

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

**ОТЧЕТ**  
**по лабораторной работе №7**  
**по дисциплине «Операционные системы»**  
**Тема: Построение модуля оверлейной структуры**

Студент гр.0382

\_\_\_\_\_

Бочаров Г.С.

Преподаватель

\_\_\_\_\_

Ефремов М.А.

Санкт-Петербург

2022

### **Цель работы.**

Исследование возможности построения загрузочного модуля оверлейной структуры. Исследуется структура оверлейного сегмента и способ загрузки и выполнения оверлейных сегментов. Для запуска вызываемого оверлейного модуля используется функция 4B03h прерывания int 21h. Все загрузочные оверлейные модули находятся в одном каталоге.

### **Задание.**

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

- Освобождает память для загрузки оверлеев;
- Читает размер файла оверлея и запрашивает объем памяти, достаточный для его загрузки;
- Файл оверлейного сегмента загружается и выполняется;
- Освобождается память, отведенная для оверлейного сегмента;
- Затем действия 1)-4) выполняются для оверлейного сегмента;

2. Также необходимо написать и отладить оверлейные сегменты. Оверлейный сегмент выводит адрес сегмента, в который он загружен.

3. Запустите отлаженное приложение. Оверлейные сегменты должны загружаться с одного и того же адреса, перекрывая друг друга.

4. Запустите приложение из другого каталога. Приложение должно быть выполнено успешно.

5. Запустите приложение в случае, когда одного оверлея нет в каталоге. Приложение должно закончиться аварийно.

6. Занесите полученные результаты в виде скриншотов в отчет. Оформите отчет в соответствии с требованиями.

## Выполнение работы.

1. Был написан программный модуль типа .EXE, осуществляющий освобождение неиспользуемой программой памяти, выделение памяти под оверлейные модули, и обработку ошибок возникающих в процессе выполнения программы.

Для выполнения поставленных задач были написаны следующие функции:

- free\_memory – подготавливает место в памяти, необходимое для программы. В случае возникновения ошибок выводит соответствующее сообщение

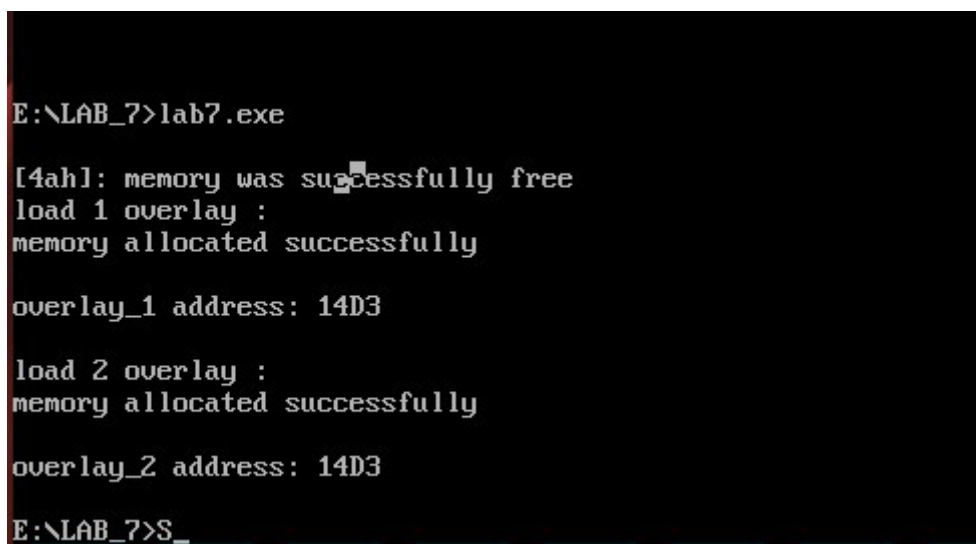
- str\_plus\_str функция сложения двух строк.

- set\_full\_name – устанавливает полный путь к загружаемому модулю

- alloc – определение и отведение памяти для оверлейного модуля

- load\_over – загружает вызываемый оверлейный модуль.

