МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра математического обеспечения и применения ЭВМ

ОТЧЕТ

по практической работе №6

по дисциплине «Операционные системы»

Тема: Построение модуля динамической структуры

Студентка гр. 7382 _____ Чемова К.А.

Преподаватель _____ Ефремов М.А.

Санкт-Петербург

2019

Цель работы.

Исследование возможности построения загрузочного модуля динамической структуры. В отличии от предыдущих лабораторных работ в этой работе рассматривается приложение, состоящее из нескольких модулей, а не из одного модуля простой структуры. В этом случае разумно предположить, что все модули приложения находятся в одном каталоге и полный путь в этот каталог можно взять из среды, как это делалось в работе 2. Понятно, что такое приложение должно запускаться в соответствии со стандартами ОС.

В работе исследуется интерфейс между вызывающим и вызываемым модулями по управлению и по данным. Для запуска вызываемого модуля используется функция 4B00h прерывания int 21h. Все загрузочные модули находятся в одном каталоге. Необходимо обеспечить возможность запуска модуля динамической структуры из любого каталога.

Необходимые сведения для составления программы.

Для загрузки и выполнения одной программы из другой используется функция 4B00h прерывания int 21h (загрузчик ОС). Перед обращением к этой функции необходимо выполнить следующие действия:

1) Подготовить место в памяти. При начальном запуске программы ей отводится вся доступная в данный момент память OS, поэтому необходимо освободить место в памяти. Для этого можно использовать функцию 4Ah прерывания int 21h. Эта функция позволяет уменьшить отведенный программе блок памяти. Перед вызовом функции надо определить объем памяти, необходимый программе ЛР6 и задать в регистре BX число параграфов, которые будут выделяться программе. Если функция 4Ah не может быть выполнена, то устанавливается флаг переноса CF=1 и в AX заносится код ошибки:

7 - разрушен управляющий блок памяти;

8 - недостаточно памяти для выполнения функции;

9- неверный адрес блока памяти.

Поэтому после выполнения каждого прерывания int 21h следует проверять флаг переноса CF=1.

2) Создать блок параметров. Блок параметров - это 14-байтовый блок памяти, в который помещается следующая информация:

dw сегментный адрес среды

dd сегмент и смещение командной строки

dd сегмент и смещение первого FCB

dd сегмент и смещение второго FCB

Если сегментный адрес среды 0, то вызываемая программа наследует среду вызывающей программы. В противном случае вызывающая программа должна сформировать область памяти в качестве среды, начинающуюся с адреса кратного 16 и поместить этот адрес в блок параметров.

Командная строка записывается в следующем формате:

первый байт - счетчик, содержащий число символов в командной строке, затем сама командная строка, содержащая не более 128 символов.

На блок параметров перед загрузкой вызываемой программы должны указывать ES:BX.

3) Подготовить строку, содержащую путь и имя вызываемой программы. В конце строки должен стоять код ASCII 0. На подготовленную строку должны указывать DS:DX.

4) Сохранить содержимое регистров SS и SP в переменных. При восстановлении SS и SP нужно учитывать, что DS необходимо также восстановить.

Когда вся подготовка выполнена, вызывается загрузчик OS следующей последовательностью команд:

mov AX,4B0 0h int 21h

Если вызываемая программа не была загружена, то устанавливается флаг переноса CF=1 и в AX заносится код ошибки:

1 - если номер функции неверен;

2 - если файл не найден;

5 - при ошибке диска;

8 - при недостаточном объеме памяти;

10 - при неправильной строке среды;

11 - если не верен формат.

Если CF=0, то вызываемая программа выполнена и следует обработать ее завершение. Для этого необходимо воспользоваться функцией 4Dh прерывания int 21h. В качестве результата функция возвращает в регистре AH причину, а в регистре AL код завершения.

Причина завершения в регистре AH представляется следующими кодами:

0 - нормальное завершение;

1 - завершение о Ctrl-Break;

2 - завершение о ошибке устройства;

3 - завершение по функции 31h, оставляющей программу резидентной.

Код завершения формируется вызываемой программой в регистре AL перед выходом в OS с помощью функции 4Ch прерывания int 21h.

