

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

ОТЧЕТ
по лабораторной работе №6
по дисциплине «Операционные системы»
Тема: Построение модуля динамической структуры

Студентка гр. 8383

Максимова А.А.

Преподаватель

Ефремов М.А.

Санкт-Петербург

2020

Цель работы.

Исследование возможности построения загрузочного модуля динамической структуры.

Основные теоретические положения.

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) Освободить место в памяти, используя функцию 4Ah прерывания int 21h. Эта функция позволяет уменьшить отведенный программе блок памяти.
- 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
```

Рисунок 1 – Блок параметров

- 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 ОС следующей последовательностью команд:

```
mov AX, 4B00h
Int 21h
```

Процедуры, используемые в программе.

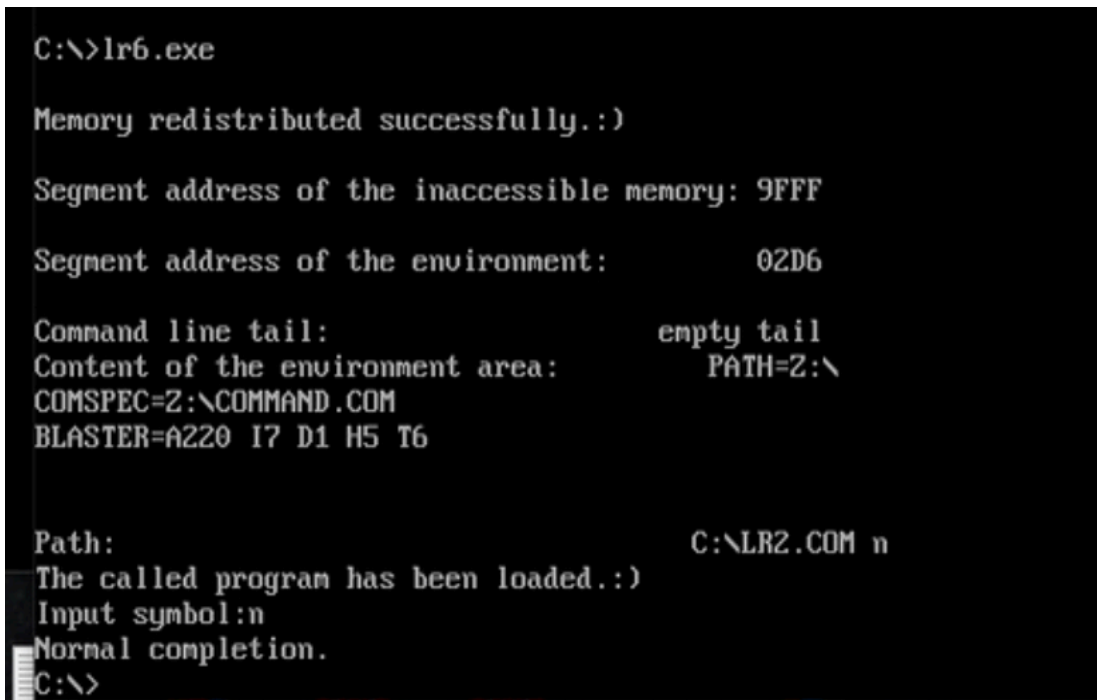
Таблица 1 - Процедуры

Название:	Предназначение:
FREE_MEMORY	Процедура, используемая для освобождения места в памяти, которое не требуется программе. В случае если освобождение не может быть выполнено, то выводится ошибка.
CATCH_ERROR	Процедура, используемая для определения ошибки, возникшей при освобождении памяти, определяемой по коду в AX.
FILL_PARAM_BLOCK	Процедура, используемая для заполнения блока параметров.
GET_PROG_PATH	Процедура, используемая для записи местоположения вызываемого модуля.
LOADER	Процедура, используемая для загрузки вызываемого модуля.
COMPL_PROC	Процедура, используемая для обработки причины и вывода кода завершения.
FIND_ERROR	Процедура, используемая для обработки кода ошибки в случае, если вызываемая программа не загружена.
BEGIN	Головная процедура.
PRINTF	Процедура печати.

Выполнение работы.

Шаг 1. Для выполнения лабораторной работы был написан и отлажен 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 проверить причину завершения и, в зависимости от значения, выводить соответствующее сообщение. Если причина завершения 0, то выводится код завершения. В качестве вызываемой программы была взята программа ЛР 2, печатающая среду и командную строку. Код программы см. в приложении Б.



