

**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**

Студент гр. 0382

Самулевич В.А.

Преподаватель

Ефремов М.А.

Санкт-Петербург

2022

### **Цель работы.**

Исследование возможности построения загрузочного модуля динамической структуры, а также интерфейса между вызывающим и вызываемым модулем по управлению и данным.

### **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:**

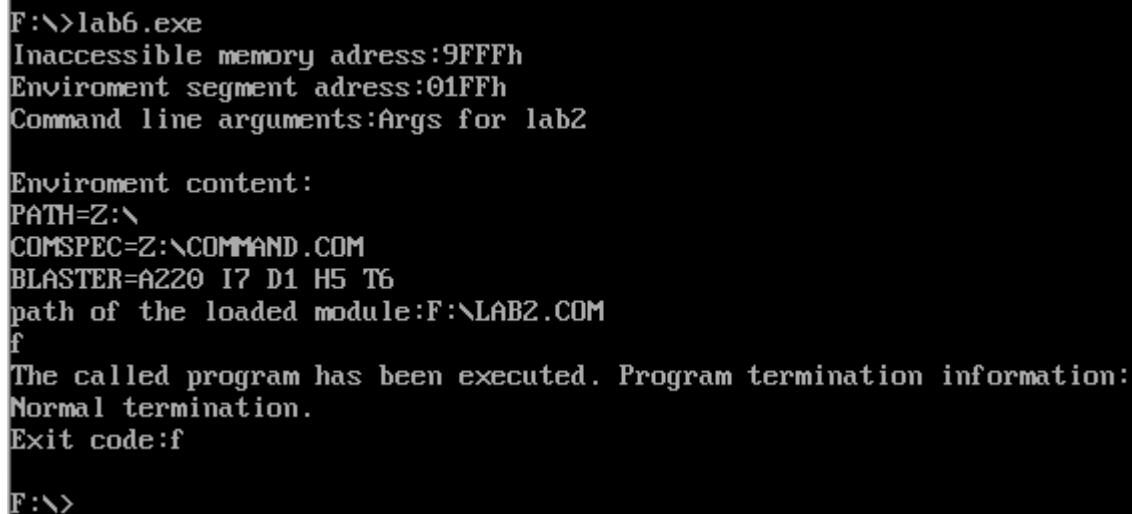
При написании требуемого программного модуля были реализованы следующие функции:

- `Free_memory` – освобождает неиспользуемую память вызывающей программы (по умолчанию она занимает все доступное место).
- `Init_file_path` – инициализирует строку `file_path`, помещая в нее путь и имя вызываемой программы (путь берется из PSP вызывающей программы, а имя – из поля `file_name`).
- `Init_command_line` – инициализирует значение первого байта командой строки (поле `command_line_size`), после чего устанавливает соответствующие значения в поля `cmd_offset` и `cmd_segment`.
- `Run_loader` – подготавливает все необходимые параметры для загрузки `lb2` из `lb5` (используя, в том числе, `init_file_path` и `init_command_line`), после чего активирует прерывание `int 21h` (`AH = 4bh`) и вызывает `process_the_result` для обработки результата.

- Process\_the\_result – Обрабатывает результат работы вызываемой программы и выводит соответствующие ему сообщения.

### Шаг 2:

Написанная программа была запущена, когда текущим каталогом являлся каталог с разработанными программными модулями. В качестве кода завершения для lab2 был введен 'f'. Результат работы представлен на рисунке 1.



```
F:\>lab6.exe
Inaccessible memory address:9FFFh
Enviroment segment address:01FFFh
Command line arguments:Args for lab2

Enviroment content:
PATH=Z:\
COMSPEC=Z:\COMMAND.COM
BLASTER=A220 I7 D1 H5 T6
path of the loaded module:F:\LAB2.COM
f
The called program has been executed. Program termination information:
Normal termination.
Exit code:f
F:\>
```

Рисунок 1- Запуск lab6.exe из каталога с разработанными программными модулями и кодом завершения 'f'.

Как видно из рисунка, запуск lab2 из lab5 прошел без ошибок.

### Шаг 3:

Программа была запущена, когда текущим каталогом все также являлся каталог с разработанными модулями. Однако в этот раз, при запросе из lab2 кода завершения, была нажата комбинация Ctrl – C. Результат представлен на рисунке 2.

```

F:\>lab6.exe
Inaccessible memory adress:9FFFh
Enviroment segment adress:01FFFh
Command line arguments:Args for lab2

Enviroment content:
PATH=Z:\
COMSPEC=Z:\COMMAND.COM
BLASTER=A220 I7 D1 H5 T6
path of the loaded module:F:\LAB2.COM
♥
The called program has been executed. Program termination information:
Normal termination.
Exit code:♥
F:\>

```

Рисунок 2 – Запуск lab6.exe из каталога с разработанными программными модулями и завершением через Ctrl-C.