В качестве вызываемой программы целесообразно использовать программу, разработанную в Лабораторной работе №2, модифицировав ее следующим образом. Перед выходом из программы перед выполнением функции 4Ch прерывания int 21h следует за росить с клавиатуры символ и поместить введенный символ в регистр AL, в качестве кода завершения. Это можно сделать с помощью функции 01h прерывания int 21h.

mov AH,01h int 21h

Введенный символ остается в регистре AL и служит аргументом для функции 4Ch прерывания int 21h.

Постановка задачи.

Шаг 1. Для выполнения лабораторной работы необходимо написать и отладить программный модуль типа .EXE, который выполняет функции:

1) Подготавливает параметры для запуска загрузочного модуля из того же каталога, в котором находится он сам. Вызываемому модулю передается новая среда, созданная вызывающим модулем и новая командная строка.

2) Вызываемый модуль запускается с использованием загрузчика.

3) После запуска проверяется выполнение загрузчика, а затем результат выполнения вызываемой программы. Необходимо проверять причину завершения и, в зависимости от значения, выводить соответствующее сообщение. Если причина завершения 0, то выводится код завершения.

В качестве вызываемой программы необходимо взять программу ЛР 2, которая распечатывает среду и командную строку. Эту программу следует немного модифицировать, вставив перед выходом из нее обращение к функции ввода символа с клавиатуры. Введенное значение записывается в регистр AL и затем происходит обращение к функции выхода 4Ch прерывания int 21h.

Шаг 2. Запустите отлаженную программу, когда текущим каталогом является каталог с разработанными модулями. Программа вызывает другую программу, которая останавливается, ожидая символ с клавиатуры.

Введите произвольный символ из числа A-Z. Посмотрите причину завершения и код. Занесите полученные данные в отчет.

Шаг 3. Запустите отлаженную программу, когда текущим каталогом является каталог с разработанными модулями. Программа вызывает другую программу, которая останавливается, ожидая символ с клавиатуры.

Введите комбинацию символов Ctrl-C. Посмотрите причину завершения и код. Занесите полученные данные в отчет.

Шаг 4. Запустите отлаженную программу, когда текущим каталогом является какой- либо другой каталог, отличный от того, в котором содержатся разработанные программные модули.

Повторите ввод комбинаций клавиш. Занесите полученные данные в отчет.

Шаг 5. Запустите отлаженную программу, когда модули находятся в разных каталогах. Занесите полученные данные в отчет.

Процедуры используемые в программе.

- PRINT – выводит сообщение на экран;
- TETR_TO_HEX – вспомогательная функция, которая переводит из двоичной в шестнадцатеричную систему счисления, используется для работы функции BYTE_TO_HEX;
- BYTE_TO_HEX – переводит байтовое число из регистра AL в шестнадцатеричную систему счисления;
- FREE_MEMORY – освобождает место в памяти, используя функцию 4Ah прерывания int 21h;
- PROCESSING – запуск вызываемого модуля LR2.COM;
- CREATE_BP – создание блока параметров;
- RETURN_CODE – вывод кода завершения программы;
- NOT_LOADED_ERROR – обработка ошибок, если программа не была выполнена;
- BEGINNING – главная процедура.

Структуры данных.

Таблица 2 – Структуры данных используемые в программе

Название поля данных	Тип	Назначение
Mem_7	db	Строка – сообщение об ошибке освобождения памяти: 7 – разрушен управляющий блок памяти; 8 – недостаточно памяти для выполнения функции; 9 – неверный адрес блока памяти
Mem_8	db	
Mem_9	db	
Err_11	db	Строка – сообщение об ошибке освобождения памяти: 7 – разрушен управляющий блок памяти; 8 – недостаточно памяти для выполнения функции;
Err_10	db	
Err_1	db	
Err_2	db	
Err_5	db	

Err_8	db	9 – неверный адрес блока памяти
End_0	db	Строка – сообщение, содержащая причину завершения вызываемой программы: 0 – нормальное завершение; 1 – завершение по Ctrl-Break; 2 – завершение по ошибке устройства; 3 – завершение по функции 31h, оставляющей программу резидентной
End_1	db	
End_2	db	
End_3	db	
PATH	db	Строка, содержащая название вызываемого модуля
KEEP_SS	dw	Переменная для сохранение содержимого регистра SS
KEEP_SP	dw	Переменная для сохранение содержимого регистра SP
PATH	db	Строка, содержащая название вызываемого модуля
KEEP_SS	dw	Переменная для сохранение содержимого регистра SS

Ход работы.