```
C:\>lr6.exe

Memory redistributed successfully :)

Segment address of the inaccessible memory: 9FFF

Segment address of the environment:          02D6

Command line tail:                          empty tail
Content of the environment area:             PATH=Z:\
COMSPEC=Z:\COMMAND.COM
BLASTER=A220 17 D1 H5 T6

Path:                                         C:\LR2.COM n
The called program has been loaded :)
Input symbol:n
Normal completion.
C:\>
```

Рисунок 2 – Пример запуска программы

Шаг 2.

Был выполнен запуск программы, когда текущим каталогом является каталог с разработанными модулями. Программа вызывает другую программу, которая останавливается, ожидая символ с клавиатуры. Вводится символ А. Полученный результат см. на рис. 3.

```
Normal completion.  
C:\>lr6.exe  
  
Memory redistributed successfully.:)  
  
Segment address of the inaccessible memory: 9FFF  
  
Segment address of the environment:          02D6  
  
Command line tail:                          empty tail  
Content of the environment area:             PATH=Z:\  
COMSPEC=Z:\COMMAND.COM  
BLASTER=A220 I7 D1 H5 T6  
  
Path:                                         C:\LR2.COM A  
The called program has been loaded.:)  
Input symbol:A  
Normal completion.  
C:\>
```

Рисунок 3 – Результат запуска программы при вводе символа А

Шаг 3.

Был выполнен повторный запуск программы с тем же текущим каталогом, но вместо символа была введена комбинация символов Ctrl-C.

```
Normal completion.  
C:\>lr6.exe  
  
Memory redistributed successfully.:)  
  
Segment address of the inaccessible memory: 9FFF  
  
Segment address of the environment:          02D6  
  
Command line tail:                          empty tail  
Content of the environment area:             PATH=Z:\  
COMSPEC=Z:\COMMAND.COM  
BLASTER=A220 I7 D1 H5 T6  
  
Path:                                         C:\LR2.COM ♥  
The called program has been loaded.:)  
Input symbol:♥  
Normal completion.  
C:\>
```

Рисунок 4 - Результат запуска программы при вводе Ctrl-C

Шаг 4.

Был выполнен запуск программы, когда текущим каталогом является какой-либо другой каталог, отличный от того, в котором содержатся разработанные программные модули.

```
C:\>\OTHER\LR6.EXE

Memory redistributed successfully.:)

Segment address of the inaccessible memory: 9FFF

Segment address of the environment:          02D6

Command line tail:                          empty tail
Content of the environment area:             PATH=Z:\
COMSPEC=Z:\COMMAND.COM
BLASTER=A220 I7 D1 H5 T6

Path:                                         C:\OTHER\LR2.COM b
The called program has been loaded.:)
Input symbol:b
Normal completion.
C:\>
```

Рисунок 5 – Результат запуска из другого каталога. Введенный символ b

```
C:\>\OTHER\LR6.EXE

Memory redistributed successfully.:)

Segment address of the inaccessible memory: 9FFF

Segment address of the environment:          02D6

Command line tail:                          empty tail
Content of the environment area:             PATH=Z:\
COMSPEC=Z:\COMMAND.COM
BLASTER=A220 I7 D1 H5 T6

