

**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

\_\_\_\_\_

Крючков А.М.

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 и оверлейные модули находятся в одном каталоге.

В этой работе также рассматривается приложение, состоящее из нескольких модулей, поэтому все модули помещаются в один каталог и вызываются с использованием полного пути.

### **Задание.**

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

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

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

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

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

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

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

### **Выполнение работы.**

#### **Шаг 1.**

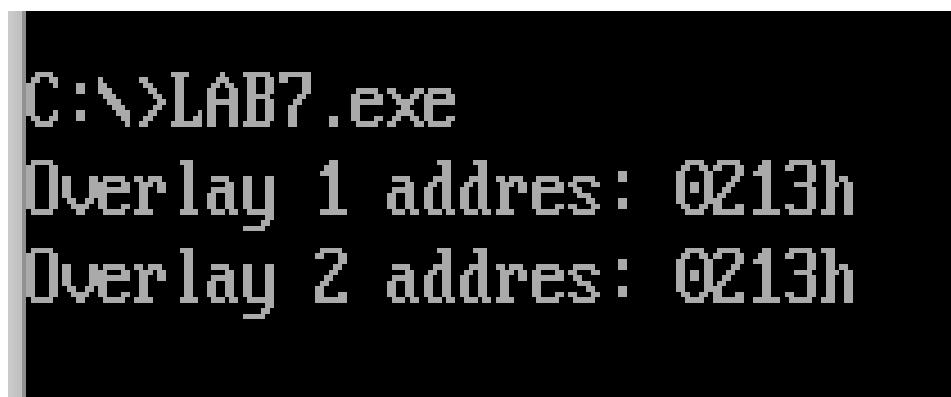
На первом шаге был написан и отлажен .EXE модуль, который освобождает память для загрузки оверлеев, читает размер файла оверлея и запрашивает объем памяти, достаточный для его загрузки, запускает оверлеи. В качестве запускаемых оверлейных модулей были написаны две программы, каждая из которых выводит адрес, начиная с которого она находится в памяти.

#### **Шаг 2.**

На втором шаге были написаны оверлейные модули, каждый из которых выводит адрес сегмента, куда он загружен.

#### **Шаг 3.**

На третьем шаге был запущен модуль .EXE. Результаты работы программ см. на рис. 1.

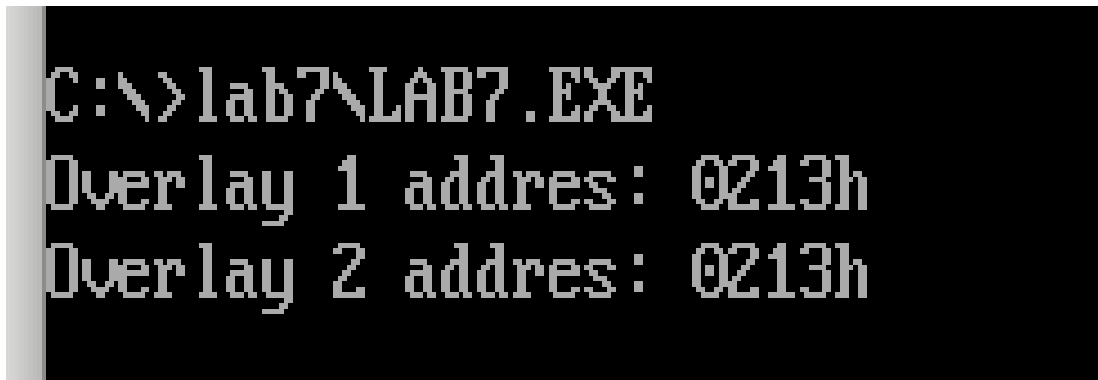


```
C:\>LAB7.exe
Overlay 1 address: 0213h
Overlay 2 address: 0213h
```

Рисунок 1 - Результаты третьего шага

#### Шаг 4.

На четвертом шаге приложение было запущено из другого каталога. По рис. 2 видно, что оно запустилось и отработало успешно.

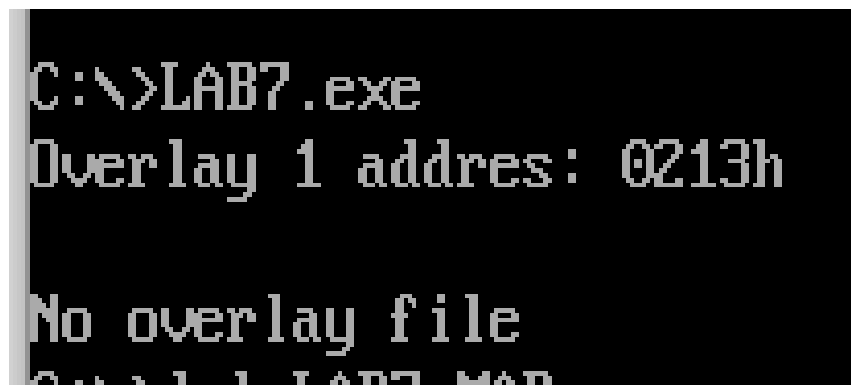


```
C:\>lab7\LAB7.EXE
Overlay 1 address: 0213h
Overlay 2 address: 0213h
```

Рисунок 2 - Результаты четвёртого шага

#### Шаг 5.

На пятом шаге приложение было запущено для случая, когда одного оверлейного модуля нет в каталоге. На рис. 3 видно, что программа работает корректно.



```
C:\>LAB7.exe
Overlay 1 address: 0213h
No overlay file
```

Рисунок 3 - Результаты пятого шага

Исходный код программы см. в приложении А.