2. Результаты выполнения программы в условия описанных в требованиях представлены на рисунках 1-3:



```
E:\LAB_7>lab7.exe

[4ah]: memory was successfully free
load 1 overlay :
memory allocated successfully

overlay_1 address: 14D3

load 2 overlay :
memory allocated successfully

overlay_2 address: 14D3

E:\LAB_7>S_
```

Рисунок 1 – Результат запуска приложения из того же каталога, где само приложение.

```

E:\LAB_7\SCREENS> E:\LAB_7\lab7.exe

[4ah]: memory was successfully free
load 1 overlay :
memory allocated successfully

overlay_1 address: 14D3

load 2 overlay :
memory allocated successfully

overlay_2 address: 14D3

E:\LAB_7\SCREENS>

```

Рисунок 2 – Результат запуска приложения из каталога, отличного от того, где расположено само приложение.

```

E:\LAB_7>lab7.exe

[4ah]: memory was successfully free
load 1 overlay :
memory allocated successfully

overlay_1 address: 14D3

load 2 overlay :
file not found
[4b03h]: file not found

E:\LAB_7>S

```

Рисунок 3 – Результат запуска приложения, когда одного оверлейного модуля нет или указано неверное название модуля.

Исходный код программы см. в приложении А.

### Ответы на вопросы.

1. Как должна быть устроена программа, если в качестве оверлейного сегмента использовать .COM модули?

При обращении к .COM модулю, в качестве оверлея нужно учитывать смещение 100h, т.к. в начале .COM модуля эту память занимает PSP.

### **Выводы.**

В ходе работы были исследованы возможности построения загрузочного модуля оверлейной структуры и структура оверлейного сегмента, а также способ их загрузки и выполнения.

# ПРИЛОЖЕНИЕ А

## ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: lab7.asm

```
stack segment stack
    dw 128 dup(?)
stack ends
```

```
data segment
```

```
ovl_1_name db 'over_1.ovl', 0
ovl_2_name db 'over_2.ovl', 0
```

```
curent_f_name dw 0
dta db 43 dup(0)
full_p db 128 dup(0)
```

```
param dd 0
keep_psp dw 0
```

```
    ; message 4ah    free mem
    mem_free_mes db                                '[4ah]:    memory
was successfully free ', 0dh, 0ah, '$'
    mcb_crash_mes db                                '[4ah]:    mcb    was
crashed.', 0dh, 0ah, '$'
    no_memory_mes db                                '[4ah]:    not
enough memory', 0dh, 0ah, '$'
    ivalid_address_mes db                            '[4ah]:    invalid
memory addressess', 0dh, 0ah, '$'

    ;load message
    load_1_mes db                                    'load                1
overlay :', 13, 10, '$'
    load_2_mes db                                    'load                2
overlay :', 13, 10, '$'

    ; message 4b03h
    err_1_mes db                                    '[4b03h]:    wrong
function', 13, 10, '$'
    err_2_mes db                                    '[4b03h]:    file
not found', 13, 10, '$'
    err_3_mes db                                    '[4b03h]:    path
not found', 13, 10, '$'
    err_4_mes db                                    '[4b03h]:    too
many open files', 13, 10, '$'
    err_5_mes db                                    '[4b03h]:    disk
error', 13, 10, '$'
```

```

        err_8_mes db                                '[4b03h]:      not
enough memory to load programm', 13, 10, '$'
        err_10_mes db                               '[4b03h]:      error
environment', 13, 10, '$'
        err_11_mes db                               '[4b03h]:      error
format', 13, 10, '$'

        alloc_success_mes db                       'memory allocated
successfully', 13, 10, '$'
        file_err_mes db                            'file not found',
13, 10, '$'
        route_err_mes db                          'route          not
found', 13, 10, '$'

        endl_s db                                  0dh,0ah,'$'

        data_end db 0
data ends

code segment
        assume cs:code,ds:data,ss:stack

print proc
        push ax
        mov ah, 09h
        int 21h
        pop ax
        ret
print endp

free_memory proc  near
        push ax
        push bx
        push cx
        push dx

        mov bx, offset end_address
        mov ax, es
        sub bx, ax
        mov cl, 4
        shr bx, cl          ; byte to par
        mov ah, 4ah
        int 21h

        jnc end_proc

        cmp ax, 7
        je error_crash