Path:                                         C:\OTHER\LR2.COM ♥
The called program has been loaded.:)
Input symbol:♥
Normal completion.
C:\>_
```

Рисунок 6 – Результат запуска из другого каталога при вводе Ctrl-C

Шаг 5.

Был выполнен запуск отлаженной программы, когда модули находятся в разных каталогах.

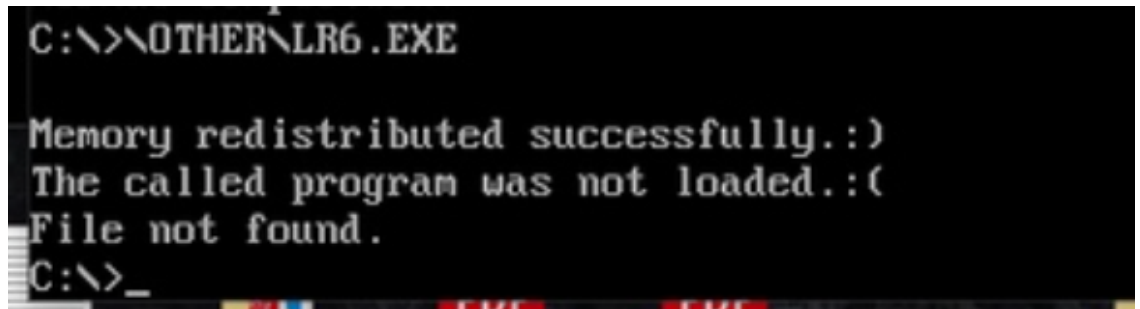


Рисунок 7 – Результат запуска программы, для модулей из разных директорий

Ответы на контрольные вопросы.

1) Как реализовано прерывание Ctrl-C?

1. Определяется нажато ли сочетание клавиш Ctrl-C (код 03 в кольцевом буфере клавиатуры).
2. Если нажато, то вызывается прерывание int 23h, завершающее работу текущей программы.
3. Если пользователю не нужно завершать программу, при нажатии данной комбинации, то подключается собственный обработчик для обработки прерывания.

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

Код завершения 0 – это код нормального завершения программы. Вызываемая программа, в таком случае, заканчивается в точке вызова функции 4Ch (прерывания int 21h).

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

Вызываемая программа, в таком случае, заканчивается в точке вызова функции 01h (прерывания int 21h) – ввод символа с клавиатуры, так

как при обнаружении Ctrl-C вызывается прерывание int 23h, незамедлительно завершающее работу программы.

Выводы.

В результате выполнения лабораторной работы была изучена возможность построения загрузочного модуля динамической структуры. Была написана программа, реализовывающая интерфейс вызываемого и вызывающего модуля. В качестве вызываемого – использовался модифицированный код из второй лабораторной работы.

ПРИЛОЖЕНИЕ А

Содержимое файлы LR6.ASM

```
COMMENT *
Максимова Анастасия, группа 8383 - лабораторная 6
*
;-----
Astack SEGMENT STACK
        DW 256 dup(?)
Astack ENDS
;-----
DATA    SEGMENT                                ;ДАННЫЕ
EOF      EQU      '$'
SETPR    EQU      15

        KEEP_PSP      DW      0
        KEEP_AX        DW      0
        KEEP_SP        DW      0
        KEEP_SS        DW      0
        THIS_          DW      0
                                ;/

        PROG_NAME      DB      'lr2.com', 0
        PROG_PATH      DB      64 dup(?)

        INPUT          DB      0DH, 0AH, 'Input symbol: ', EOF

                                ;memory
        ERROR_MEM      DB      0DH, 0AH, 'Memory block size not
changed.:(', EOF
        NOT_ERROR_MEM  DB      0DH, 0AH, 'Memory redistributed
successfully.:)', EOF
        ERROR_MEM_7    DB      0DH, 0AH, 'Memory control
block destroyed.', EOF
```

```

ERROR_MEM_8      DB      0DH, 0AH, 'Not enough memory
to execute function.', EOF
ERROR_MEM_9      DB      0DH, 0AH, 'Invalid memory
block address.', EOF
ERROR_MEM_FLAG   DB      0

