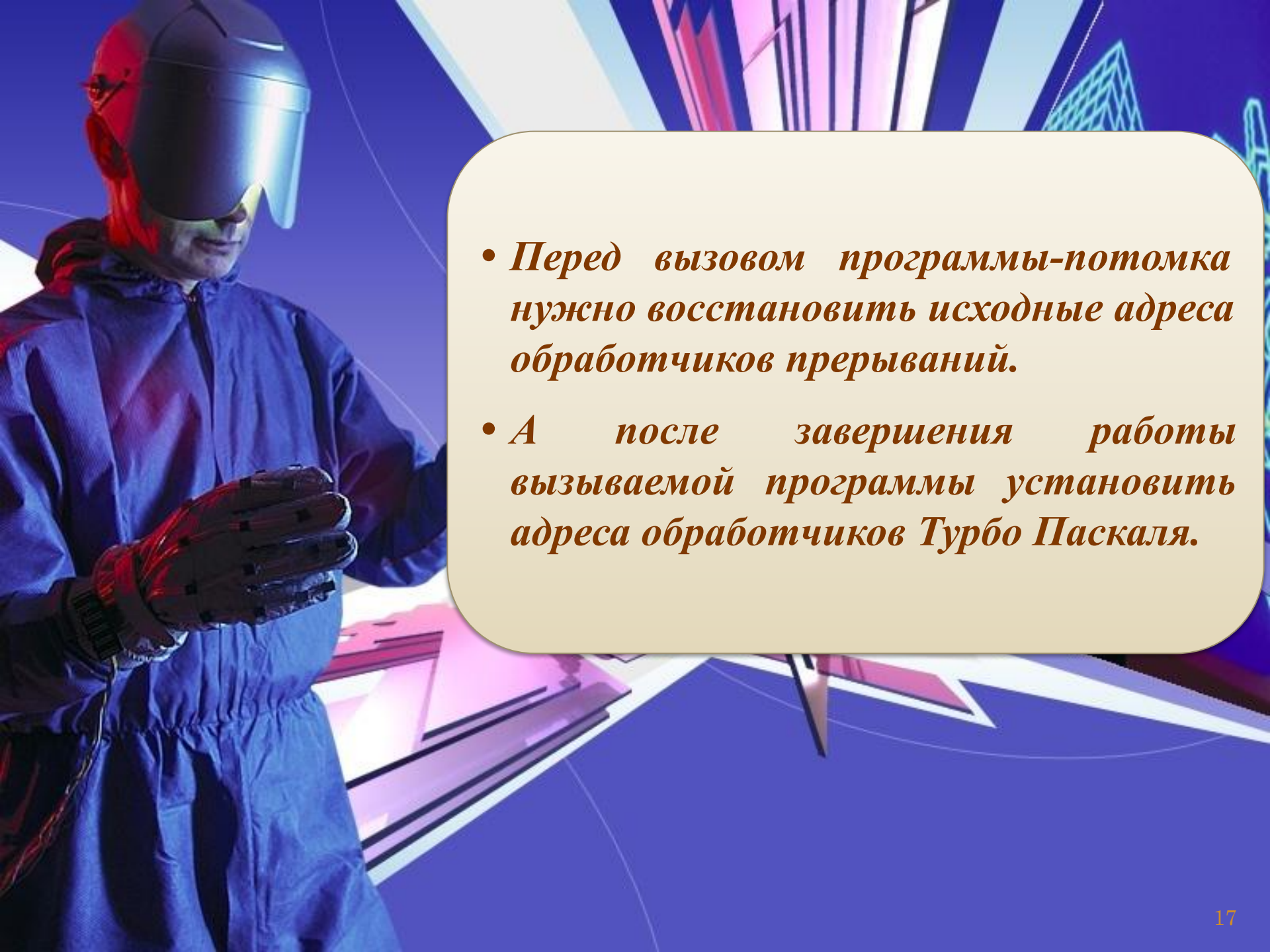
Если вызов программы не произошел, то это никак не отразится на работе вызвавшей программы, т.е. она будет вести себя одинаково как при успешном, так и при неудачном вызове.

- ***Для того чтобы определить, как прошел вызов, используется системная переменная* DosError.**
- ***Сразу после вызова процедуры* Exec *необходимо проверить значение переменной* DosError.**
  - *Если она равна 0, то это значит, что вызов прошел успешно.*
  - *Ненулевое значение свидетельствует об ошибке.*

Процедуры управления процессами.

Значения переменной **DosError** *соответствуют кодам, вырабатываемым операционной системой, и могут быть следующими :*

Код ошибки	Значение
0	Нормальное завершение
2	Файл не найден
3	Путь не найден
4	Слишком много открытых файлов
5	Доступ закрыт
6	Нарушена информация в полях файла или в системных областях
8	Недостаточно памяти
10	Несовместимые параметры окружения
11	Нераспознаваемый формат диска
18	Нет больше файлов (при работе процедуры FindNext)



- *Перед вызовом программы-потомка нужно восстановить исходные адреса обработчиков прерываний.*
- *А после завершения работы вызываемой программы установить адреса обработчиков Турбо Паскаля.*

Процедуры управления процессами.

Специфические особенности исполнения Турбо-Паскалевых программ требуют изменения стандартных значений некоторых векторов прерываний.

- *К ним относятся векторы со следующими шестнадцатеричными номерами:*

$00, $02, $18, $23, $24, $34, $35, $36, $37, $38, $39, $3A, $3B, 3C, $3D, $3E, $3F, $75.

