

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

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

Студентка гр. 0382

\_\_\_\_\_

Морева Е.С.

Преподаватель

\_\_\_\_\_

Ефремов М.А.

Санкт-Петербург

2022

## Постановка задачи

### Цель работы.

Исследование возможности построения загрузочного модуля динамической структуры.

### Сведения о функциях и структурах данных управляющей программы.

Процедура	Описание
MODULE_PATH	Получение пути до вызываемого модуля
GET_PATH	Получение пути до вызываемого каталога
FREE	Освобождение памяти выделенную под программу
LOAD	Загрузка вызываемого модуля
MAIN	Главная функция программы

### Задание.

**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.

### **Выполнение работы**

**Шаг 1.** Для выполнения лабораторной работы был написан и отлажен программный модуль типа .EXE, который выполняет требуемые по заданию функции.

**Шаг 2.** Была запущена и отлажена программа, когда текущим каталогом является каталог с разработанными модулями. Программа вызывает другую программу, которая останавливается, ожидая символ с клавиатуры. Был введен символ «j».

```

D:\>exe
Memory is freed
Address of unavailable memory segment: 9FFF
Address of environment segment: 02D2
End of command line:
Contents of environment area:
PATH=Z:\
COMSPEC=Z:\COMMAND.COM
BLASTER=A220 I7 D1 H5 T6
Path of loaded module:D:\LAB2.COMj
Program was finished: exit with code:  j
D:\>_

```

Рисунок 1 — Иллюстрация работы программы (текущий каталог — каталог с разработанным модулем, вводимый символ - «j»)

**Шаг 3.** Была запущена и отлажена программа, когда текущим каталогом является каталог с разработанными модулями. Программа вызывает другую программу, которая останавливается, ожидая символ с клавиатуры. Была введена комбинация символов Ctrl-C. Результат повторяется в силу того, что в DOS BOX не реализовано прерывание Ctrl-C.

```

D:\>exe
Memory is freed
Address of unavailable memory segment: 9FFF
Address of environment segment: 02D2
End of command line:
Contents of environment area:
PATH=Z:\
COMSPEC=Z:\COMMAND.COM
BLASTER=A220 I7 D1 H5 T6
Path of loaded module:D:\LAB2.COM
Program was finished: exit with code:  ♥
D:\>_

```

Рисунок 2 — Иллюстрация работы программы (текущий каталог — каталог с разработанным модулем, вводимый символ — комбинация Ctrl-C)

**Шаг 4.** Была запущена отлаженная программа, когда текущим каталогом является другой каталог, отличный от того, в котором содержатся разработанные программные модули. Ввод комбинаций клавиш тоже был повторен.

```

D:\LAB6_EXP>exe
Memory is freed
Address of unavailable memory segment: 9FFF
Address of environment segment: 02D2
End of command line:
Contents of environment area:
PATH=Z:\
COMSPEC=Z:\COMMAND.COM
BLASTER=A220 I7 D1 H5 T6
Path of loaded module:D:\LAB6_EXP\LAB2.COMj
Programm was finished: exit with code:  j
D:\LAB6_EXP>_

```

Рисунок 3 — Иллюстрация работы программы (текущий каталог — «другой» каталог, вводимый символ — «j»)

```

D:\LAB6_EXP>exe
Memory is freed
Address of unavailable memory segment: 9FFF
Address of environment segment: 02D2
End of command line:
Contents of environment area:
PATH=Z:\
COMSPEC=Z:\COMMAND.COM
BLASTER=A220 I7 D1 H5 T6
Path of loaded module:D:\LAB6_EXP\LAB2.COM
Programm was finished: exit with code:  ♥
D:\LAB6_EXP>

```

Рисунок 4 — Иллюстрация работы программы (текущий каталог — «другой» каталог, вводимый символ — комбинация Ctrl-C)

**Шаг 5.** Была запущена и отлажена программа, когда модули находятся в разных каталогах.

```

D:\LAB6_EXP\MYR>exe
Memory is freed
Error: file is not found
D:\LAB6_EXP\MYR>_

```

Рисунок 5 — Иллюстрация работы программы (модули находятся в разных каталогах)

1) Как реализовано прерывание Ctrl-C?

Ответ. При нажатии сочетания клавиш Ctrl+C срабатывает прерывание int 23h, управление передается по адресу — (0000:008C), адрес копируется в PSP (с помощью функций 26h и 4ch), при выходе из программы исходное значение адреса восстанавливается.

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

Ответ. В точке вызова функции 4ch прерывания int 21h.