;load
NOT_LOADED      DB      0DH, 0AH, 'The called program was
not loaded.:(', EOF
LOADED          DB      0DH, 0AH, 'The called program has
been loaded.:)', EOF

LOAD_ERROR_1     DB      0DH, 0AH, 'Function number is
incorrect.', EOF
LOAD_ERROR_2     DB      0DH, 0AH, 'File not found.', EOF
LOAD_ERROR_5     DB      0DH, 0AH, 'Disk error.', EOF
LOAD_ERROR_8     DB      0DH, 0AH, 'Insufficient memory',
EOF
LOAD_ERROR_10    DB      0DH, 0AH, 'Wrong environment
string.', EOF
LOAD_ERROR_11    DB      0DH, 0AH, 'Format is not correct',
EOF

;the end
COMPL_PROC_0     DB      0DH, 0AH, 'Normal completion.', EOF
COMPL_PROC_1     DB      0DH, 0AH, 'Completion by Ctrl-
Break.', EOF
COMPL_PROC_2     DB      0DH, 0AH, 'Device error
termination.', EOF
COMPL_PROC_3     DB      0DH, 0AH, 'Completion by function
31h.', EOF

;parameters

```

```

        BLOCK_PARAM    DW      ?                ; сегментный адрес
среды
                                DD      ?                ; сегмент
и
смещение командной строки
                                DD      ?                ; сегмент
и
смещение первого FCB
                                DD      ?                ; сегмент
и
смещение второго FCB

```

```

        DATA_END_FLAG DB      0
DATA    ENDS
;-----
CODE    SEGMENT
        ASSUME CS:CODE, DS:DATA, SS:AStack, ES:NOTHING
;-----
PRINTF    PROC NEAR
        push    AX

        mov     AH, 09h
        int     21h

        pop     AX
        retn

PRINTF    ENDP
;-----
FREE_MEMORY    PROC NEAR                                ; уменьшить
отведенный блок памяти
        push    AX
        push    BX
        push    CX
        push    DX

        mov     BX, offset CODE_END_FLAG    ; определить
размер памяти необходимый программе

```

```

mov      AX, offset DATA_END_FLAG    ;положить в
BX число параграфов
add      BX, AX
mov      CL, 4                        ;деление
на 16
shr      BX, CL
add      BX, 100h
;дополнительная память

xor      AL, AL
mov      AH, 4Ah
;4Ah - изменить размер блока памяти
int      21h                          ;BX
- новый размер в 16байтных параграфах

;если освобождение памяти не может быть выполнено
;устанавливается флаг переноса CF = 1 и AX
заносится код ошибки (7,8,9)

jnc      NOT_ERROR
;переход, если перенос не установлен
mov      DX, offset ERROR_MEM
call     PRINTF
call     CATCH_ERROR

NOT_ERROR:
mov      DX, offset NOT_ERROR_MEM
call     PRINTF

pop      DX
pop      CX
pop      BX
pop      AX
retn

```

```

FREE_MEMORY      ENDP
;-----
CATCH_ERROR      PROC NEAR
                  push    AX
                  push    DX

                  mov     ERROR_MEM_FLAG, 1

                  cmp     AX, 7
                  jne     NEXT_ERROR
                  mov     DX, offset ERROR_MEM_7
                  jmp     END_CATCH

NEXT_ERROR:
                  cmp     AX, 8
                  jne     NEXT_ERROR2
                  mov     DX, offset ERROR_MEM_8
                  jmp     END_CATCH

NEXT_ERROR2:
                  cmp     AX, 9
                  mov     DX, offset ERROR_MEM_9

END_CATCH:
                  call    PRINTF

                  pop     DX
                  pop     AX
                  retn

CATCH_ERROR      ENDP
;-----

FILL_PARAM_BLOCK PROC NEAR
                  push    AX

```

```

                                push    ES

                                mov     AX, ES:[002Ch]
; сегментный адрес среды

                                mov     BLOCK_PARAM, AX

                                mov     AX, ES
                                mov     BLOCK_PARAM + 2, AX
                                mov     BLOCK_PARAM + 4, 0080h ; число
символов в хвосте командной строки

```