Начальные значения этих векторов сохраняются в восемнадцати переменных с именами **SaveIntXX** *,*
где **XX** *- шестнадцатеричный номер прерывания.*

- ***Непосредственно перед запуском внешней программы и сразу после возврата из нее рекомендуется вызвать библиотечную процедуру не имеющая параметров* SwapVectors, *которая обменивает содержимое векторов прерывания и перечисленных переменных.***

Пример использования процедур Exec и SwapVectors .

```
{Использование процедур Exec и SwapVectors}
{$M 1024,0,0}   {освобождение памяти для потомка}
USES Dos;
VAR
      ProgName, CmdLine: STRING;
BEGIN
      Write('Введите путь и имя исполняемого файла ');
      ReadLn(ProgName);
      Write('Введите командную строку ');
      ReadLn(CmdLine);
      SwapVectors;                  {переустановка векторов}
      Exec(ProgName, CmdLine);
      SwapVectors;                  {восстановление векторов}
      IF DosError <> 0                    {проверка на ошибку запуска}
            THEN
      WriteLn('Ошибка DOS ', DosError)
      ELSE
      WriteLn('Выполнено успешно, код возврата ',Lo(DosExitCode));
END.
```

Процедуры управления процессами.

Функция **DosExitCode : WORD;**

*Эта функция возвращает значение типа **WORD**, в младшем байте которого содержится код возврата, переданный через процедуру завершения, а в старшем — признак того, как завершилась программа (см. табл.).*

Код	Значение
0	Нормальное завершение
1	Прервана нажатием [Ctrl+Break]
2	Прервана из-за ошибки устройства
3	Завершена процедурой Keep

Процедуры управления процессами.

В заключение рассмотрим еще один весьма полезный пример.
Программа воспринимает с клавиатуры любую команду DOS, затем вызывает командный процессор COMMAND.COM операционной системы и передает ему эту команду.

```
PROGRAM ExecDemo;
{$M 1024, 0, 0}
Uses DOS;
var
      st: string f79];
BEGIN
      write ( ‘Введите команду DOS:’  );
      read1n (st);
      if st <> " then
            begin
                  st := '/C '+st,;
                  SwapVectors;
                  Exec (GetEnv ('COMSPEC'), st);
                  SwapVectors
            end;
END.
```

- ***Обратите внимание: для указания файла COMMAND.COM и пути к нему используется обращение к библиотечной функции GetEnv, с помощью которой можно получить параметры настройки операционной системы.***
- ***В частности, параметр COMSPEC определяет спецификацию файла, содержащего командный процессор.***

Процедуры обслуживания прерываний.

- ***Несмотря на то, что Турбо Паскаль имеет большой набор инструментальных средств для использования возможностей MS-DOS, во многих случаях возникает необходимость прямого обращения к функциям MS-DOS для использования некоторых специфических средств операционной системы, в частности, для организации обмена информацией в оперативной памяти.***

- ***Рассмотрим более подробно принцип такого обмена.***

Для более полного использования возможностей MS-DOS в модуле Dos имеются две процедуры:

Intr и MsDos.

Процедуры обслуживания прерываний.

Процедура

Intr(IntNo :BYTE; VAR Regs :REGISTERS)

выполняет заданное программное прерывание.

IntNo — *номер программного прерывания*;

REGISTERS — *является типом записи, определенным в модуле Dos следующим образом:*

```
Type Registers = Record
      Case Integer Of
   0: (AX, BX, CX, DX, BP, SI, DI, DS, ES, Flags : Word);
   1: (AL, AH, BL, BH, CL, CH, DL, DH : Byte);
     End;
```

Поля записи соответствуют регистрам процессора.

Процесс выполнения процедуры Intr.

Таким образом, можно получить доступ к результатам работы процедуры прерывания

Процедуры обслуживания прерываний.

Для доступа к отдельным битам регистра флагов процессора используются следующие константы масок

ИМЯ	Значение
FCarry	**$0001**
FParity	**$0004**
FAuxiliary	**$0010**
FZero	**$0040**
FSign	**$0080**
FOverflow	**$0800**

Пример использования процедуры Intr.

прерывание с номером 18 ($12) возвращает в регистре AX объем оперативной памяти ЭВМ

```
PROGRAM IntrDem;    {Использование процедуры Intr}
Uses DOS;
Var
    r : registers;
BEGIN
    Intr ($12, r);
    writeln('Объем памяти = ', r.AX, ' Кбайт')
END.
```

Прерывание номер 33 ($21) стоит особняком и называется прерывание DOS.

Оно дает доступ к большому количеству функций различных функций DOS

(этим прерыванием вызывается 85 функций).

Процедуры обслуживания прерываний.

Процедура

MsDos(VAR Regs:REGISTERS)

выполняет вызов функции DOS.

*Результат обращения к процедуре **MsDos** будет тот же, что и при обращении к процедуре **Intr** с номером прерывания $21.*

Пример использования процедуры MsDos.

```
PROGRAM MsDosDemo;
Uses DOS;
Var
    R: registers;
BEGIN
    r.AH: = $30;
    MsDos (r);
    WriteLN ('Версия операционной системы :’,r.AL,'.',r.AH)
END.
```

Процедуры обслуживания прерываний.

Довольно распространенной является ситуация, когда в программе необходимо определить собственные алгоритмы реакции на прерывания операционной системы.

- При этом обычно нужно либо отменить стандартные реакции, либо сделать так, чтобы выполнялась и стандартная реакция, и своя собственная.

Для этого Турбо Паскаль позволяет создавать процедуры специального вида — обработчики прерываний

- хотя обычно такие процедуры пишутся на языке ассемблера

Обработчики прерываний.