Запуск отлаженной программы, когда текущей каталог является каталогом с разработанными модулями. Выведенная информация показана на рис.1.

```
C:\>LAB6.EXE
Segment memory address:  9FFF
The environment segment address:  1191
The tail of the command line:
Empty tail

Content of the environment area in symbolic form
PATH=Z:\
COMSPEC=Z:\COMMAND.COM
BLASTER=A220 I7 D1 H5 T6

The path of the loaded module
:\LAB2.COM
```

Рисунок 1 – Запуск LAB2.COM программой LAB6.EXE

Была запущена LAB2.COM, которая ожидает ввода символа с клавиатуры. После ввода R на экран было выведено сообщение о нормальном завершении программы, показанное на рис.2

Рисунок 2 – Нормальное завершение программы

На рис.3 представлено прерывание программы LAB2.COM при помощи Ctrl-C. Причиной завершения является нормальное завершение работы программы, код завершения – 03. В данном случае, причиной должно было бы являться прерывание по Ctrl-Break, но в DOSBOX игнорируется это прерывание.

Рисунок 3 – Прерывание Ctrl-C

На рис. 4 и 5 показана работа программы LAB6.EXE, когда текущий каталог не является каталогом с разработанными модулями.

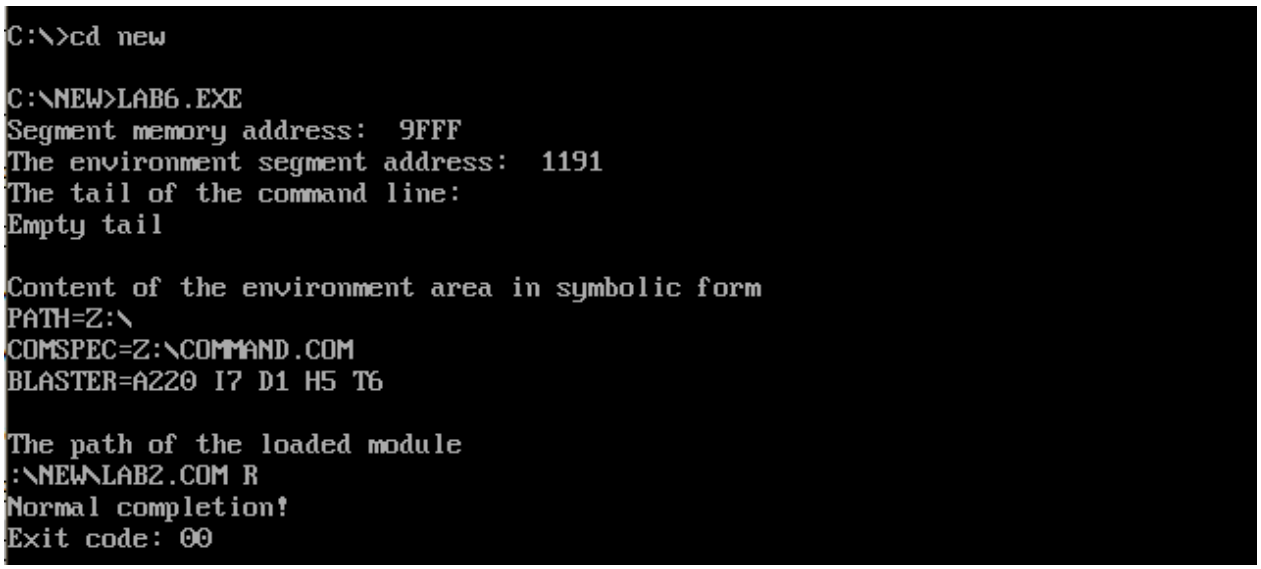

Рисунок 4 – Прерывание при нажатии R

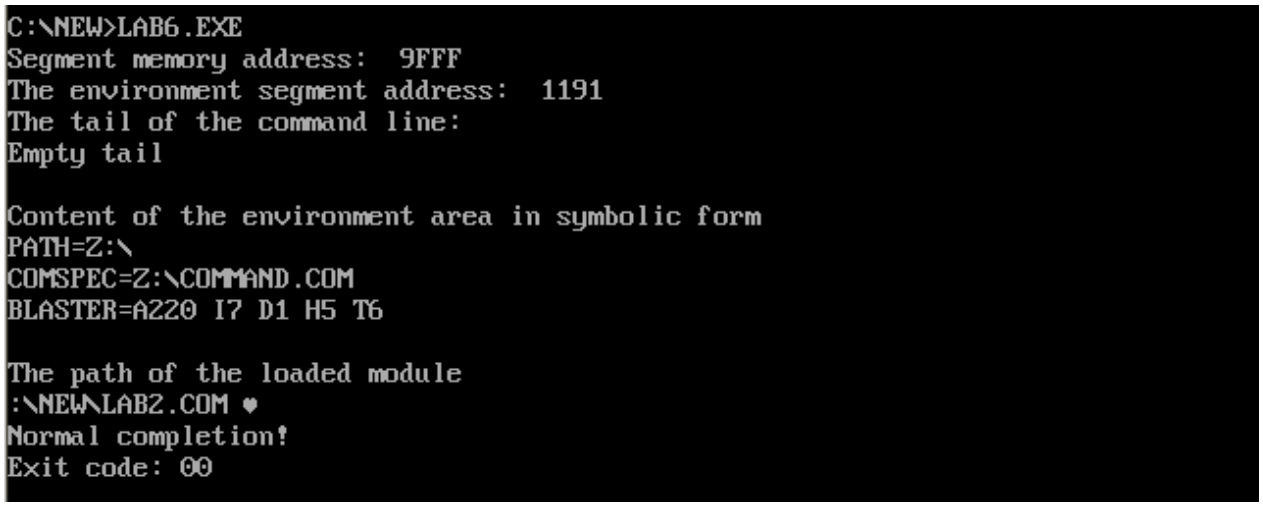