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

Ответ. В данном случае, программа завершится в точке, в который была введена и считана комбинация Ctrl+C.

**Вывод.**

Было произведено исследование возможности построения загрузочного модуля динамической структуры.

## Приложение А

### Исходный код программы

Файл exe.asm:

```
MYSTACK SEGMENT STACK
    DW 256 DUP(?)
MYSTACK ENDS
```

```
DATA SEGMENT
```

```
    block_param dw 0
    com_off dw 0
    com_seg dw 0
    fcb1 dd 0
    fcb2 dd 0
```

```
    next_com_line db 1h, 0dh
    file_name db 'LAB2.com', 0h
    file_path db 128 DUP(0)
```

```
    keep_SS dw 0
    keep_SP dw 0
```

```
    mem_error db 0
    free_memory_mcb_str db 'Error: MCB crashed', 0DH, 0AH,
    '$'
    free_memory_need_more_str db 'Error: It needs more
memory', 0DH, 0AH, '$'
    free_memory_address_str db 'Errorr: Wrong address',
0DH, 0AH, '$'
```

```
free_memory_success_str db 'Memory is freed', 0DH,  
0AH, '$'
```

```
load_function_str db 'Error: Function number is
```



```

incorrect', 0DH, 0AH, '$'

    load_file_not_found_str db 'Error: file is not found',
0DH, 0AH, '$'

    load_disk_str db 'Error: Disk_problem', 0DH, 0AH, '$'
    load_need_more_str db 'Error(load): It needs more
memory', 0DH, 0AH, '$'

    load_path_str db 'Error: Wrong path', 0DH, 0AH, '$'
    load_format_str db 'Error: Wrong format', 0DH, 0AH,
'$'

    exit_str db 'Programm was finished: exit with code:
', 0DH, 0AH, '$'

    exit_ctrl_c_str db 'Exit with Ctrl+Break', 0DH, 0AH,
'$'

    exit_error_str db 'Exit with device error', 0DH, 0AH,
'$'

    exit_int_31h_str db 'Exit with int 31h', 0DH, 0AH, '$'

    end_of_data db 0
DATA ENDS

CODE SEGMENT

    ASSUME CS:CODE, DS:DATA, SS:MYSTACK

    MODULE_PATH PROC near
        push AX
        push BX
        push BP
        push DX

```

```

push ES
push DI

mov BX, offset file_path
add DI, 3

loop1:
    mov DL, ES:[DI]
    mov [BX], DL
    cmp DL, '.'
    je slash
    inc DI
    inc BX
    jmp loop1

slash:
    mov DL, [BX]
    cmp DL, '\'
    je module_name
    mov DL, 0h
    mov [BX], DL
    dec BX
    jmp slash

module_name:
    mov DI, offset file_name
    inc BX

add_name:
    mov DL, [DI]

```

```

        cmp DL, 0h
        je module_path_end
        mov [BX], DL
        inc BX
        inc DI
        jmp add_name

module_path_end:
        mov [BX], DL
        pop DI
        pop ES
        pop DX
        pop BP
        pop BX
        pop AX
    ret
MODULE_PATH ENDP

GET_PATH PROC near
    push AX
    push DX
    push ES
    push DI

    xor DI, DI
    mov AX, ES:[2ch]
    mov ES, AX

    loop2:
        mov DL, ES:[DI]

```

```

        cmp DL, 0
        je end1
        inc DI
        jmp loop2

end1:
        inc DI
        mov DL, ES:[DI]
        cmp DL, 0
        jne loop2

call MODULE_PATH

pop DI
pop ES
pop DX
pop AX
ret
GET_PATH ENDP

FREE PROC far
        push AX
        push BX
        push CX
        push DX
        push ES

        xor DX, DX
        mov mem_error, 0h

```

```

    mov AX, offset end_of_data
    mov BX, offset main_fin
    add AX, BX
    mov BX, 10h
    div BX
    add AX, 100h
    mov BX, AX
    xor AX, AX

    mov AH, 4ah
    int 21h

    jnc free_memory_success
    mov mem_error, 1h
    cmp AX, 7
    jne free_memory_need_more

    mov DX, offset free_memory_mcb_str
    call WRITE_MESSAGE_WORD
    jmp free_end

free_memory_need_more:
    cmp AX, 8
    jne free_memory_address

    mov DX, offset free_memory_need_more_str
    call WRITE_MESSAGE_WORD
    jmp free_end

free_memory_address:

```

```

    cmp AX, 9
    jne free_end

    mov DX, offset free_memory_address_str
    call WRITE_MESSAGE_WORD
    jmp free_end

free_memory_success:
    mov DX, offset free_memory_success_str
    call WRITE_MESSAGE_WORD

free_end:
    pop ES
    pop DX
    pop CX
    pop BX
    pop AX
    ret
FREE ENDP

LOAD PROC far
    push AX
    push BX
    push CX
    push DX
    push DS
    push ES

    mov keep_SP, SP

```

```

mov keep_SS, SS

call GET_PATH

mov AX, DATA
mov ES, AX
mov BX, offset block_param
mov DX, offset next_com_line
mov com_off, DX
mov com_seg, DS
mov DX, offset file_name

mov AX, 4b00h
int 21h

mov SS, keep_SS
mov SP, keep_SP
pop ES
pop DS

call NEXT_LINE

jnc success_load

cmp AX, 1
jne load_file_not_found

mov DX, offset load_function_str
call WRITE_MESSAGE_WORD
jmp load_end

```

```
load_file_not_found:
    cmp AX, 2
    jne load_disk
    mov DX, offset load_file_not_found_str
    call WRITE_MESSAGE_WORD
    jmp load_end
```

```
load_disk:
    cmp AX, 5
    jne load_need_more
    mov DX, offset load_disk_str
    call WRITE_MESSAGE_WORD
    jmp load_end
```

```
load_need_more:
    cmp AX, 8
    jne load_path
    mov DX, offset load_need_more_str
    call WRITE_MESSAGE_WORD
    jmp load_end
```

```
load_path:
    cmp AX, 10
    jne load_format
    mov DX, offset load_path_str
    call WRITE_MESSAGE_WORD
    jmp load_end
```

```
load_format:
```



```

        cmp AX, 11
        jne load_end
        mov DX, offset load_format_str
        call WRITE_MESSAGE_WORD
        jmp load_end

success_load:
        mov ax, 4d00h
        int 21h

cmp AH, 0
jne ctrl_exit
mov DI, offset exit_str

add DI, 41
mov [DI], AL
mov DX, offset exit_str
call WRITE_MESSAGE_WORD
jmp load_end

ctrl_exit:
        cmp AH, 1
        jne exit_error
        mov DX, offset exit_ctrl_c_str
        call WRITE_MESSAGE_WORD
        jmp load_end

exit_error:
        cmp AH, 2
        jne exit_int_31h

```

```

        mov DX, offset exit_error_str
        call WRITE_MESSAGE_WORD
        jmp load_end

exit_int_31h:
        cmp AH, 3
        jne load_end
        mov DX, offset exit_int_31h_str
        call WRITE_MESSAGE_WORD
        jmp load_end

load_end:
        pop DX
        pop CX
        pop BX
        pop AX

ret

LOAD ENDP

WRITE_MESSAGE_WORD PROC near
        push AX

        mov AH, 9
        int 21h

        pop AX
        ret

WRITE_MESSAGE_WORD ENDP

```

```

WRITE_MESSAGE_BYTE PROC near
    push AX

    mov AH, 02h
    int 21h

    pop AX
    ret
WRITE_MESSAGE_BYTE ENDP

NEXT_LINE PROC near
    push AX
    push DX

    mov DL, 0DH
    call WRITE_MESSAGE_BYTE

    mov DL, 0AH
    call WRITE_MESSAGE_BYTE

    pop DX
    pop AX
    ret
NEXT_LINE ENDP

MAIN PROC far
    mov AX, DATA
    mov DS, AX

    call FREE

```

```

        cmp mem_error, 0h
        jne main_end

        call GET_PATH
        call LOAD

        main_end:
            xor AL, AL
            mov AH, 4ch
            int 21h

MAIN ENDP

main_fin:
CODE ENDS

END MAIN

файл lab2.asm

TESTPC SEGMENT
    ASSUME CS:TESTPC, DS:TESTPC, ES:NOTHING, SS:NOTHING
    ORG 100H
START: JMP BEGIN

UNAVAILABLE_MEMORY db 'Address of unavailable memory
segment:      ', 0DH, 0AH, '$'
ENVIRONMENT db 'Address of environment segment:      ', 0DH,
0AH, '$'
CONTENT_ENV_AREA db 'Contents of environment area:      ',

```

```

0DH, 0AH, '$'
COMMAND_LINE_END_EMPTY db 'End of command line: empty',
0DH, 0AH, '$'
COMMAND_LINE_END db 'End of command line:$'
LOADED_MODULE_PATH db 'Path of loaded module:$'