Заголовок таких процедур должен иметь стандартный вид:

```
PROCEDURE имя_процедуры  (Flags,CS,IP,AX,BX,CX,DX,SI,DI,DS,ES,BP:WORD);
                                                                INTERRUPT;
    begin
        . . . . . .
    end;
```

При активизации процедуры обработки прерываний в стек автоматически сохраняется содержимое всех регистров,

а процедуре передаются копии содержимого тех регистров, которые указаны к качестве формальных параметров.

Поэтому в процедуре их можно изменять и использовать.

Порядок следования параметров должен точно соответствовать приведенному выше,

но допускается указывать только необходимые из них.

Процедуры обслуживания прерываний.

При написании процедур обработки прерываний существенными являются следующие обстоятельства:

Процедура обработки прерывания не должна искажать работу прерванной программы.

- *Для этого необходимо сначала сохранить регистры центрального процессора, а перед выходом из процедуры — восстановить их.*

Процедура должна строиться по принципу реентерабельности (повторной входимости)

- *Ее работа может быть прервана в любой момент другими прерываниями и DOS может обратиться к соответствующей функции до завершения обработки предыдущего прерывания*

В самой процедуре обработки прерывания не рекомендуется обращение к другим функциям DOS.

- *Так как некоторые из них, в том числе все функции ввода-вывода, нереентерабельны.*

Процедуры обслуживания прерываний.

Процедура

SetIntVec(IntNo :BYTE; Vector :POINTER);

используется для установки нового адреса обработчика прерывания .

IntNo — *номер прерывания, которое необходимо переопределить*;

Vector — *адрес новой процедуры обработки прерывания.*

Процедуры обслуживания прерываний.

Для того чтобы изменить реакцию системы на прерывание и определить пользовательский обработчик, **SetIntVec** просто изменяет запись в системной области DOS, которая называется областью векторов прерываний.

- *но при этом старый вектор, т.е. адрес старой процедуры обработки прерываний, не сохраняется.*

Адрес старой процедуры обработки прерываний может быть необходим, чтобы через некоторое время восстановить старую реакцию на прерывание.

- *или чтобы при вызове прерывания сначала срабатывал новый обработчик, который затем передавал бы управление оригинальному (старому) обработчику.*

Процедуры обслуживания прерываний.

Этот принцип использует большинство резидентных программ.

- *Таким образом, работоспособность системы не нарушается, когда несколько программ перехватывают одно прерывание.*

Резидентная программа сначала отрабатывает сама, затем передает управление другой программе и так далее, а в конце управление передается операционной системе. .

Поэтому очень важно сохранить адрес старого обработчика прерывания.

Для этого можно использовать процедуру

GetlntVec(IntNo :BYTE; VAR Vector :POINTER);

которая присваивает параметру-переменной **Vector** *адрес текущего обработчика прерывания, номер которого задан в параметре* **IntNo.**

Процедуры обслуживания прерываний.

Следующий пример выводит на экран содержимое всех ненулевых векторов прерываний.

```
Uses DOS;
var
    i : byte;
    p : pointer;
BEGIN
    for i := 0 to 255 do
    begin
        GetIntVec (i, p);
        if (Seg (p^) <> 0) or (Ofs (p^) <> 0) then
        write1n (' N =', i:3, ' Seg =', Seg (p^):5, Ofs =', Ofs (p^):5)
    end
END.
```

Программы, резидентные в памяти.

- *Существует еще одна процедура, относящаяся к процедурам управления процессами - это процедура* **Keep**.

- **Keep(ExitCode: WORD);**

- *Вызов этой процедуры приводит к завершению работы программы, но при этом оставляет ее в памяти.*

- *Такие программы носят название программ, резидентных в памяти (**Terminate and Stay Resident, TSR**), или просто резидентных программ.*

- *На этом принципе построены драйверы устройств и различные сервисные программы, например резидентные словари или калькуляторы.*

Программы, резидентные в памяти.

После того как программа осталась резидентной в памяти, она передает управление командному процессору.

- *а сама как бы «замирает».*

Для того чтобы активизироваться в нужный момент, программа обязательно должна перехватывать какое-нибудь прерывание

- *например, прерывание от клавиатуры.*

После нажатия нужной комбинации клавиш программа перехватывает управление и выполняет свою задачу.

Пример резидентной программы.

Переопределим прерывание от клавиатуры.

```
USES Dos;
CONST
    Segment = $B800;
    Col = 39;
    Row = 12;
    Attr = $0C;
VAR
    Offset      : WORD;
    Vector      : PROCEDURE;
    Activ       : BOOLEAN;
{$F+}           {должен использоваться дальний тип вызова}
```

Пример резидентной программы.

```
PROCEDURE Int09;   INTERRUPT;       {процедура обработки прерывания 09}
VAR
      Time62,  Time60, Portfo     : BYTE;
BEGIN
Time60:=Port[$60);                  {чтение регистров портов}
 Portfo:=Port[$61];
  Time62:=Port[$62];
Port[$61]:=Portfo OR 128;           {запись новых значений в регистры порта}
Port[$61]:=Portfo AND  127;
IF   ((Mem[$40:$17]=Mem[$40:$17] OR 12) AND (Time60=19)) OR (activ) THEN
      BEGIN
      Activ:=TRUE;
      IF Time60=l  THEN Activ:=FALSE;
      Mem[Segment:Offset+1]:=Attr;        {прямой вывод в видеопамять}
      Mem[Segment:Off3et+3]:=Attr;
      Mem[Segment:Offset+5]:=Attr;
      Mem[Segment:Offset]:=(Time60 DIV 100J+48;
      Mem[Segment:0ffset+2]:=((Time60 MOD 100) DIV 10)+48;
      Mem[Segment:Offset+4]:=((Time60 MOD 100) MOD 10)+48;
      END;
INLINE   ($9C);           {вызов новой процедуры обработки прерывания}
Vector;
End;
```