Рисунок 5 – Прерывание Ctrl-C

Запуск отлаженной программы, когда модули находятся в разных каталогах (LAB2.COM в другом каталоге). Выведенное сообщение представлено на рис. 6.

Рисунок 6 – Разные каталоги

Вывод.

В ходе лабораторной работы был построен загрузочный модуль динамической структуры, а также модифицирован ранее построенный программный модуль. Изучены дополнительные функции работы с памятью и исследованы возможности использования интерфейса между вызывающим и вызываемым модулями по управлению и по данным.

Ответы на контрольные вопросы.

1) **Как реализовано прерывание Ctrl-C?**

При нажатии клавиш Ctrl-C управление передаётся по адресу 0000:008Ch. Этот адрес копируется в PSP функциями 26h и 4Ch и восстанавливается из PSP при выходе из программы.

2) **В какой точке заканчивается вызываемая программа, если код причины завершения 0?**

Если код завершения 0, то программа завершается при выполнении функции 4Ch прерывания int 21h;

3) **В какой точке заканчивается вызываемая программа по прерыванию Ctrl-C?**

Если во время выполнения программы было нажато Ctrl-C, то программа завершится непосредственно в том месте, в котором произошло нажатие сочетания клавиш (то есть в месте ожидания нажатия клавиши: 01h вектора прерывания 21h);

