

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

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

Студент гр. 8382

\_\_\_\_\_

Янкин Д.О.

Преподаватель

\_\_\_\_\_

Ефремов М.А.

Санкт-Петербург

2020

### **Цель работы.**

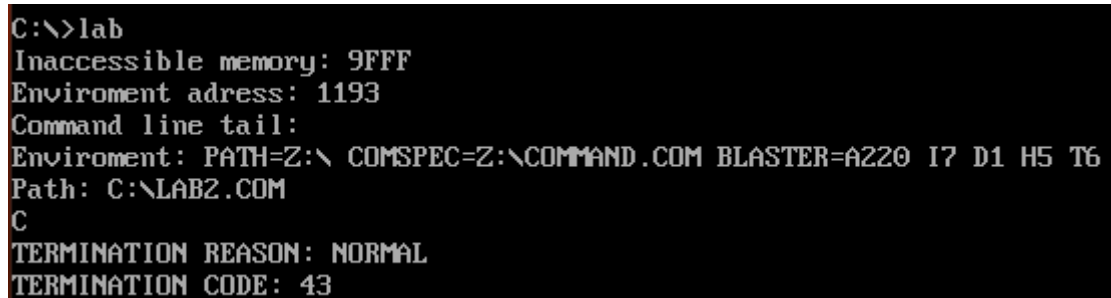
Исследование возможности построения загрузочного модуля динамической структуры. Исследуется интерфейс между вызывающим и вызываемым модулями по управлению и по данным.

### **Ход работы.**

Был написан программный модуль типа .EXE, который:

- 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.

Запущена программа, когда оба модуля находятся в одной директории, введен символ C. Смотреть рисунок 1.



```
C:\>lab
Inaccessible memory: 9FFF
Environment address: 1193
Command line tail:
Environment: PATH=Z:\ COMSPEC=Z:\COMMAND.COM BLASTER=A220 I7 D1 H5 T6
Path: C:\LAB2.COM
C
TERMINATION REASON: NORMAL
TERMINATION CODE: 43
```

Рисунок 1. Результат работы программы при модулях в одном каталоге

Запущена программа, когда оба модуля находятся в одной директории, введена комбинация Ctrl-C. Смотреть рисунок 2.

```
C:\>lab
Inaccessible memory: 9FFF
Environment address: 1193
Command line tail:
Environment: PATH=Z:\ COMSPEC=Z:\COMMAND.COM BLASTER=A220 I7 D1 H5 T6
Path: C:\LAB2.COM
♥
TERMINATION REASON: NORMAL
TERMINATION CODE: 03
```

Рисунок 2. Результат работы программы при модулях в одном каталоге и вводе Ctrl-C

Запущена программа, когда оба модуля находятся в другой директории, введен символ C. Смотреть рисунок 3.

```
C:\>TEMP\LAB.EXE
Inaccessible memory: 9FFF
Environment address: 1193
Command line tail:
Environment: PATH=Z:\ COMSPEC=Z:\COMMAND.COM BLASTER=A220 I7 D1 H5 T6
Path: C:\TEMP\LAB2.COM
C
TERMINATION REASON: NORMAL
TERMINATION CODE: 63
```

Рисунок 3. Результат работы программы при модулях в другом каталоге

Запущена программа, когда оба модуля находятся в другой директории, введена комбинация Ctrl-C. Смотреть рисунок 4.

```
C:\>TEMP\LAB.EXE
Inaccessible memory: 9FFF
Environment address: 1193
Command line tail:
Environment: PATH=Z:\ COMSPEC=Z:\COMMAND.COM BLASTER=A220 I7 D1 H5 T6
Path: C:\TEMP\LAB2.COM
♥
TERMINATION REASON: NORMAL
TERMINATION CODE: 03
```

Рисунок 4. Результат работы программы при модулях в другом каталоге и вводе Ctrl-C

Запущена программа, когда модули были в разных директориях. Смотреть рисунок 5.



```
C:\>TEMP\LAB.EXE  
PROGRAM LOAD ERROR: FILE NOT FOUND
```

Рисунок 5. Результат работы программы при модулях в разных каталогах

### **Контрольные вопросы.**

- 1) Как реализовано прерывание Ctrl-C

При обнаружении в буфере клавиатуры сочетания Ctrl-C вызывается прерывание int 23h, приводящее к завершению текущего процесса.