Поскольку в DOS не реализовано прерывание по Ctrl –C, его не происходит. Вместо этого, нажатие этих клавиш воспринимается как ввод символа ‘ ♥’.

#### Шаг 4:

lab6.exe была запущена когда текущим являлся каталог, отличный от того, в котором содержатся разработанные программные модули. Ввод комбинаций клавиш был повторен. Результаты работы представлены на рисунках 3 и 4.

```

C:\>oc\lab6.exe
Inaccessible memory adress:9FFFh
Enviroment segment adress:01FFFh
Command line arguments:Args for lab2

Enviroment content:
PATH=Z:\
COMSPEC=Z:\COMMAND.COM
BLASTER=A220 I7 D1 H5 T6
path of the loaded module:C:\OC\LAB2.COM
f
The called program has been executed. Program termination information:
Normal termination.
Exit code:f

```

Рисунок 3 – Запуск lab6.exe из C:\ и кодом завершения ‘f’.

```

C:\>oc\lab6.exe
Inaccessible memory address:9FFFh
Enviroment segment address:01FFh
Command line arguments:Args for lab2

Enviroment content:
PATH=Z:\
COMSPEC=Z:\COMMAND.COM
BLASTER=A220 I7 D1 H5 T6
path of the loaded module:C:\OC\LAB2.COM
♥
The called program has been executed. Program termination information:
Normal termination.
Exit code:♥

```

Рисунок 4 – Запуск lab6.exe из C:\ и завершением через Ctrl-C.

### Шаг 5:

Программа была запущена, когда модули находились в разных каталогах.

Результат представлен на рисунке 5.

```

C:\>oc\lab6.exe
File not found.

C:\>_

```

Рисунок 5- запуск lab6.exe, когда она не находится в одном каталоге с lab2.com.

### Контрольные вопросы.

1) Как реализовано прерывание Ctrl-C?

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?

В этом случае вызываемая программа заканчивается при вызове int 21h(АН = 4ch).

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

В точке, в которой была введена и считана эта комбинация.

### **Выводы.**

Было произведено исследование возможности построения загрузочного модуля динамической структуры.

## ПРИЛОЖЕНИЕ А

### ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: lab6.asm

```
Astack SEGMENT STACK
    DW 128 DUP(?)
Astack ENDS
```

```
DATA SEGMENT
    first_clear_error DB "Error clearing memory: control block
destroyed.",0Dh,0Ah,'$'
    second_clear_error DB "Error while clearing memory: Not enough
memory to execute function.",0Dh,0Ah,'$'
    first_clear_error DB "Error clearing memory: invalid memory block
address.",0Dh,0Ah,'$'
    file_name DB "lab2.com",'$'
    file_path DB 128 dup(0)
    segment_adress DW 0
    cmd_offset DW 0
    cmd_segment DW 0
    first_FCB DD 0
    second_FCB DD 0

    command_line_size DB 0
    command_line_args DB "Args for lab2",0Dh,0Ah,0
    normal_termination DB 0Dh,0Ah,"The called program has been
executed. Program termination information:",0Dh,0Ah,'$'
    termination_reason_0 DB "Normal termination.",0Dh,0Ah,'$'
    termination_reason_1 DB "Terminate by pressing ctrl-
break.",0Dh,0Ah,'$'
    termination_reason_2 DB "Device error termination.",0Dh,0Ah,'$'
    termination_reason_3 DB "Termination by function
31h.",0Dh,0Ah,'$'
    exit_code DB "Exit code: ",0Dh,0Ah,'$'

    abnormal_termination DB 0Dh,0Ah,"The called program was not
loaded, error information:",0Dh,0Ah,'$'
    load_error_1 DB "Function number is invalid.",0Dh,0Ah,'$'
    load_error_2 DB "File not found.",0Dh,0Ah,'$'
    load_error_3 DB "Disk error.",0Dh,0Ah,'$'
    load_error_4 DB "Insufficient memory size.",0Dh,0Ah,'$'
    load_error_5 DB "Wrong environment string.",0Dh,0Ah,'$'
    load_error_6 DB "Wrong format.",0Dh,0Ah,'$'

    end_data DB 0
DATA ENDS

CODE SEGMENT
ASSUME CS:CODE, DS:DATA,SS:Astack,ES:NOTHING

KEEP_SS DW 0
KEEP_SP DW 0

Main PROC FAR
    push DS
    sub AX,AX
```



```

    push AX
    mov AX, DATA
    mov DS,AX
    call free_memory
    jb programm_end
    call run_loader

programm_end:
    mov AX,0
    mov AH,4ch
    int 21h

main ENDP

Free_memory PROC NEAR
    push AX
    push BX
    push DX

    mov DX,0
    mov AX,offset end_code
    add AX,offset end_data
    inc AX
    add AX,228h
    mov BX,16
    div BX
    inc AX
    mov BX,AX
    mov AX,0
    mov AH,4ah
    int 21h
    jae free_without_errors
    cmp AX,7
    jne second_free_check
    mov DX,offset first_clear_error
    jmp handle_free_error

second_free_check:
    cmp AX,8
    jne fird_free_check
    mov DX,offset second_clear_error
    jmp handle_free_error

fird_free_check:
    mov DX,offset fird_clear_error

handle_free_error:
    call print
    STC ; флаг того, что очистка памяти не удалась
    jmp free_end

free_without_errors:
    CLC
free_end:
    pop DX
    pop BX
    pop AX
    ret