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ LAB2.ASM

```
TESTPC SEGMENT
        ASSUME CS:TESTPC, DS:TESTPC, ES:NOTHING, SS:NOTHING
        ORG 100H
START: JMP BEGIN
;-------------------------------------------------------------------------
-----------------------
;ДАННЫЕ
mem_add         db 'Segment memory adфdress:     ', 0DH, 0AH, '$'
env_add    db 'The environment segment address:     ', 0DH, 0AH, '$'
env_cont   db 'Content of the environment area in symbolic form      ',
0DH, 0AH, '$'
m_path          db 'The path of the loaded module       ', 0DH, 0AH,
'$'
tail_str   db 'The tail of the command line: ', 0DH, 0AH, '$'
empty_tail db 'Empty tail', 0DH, 0AH, '$'
endl       db ' ', 0DH, 0AH, '$'
;ПРОЦЕДУРЫ
;-------------------------------------------------------------------------
-----------------------
TETR_TO_HEX  PROC near
     and   AL, 0Fh
     cmp   AL, 09
     jbe   NEXT
     add   AL, 07
NEXT:   add     AL, 30h
     ret
TETR_TO_HEX  ENDP
;-------------------------------------------------------------------------
-----------------------
BYTE_TO_HEX  PROC near
;байт в AL переводиться в два символа шестнадцатеричного числа в AX
     push       CX
     mov        AH,AL
     call       TETR_TO_HEX
     xchg       AL,AH
     mov        CL,4
     shr        AL,CL
     call       TETR_TO_HEX
     pop        CX
     ret
BYTE_TO_HEX  ENDP
;-------------------------------------------------------------------------
-----------------------
WRD_TO_HEX  PROC near
;перевод в 16 СС 16-ти разрядного числа
;в AX - число, DI - адрес последнего символа
```

```
      push       BX
      mov        BH,AH
      call       BYTE_TO_HEX
      mov        [DI],AH
      dec        DI
      mov        [DI],AL
      dec        DI
      mov        AL,BH
      call       BYTE_TO_HEX
      mov        [DI],AH
      dec        DI
      mov        [DI],AL
      pop        BX
      ret
WRD_TO_HEX  ENDP
;-------------------------------------------------------------------------
-----------------------
BYTE_TO_DEC  PROC near
;перевод в 10 СС, SI - адрес поля младшей цифры
      push  CX
      push  DX
      xor   AH,AH
      xor   DX,DX
      mov   CX,10
loop_bd:
      div   CX
      or          DL,30h
      mov   [SI],DL
      dec   SI
      xor   DX,DX
      cmp   AX,10
      jae   loop_bd
      cmp   AL,00h
      je    end_l
      or    AL,30h
      mov   [SI],AL
end_l:
      pop   DX
      pop   CX
      ret
BYTE_TO_DEC ENDP
;-------------------------------------------------------------------------
-----------------------
PRINT_MSG  PROC near
;вывод строки на экран
        push    AX
        mov     AH, 09h
      int   21H
      pop   AX
      ret
PRINT_MSG  ENDP
```

```
;-------------------------------------------------------------------------
----------------------
SEG_ADDRESS_INACCESS_MEM  PROC near
;получение сегментного адреса недоступной памяти
        push    AX
      push DI
      mov  AX, DS:[02h]
      mov  DI, OFFSET mem_add
      add  DI, 28
      call WRD_TO_HEX
      pop  DI
      pop  AX
      ret
SEG_ADDRESS_INACCESS_MEM  ENDP
;-------------------------------------------------------------------------