```

```

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
    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
    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
        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
end_l:   pop DX
        pop CX
        ret
BYTE_TO_DEC ENDP

```

```

WRITE_MESSAGE_WORD PROC near
    push AX
    mov AH, 09h
    int 21h
    pop AX
    ret
WRITE_MESSAGE_WORD ENDP

```

```

WRITE_MESSAGE_BYTE PROC near
    push AX
    mov AH, 02h
    int 21h
    pop AX
    ret
WRITE_MESSAGE_BYTE ENDP

```

```

PRINT_UNAVAILABLE_MEMORY PROC near
    push AX
    push DI
    push DX

    mov AX, DS:[02h]
    mov DI, offset UNAVAILABLE_MEMORY
    add DI, 42

```

```

    call WRD_TO_HEX
    mov DX, offset UNAVAILABLE_MEMORY

    call WRITE_MESSAGE_WORD

    pop DX
    pop DI
    pop AX

    ret
PRINT_UNAVAILABLE_MEMORY ENDP

PRINT_ENVIRONMENT PROC near
    push AX
    push DI
    push DX

    mov AX, DS:[02Ch]
    mov DI, offset ENVIRONMENT
    add DI, 35
    call WRD_TO_HEX
    mov DX, offset ENVIRONMENT

    call WRITE_MESSAGE_WORD

    pop DX
    pop DI
    pop AX

    ret

```



```
PRINT_ENVIRONMENT ENDP
```

```
PRINT_COMMAND_LINE_END PROC near
```

```
    push AX
```

```
    push DI
```

```
    push CX
```

```
    push DX
```

```
    xor CX, CX
```

```
    mov CL, DS:[80h]
```

```
    cmp CL, 0h
```

```
    je empty_cont
```

```
    xor DI, DI
```

```
    mov DX, offset COMMAND_LINE_END
```

```
    call WRITE_MESSAGE_WORD
```

```
cycle:
```

```
    mov DL, DS:[81h+DI]
```

```
    call WRITE_MESSAGE_BYTE
```

```
    inc DI
```

```
    loop cycle
```

```
    mov DL, 0Dh
```

```
    call WRITE_MESSAGE_BYTE
```

```
    mov DL, 0Ah
```

```
    call WRITE_MESSAGE_BYTE
```

```
    jmp final
```

```
empty_cont:
```

```

        mov DX, offset COMMAND_LINE_END_EMPTY
        call WRITE_MESSAGE_WORD

final:
        pop DX
        pop CX
        pop DI
        pop AX
    ret
PRINT_COMMAND_LINE_END ENDP

PRINT_CONTENT_ENV_AREA_AND_LOADED_MODULE_PATH PROC near
    push AX
    push DI
    push DX
    push ES

    mov DX, offset CONTENT_ENV_AREA
    call WRITE_MESSAGE_WORD
    xor DI, DI
    mov AX, DS:[2Ch]
    mov ES, AX

cycle_02:
    mov DL, ES:[DI]
    cmp DL, 0h
    je end_word
    call WRITE_MESSAGE_BYTE
    inc DI
    jmp cycle_02

```

```

end_word:
    mov DL, 0Ah
    call WRITE_MESSAGE_BYTE
    inc DI
    mov DL, ES:[DI]
    cmp DL, 0h
    je final_02
    call WRITE_MESSAGE_BYTE
    inc DI
    jmp cycle_02

final_02:
    mov DX, offset LOADED_MODULE_PATH
    call WRITE_MESSAGE_WORD
    add DI, 3
    cycle_03:
        mov DL, ES:[DI]
        cmp DL, 0h
        je final_03
        call WRITE_MESSAGE_BYTE
        inc DI
        jmp cycle_03

final_03:
    pop ES
    pop DX
    pop DI
    pop AX
ret

```

```
PRINT_CONTENT_ENV_AREA_AND_LOADED_MODULE_PATH ENDP
```

```
BEGIN:
```

```
    call PRINT_UNAVAILABLE_MEMORY
```

```
    call PRINT_ENVIRONMENT
```

```
    call PRINT_COMMAND_LINE_END
```

```
    call PRINT_CONTENT_ENV_AREA_AND_LOADED_MODULE_PATH
```

```
    xor AL, AL
```

```
    mov AH, 01h ;запросить с клавиатуры символ и поместить  
введенный символ в регистр AL
```

```
    int 21h
```

```
    mov AH, 4Ch
```

```
    int 21h
```

```
TESTPC ENDS
```

```
END START
```