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

В точке вызова функции 4Ch прерывания int 21h.

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

При считывании символа функцией 01h прерывания int21h.

### **Выводы.**

В ходе лабораторной работы была исследована возможность построения загрузочного модуля динамической структуры.

# ПРИЛОЖЕНИЕ А

## ИСХОДНЫЙ КОД ПРОГРАММЫ LAB.ASM

```
ASTACK    SEGMENT    STACK
```

```
        DW 100 DUP(0)
```

```
ASTACK    ENDS
```

```
DATA SEGMENT
```

```
        MEM_ALLOC_ERROR_07_MESSAGE    DB    "MEM ALLOC ERROR: MCB  
WAS DESTROYED$"
```

```
        MEM_ALLOC_ERROR_08_MESSAGE    DB    "MEM ALLOC ERROR: NOT  
ENOUGH MEMORY$"
```

```
        MEM_ALLOC_ERROR_09_MESSAGE    DB    "MEM ALLOC ERROR: WRONG  
MCB ADDRESS$"
```

```
        LOAD_ERROR_01_MESSAGE          DB    "PROGRAM LOAD ERROR:  
WRONG FUNCTION NUMBER$"
```

```
        LOAD_ERROR_02_MESSAGE          DB    "PROGRAM LOAD ERROR:  
FILE NOT FOUND$"
```

```
        LOAD_ERROR_05_MESSAGE          DB    "PROGRAM LOAD ERROR:  
DISC ERROR$"
```

```
        LOAD_ERROR_08_MESSAGE          DB    "PROGRAM LOAD ERROR: NOT  
ENOUGH MEMORY$"
```

```
        LOAD_ERROR_10_MESSAGE          DB    "PROGRAM LOAD ERROR:  
WRONG ENVIROMENT STRING$"
```

```
        LOAD_ERROR_11_MESSAGE          DB    "PROGRAM LOAD ERROR:  
WRONG FORMAT$"
```

```
        TERMINATION_REASON_00_MESSAGE  DB    "TERMINATION REASON:  
NORMAL$"
```

```
        TERMINATION_REASON_01_MESSAGE  DB    "TERMINATION REASON: BY  
CTRL-BREAK$"
```

```
        TERMINATION_REASON_02_MESSAGE  DB    "TERMINATION REASON:  
DEVICE ERROR$"
```

```

        TERMINATION_REASON_03_MESSAGE    DB    "TERMINATION REASON:
RESIDENT HAS BEEN SET$"

        TERMINATION_CODE_MESSAGE          DB    "TERMINATION CODE:  $"

        PARAMETER_BLOCK                   DB    0
                                           DD    0
                                           DD    0
                                           DD    0

        PATH                              DB    128 DUP(0)

        KEEP_SS                           DW    0
        KEEP_SP                           DW    0

DATA ENDS

```

#### CODE SEGMENT

```

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

```

```

        KEEP_DS                           DW    0

```

```

TETR_TO_HEX PROC near

```

```

; младшая шестн. цифра AL в шестн. цифру ASCII в AL

```

```

        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 шестн. числа ASCII 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

```

; Вывод строки по DS:DX. Логично

```
PRINT_STRING PROC
```

```
    push ax
```

```
    mov ah, 09h
```

```
    int 21h
```

```
    pop ax
```

```
    ret
```

```
PRINT_STRING ENDP
```

; Перевод строки

```
PRINT_ENDL PROC
```

```
    push ax
```

```
    push dx
```

```
    mov dl, 13
```

```
    mov ah, 02h
```

```
    int 21h
```

```
    mov dl, 10
```

```
    int 21h
```

```
    pop dx
```

```

        pop ax
        ret
PRINT_ENDL ENDP

```

```

FREE_EXTRA_MEMORY PROC

```

```

        push ax
        push bx

```

```

        ; В конце сегмента, вроде, стек должен валяться,
        ; поэтому 65536/16 = 4096 параграфов

