

**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

Сергеев Д.А.

Преподаватель

Ефремов М.А.

Санкт-Петербург

2022

### **Цель работы.**

Исследование структуры оверлейного сегмента и способа загрузки и выполнения оверлейных сегментов. Написание программы, состоящей из нескольких модулей.

### **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.

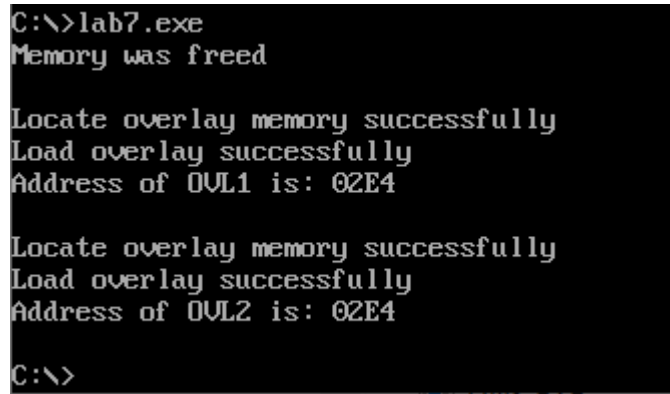
**Шаг 5.** Запустите приложение в случае, когда одного оверлея нет в каталоге. Приложение должно закончиться аварийно.

### **Выполнение работы.**

**Шаг 1.** Был написан и отлажен программный модуль типа .EXE, требуемый по заданию.

**Шаг 2.** Были написаны и отлажены оверлейные сегменты, требуемые по заданию.

**Шаг 3.** Программа была запущена из каталога с разработанными оверлейными сегментами.



```
C:\>lab7.exe
Memory was freed

Locate overlay memory successfully
Load overlay successfully
Address of OVL1 is: 02E4

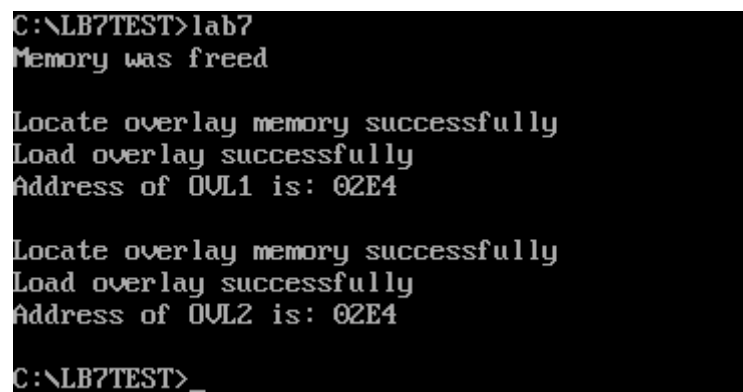
Locate overlay memory successfully
Load overlay successfully
Address of OVL2 is: 02E4

C:\>
```

Рисунок 1 – Результат работы программы

Как видно на рисунке, программа работает корректно: оверлейные сегменты загружаются с одно и того же адреса, перекрывая друг друга.

**Шаг 4.** Программа была запущена из другого каталога.



```
C:\LB7TEST>lab7
Memory was freed

Locate overlay memory successfully
Load overlay successfully
Address of OVL1 is: 02E4

Locate overlay memory successfully
Load overlay successfully
Address of OVL2 is: 02E4

C:\LB7TEST>_
```

Рисунок 2 – Результат работы программы из каталога LB7TEST

Как видно на рисунке 2, программа работает корректно.

**Шаг 5.** Программа была запущена, при этом из каталога был удалён файл ovl2.ovl

```
C:\LB7TEST>lab7
Memory was freed

Locate overlay memory successfully
Load overlay successfully
Address of OVL1 is: 02E4

Error locate overlay: file not found
Error load overlay: file not found

C:\LB7TEST>
```

Рисунок 3 – Результат работы программы при выполнении шага 5.

Как видно на рисунке 3, разработанная программа корректно обрабатывает ошибки.

Исходный программный код смотрите в приложении А.