----------------------
SEG_ADDRESS_ENV  PROC near
;получение сегментного адреса среды
        push    AX
      push DI
      mov  AX, DS:[2Ch]
      mov  DI, OFFSET env_add
      add  DI, 37
      call WRD_TO_HEX
      pop  DI
      pop  AX
      ret
SEG_ADDRESS_ENV  ENDP
;-------------------------------------------------------------------------
----------------------
TAIL  PROC near
;получение хвоста командной строки
        push    AX
      push CX
      push DX
      push SI
        mov     DX, OFFSET tail_str
      call PRINT_MSG
      mov  CL, DS:[80h]
      cmp  CL, 00h
      je   empty_str
      mov  SI, 81h
      mov  AH, 02h
str_content:
      mov  DL, DS:[SI]
      int  21h
      inc  SI
      loop str_content
      mov  DX, OFFSET endl
      call PRINT_MSG
      jmp  str_end
```

```
empty_str:
        mov     AL, 00h
      mov  [DI], AL
      mov  DX, OFFSET empty_tail
      call PRINT_MSG
str_end:
      pop  SI
      pop  DX
      pop  CX
      pop  AX
      ret
TAIL  ENDP
;-------------------------------------------------------------------------
-----------------------
ENV_CONTENT  PROC near
;получение содержимого области среды
        push    AX
      push DX
      push DS
      push ES
      mov  DX, OFFSET env_cont
      call PRINT_MSG
      mov  AH, 02h
      mov  ES, DS:[2Ch]
      xor  SI, SI
environment1:
        mov     DL, ES:[SI]
      int  21h
      cmp  DL, 00h
      je   environment2
      inc  SI
      jmp  environment1
environment2:
        mov     DX, OFFSET endl
      call PRINT_MSG
      inc  SI
      mov  DL, ES:[SI]
      cmp  DL, 00h
      jne  environment1
      mov  DX, OFFSET endl
      call PRINT_MSG
      pop  ES
      pop  DS
      pop  DX
      pop  AX
      ret
ENV_CONTENT  ENDP
;-------------------------------------------------------------------------
-----------------------
PATH  PROC near
;получение пути загружаемого модуля
```

```
        push    AX
      push  DX
      push  DS
      push  ES
      mov   DX, OFFSET m_path
         call    PRINT_MSG
         add     SI, 04h
         mov     AH, 02h
         mov     ES, DS:[2Ch]
path_:
         mov     DL, ES:[SI]
         cmp     DL, 00h
         je      _path_
         int     21h
         inc     SI
         jmp     path_
_path_:
         pop     ES
         pop     DS
         pop     DX
         pop     AX
         ret
PATH  ENDP
;-------------------------------------------------------------------------
-----------------------
BEGIN:
         call    SEG_ADDRESS_INACCESS_MEM
         mov     DX, OFFSET mem_add
         call    PRINT_MSG
         call    SEG_ADDRESS_ENV
         mov     DX, OFFSET env_add
         call    PRINT_MSG
         call    TAIL
         mov     DX, OFFSET endl
         call    PRINT_MSG
         call    ENV_CONTENT
         call    PATH

      mov   DL, 0020h
      mov   AH, 0002h
      int   21h
      mov   AH, 01h
      int   21h
;ВЫХОД ИЗ DOS
;-------------------------------------------------------------------------
-----------------------
         xor     AL, AL
         mov     AH, 4CH
         int     21h
TESTPC  ENDS
         END  START
```