```

```

        mov  bx, 4096
        mov  ah, 4Ah
        int  21h

```

```

        jnc      FREE_EXTRA_MEMORY_RET

```

```

        cmp  ax, 7
        je    MEM_ALLOC_ERROR_07
        cmp  ax, 8
        je    MEM_ALLOC_ERROR_08
        cmp  ax, 9
        je    MEM_ALLOC_ERROR_09

```

```

MEM_ALLOC_ERROR_07:

```

```

        mov  dx, offset MEM_ALLOC_ERROR_07_MESSAGE
        call PRINT_STRING
        call PRINT_ENDL
        jmp  MEM_ALLOC_ERROR_EXIT

```

```

MEM_ALLOC_ERROR_08:

```

```

        mov  dx, offset MEM_ALLOC_ERROR_08_MESSAGE
        call PRINT_STRING
        call PRINT_ENDL

```



```
jmp      MEM_ALLOC_ERROR_EXIT
```

```
MEM_ALLOC_ERROR_09:
```

```
mov  dx, offset MEM_ALLOC_ERROR_09_MESSAGE
```

```
call PRINT_STRING
```

```
call PRINT_ENDL
```

```
jmp      MEM_ALLOC_ERROR_EXIT
```

```
MEM_ALLOC_ERROR_EXIT:
```

```
mov      ax, 4C00h
```

```
int      21h
```

```
FREE_EXTRA_MEMORY_RET:
```

```
pop  bx
```

```
pop  ax
```

```
ret
```

```
FREE_EXTRA_MEMORY ENDP
```

```
PREPARE_BLOCK PROC
```

```
push ax
```

```
push bx
```

```
push dx
```

```
mov  bx, offset PARAMETER_BLOCK
```

```
mov      dx, es
```

```
; Сегментный адрес среды
```

```
mov      ax, 0
```

```
mov  [bx], ax
```

```
; Сегмент и смещение командной строки
```

```
mov  [bx+2], dx
```

```
mov      ax, 80h
```

```

mov    [bx+4], ax

; Сегмент и смещение первого FCB
mov    [bx+6], dx
mov     ax, 5Ch
mov    [bx+8], ax

; Сегмент и смещение второго FCB
mov    [bx+10], dx
mov     ax, 6Ch
mov    [bx+12], ax

pop     dx
pop    bx
pop    ax
ret

PREPARE_BLOCK ENDP

```

```

RUN_PROGRAM PROC

; Взятие пути программы
mov    es, es:[2Ch]
mov    si, 0

ENVIROMENT_STRING_LOOP:
mov    dl, es:[si]
cmp    dl, 00h
je     ENVIROMENT_STRING_ENDL
inc    si
jmp    ENVIROMENT_STRING_LOOP

ENVIROMENT_STRING_ENDL:
inc    si
mov    dl, es:[si]

```

```

cmp    dl, 00h
jne    ENVIROMENT_STRING_LOOP
add    si, 03h

```

```

lea    di, PATH

```

```

PATH_LOOP:
mov     dl, es:[si]
cmp     dl, 00h
je      PATH_ENDL
mov     [di], dl
inc     di
inc     si
jmp     PATH_LOOP

```

```

PATH_ENDL:
sub     di, 7
mov     [di], byte ptr 'L'
mov     [di+1], byte ptr 'A'
mov     [di+2], byte ptr 'B'
mov     [di+3], byte ptr '2'
mov     [di+4], byte ptr '.'
mov     [di+5], byte ptr 'C'
mov     [di+6], byte ptr 'O'
mov     [di+7], byte ptr 'M'
mov     [di+8], byte ptr 00h

```

```

; Подготовка к вызову
mov     KEEP_DS, ds
mov     KEEP_SS, ss
mov     KEEP_SP, sp