```

                                pop     ES
                                pop     AX
                                retn

```

FILL_PARAM_BLOCK ENDP

;-----

```

GET_PROG_PATH      PROC NEAR
                    push    AX
                    push    ES
                    push    DI
                    push    SI
                    push    CX

```

```

                    mov     ES, ES:[002Ch]
                    xor     DI, DI
                    xor     SI, SI

```

FIND_PATH:

```

                    mov     AX, ES:[DI]
                    inc     DI
                    cmp     AX, 0000h ; если встретили два нулевых
байта подряд

                    je      NEXT_STEP
                    jmp     FIND_PATH

```

NEXT_STEP:

```
inc    DI
mov     AL, ES:[DI]
cmp     AL, 01h      ;после  располагается
маршрут
jne     NEXT_STEP
add     DI, 2

mov     SI, offset PROG_PATH
```

WRITTING:

```
mov     AL, ES:[DI]
cmp     AL, 00h      ;the end
je      FIND_NAME
;последний слэш запоминаем
cmp     AL, '\'
jne     CONTINUE_
mov     THIS_, SI
```

CONTINUE_:

```
mov     [SI], AL
inc     DI
inc     SI
jmp     WRITTING
```

FIND_NAME:

```
mov     SI, THIS_
inc     SI

mov     DI, offset PROG_NAME
mov     CX, 7

xor     AL, AL
```

```

REPEAT_:
    mov     AL, [DI]
    mov     [SI], AL
    inc     DI
    inc     SI
    loop    REPEAT_

END_FIND:
    pop     CX
    pop     SI
    pop     DI
    pop     ES
    pop     AX
    retn

GET_PROG_PATH      ENDP
;-----
LOADER      PROC  NEAR
;загрузчик
    push    AX
    push    BX
    push    DX
    push    DS
    push    ES

    mov     AX, DATA
    mov     ES, AX

    mov     DX, offset PROG_PATH
    mov     BX, offset BLOCK_PARAM

    mov     KEEP_SP, SP
;для восстановления
    mov     KEEP_SS, SS

    mov     AX, 4B00h

```



```

int      21h

mov      KEEP_AX, AX
mov      AX, DATA
mov      DS, AX

mov      SS, KEEP_SS
mov      SP, KEEP_SP
mov      AX, KEEP_AX

jnc      SUCCESSFULLY
;переход, если перенос не установлен

mov      DX, offset NOT_LOADED          ;CF = 1
call     PRINTF
call     FIND_ERROR
jmp      END_LOADER

SUCCESSFULLY:
;если CF = 0, то обрабатываем ее завершение
mov      DX, offset LOADED
call     PRINTF
call     COMPL_PROC                      ;CF
= 0

END_LOADER:
pop      ES
pop      DS
pop      DX
pop      BX
pop      AX
retn

LOADER    ENDP
;-----

```

```

COMPL_PROC PROC NEAR                                ;CF = 0
    push    AX
    push    DI
    push    DX

    mov     AX, 4D00h                                ;AH
- причина, AL - код завершения
    int     21h
    mov     DI, offset INPUT
    add     DI, SETPR
    mov     [DI], AL
    mov     DX, offset INPUT

    call    PRINTF

    xor     DX, DX
    cmp     AH, 0
    jne     NOT_0
    mov     DX, offset COMPL_PROC_0
    jmp     COMPL_END

NOT_0:
    cmp     AH, 1
    jne     NOT_1
    mov     DX, offset COMPL_PROC_1
    jmp     COMPL_END

NOT_1:
    cmp     AH, 2
    jne     NOT_2
    mov     DX, offset COMPL_PROC_2
    jmp     COMPL_END