ПРИЛОЖЕНИЕ Б
ИСХОДНЫЙ КОД ПРОГРАММЫ LAB6.ASM

```
ASTACK SEGMENT STACK
     DW 64 DUP (?)
ASTACK ENDS
;ДАННЫЕ
;-------------------------------------------------------------------
-----------------------

DATA SEGMENT
ParameterBlock     dw  0000h  ;сегментный адрес среды
                   dd  0000h  ;сегмент и смещение командной строки
                   dd   0000h   ;сегмент и смещение первого FCB (File
Control Block)
                   dd  0000h  ;сегмент и смещение второго FCB
     Mem_8          db   0DH, 0AH, 'Not enough memory to perform the
function!', 0DH, 0AH, '$'
        PATH       db                                                '
', 0DH, 0AH, '$', 0
     End_2        db  0DH, 0AH, 'The completion of the device error!',
0DH, 0AH, '$'
     Mem_9         db  0DH, 0AH, 'Wrong address of the memory block!',
0DH, 0AH, '$'
     Err_1       db  0DH, 0AH, 'The number of function is wrong!', 0DH,
0AH, '$'
     Mem_7         db  0DH, 0AH, 'Memory control unit destroyed!', 0DH,
0AH, '$'
     Err_10        db  0DH, 0AH, 'Incorrect environment string!', 0DH,
0AH, '$'
     End_3       db  0DH, 0AH, 'Completion by function 31h!', 0DH, 0AH,
'$'
     Err_8       db  0DH, 0AH, 'Insufficient memory!', 0DH, 0AH, '$'
     End_0       db  0DH, 0AH, 'Normal completion!', 0DH, 0AH, '$'
     End_1       db  0DH, 0AH, 'End by Ctrl-Break!', 0DH, 0AH, '$'
     Err_11      db  0DH, 0AH, 'Wrong format!', 0DH, 0AH, '$'
     Err_5       db  0DH, 0AH, 'Disk error!', 0DH, 0AH, '$'
     Err_2       db  0DH, 0AH, 'File not found!', 0DH, 0AH, '$'
     END_CODE    db  'Exit code:   ', 0DH, 0AH, '$'
     KEEP_SS     dw  0000h
     KEEP_SP     dw  0000h
DATA ENDS
;-------------------------------------------------------------------
-----------------------
CODE SEGMENT
        ASSUME CS:CODE, DS:DATA, ES:DATA, SS:ASTACK
;ПРОЦЕДУРЫ
;-------------------------------------------------------------------
-----------------------
```

```
PRINT PROC NEAR
;вывод строки
      mov  AH, 0009h
      int  21h
      ret
PRINT ENDP
;-------------------------------------------------------------------------
----------------------
TETR_TO_HEX PROC near
;из двоичной в шестнадцатеричную сс
      and  AL,  0Fh
      cmp  AL,  09
      jbe  NEXT
      add  AL,  07
NEXT:
      add  AL,  30h
      ret
TETR_TO_HEX ENDP
;-------------------------------------------------------------------------
----------------------
BYTE_TO_HEX PROC near
;байтовое число в шестнадцатеричную сс
      push CX
      mov  AH,  AL
      call TETR_TO_HEX
      xchg AL,  AH
      mov  CL,  4
      shr  AL,  CL
      call TETR_TO_HEX
      pop  CX
      ret
BYTE_TO_HEX ENDP
;-------------------------------------------------------------------------
----------------------
FREE_MEMORY PROC NEAR
;освобождение места в памяти, перед вызовом функции надо определить
объём памяти, необходимый lr6
      mov  BX,  offset LAST_BYTE  ;кладём в BX адрес конца программы
      mov  AX,  ES                ;ES - начало программы
      sub  BX,   AX                   ;BX = BX - ES, число параграфов,
которые будут выделяться прграмме
      mov  CL,  0004h
      shr  BX,  CL                ;переводим в параграфы
      mov  AH,  4Ah               ;функция сжатия блока памяти
      int  21h
      jnc  WITHOUT_ERROR          ;флаг CF = 0, если нет ошибки
      jmp  WITH_ERROR                 ;обработка ошибок CF=1 AX = код
ошибки, если CF установлен
;*************************************************************
**********************
WITHOUT_ERROR:
```

```
      ret
;*************************************************************************
**********************
WITH_ERROR:
;*************************************************************************
**********************
MEM_7_ERROR:
;разрушен управляющий блок памяти
      cmp   AX,  0007h
         jne      MEM_8_ERROR
      mov   DX,  offset Mem_7
      jmp   END_ERROR
;*************************************************************************
**********************
MEM_8_ERROR:
;недостаточно памяти для выполнения функции
      cmp   AX,  0008h
      jne   MEM_9_ERROR
      mov   DX,  offset Mem_8
      jmp   END_ERROR
;*************************************************************************
**********************
MEM_9_ERROR:
;неверный адрес блока памяти
         mov      DX,  offset Mem_9
;*************************************************************************
**********************
END_ERROR:
      call  PRINT
      xor   AL,  AL
      mov   AH,  4Ch
      int   21h
FREE_MEMORY ENDP
;-------------------------------------------------------------------------
----------------------
PROCESSING PROC NEAR
      mov   ES,     ES:[2Ch]     ;сегментный   адрес   среды,   передаваемый