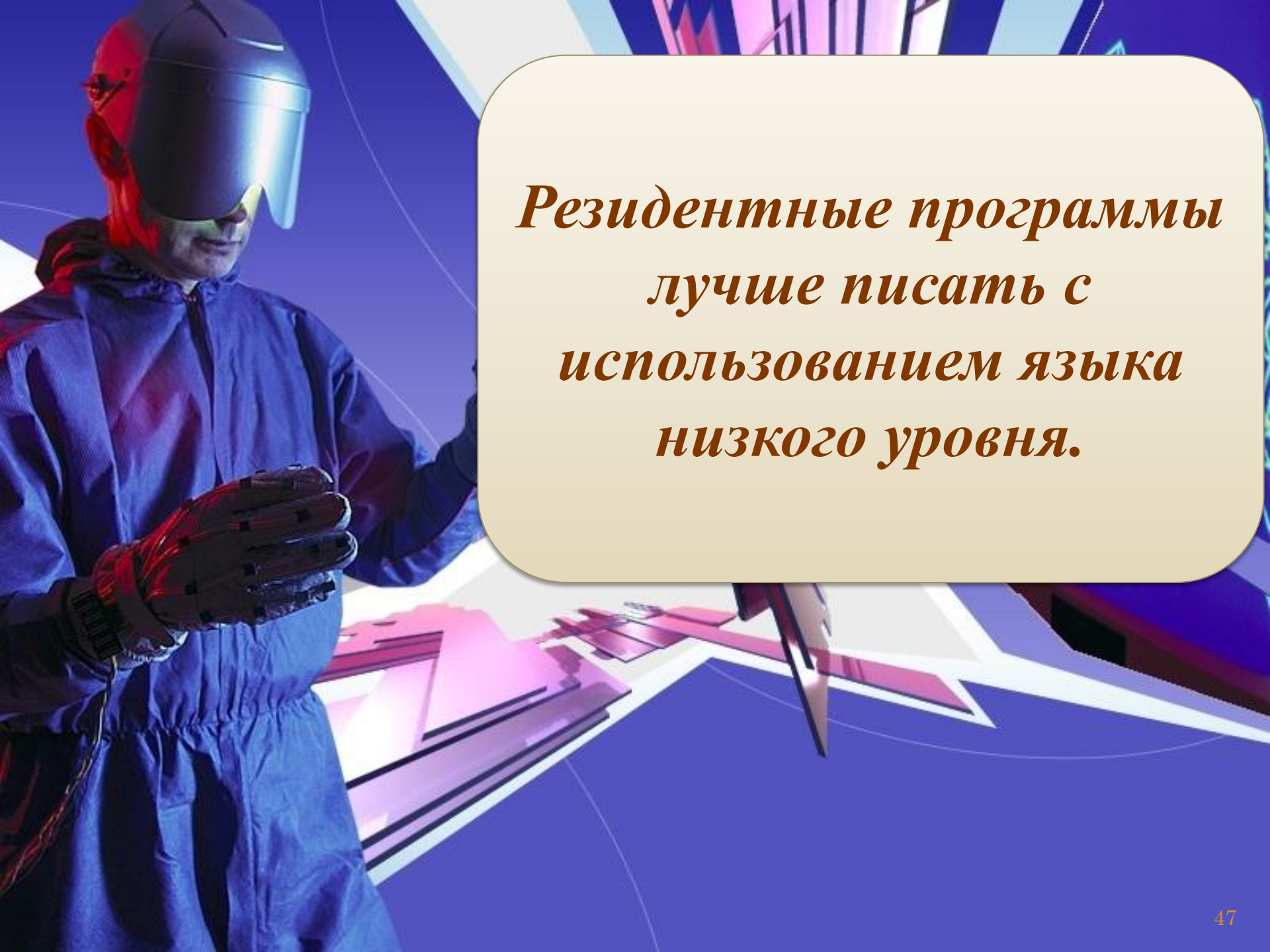
Пример резидентной программы.

```
{$F-}
BEGIN
    Activ:=FALSE;
    Offset:=(Row-l)*160+(Col-l) SHL 1;
    GetIntVec($09,@Vector) ;        {сохранение старого вектора}
    SetIntVec($09,Addr(Int09));     {установка нового вектора}
    Keep(0);                        {выйти и остаться резидентной}
END.
```

Данная программа остается резидентной в памяти и выводит на экран значение скэн-кодов нажатых клавиш.

Процедура Int09 перехватывает прерывание клавиатуры и активизируется после нажатия комбинации клавиш [Ctrl+Alt+R], а затем используя прямой доступ к видеопамяти, выводит скэн-коды

После нажатия клавиши [Esc] вывод кодов прекращается.



Резидентные программы лучше писать с использованием языка низкого уровня.

Процедуры работы с датой и временем.

- *Модуль Dos предоставляет программисту возможность доступа к системным часам и календарю, а также дает возможность изменять дату и время создания файла.*

- *Для доступа к системной дате используют процедуры* **GetDate** *и* **SetDate**.

- *Для доступа к системным часам используются процедуры* **SetTime** *и* **GetTime**.

Процедуры работы с датой и временем.

Процедура **GetDate**

- **GetDate(var year,month,day,day_of_week : WORD);**
- *возвращает текущую дату, установленную в системе.*

Процедура **SetDate**

- **SetDate(year, month,day : WORD);**
- *устанавливает текущую дату в операционной системе.*

Процедуры работы с датой и временем.

Значения параметров

year	• может принимать значения от **1980** до **2099**
month	• значения от **1** до **12**
day	• значения от **1** до **31**
day_of_week	• значения от **0** до **6**, • где **0** означает воскресенье.