```

```

        cmp ax, 8
        je error_no_memory

        cmp ax, 9
        je error_address

error_crash:
        mov dx, offset mcb_crash_mes
        call print
        jmp ret_p

error_no_memory:
        mov dx, offset no_memory_mes
        call print
        jmp ret_p

error_address:
        mov dx, offset ivalid_address_mes
        call print
        jmp ret_p

end_proc:
        mov dx, offset endl_s
        call print
        mov dx, offset mem_free_mes
        call print

ret_p:
        pop dx
        pop cx
        pop bx
        pop ax
        ret
free_memory endp

str_plus_str proc near
push dx

add_loop:  ;si - source    di - dest

        mov dl, byte ptr [si]
        mov byte ptr [di], dl
        inc di
        inc si
        cmp dl, 0

```



```

        jne add_loop

pop dx
    ret
str_plus_str endp


set_full_name proc near
    push ax
    push bx
    push cx
    push dx
    push di
    push si
    push es

    mov curent_f_name, dx
    mov ax, keep_psp
    mov es, ax
    mov es, es:[2ch]
    mov bx, 0

skip_env:                                ;skip_env
    cmp byte ptr es:[bx], 0
    je separator_word
    inc bx
    jmp skip_env
separator_word:
    inc bx

    cmp byte ptr es:[bx], 0
    je read_p

    inc bx
    jmp skip_env

read_p:
    add bx, 3
    mov di, 0
read_loop:

    mov dl, es:[bx]
    mov byte ptr [full_p + di], dl
    inc di
    inc bx
    cmp dl, 0

```

```

        je create_full_name
        cmp dl, '\\'
        jne read_loop
        mov cx, di
        jmp read_loop

create_full_name:
        mov si, curent_f_name
        mov di, offset full_p
        add di, cx
        call str_plus_str

        pop es
        pop si
        pop di
        pop dx
        pop cx
        pop bx
        pop ax
        ret

set_full_name endp

```

```

alloc proc
        push ax
        push bx
        push cx
        push dx
        push dx

        mov dx, offset dta
        mov ah, 1ah
        int 21h
        pop dx
        mov cx, 0
        mov ah, 4eh
        int 21h

        jnc success_alloc

        cmp ax, 2
        je path_not_found

file_not_found:

```

```

        mov dx, offset file_err_mes
        call print
        jmp end_alloc

path_not_found:
        mov dx, offset route_err_mes
        call print
        jmp end_alloc

success_alloc:
        push di
        push cx

        mov di, offset dta
        mov bx, [di + 1ah]
        mov ax, [di + 1ch]

        mov cl, 4
        shr bx, cl
        mov cl, 12
        shl ax, cl

        add bx, ax
        add bx, 1

        pop cx
        pop di

        mov ah, 48h
        int 21h

        mov word ptr param, ax
        mov dx, offset alloc_success_mes
        call print
end_alloc:
        pop dx
        pop cx
        pop bx
        pop ax
        ret
alloc endp

```

```
load_over proc near
```

```

push ax
push bx
push cx
push dx
push ds
push es
mov ax, data
mov es, ax
mov bx, offset param
mov dx, offset full_p
mov ax, 4b03h
int 21h
jnc transition

cmp ax, 1
je error_1
cmp ax, 2
je error_2
cmp ax, 3
je error_3
cmp ax, 4
je error_4
cmp ax, 5
je error_5
cmp ax, 8
je error_8
cmp ax, 10
je error_10
cmp ax, 11
je error_11

error_1:
    mov dx, offset err_1_mes
    call print
    jmp load_over_end

error_2:
    mov dx, offset err_2_mes
    call print
    jmp load_over_end

error_3:
    mov dx, offset err_1_mes
    call print
    jmp load_over_end

error_4:
    mov dx, offset err_2_mes
    call print
    jmp load_over_end