программе
      mov   SI,  0000h
;*************************************************************************
**********************
cycle:
      mov   DL, ES:[SI]
      cmp   DL, 0000h   ;конец строки?
      je    end_cycle
      inc   SI
      jmp   cycle
;*************************************************************************
**********************
end_cycle:
      inc   SI
```

```
      mov   DL,  ES:[SI]
      cmp   DL,  0000h   ;конец среды?
      jne   cycle
      add   SI,  0003h   ;SI указывает на начало маршрута
      push  DI
      lea   DI,  PATH
;*****************************************************************
**********************
loop_:
        mov     DL,  ES:[SI]
      cmp   DL,  0000h    ;конец маршрута?
      je    end_loop
      mov   [DI],  DL
      inc   DI
      inc   SI
      jmp   loop_
;*****************************************************************
**********************
end_loop:
      sub   DI,  0008h
      mov   [DI],  byte ptr 'L'
      mov   [DI+0001h],  byte ptr 'A'
      mov   [DI+0002h],  byte ptr 'B'
      mov   [DI+0003h],  byte ptr '2'
      mov   [DI+0004h],  byte ptr '.'
      mov   [DI+0005h],  byte ptr 'C'
      mov   [DI+0006h],  byte ptr 'O'
      mov   [DI+0007h],  byte ptr 'M'
      mov   [DI+0008h],  byte ptr 0h
      pop   DI

      mov   KEEP_SP,  SP                ;сохраняем содержимое регистров
SS и SP
      mov   KEEP_SS,  SS
      push  DS
      pop   ES
      mov   BX,  offset ParameterBlock
      mov   DX,  offset PATH
      mov   AX,  4B00h                  ;вызываем загрузчик OS
      int   21h
      jnc   is_loaded                     ;если вызываемая программа не
была загружена,
                                          ;то   устанавливается   флаг
переноса CF=1 и в AX заносится код ошибки
      push  AX
      mov   AX,  DATA
      mov   DS,  AX
      pop   AX
      mov   SS,  KEEP_SS                ;восстанавление DS, SS, SP
      mov   SP,  KEEP_SP
      call  NOT_LOADED_ERROR
```

```
is_loaded:
     mov   AX,  4d00h                        ;в AH - причина, в AL - код
завершения
     int   21h
        call    RETURN_CODE
     ret
PROCESSING ENDP
;-------------------------------------------------------------------------
-----------------------
CREATE_BP PROC NEAR
        mov     AX,  ES:[2Ch]
     mov   ParameterBlock,  AX
     mov   ParameterBlock+0002h,   ES      ;сегментный адрес параметров
командной строки
     mov   ParameterBlock+0004h,   0080h   ;смещение параметров комадной
строки
     ret
CREATE_BP ENDP
;-------------------------------------------------------------------------
-----------------------
RETURN_CODE PROC NEAR
     cmp   AH,  0000h          ;нормальное завершение
     mov   DX,  offset End_0
     je    EXIT_CODE
     cmp   AH,  0001h          ;завершение по Ctrl-Break
     mov   DX,  offset End_1
     je    EXIT_CODE
     cmp   AH,  0002h          ;завершение по ошибке устройства
     mov   DX,  offset End_2
     je    EXIT_CODE
     cmp   AH,  0003h           ;завершение по функции 31h, оставляющей
программу резидентной
     mov   DX,  offset End_3

EXIT_CODE:
     call  PRINT            ;выводим код завершения на экран
     mov   DI,  offset END_CODE
     call  BYTE_TO_HEX
     add   DI,  000Bh
     mov   [DI],  AL
     add   DI,  0001h
     xchg  AH,  AL
     mov   [DI],  AL
     mov   DX,  offset END_CODE
     call  PRINT
     xor   AL,  AL
     mov   AH,  4Ch
     int   21h
RETURN_CODE ENDP
```

```
;-------------------------------------------------------------------------
----------------------
NOT_LOADED_ERROR PROC NEAR
;обработка ошибок, если программа не была выполнена
      cmp   AX,  0001h          ;если номер функции неверен
      mov   DX,  offset Err_1
      je    NOT_LOADED
      cmp   AX,  0002h          ;если файл не найден
      mov   DX,  offset Err_2
      je    NOT_LOADED
      cmp   AX,  0005h          ;при ошибке диска
      mov   DX,  offset Err_5
      je    NOT_LOADED
      cmp   AX,  0008h          ;при недостаточном объёме памяти
      mov   DX,  offset Err_8
      je    NOT_LOADED
      cmp   AX,  000Ah          ;при неправильной строке среды
      mov   DX,  offset Err_10
      je    NOT_LOADED
      cmp   AX,  000Bh          ;если неверен формат
      mov   DX,  offset Err_11

NOT_LOADED:
      call  PRINT
      xor   AL, AL
      mov   AH, 4Ch
      int   21h
NOT_LOADED_ERROR ENDP
;-------------------------------------------------------------------------
----------------------
MAIN:
      mov   AX,DATA
      mov   DS,AX
      call  FREE_MEMORY
         call    CREATE_BP
      call  PROCESSING
      xor   AL, AL
      mov   AH, 4Ch
      int   21h
LAST_BYTE:
CODE ENDS
          END MAIN
```