Пример программы, демонстрирующий использование процедур GetDate и SetDate.

```
USES  Dos;
CONST
      days : ARRAY[0..6] OF STRING[11] =('Воскресенье','Понедельник','Вторник',
      'Среда', 'Четверг' , 'Пятница', 'Суббота');
VAR
      year,month,day,day_of_week : WORD;

PROCEDURE Get_Date;
BEGIN
      GetDate(year,month, day,day_of_week);
      WriteLn('Сегодня ',days[day_of_week], ‘, ',day,'/',month,’/', year) ;
END;

PROCEDURE Set_Date;
BEGIN
      Write('Введите число,  месяц и год,   используя пробел в качестве разделителя’);
      ReadLn(day,month,year); SetDate(year,month,day);
END;
```

Пример программы, демонстрирующий использование процедур GetDate и SetDate.

```
BEGIN
    Get_Date;     {получение  текущей даты}
    Set_Date;     {установка  новой  даты}
END.
```

Данная программа представляет собой аналог команды **DOS DATE.**

Если параметры в процедуре **SetDate** *будут заданы некорректно, то вызов процедуры будет проигнорирован и дата не изменится.*

Процедуры работы с датой и временем.

Процедура **GetTime**

- **GetTime (var hour, minute, second, hund: WORD);**
- *возвращает текущее время, установленное в системе.*

Процедура **SetTime**

- **SetTime (hour, minute, second, hund: WORD);**
- *устанавливает текущее время в операционной системе.*

Процедуры работы с датой и временем.

Значения параметров

hour	• может принимать значения от **0** до **23**
minute	• значения от **0** до **59**
second	• значения от **0** до **59**
hund	• значения от **0** до **99**, • сотые доли секунды

Пример программы, демонстрирующий использование процедур GetTime и SetTime.

```
USES  Dos;
VAR
      hour,minute,second,hund   :   WORD;
PROCEDURE  Get_Time;
      BEGIN
      GetTime(hour,   minute,second,hund);
      WriteLn('Текущее время',hour,':',minute,':',second,':',hund’);
      END;
PROCEDURE Set_Time;
      BEGIN
            Write('Введите часы, минуты и секунды');
            ReadLn(hour, minute, second);
            SetTime(hour,minute,second,0);
      END;
BEGIN
            Get_Time;   {получение текущего времени}
            Set_Time;   {установка нового времени}
END.
```

Процедуры работы с датой и временем.

Процедура **GetFTime**

- **GetFTime (var F; var Time:Longlnt);**
- *возвращает время и дату создания*

Переменная **F** *должна быть файловой переменной.*

Время и дата возвращаются в переменной **Time** *в упакованном формате в виде двойного слова (32 бита)*

- *для их распаковки используется процедура* **UnpackTime**

Процедуры работы с датой и временем.

Процедура **UnpackTime**

- **UnpackTime(Time : Longlnt; VAR DT: DateTime);**
- *преобразовывает значение, переданное в параметре* **Time**, *в запись* **DT** *типа* **DateTime** *(дата и время).*

Тип **DateTime** *описан в модуле* **Dos** *следующим образом:*

```
TYPE
    DataTime = RECORD
    Year, Month, Day, Hour, Min, Sec  :  WORD;
    END;
```

Процедуры работы с датой и временем.

Процедура **SetFTime**

- **SetFTime(VAR F; Time: LongInt);**
- *устанавливает время и дату создания файла.*

Процедура **PackTime**

- **PackTime(VAR T: DateTime; VAR Time: LongInt);**
- *преобразовывает запись типа* **DateTime** *в двойное слово, используемое процедурой* **SetFTime**.

- ***О том, как прошла операция установки даты создания файла, можно узнать из переменной*** **DosError.**
  - ***В случае успешного завершения переменная*** **DosError** ***будет содержать 0.***
  - ***В случае неудачи код ошибки может быть только 6 (разрушен заголовок файла).***

- *Следует запомнить, что файл, для которого необходимо изменить дату создания, должен быть открыт для чтения.*
  - *Если файл открыт процедурой* **Rewrite**, *то после того, как дата его создания изменена и в конце работы файл будет закрыт процедурой* **Close**, *дата создания автоматически обновится, т. е. установится текущая системная дата.*
- *Поэтому, если происходит запись в файл, а затем требуется изменить дату его создания, можно просто выполнить перед процедурой* **SetFTime** *команду* **Reset**, *т.е. открыть файл для чтения.*

Пример программы, использующей процедуры работы с временем создания файла.

```
USES  Dos;
VAR
      ff     : TEXT;
      hour, minute, second, hund      : WORD;        {переменные для GetTime}
      FileTime        : LongInt;      {переменная для GetFTime и SetFTime}
      dt     : DateTime;   {переменная  для   PackTime  и  UnpackTime}

FUNCTION FZero(w : WORD) :STRING;
VAR
      S   :   STRING;
BEGIN
      Str(w:0, S);
      IF Length(S)   = 1 THEN S:= '0'   + S;
      FZero:= S ;
END;
```

Пример программы, использующей процедуры работы с временем создания файла.

```
BEGIN
      Assign(ff, Paramstr(1));
      GetTime(hour, minute,second,hund);
      WriteLn('Текущее время',FZero(hour),':',FZero(minute),':‘,FZero(second));
      Rewrite(ff);          {создание нового файла}
      GetFTime(ff,filetime);       {получение времени создания}
      UnpackTime(filetime, dt);
      WITH dt DO
      BEGIN
            WriteLn('Файл создан ', FZero(hour),':',FZero(min),':',FZero(sec));
              hour:= 13;
               min:= 0;
                 sec:= 0;
             year:=2009;
               month:=4;
                 day:=2;
            PackTime(dt, FileTime) ;
            WriteLn('Устанавливаем время создания 13:00 2 апреля 2009 г.');
            Reset (ff);      {Открытие файла для чтения}
            SetFTime(ff,FileTime);
      END;
      Close(ff);              {Закрытие файла}
END.
```

Процедуры и функции проверки состояния диска.