NOT_2:
    cmp     AH, 3
    mov     DX, offset COMPL_PROC_3

```

```

COMPL_END:
        call    PRINTF

        pop     DX
        pop     DI
        pop     AX
        retn

COMPL_PROC ENDP

;-----
FIND_ERROR PROC NEAR                                ;CF = 1
        push    AX
        push    DX

        cmp     AX, 1
        jne     NEXT2
        mov     DX, offset LOAD_ERROR_1
        jmp     FIND_END

NEXT2:
        cmp     AX, 2
        jne     NEXT5
        mov     DX, offset LOAD_ERROR_2
        jmp     FIND_END

NEXT5:
        cmp     AX, 5
        jne     NEXT8
        mov     DX, offset LOAD_ERROR_5
        jmp     FIND_END

NEXT8:
        cmp     AX, 8
        jne     NEXT10
        mov     DX, offset LOAD_ERROR_8
        jmp     FIND_END

NEXT10:

```

```

        cmp     AX, 10
        jne     NEXT11
        mov     DX, offset LOAD_ERROR_10
        jmp     FIND_END
NEXT11:
        cmp     AX, 11
        mov     DX, offset LOAD_ERROR_11
FIND_END:
        call    PRINTF
        pop     DX
        pop     AX
        retn
FIND_ERROR ENDP
;-----
BEGIN      PROC  FAR

        push    DS
        xor     AX, AX
        push    AX

        mov     AX, DATA
        mov     DS, AX
        mov     KEEP_PSP, ES

        call    FREE_MEMORY

        cmp     ERROR_MEM_FLAG, 1
свободной памяти      ;освобождение

        jne     CONTINUE
        jmp     END_BEGIN

CONTINUE:
        call    FILL_PARAM_BLOCK
параметров      ;блок

```

```

                                call    GET_PROG_PATH
                                call    LOADER

END_BEGIN:
                                xor     AL, AL                                ; Выход
B DOS
                                mov     AH, 4Ch
                                int     21h

BEGIN      ENDP
;-----
CODE_END_FLAG:
CODE      ENDS
                                END      BEGIN

```

ПРИЛОЖЕНИЕ Б

Содержимое файлы LR2.ASM

COMMENT @

Максимова Анастасия, группа 8383, 2 лабораторная

@

CODE

SEGMENT

ASSUME CS:CODE, DS:CODE, ES:NOTHING, SS:NOTHING

ORG 100H

START: JMP

MAIN

EOF

EQU

'\$'

SETPRECISION

EQU

50

;ДАННЫЕ

ADDRESS_MEMORY

DB

0DH, 0AH, 0AH, 'Segment

address of the inaccessible memory:

', EOF

ADDRESS_ENVIRONMENT

DB

0DH, 0AH, 0AH, 'Segment

address of the environment:

', EOF

TAIL_STRING

DB

0DH, 0AH, 0AH, 'Command

line tail:

', EOF

WHERE_MY_TAIL

DB

'empty tail', EOF

CONTENT_AREA

DB

0DH, 0AH, 'Content of

the environment area:

', EOF

ENDL

DB

0DH, 0AH, EOF

WAY

DB

0DH, 0AH, 'Path:

', EOF

;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 переводится в два символа шестн. 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
        ;в AL - старшая цифра
        pop     CX
        ;в AH - младшая
        ret
BYTE_TO_HEX    ENDP
;-----
WRD_TO_HEX     PROC NEAR
        ;перевод в 16 с/с 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

```

```

;-----

```

```

PRINTF      PROC NEAR
    push     AX
    mov      AH, 09h
    int      21h
    pop      AX
    retn

```

```

PRINTF      ENDP

```

```

;-----

```

```

GET_ADDRESS_PR      PROC NEAR
    push     AX
    push     DI
    push     DX

```

;Первое - сегментный адрес недоступной памяти,

взятый из PSP

```

mov      AX, DS:[0002h]
mov      DI, offset ADDRESS_MEMORY
add      DI, SETPRECISION
call     WRD_TO_HEX

```


программ

```
mov     DX, offset ADDRESS_MEMORY
call PRINTF
```

```
sub     AX, AX
sub     DI, DI
sub     DX, DX
```