```

```

Free_memory ENDP

print PROC NEAR
    push AX
    mov AH,09h
    int 21h
    pop AX
    ret
print ENDP

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

    mov AX, ES:[2ch]
    mov ES,AX
    mov CX,0
    mov SI,0
find_file_path_cycle:
    mov AL,ES:[SI]
    cmp AL,0
    je two_zeros_checking
    mov CX,0
    inc SI
    jmp find_file_path_cycle
two_zeros_checking:
    inc CX
    cmp CX,2
    je stop_find_file_loop
    inc SI
    jmp find_file_path_cycle

stop_find_file_loop:
    add SI,3
    mov DI,offset file_path
file_path_copy_loop:
    cmp BYTE PTR ES:[SI],0
    je stop_file_path_copy_loop
    mov AL,ES:[SI]
    mov DS:[DI],AL
    inc SI
    inc DI
    cmp AL,'\'
    je update_last_dir
    jmp file_path_copy_loop

update_last_dir:
    mov CX,DI
    jmp file_path_copy_loop

stop_file_path_copy_loop:
    mov DI,CX
    mov SI,offset file_name
copy_file_name_loop:

```

```

        mov AL,[SI]
        cmp AL,'$'
        je stop_copy_file_name_loop
        mov [DI],AL
        inc SI
        inc DI
        jmp copy_file_name_loop

stop_copy_file_name_loop:
        mov BYTE PTR [DI],0

        pop DI
        pop SI
        pop CX
        pop ES
        pop AX
        ret
init_file_path ENDP

init_command_line PROC NEAR
        push CX
        push SI

        mov CX,0
        mov SI,offset command_line_args
get_cmd_size_loop:
        cmp BYTE PTR [SI],0
        je end_cmd_size_loop
        inc SI
        inc CX
        jmp get_cmd_size_loop

end_cmd_size_loop:
        mov DS:command_line_size, CL
        mov cmd_offset,offset command_line_size
        mov cmd_segment, SEG command_line_size

        pop SI
        pop CX
        ret
init_command_line ENDP

run_loader PROC NEAR
        push DS
        push ES
        push AX
        push DX
        push BX

        mov CS:KEEP_SS,SS
        mov CS:KEEP_SP,SP
        call init_command_line
        call init_file_path
        mov AX,DS
        mov ES,AX
        mov BX,offset segment_adress

```

```

    mov DX,offset file_path

    mov AX,4B00h
    int 21h

    mov SS,CS:KEEP_SS
    mov SP,CS:KEEP_SP
    call process_the_result

    pop BX
    pop DX
    pop AX
    pop ES
    pop DS

    ret
run_loader ENDP

process_the_result PROC NEAR
    push AX
    push DX
    push SI

    jb hanlde_error_code
    mov DX,offset normal_termination
    call print
    mov AH,4Dh
    int 21h
    cmp AH,0
    jne completion_check_2
    mov DX,offset termination_reason_0
    jmp print_exit_code

completion_check_2:
    cmp AH,1
    jne completion_check_3
    mov DX,offset termination_reason_1
    jmp print_exit_code

completion_check_3:
    cmp AH,2
    jne completion_check_4
    mov DX,offset termination_reason_2
    jmp print_exit_code

completion_check_4:
    mov DX,offset termination_reason_3

print_exit_code:
    call print
    mov SI,offset exit_code
    add SI,10
    mov [SI],AL
    mov DX,offset exit_code
    call print
    jmp process_the_result_end

hanlde_error_code:

```

```

        cmp AX,1
        jne load_error_check_2
        mov DX,offset load_error_1
        jmp print_load_error

load_error_check_2:
        cmp AX,2
        jne load_error_check_3
        mov DX,offset load_error_2
        jmp print_load_error

load_error_check_3:
        cmp AX,5
        jne load_error_check_4
        mov DX,offset load_error_3
        jmp print_load_error

load_error_check_4:
        cmp AX,8
        jne load_error_check_5
        mov DX,offset load_error_4
        jmp print_load_error

load_error_check_5:
        cmp AX,10
        jne load_error_check_6
        mov DX,offset load_error_5
        jmp print_load_error

load_error_check_6:
        mov DX,offset load_error_6
        jmp print_load_error
print_load_error:
        call print

process_the_result_end:

        pop SI
        pop DX
        pop AX
        ret
process_the_result ENDP

end_code:
CODE ENDS
END Main

```