- *Модуль Dos содержит ряд функций и процедур для работы с диском.*

- *С их помощью можно осуществлять контроль за наличием свободного пространства на диске, контроль правильности записи на диск, а также определять общий объем памяти диска.*

Процедуры и функции проверки состояния диска.

Функция **DiskFree**

- **DiskFree(Drive: BYTE) :Longlnt;**
- *возвращает количество свободных байтов на указанном диске.*

Drive — *задает номер диска:*

0	**текущий диск, т.е. тот диск, с которого запущена программа**
1	**дисковод A**
2	**дисковод B**
3	**диск C**
4	**диск D**

и так далее …..

- *Если номер диска задан некорректно, то функция* **DiskFree** *возвращает значение* **1.**
- *Таким образом, всегда есть возможность проконтролировать правильность вызова функции.*

Процедуры и функции проверки состояния диска.

Функция **DiskSize**

- **DiskSize(Drive: BYTE) :Longlnt;**
- *возвращает общее количество байтов на указанном диске.*

Drive — *задает номер диска:*

0	**текущий диск, т.е. тот диск, с которого запущена программа**
1	**дисковод A**
2	**дисковод B**
3	**диск C**
4	**диск D**

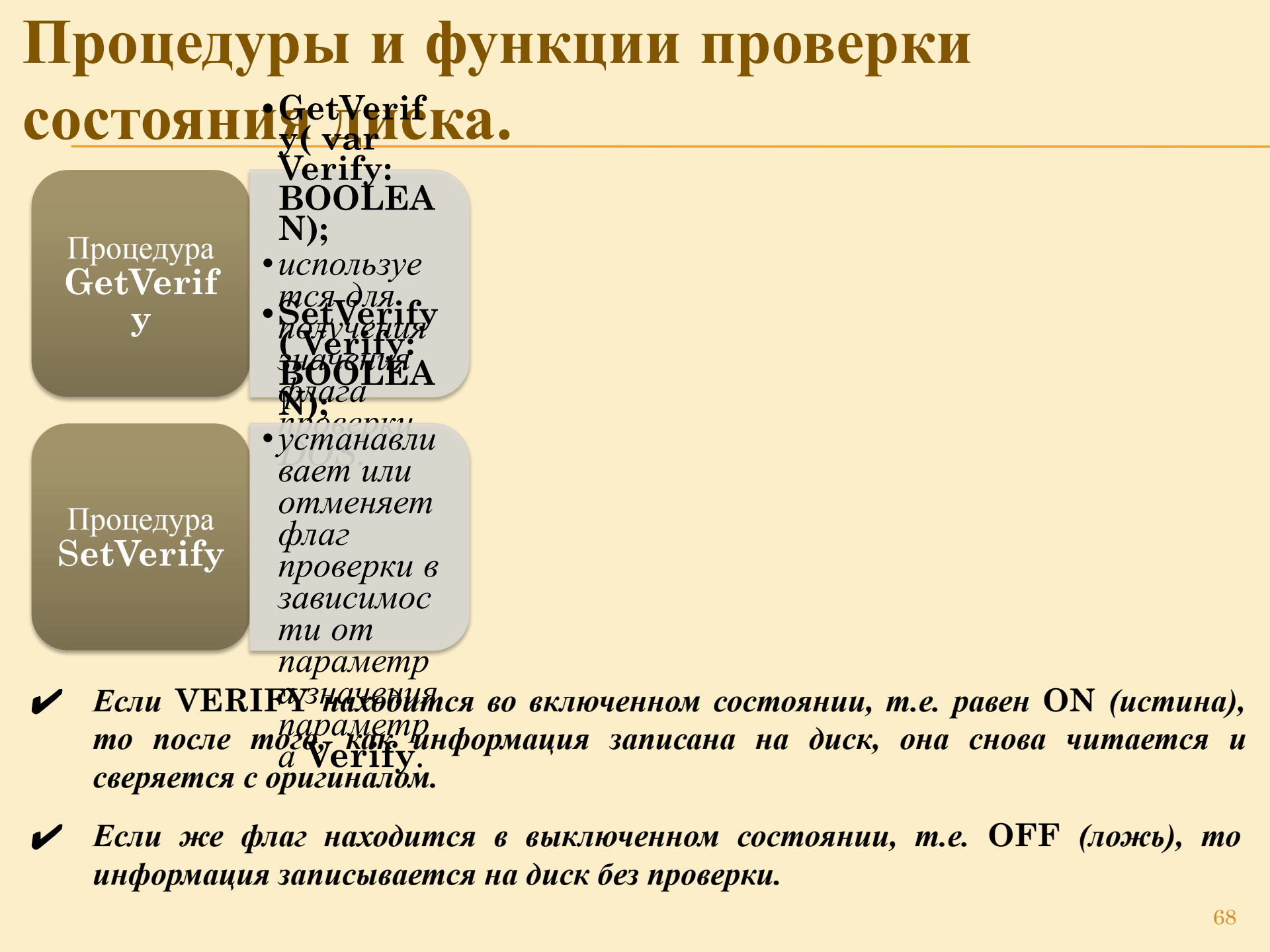
и так далее …..

Процедуры и функции проверки состояния диска.

```
{Использование функций DiskFree и DiskSize}

USES Dos;

BEGIN

    WriteLn('Объем вашего диска ', DiskSize(0) DIV 1024,' Kb');

    WriteIn('На диске свободно ', DiskFree(0) DIV 1024,' Kb');

END.
```

Процедуры и функции проверки состояния диска.