### **Контрольные вопросы.**

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

Ответ: так как речь идет о .COM модуле, в начале у нас будет смещение в 100h. Из этого следует, что в дальнейшем при обращении к данным нужно его учитывать и вычитать его из адреса этих данных.

### **Выводы.**

В результате выполнения лабораторной работы, были исследованы структуры оверлейного сегмента и способ загрузки и выполнения оверлейных сегментов. Написана программы, состоящая из нескольких модулей.

## ПРИЛОЖЕНИЕ А

### ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: lab7.asm

```
ASTACK SEGMENT STACK
```

```
    DW 256 DUP(?)
```

```
ASTACK ENDS
```

```
DATA SEGMENT
```

```
    data_buffer db 43 DUP(0)
```

```
    overlay_address dd 0
```

```
    overlay_1_name db 'ovl1.ovl', 0h
```

```
    overlay_2_name db 'ovl2.ovl', 0h
```

```
    overlay_path db 128 DUP(0)
```

```
    keep_ss dw 0
```

```
    keep_sp dw 0
```

```
    free_memory db 0
```

```
    str_free_memory_mcb_error db 'Free memory error: MCB crashed', 0DH,  
0AH, '$'
```

```
    str_free_memory_not_enough_error db 'Free memory error: not enough  
memory', 0DH, 0AH, '$'
```

```
    str_free_memory_address_error db 'Free memory error: wrong address',  
0DH, 0AH, '$'
```

```
    str_free_memory_successfully db 'Memory was freed', 0DH, 0AH, '$'
```

```
    str_error_locate_overlay_file db 'Error locate overlay: file not found',  
0DH, 0AH, '$'
```

```
    str_error_locate_overlay_route db 'Error locate overlay: route not  
found', 0DH, 0AH, '$'
```

```
    str_locate_overlay_memory db 'Locate overlay memory successfully', 0DH,  
0AH, '$'
```

```
    str_error_load_overlay_function db 'Error load overlay: function is  
not exist', 0DH, 0AH, '$'
```

```
    str_error_load_overlay_file db 'Error load overlay: file not found',  
0DH, 0AH, '$'
```

```
    str_error_load_overlay_route db 'Error load overlay: route not found',  
0DH, 0AH, '$'
```

```
    str_error_load_overlay_too_many_files db 'Error load overlay: too many  
files opened', 0DH, 0AH, '$'
```

```
    str_error_load_overlay_access db 'Error load overlay: no access', 0DH,  
0AH, '$'
```

```
    str_error_load_overlay_memory db 'Error load overlay: not enough  
memory', 0DH, 0AH, '$'
```

```
    str_error_load_overlay_env db 'Error load overlay: wrong environment',  
0DH, 0AH, '$'
```

```
    str_load_overlay_successfully db 'Load overlay successfully', 0DH, 0AH,  
'$'
```

```
    data_end db 0
```

```
DATA ENDS
```

```
CODE SEGMENT
```