```

```

; Адрес блока параметров

```

```
mov     ax, ds
mov     es, ax
mov     bx, offset PARAMETER_BLOCK
```

```
; Адрес пути до программы
mov     dx, offset PATH
```

```
; Вызов
mov     ax, 4B00h
int     21h
```

```
mov     ds, KEEP_DS
mov     ss, KEEP_SS
mov     sp, KEEP_SP
call    PRINT_ENDL
```

```
jnc     LOAD_OK
```

```
LOAD_ERROR:
```

```
cmp     ax, 1
je      LOAD_ERROR_01
cmp     ax, 2
je      LOAD_ERROR_02
cmp     ax, 5
je      LOAD_ERROR_05
cmp     ax, 8
je      LOAD_ERROR_08
cmp     ax, 10
je      LOAD_ERROR_10
cmp     ax, 11
je      LOAD_ERROR_11
```

```
LOAD_ERROR_01:
    lea     dx, LOAD_ERROR_01_MESSAGE
    jmp     LOAD_ERROR_PRINT
```

```
LOAD_ERROR_02:
    lea     dx, LOAD_ERROR_02_MESSAGE
    jmp     LOAD_ERROR_PRINT
```

```
LOAD_ERROR_05:
    lea     dx, LOAD_ERROR_05_MESSAGE
    jmp     LOAD_ERROR_PRINT
```

```
LOAD_ERROR_08:
    lea     dx, LOAD_ERROR_08_MESSAGE
    jmp     LOAD_ERROR_PRINT
```

```
LOAD_ERROR_10:
    lea     dx, LOAD_ERROR_10_MESSAGE
    jmp     LOAD_ERROR_PRINT
```

```
LOAD_ERROR_11:
    lea     dx, LOAD_ERROR_11_MESSAGE
```

```
LOAD_ERROR_PRINT:
    call    PRINT_STRING
    call    PRINT_ENDL
```

```
mov  ax,  4C00h
int  21h
```

```
LOAD_OK:
    mov     ax, 4D00h
```

```

int          21h

cmp          ah, 0
je           TERMINATION_REASON_00
cmp          ah, 1
je           TERMINATION_REASON_01
cmp          ah, 2
je           TERMINATION_REASON_02
cmp          ah, 3
je           TERMINATION_REASON_03

TERMINATION_REASON_00:
lea          dx, TERMINATION_REASON_00_MESSAGE
call PRINT_STRING
call PRINT_ENDL

call BYTE_TO_HEX
mov          di, offset TERMINATION_CODE_MESSAGE
add          di, 18
mov          [di], al
mov          [di+1], ah
mov          dx, offset TERMINATION_CODE_MESSAGE
call PRINT_STRING
call PRINT_ENDL

ret

TERMINATION_REASON_01:
lea          dx, TERMINATION_REASON_01_MESSAGE
jmp          TERMINATION_REASON_PRINT

TERMINATION_REASON_02:
lea          dx, TERMINATION_REASON_02_MESSAGE
jmp          TERMINATION_REASON_PRINT

```

```

    TERMINATION_REASON_03:
    lea        dx, TERMINATION_REASON_03_MESSAGE

    TERMINATION_REASON_PRINT:
    call PRINT_STRING
    call PRINT_ENDL

    ret
RUN_PROGRAM ENDP

MAIN PROC
    mov        ax, DATA
    mov        ds, ax

    call FREE_EXTRA_MEMORY
    call PREPARE_BLOCK
    call RUN_PROGRAM

    mov        ax, 4C00h
    int        21h
MAIN ENDP
CODE ENDS

END MAIN

```

## ПРИЛОЖЕНИЕ Б

### ИСХОДНЫЙ КОД ПРОГРАММЫ LAB2.ASM

TESTPC SEGMENT

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

ORG 100H

START: JMP BEGIN

; ДАННЫЕ

INACCESSIBLE\_MEMORY\_MESSAGE                db 'Inaccessible memory: ',  
'\$'

INACCESSIBLE\_MEMORY                        db '0000', 13, 10, '\$'

ENVIROMENT\_ADRESS\_MESSAGE                db 'Enviroment adress: ', '\$'

ENVIROMENT\_ADRESS                         db '0000', 13, 10, '\$'

COMMAND\_LINE\_TAIL\_MESSAGE                db 'Command line tail: ', '\$'

ENVIROMENT\_MESSAGE                        db 'Enviroment: ', '\$'

PATH\_MESSAGE                               db 'Path: ', '\$'