;Второе - сегментный адрес среды, передаваемый

```
mov     AX, DS:[002Ch]
mov     DI, offset ADDRESS_ENVIRONMENT
add     DI, SETPRECISION
call WRD_TO_HEX
```

```
mov     DX, offset ADDRESS_ENVIRONMENT
call PRINTF
```

```
pop     DX
pop     DI
pop     AX
```

```
retn
```

```
GET_ADDRESS_PR      ENDP
```

;-----

```
GET_TAIL_PR      PROC NEAR ;если хвост не пустой - выводим его
```

```
push     DI
push     AX
push     DX
```

```
sub     DI, DI
sub     AX, AX
```

REPEAT:

```
mov     AL, DS:[0081h + DI]
```

```
push    AX
```

```
sub     DX, DX
```

```
mov     DL, AL
```

```
mov     AH, 02h
```

```
int     21h
```

```
pop     AX
```

```
inc     DI
```

```
loop    REPEAT
```

```
pop     DX
```

```
pop     AX
```

```
pop     DI
```

```
retn
```

```
GET_TAIL_PR      ENDP
```

```
;-----
```

;Третье - кол-во символов в хвосте, если 0, то печатаем
предупреждение

```
GET_NUMBER_CHAR  PROC NEAR
```

```
push    CX
```

```
push    DX
```

```
mov     DX, offset TAIL_STRING
```

```
call    PRINTF
```

```
sub     CX, CX
```

```
mov     CL, DS:[0080h] ;количество символов в хвосте
```

```
cmp     CL, 00h
```

```

        je          EMPTY
        call GET_TAIL_PR
        jmp         EXIT

EMPTY:   mov         DX, offset WHERE_MY_TAIL
        call PRINTF

EXIT:

        pop        DX
        pop        CX

        retn

GET_NUMBER_CHAR ENDP
;-----
;четвертое - пятое - содержимое области среды и путь
GET_CONTENT_AREA PROC NEAR
        push AX
        push DX
        push DI

        mov         DX, offset CONTENT_AREA
        call PRINTF

        sub  DI, DI
        sub  AX, AX

        mov ES, DS:[002Ch]

CICLE_READ:
        mov         AL, ES:[DI]
        cmp         AL, 00h           ;первый нуль
        je          NEW_STRING       ;печатаем новую строку

```

```

push    AX
sub     DX, DX
mov     DL, AL
mov     AH, 02h           ;печатаем символ
int     21h
pop     AX

inc     DI
jmp     CICLE_READ

```

NEW_STRING:

```

mov     DX, offset ENDL
call    PRINTF
inc     DI

mov     AL, ES:[DI]
cmp     AL, 00h           ;второй нуль
jne     CICLE_READ

mov     DX, offset ENDL
call    PRINTF

```

FIND_PATH: inc DI

```

mov     AL, ES:[DI]
cmp     AL, 01h
jne     FIND_PATH
add     DI, 2

```

```

mov     DX, offset WAY
call    PRINTF

```

COUT_PATH: cmp AL, 00h
je BYE

```

mov     AL, ES:[DI]
push    AX
sub     DX, DX
mov     DL, AL
mov     AH, 02h      ;печатаем символ
int     21h
pop     AX

inc     DI
jmp     COUT_PATH

```

BYE:

```

pop     DI
pop     DX
pop     AX

```

```

retn

```

GET_CONTENT_AREA ENDP

;-----

MAIN:

```

call    GET_ADDRESS_PR      ; 1 и 2 задания
call    GET_NUMBER_CHAR     ; 3 задание
call    GET_CONTENT_AREA; 4 и 5 задание

```

;выход в DOS

```

sub     AL, AL

```

```

mov     AH, 01h              ;модификация программы
int     21h                  ;01h - ввод символа с

```

клавиатуры

```

mov     AH, 4Ch
int     21h

```

CODE ENDS

END START

;конец модуля, START - точка входа