```

```

error_5:
    mov dx, offset err_5_mes
    call print
    jmp load_over_end

error_8:
    mov dx, offset err_8_mes
    call print
    jmp load_over_end

error_10:
    mov dx, offset err_10_mes
    call print
    jmp load_over_end

error_11:
    mov dx, offset err_11_mes
    call print
    jmp load_over_end

transition:

    mov ax, word ptr param
    mov es, ax

    mov word ptr param, 0
    mov word ptr param + 2, ax

    call param

    mov es, ax
    mov ah, 49h
    int 21h

load_over_end:
    pop es
    pop ds
    pop dx
    pop cx
    pop bx
    pop ax
    ret
load_over_endp

```

```

main proc far
    push ds
    xor ax, ax
    push ax
    mov ax, data
    mov ds, ax
    mov keep_psp, es
    call free_memory

f1:
    mov dx, offset load_1_mes
    call print
    mov dx, offset ovl_1_name
    push dx
    call set_full_name
    mov dx, offset full_p
    call alloc
    call load_over
    pop dx

    mov dx, offset endl_s
    call print

f2:
    mov dx, offset load_2_mes
    call print
    mov dx, offset ovl_2_name
    push dx
    call set_full_name
    mov dx, offset full_p
    call alloc
    call load_over
    pop dx

    xor al, al
    mov ah, 4ch
    int 21h

main endp
end_address:
code ends
end main

```

**Название файла: over\_1.asm**

```

overlay segment
    assume cs:overlay, ds:nothing, ss:nothing

```

```

main proc far
    push ax
    push dx
    push ds
    push di

    mov ax, cs ;;
    mov ds, ax

    mov di, offset message_add
    add di, 24
    call wrd_to_hex
    mov dx, offset message_add
    call print
    pop di
    pop ds
    pop dx
    pop ax
    retf
main endp

```

```

message_add db 13, 10, "overlay_1 address:      ", 13, 10, '$'

```

```

print proc
    push dx
    push ax
    mov ah, 09h
    int 21h
    pop ax
    pop dx
    ret
print endp

```

```

tetr_to_hex proc
    and al, 0fh
    cmp al, 09
    jbe next
    add al, 07
next:
    add al, 30h
    ret
tetr_to_hex endp

```

```

byte_to_hex proc
    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
    push bx
    mov bh, ah
    call byte_to_hex
    mov [di], ah
    dec di
    mov [di], al
    dec di
    mov al, bh
    xor ah, ah
    call byte_to_hex
    mov [di], ah
    dec di
    mov [di], al
    pop bx
    ret
wrd_to_hex endp
overlay ends

```

```

end main

```

**Название файла: over\_2.asm**

```

overlay segment
    assume cs:overlay, ds:nothing, ss:nothing

main proc far
    push ax
    push dx
    push ds
    push di

    mov ax, cs ;;
    mov ds, ax

    mov di, offset message_add

```



```

        add di, 24
        call wrd_to_hex
        mov dx, offset message_add
        call print
        pop di
        pop ds
        pop dx
        pop ax
        retf
main endp

```

```

message_add db 13, 10, "overlay_2 address:      ", 13, 10, '$'

```

```

print proc
        push dx
        push ax
        mov ah, 09h
        int 21h
        pop ax
        pop dx
        ret
print endp

```

```

tetr_to_hex proc
        and al, 0fh
        cmp al, 09
        jbe next
        add al, 07
next:
        add al, 30h
        ret
tetr_to_hex endp

```

```

byte_to_hex proc
        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
    push bx
    mov bh, ah
    call byte_to_hex
    mov [di], ah
    dec di
    mov [di], al
    dec di
    mov al, bh
    xor ah, ah
    call byte_to_hex
    mov [di], ah
    dec di
    mov [di], al
    pop bx
    ret
wrd_to_hex endp
overlay ends

end main
```