Процедура **GetVerify**	• **GetVerify(var Verify: BOOLEAN);** • *используется для получения значения флага проверки*
Процедура **SetVerify**	• **SetVerify (Verify: BOOLEAN);** • *устанавливает или отменяет флаг проверки в зависимости от параметр значения параметра* **Verify**.



- ✔ ***Если* VERIFY *находится во включенном состоянии, т.е. равен* ON *(истина), то после того, как информация записана на диск, она снова читается и сверяется с оригиналом.***
- ✔ ***Если же флаг находится в выключенном состоянии, т.е.* OFF *(ложь), то информация записывается на диск без проверки.***

Процедуры и функции проверки состояния диска.

```
{Контроль  флага VERIFY}
USES Dos;
VAR
      f     : BOOLEAN;
      OffOn   : STRING[14];
BEGIN
      GetVerify(f);
      If  f  Then OffOn := 'ON - включен'
      ELSE OffOn := 'OFF - выключен'
      WriteLn('Флаг проверки ', OffOn);
        f := Not(f);
          WriteLn(Переключаем флаг проверки');
      SetVerify(f);
END.
```

Данная программа получает флаг проверки и изменяет его значение на противоположное.

Процедуры и функции работы с файлами.

- *Работа с файлами в Турбо Паскале приводит к необходимости использования терминологии и средств MS-DOS.*

- *Иногда возникает необходимость поиска файлов с одинаковым именем, поиска по шаблону, выделения нужного файла из найденной группы файлов.*

  - *Процедуры* **FSplit**, **FExpand**, **FSearch**, **FindFirst**, **FindNext** *упрощают решение этих задач.*

Процедуры и функции работы с файлами.

Для работы с файлами в модуле Dos определены следующие процедуры и функции :

Название	Тип	Назначение
FExpand	**FUNCTION**	*Дополнение имени файла до полного значения (текущий путь, имя, расширение)*
FSearch	**FUNCTION**	*Поиск файла в списке каталогов*
FindFirst	**PROCEDURE**	*Поиск первого файла с заданным именем и атрибутами*
FindNext	**PROCEDURE**	*Поиск следующего файла с заданным именем и атрибутами*
FSplit	**PROCEDURE**	*Разбиение полного имени файла на составные части (путь, имя, расширение)*
GetFAttr	**PROCEDURE**	*Получение атрибутов файла*
SetFAttr	**PROCEDURE**	*Задание атрибутов файла*

Процедуры и функции работы с файлами.

Процедура FindFirst

- **FindFirst(Path :STRING; Attr :WORD; VAR S :SearchRec);**
- *Поиск первого файла с заданным именем и атрибута в указанном каталоге.*

Процедура FindNext

- **FindNext (VAR S :SearchRec);**
- *Поиск следующего файла с заданным именем в указанном каталоге.*

Path — *путь и имя для поиска;*

Attr — *задаваемые атрибуты файла;*

S — *переменная типа* ***SearchRec.***

Тип SearchRec описан в модуле Dos следующим образом:

```
TYPE
    SearchRec = RECORD
        Fill : ARRAY[1..21] OF BYTE;
        Attr : BYTE;
        Time : Longlnt;
        Size : Longlnt;
        Name : STRING[12];
    END;
```

*Поле **Attr*** *содержит атрибуты файла, которые определяются константами атрибутов файла, описанными в модуле* **Dos** *(см. таблицу).*

*Поле **Time*** *содержит дату и время создания файла в упакованном виде (для распаковки необходимо использовать процедуру* **UnpackTime***).*

*Поле **Size*** *содержит размер файла в байтах.*

*Поле **Name*** *содержит имя найденного файла.*

*Поле **Fill*** *содержит служебную информацию **MS-DOS** и не должно модифицироваться.*

Атрибуты файла	
Название	**Значение**
Readonly	$01
Hidden	$02
SysFile	$04
VolumelD	$08
Directory	$10
Archive	$20
AnyFile	$3F

- *Процедура* **FindNext** *ищет следующий файл с именем и атрибутами, заданными при вызове* **FindFirst**, *если имя файла было задано шаблоном.*
- *Если* **FindNext** *не находит больше файлов, в системной переменной* **DosError** *устанавливается значение 18.*

Пример, демонстрирующий использование процедур поиска файлов.

```
USES   Dos;
VAR
      DT        :  DateTime;
      S         :  SearchRec;
      Function  FZero(d : WORD)     : STRING;
      var       t      :  STRING;
      Begin
                Str(d:0,t);
                     IF   Length(t)=1   THEN   t:='0'+t;
                FZero:=t;
      End;
BEGIN
      WriteLn;
      FindFirst ('*.exe ', AnyFile, S);
      While DosError=0 Do
      begin
            UnpackTime( S.Time, DT);
                  Write(S.Name, '  ',FZero (DT.Hour),':',FZero(DT.Min),':',FZero(DT.Sec));
                  WriteLn ('  ',FZero(DT.Day),':',FZero(DT.Month),':', DT.Year);
            FindNext(S);
      end;
      ReadLN
END.
```

Процедуры и функции работы с файлами.

Функция **FSearch**

- **FSearch(Path : PathStr; DirList : STRING) : PathStr;**
- *Поиск файла в списке каталогов.*

Path — *путь и имя файла, который необходимо найти;*

Dirlist — *это список каталогов, в которых будет проводиться поиск;*

Тип ***PathStr*** *описан в модуле* ***Dos,*** *как строковый тип* **STRING[79].**

Каталоги в списке должны быть разделены точкой с запятой.

- аналогично тому, как это делается в команде MS-DOS **PATH.**

Поиск всегда начинается с текущего каталога текущего диска.