; ПРОЦЕДУРЫ

;-----

TETR\_TO\_HEX PROC near

; младшая шестн. цифра AL в шестн. цифру ASCII

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 шестн. числа ASCII 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
    ;xchg al, ah          ;; а теперь наоборот!
    ret
BYTE_TO_HEX ENDP

```

```

;-----
WRD_TO_HEX PROC near
; перевод в 16 с/с 16-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 с/с, 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_1
    or AL,30h
    mov [SI],AL
    dec si
end_1:    pop DX
        pop CX
        ret
BYTE_TO_DEC ENDP

```

```

;-----
PRINT_STRING PROC near
; Просто выводит строку с уже указанным в dx смещением, очень
сложная функция
    push ax

```

```

        mov ah, 09h
        int 21h

        pop ax
        ret
PRINT_STRING ENDP

```

```

;-----
PRINT_WORD PROC near
; Выводит регистр AX
        push ax
        push dx

        mov dl, ah
        mov ah, 02h
        int 21h

        mov dl, al
        int 21h

        pop dx
        pop ax
        ret
PRINT_WORD ENDP

```

```

;-----
PRINT_ENDL PROC near
; Выводит 13, 10
        push ax
        push dx

        mov dl, 13
        mov ah, 02h
        int 21h

```

```

        mov dl, 10
        int 21h

        pop dx
        pop ax
        ret
PRINT_ENDL ENDP

```

```

;-----

```

```

; КОД

```

```

BEGIN:

```

```

        ; Недоступная память

```

```

        mov dx, offset INACCESSIBLE_MEMORY_MESSAGE
        call PRINT_STRING

```

```

        mov bx, 2h
        mov ax, [bx]

```

```

        mov di, offset INACCESSIBLE_MEMORY
        add di, 3
        call WRD_TO_HEX

```

```

        mov dx, offset INACCESSIBLE_MEMORY
        call PRINT_STRING

```

```

        ; Сегментный адрес среды, передаваемый программе
        mov dx, offset ENVIROMENT_ADRESS_MESSAGE
        call PRINT_STRING

```

```

        mov bx, 2Ch

```

```
mov ax, [bx]
```

```
mov di, offset ENVIROMENT_ADRESS
```

```
add di, 3
```

```
call WRD_TO_HEX
```

```
mov dx, offset ENVIROMENT_ADRESS
```

```
call PRINT_STRING
```

```
; Хвост командной строки
```

```
mov dx, offset COMMAND_LINE_TAIL_MESSAGE
```

```
call PRINT_STRING
```

```
mov bx, 80h
```

```
mov ch, 0
```

```
mov cl, [bx]
```

```
cmp cx, 0
```

```
je COMMAND_LINE_TAIL_END
```

```
mov bx, 81h
```

```
mov ah, 02h
```

```
COMMAND_LINE_TAIL_LOOP:
```

```
mov dl, [bx]
```

```
int 21h
```

```
add bx, 1
```

```
loop COMMAND_LINE_TAIL_LOOP
```

```
COMMAND_LINE_TAIL_END:
```

```
call PRINT_ENDL
```

```
; Область среды
```

```
mov dx, offset ENVIROMENT_MESSAGE
```

```

        call PRINT_STRING

        mov bx, 2Ch
        mov es, [bx]
        mov bx, 0

        mov ah, 02h
ENVIROMENT_LOOP:
        mov dl, 0
        cmp dl, es:[bx]
        je ENVIROMENT_END
ENVIROMENT_VARIABLE_LOOP:
        mov dl, es:[bx]
        int 21h
        add bx, 1
        cmp dl, 0
        jne ENVIROMENT_VARIABLE_LOOP
        je ENVIROMENT_LOOP
ENVIROMENT_END:
        call PRINT_ENDL

; Путь загружаемого модуля
mov dx, offset PATH_MESSAGE
call PRINT_STRING

add bx, 3
PATH_LOOP:
        mov dl, es:[bx]
        int 21h
        add bx, 1
        cmp dl, 0
        jne PATH_LOOP
        call PRINT_ENDL

```

```

mov      ah, 01h
int      21h
mov      ah, 4Ch
int      21h

TESTPC   ENDS

END START ; конец модуля, START – точка входа

```