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

```
;-----  
PRINT_MESSAGE PROC NEAR  
    push ax  
    mov ah, 9  
    int 21h  
    pop ax  
    ret  
PRINT_MESSAGE ENDP  
  
PRINT_EOF PROC NEAR  
    push ax  
    push dx  
    mov dl, 0dh  
    push ax  
    mov ah, 02h  
    int 21h  
    pop ax  
    mov dl, 0ah  
    push ax  
    mov ah, 02h  
    int 21h  
    pop ax  
    pop dx  
    pop ax  
    ret  
PRINT_EOF ENDP  
;-----  
  
;-----  
FREE_MEMORY_PROC PROC FAR  
    push ax  
    push bx  
    push cx  
    push dx  
    push es  
    xor dx, dx  
    mov free_memory, 0h  
    mov ax, offset data_end  
    mov bx, offset finish  
    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_successfully  
    mov free_memory, 1h  
    cmp ax, 7  
    jne free_memory_not_enough_error  
    mov dx, offset str_free_memory_mcb_error  
    call PRINT_MESSAGE  
    jmp free_memory_exit  
  
free_memory_not_enough_error:
```

```

        cmp ax, 8
        jne free_memory_address_error
        mov dx, offset str_free_memory_not_enough_error
        call PRINT_MESSAGE
        jmp free_memory_exit

free_memory_address_error:
        cmp ax, 9
        jne free_memory_exit
        mov dx, offset str_free_memory_address_error
        call PRINT_MESSAGE
        jmp free_memory_exit

free_memory_successfully:
        mov dx, offset str_free_memory_successfully
        call PRINT_MESSAGE
        jmp free_memory_exit

free_memory_exit:
        pop es
        pop dx
        pop cx
        pop bx
        pop ax
        ret
FREE_MEMORY_PROC ENDP
;-----

;-----
LOAD_OVERLAY PROC FAR
        push ax
        push bx
        push cx
        push dx
        push es
        push ds
        push es
        mov keep_sp, sp
        mov keep_ss, ss
        mov ax, data
        mov es, ax
        mov bx, offset overlay_address
        mov dx, offset overlay_path
        mov ax, 4b03h
        int 21h
        mov ss, keep_ss
        mov sp, keep_sp
        pop es
        pop ds
        jnc load_overlay_successfully

        cmp ax, 1
        je error_load_overlay_function
        cmp ax, 2
        je error_load_overlay_file
        cmp ax, 3
        je error_load_overlay_route
        cmp ax, 4

```

```

        je error_load_overlay_too_many_files
        cmp ax, 5
        je error_load_overlay_access
        cmp ax, 8
        je error_load_overlay_memory
        cmp ax, 10
        je error_load_overlay_env

error_load_overlay_function:
        mov dx, offset str_error_load_overlay_function
        call PRINT_MESSAGE
        jmp load_overlay_finish

error_load_overlay_file:
        mov dx, offset str_error_load_overlay_file
        call PRINT_MESSAGE
        jmp load_overlay_finish

error_load_overlay_route:
        mov dx, offset str_error_load_overlay_route
        call PRINT_MESSAGE
        jmp load_overlay_finish

error_load_overlay_too_many_files:
        mov dx, offset str_error_load_overlay_too_many_files
        call PRINT_MESSAGE
        jmp load_overlay_finish

error_load_overlay_access:
        mov dx, offset str_error_load_overlay_access
        call PRINT_MESSAGE
        jmp load_overlay_finish

error_load_overlay_memory:
        mov dx, offset str_error_load_overlay_memory
        call PRINT_MESSAGE
        jmp load_overlay_finish

error_load_overlay_env:
        mov dx, offset str_error_load_overlay_env
        call PRINT_MESSAGE
        jmp load_overlay_finish

load_overlay_successfully:
        mov dx, offset str_load_overlay_successfully
        call PRINT_MESSAGE
        mov bx, offset overlay_address
        mov ax, [bx]
        mov cx, [bx + 2]
        mov [bx], cx
        mov [bx + 2], ax
        call overlay_address
        mov es, ax
        mov ah, 49h
        int 21h

load_overlay_finish:
        pop es

```



```

        pop dx
        pop cx
        pop bx
        pop ax
        ret
LOAD_OVERLAY ENDP
;-----
;-----
PATH_1 PROC NEAR
    push ax
    push dx
    push es
    push di
    xor di, di
    mov ax, es:[2ch]
    mov es, ax

loop_for_PATH_1:
    mov dl, es:[di]
    cmp dl, 0
    je go_to_path
    inc di
    jmp loop_for_PATH_1

go_to_path:
    inc di
    mov dl, es:[di]
    cmp dl, 0
    jne loop_for_PATH_1
    call PATH_2
    pop di
    pop es
    pop dx
    pop ax
    ret
PATH_1 ENDP

PATH_2 PROC NEAR
    push ax
    push bx
    push bp
    push dx
    push es
    push di
    mov bx, offset overlay_path
    add di, 3

loop_for_symbol_boot:
    mov dl, es:[di]
    mov [bx], dl
    cmp dl, '.'
    je loop_for_symbol_slash
    inc di
    inc bx
    jmp loop_for_symbol_boot

loop_for_symbol_slash:

```

```

        mov dl, [bx]
        cmp dl, '\'
        je get_overlay_name
        mov dl, 0h
        mov [bx], dl
        dec bx
        jmp loop_for_symbol_slash

get_overlay_name:
        mov di, si
        inc bx

add_overlay_name:
        mov dl, [di]
        cmp dl, 0h
        je path_exit
        mov [bx], dl
        inc bx
        inc di
        jmp add_overlay_name

path_exit:
        mov [bx], dl
        pop di
        pop es
        pop dx
        pop bp
        pop bx
        pop ax
        ret
PATH_2 ENDP
;-----

;-----

LOCATE_OVERLAY PROC FAR
        push ax
        push bx
        push cx
        push dx
        push di
        mov dx, offset data_buffer
        mov ah, 1ah
        int 21h
        mov dx, offset overlay_path
        mov cx, 0
        mov ah, 4eh
        int 21h
        jnc locate_overlay_successfully
        cmp ax, 12h
        jne error_locate_overlay_route
        mov dx, offset str_error_locate_overlay_file
        call PRINT_MESSAGE
        jmp locate_overlay_finish

error_locate_overlay_route:
        cmp ax, 3
        jne locate_overlay_finish
        mov dx, offset str_error_locate_overlay_route

```

```

        call PRINT_MESSAGE
        jmp locate_overlay_finish

locate_overlay_successfully:
    mov di, offset data_buffer
    mov dx, [di + 1ch]
    mov ax, [di + 1ah]
    mov bx, 10h
    div bx
    add ax, 1h
    mov bx, ax
    mov ah, 48h
    int 21h
    mov bx, offset overlay_address
    mov cx, 0000h
    mov [bx], ax
    mov [bx + 2], cx
    mov dx, offset str_locate_overlay_memory
    call PRINT_MESSAGE

locate_overlay_finish:
    pop di
    pop dx
    pop cx
    pop bx
    pop ax
    ret

LOCATE_OVERLAY ENDP
;-----

;-----

MAIN PROC FAR
    mov ax, data
    mov ds, ax
    call FREE_MEMORY_PROC
    cmp free_memory, 0h
    jne main_end
    call PRINT_EOF
    mov si, offset overlay_1_name
    call PATH_1
    call LOCATE_OVERLAY
    call LOAD_OVERLAY
    call PRINT_EOF
    mov si, offset overlay_2_name
    call PATH_1
    call LOCATE_OVERLAY
    call LOAD_OVERLAY

main_end:
    xor al, al
    mov ah, 4ch
    int 21h
MAIN ENDP

finish:
CODE ENDS
END MAIN

```

## Название файла: ov11.asm

```
CODE SEGMENT
    ASSUME CS:CODE, DS:NOTHING, SS:NOTHING

MAIN PROC FAR
    push ax
    push dx
    push ds
    push di
    mov ax, cs
    mov ds, ax
    mov di, offset overlay_address
    add di, 23
    call WRD_TO_HEX
    mov dx, offset overlay_address
    call WRITEWRD
    pop di
    pop ds
    pop dx
    pop ax
    retf
MAIN ENDP

overlay_address db 'Address of OVL1 is:      ', 0DH, 0AH, '$'

WRITEWRD PROC NEAR
    push ax
    mov ah, 9
    int 21h
    pop ax
    ret
WRITEWRD ENDP

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

```

```

CODE ENDS
END MAIN

```

## Название файла: ovl2.asm

```

CODE SEGMENT
    ASSUME CS:CODE, DS:NOTHING, SS:NOTHING

MAIN PROC FAR
    push ax
    push dx
    push ds
    push di
    mov ax, cs
    mov ds, ax
    mov di, offset overlay_address
    add di, 23
    call WRD_TO_HEX
    mov dx, offset overlay_address
    call WRITEWRD
    pop di
    pop ds
    pop dx
    pop ax
    retf
MAIN ENDP

overlay_address db 'Address of OVL2 is:      ', 0DH, 0AH, '$'

WRITEWRD PROC NEAR
    push ax
    mov ah, 9
    int 21h
    pop ax
    ret
WRITEWRD ENDP

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

```

```

CODE ENDS
END MAIN

```