- В случае удачного поиска функция возвращает строку, в которой содержится полное имя файла (путь и имя).
- В случае неудачи возвращается пустая

Использование процедуры FSearch.

```
USES   Dos;
     VAR
          F  :  PathStr;
BEGIN
     F:=FSearch('NDD.EXE',GetEnv('PATH'));
     {используется системная переменная PATH для поиска}
     IF F = ' '  THEN WriteLn('Файл NDD.EXE не  найден')
          ELSE WriteLn('найден как ',F);
ReadLn;   {ожидание  нажатия  Enter}
END.
```

***Для поиска применяется переменная окружения DOS* PATH.**

Поиск проводится в каталогах, перечисленных в этой переменной, но можно задать и свой список каталогов.

Процедуры и функции работы с файлами.

Процедура **FSplit**

- **FSplit(Path : PathStr; var Dir : DirStr; var Name :Namestr; var ExtrExtStr);**
- *позволяет разбить полный путь к файлу на три компонента.*

Типы для работы с именами файлов определены в модуле Dos следующим образом:

TYPE

ComStr = STRING[127];

PathStr = STRING[79];

DirStr = STRING[67];

NameStr = STRING[8];

ExtStr = STRING[4] ;

Использование процедуры FSplit

```
Uses  Dos;
      Var
            P:  PathStr;
            D:  DirStr;
            N:  NameStr;
            E:   ExtStr;
BEGIN
      P:=FSearch('NDD.EXE', GetEnv('PATH'));  {поиск файла}
      If P ='"    Then WriteLn('NDD.EXE не найден')
                        Else
                              begin
                                    WriteLn('найден как  ', P);
                                          FSplit( P, D, N, E) ;    {разбиение на компоненты}
                                    WriteLn('путь  ',D  );
                                      WriteLn('имя   ',N);
                                          WriteLn('расширение ',E);
                              end;
      ReadLn;                                                  {ожидание  нажатия Enter}
END.
```

Использование процедур работы с атрибутами файлов.

```
USES Dos;
Var
      F                 :      File;
      FileName      :      PathStr;
      Attr              :      WORD;
      NewAttr       :      BYTE;
BEGIN
      Write('Введите имя файла : '); ReadLn(FileName);
       Assign(F, FileName);
            GetFAttr(F, Attr);                              {получение атрибутов файла}
      IF DosError <> 0
            THEN WriteLn('Ошибка DOS ', DosError)
            ELSE
                  Begin
                        WriteLn('Файл ', FileName);
                        Write('Атрибуты файла = ', Attr,', '); {Определение атрибутов файла}
                        IF Attr AND Readonly <> 0 THEN Write('только для чтения; ');
                        IF Attr AND Hidden <> 0 THEN Write ('скрытый файл;');
                        IF Attr AND SysFile <> 0 THEN Write (' системный файл; ');
                        IF Attr AND VolumelD <> 0 THEN Write('метка тома; ');
                        IF Attr AND Directory <> 0 THEN Write('имя директории; ');
                        IF Attr AND Archive <> 0 THEN Write ('архивный файл); ');
                        WriteLn;
```

Использование процедур работы с атрибутами файлов.

```
        WriteLn('Установка нового атрибута');
            WriteLn('1- только для чтения, 2- скрытый файл,');
            WriteLn('3- системный файл, 4- метка тома, ' );
            WriteLn('5 - имя директории,6 - архивный');
        Write ('Введите новый атрибут ');  ReadLn (NewAttr);
                                                {установка новых атрибутов}
        CASE NewAttr OF
                    1:   SetFAttr(F, Readonly);
                    2:   SetFAttr(F, Hidden);
                    3:   SetFAttr(F, SysFile);
                    4:   SetFAttr(F, VolumelD);
                    5:    SetFAttr(F, Directory);
            ELSE SetFAttr(F, Archive);
        end;            { of Case }
    End;
END.
```

Другие процедуры и функции.

- *В модуле Dos есть еще несколько процедур и функций, не рассмотренных нами.*

- *Это функция* **Dos-Version** *и процедуры* **GetCBreak** *и* **SetCBreak**.

Другие процедуры и функции.

Функция
DosVersion

- **DosVersion : WORD;**
- *возвращает номер версии* **DOS**.

В старшем байте содержится целая часть номера версии например 6, а в младшем байте — дробная часть номера, например 22.
В итоге получаем 6.22.

Использование функции DosVersion.

```
USES  Dos;
      VAR
            V :   WORD;
BEGIN
      V := DosVersion;
      WriteLn('Версия DOS ', Lo(V), '.',Hi(V));
END.
```

Другие процедуры и функции.

Процедура **GetCBreak**

- **GetCBreak(VAR BREAK: BOOLEAN);**
- *возвращает значение системной переменной MS-DOS* **BREAK**

Процедура **SetCBreak**

- **SetCBreak(BREAK: BOOLEAN);**
- *устанавливает системную переменную MS-DOS* **BREAK**

Если* BREAK *равна* OFF*, т.е. выключена, то прерывание программы осуществляется только во время операций ввода-вывода и выполнение программ происходит немного быстрее.

Если же* BREAK *равна* ON*, то прерывание происходит при любых системных вызовах.

Использование процедур GetBreak и SetBreak.

```
USES  Dos;
VAR
     f     : BOOLEAN;
     OffOn   : STRING [14];
BEGIN
     GetCBreak(f);
     IF   f     THEN   OffOn:='ON   -   включен'
          ELSE   OffOn:='OFF   -   выключен';
     WriteLn('BREAK   ', OffOn);
       f:=   NOT(f) ;
        WriteLn('Изменяем значение  Break');
     SetCBreak(f);
END.
```