### **Ответы на контрольные вопросы.**

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

Чтобы использовать .COM модули в качестве оверлейного сегмента нужно учитывать смещение 100h при обращении к блоку PSP, который необходимо организовать. Кроме того необходимо сохранять регистры, чтобы восстановить их в конце.

### **Выводы.**

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

## ПРИЛОЖЕНИЕ А

### КОД МОДУЛЕЙ

Название файла: lab7.asm

```
astack segment stack
    dw 100h dup (0)
astack ends

data segment
    mem_error db 0
    error_7_memory_block_destroyed db 'error. the control memory block
is destroyed', 0dh, 0ah, '$'
    error_8_memory_for_execution db 'error: not enough memory to
execute the function', 0dh, 0ah, '$'
    error_9_invalid_ba db 'error: invalid memory block address', 0dh,
0ah, '$'
    psp dw 0
    ovl_seg dw 0
    ovl_addr dd 0
    path_err db 13, 10, "No path$"
    load_err db 13, 10, "No loaded overlay$"
    ovl1_str db 13, 10, "overlay1: $"
    ovl2_str db 13, 10, "overlay2: $"
    no_overlay_err db 13, 10, "No overlay file$"
    file_err db 13, 10, "No file$"
    ovl1_name db "ovl1.ovl", 0
    ovl2_name db "ovl2.ovl", 0
    path db 100h dup(0)
    ovl_name_offset dw 0
    name_pos dw 0
    mem_err dw 0
    data_buf db 43 dup(0)
data ends

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

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

free_mem proc far
    push ax
    push bx
    push cx
    push dx
    push es
    and dx, 0
    mov mem_error, 0h
```

```

    mov ax, offset es
    mov bx, offset end_code
    add ax, bx
    mov bx, 16
    div bx
    add ax, 50h
    mov bx, ax
    and ax, 0
    mov ah, 4ah
    int 21h
    jnc end_free
    mov mem_error, 1h
    cmp ax, 7
    jne check_8
    mov dx, offset error_7_memory_block_destroyed
    call print
    jmp end_free
check_8:
    cmp ax, 8
    jne check_9
    mov dx, offset error_8_memory_for_execution
    call print
    jmp end_free
check_9:
    cmp ax, 9
    jne end_free
    mov dx, offset error_9_invalid_base
    call print
    jmp end_free
end_free:
    pop es
    pop dx
    pop cx
    pop bx
    pop ax
    ret
free_mem endp

```

```

overlay proc near
    push ax
    push bx
    push cx
    push dx
    push si
    mov ovl_name_offset, ax
    mov ax, psp
    mov es, ax
    mov es, es:[2ch]
    mov si, 0
check_0:
    mov ax, es:[si]
    inc si
    cmp ax, 0
    jne check_0

```

```

add     si, 3
mov     di, 0
path_find:
mov     al, es:[si]
cmp     al, 0
je      path_add
cmp     al, '\\'
jne     add_char
mov     name_pos, di
add_char:
mov     byte ptr [path + di], al
inc     di
inc     si
jmp     path_find
path_add:
cld
mov     di, name_pos
inc     di
add     di, offset path
mov     si, ovl_name_offset
mov     ax, ds
mov     es, ax
add_an_char:
lodsb
stosb
cmp     al, 0
jne     add_an_char
mov     ax, 1a00h
mov     dx, offset data_buf
int     21h
mov     ah, 4eh
mov     cx, 0
mov     dx, offset path
int     21h
jnc     all_good
mov     dx, offset no_overlay_err
call    print
cmp     ax, 2
je      err_file
cmp     ax, 3
je      err_path
jmp     end_path
err_file:
mov     dx, offset file_err
call    print
jmp     end_path
err_path:
mov     dx, offset path_err
call    print
jmp     end_path
all_good:
mov     si, offset data_buf
add     si, 1ah
mov     bx, [si]
mov     ax, [si + 2]

```



```

mov     cl, 4
shr     bx, cl
mov     cl, 12
shl     ax, cl
add     bx, ax
add     bx, 2
mov     ax, 4800h
int     21h
jnc     seg_load
jmp     end_path
seg_load:
mov     ovl_seg, ax
mov     dx, offset path
push    ds
pop     es
mov     bx, offset ovl_seg
mov     ax, 4b03h
int     21h
jnc     load_done
mov     dx, offset load_err
call    print
jmp     end_path
load_done:
mov     ax, ovl_seg
mov     es, ax
mov     word ptr ovl_addr + 2, ax
call    ovl_addr
mov     es, ax
mov     ah, 49h
int     21h
end_path:
pop     si
pop     dx
pop     cx
pop     bx
pop     ax
ret
overlay endp

main proc
mov     ax, data
mov     ds, ax
mov     psp, es
call    free_mem
cmp     mem_err, 1
je      main_end
mov     dx, offset ovl1_str
mov     ax, offset ovl1_name
call    overlay
mov     dx, offset ovl2_str
mov     ax, offset ovl2_name
call    overlay
main_end:
mov     ax, 4c00h
int     21h

```

```
    end_code:  
main endp  
code ends  
end    main
```