

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

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

Студент гр.8382

\_\_\_\_\_

Синельников М.Р

Преподаватель

\_\_\_\_\_

Ефремов М.А

Санкт-Петербург

2020

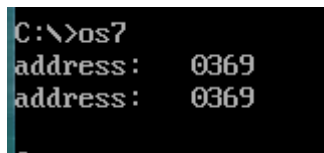
### Цель работы.

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.

### Ход работы.

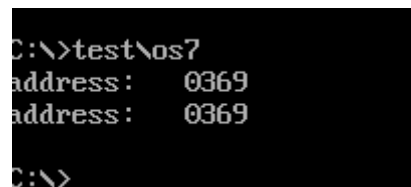
1)



```
C:\>os7
address: 0369
address: 0369
```

рисунок 1 — модули в одном каталоге

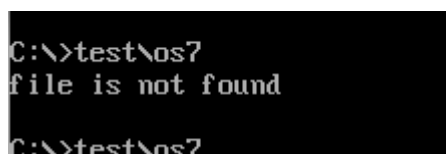
2)



```
C:\>test\os7
address: 0369
address: 0369
C:\>
```

рисунок 2 — все модули в другом каталоге

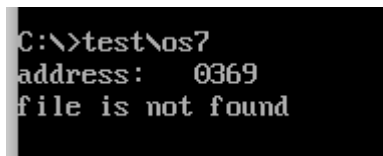
3)



```
C:\>test\os7
file is not found
C:\>test\os7
```

рисунок 3 — отсутствие первого оверлея

4)



```
C:\>test\os7  
address: 0369  
file is not found
```

рисунок 4 — отсутствие второго оверлея

### **Контрольные вопросы.**

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

Нужно вызывать оверлейный сегмент со смещением 100h, иначе не сформируется psp. Также нужно сохранять значения регистров.

### **Вывод.**

В ходе выполнения работы была исследована возможность построения модуля оверлейной структуры структуры.

## Приложение А

### Исходный код файла OS7.asm

```
        AStack SEGMENT      STACK
                dw 50 DUP(0)
AStack ENDS

DATA    SEGMENT

Message1 db 'Main memory block is destroyed', 0dh, 0ah, '$'
Message2 db 'Memory is not enough', 0dh, 0ah, '$'
Message3 db 'Address is not recognised', 0dh, 0ah, '$'

Message4 db 'file is not found', 0dh, 0ah, '$'
Message5 db 'road is not found', 0dh, 0ah, '$'

Message6 db 'such function doesnt exist', 0dh, 0ah, '$'
Message7 db 'file is not found', 0dh, 0ah, '$'
Message8 db 'road is not found', 0dh, 0ah, '$'
Message9 db 'number of open files is enormous', 0dh, 0ah, '$'
Message10 db 'access is denied', 0dh, 0ah, '$'
Message11 db 'memory is not enough', 0dh, 0ah, '$'
Message12 db 'environment is wrong', 0dh, 0ah, '$'

DTA db 43 DUP(?)
overlay_seg dw 0
overlay_address dd 0

keep_ds dw 0
keep_ss dw 0
keep_sp dw 0
path db 128 dup(?)
name1 db 'OVERLAY1.OVL', 0
name2 db 'OVERLAY2.OVL', 0

DATA ENDS

CODE    SEGMENT
```

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

print\_endl proc

push ax

push dx

mov dl, 13

mov ah, 02h

int 21h

mov dl, 10

int 21h

pop dx

pop ax

ret

print\_endl endp

print\_message proc

push ax

mov ah, 09h

int 21h

pop ax

ret

print\_message endp

set\_dta proc

push ax

push dx

mov dx, offset dta

mov ah, 1ah

int 21h

pop dx

pop ax

set\_dta endp

free\_memory proc

push ax

push dx

push es

```
    lea ax, end_of_program
    inc ax
    mov bx, ax
    mov al, 0
    mov ah, 4ah
    int 21h
    jnc finish
```

```
    cmp ax,7
    jne next1
    lea dx,message1
    call print_message
    jmp error_exit
```

```
next1:
    cmp ax,8
    jne next2
    lea dx,message2
    call print_message
    jmp error_exit
```

```
next2:
    cmp ax,9
    jne finish
    lea dx,message3
    call print_message
```

```
error_exit:
    mov ax, 4C00h
    int 21h
```

```
finish:
    pop es
    pop dx
    pop ax
    ret
```

```
free_memory endp
```

```
prepare_path proc
    push ax
```

```
push bx
push di
push si
push es
mov es,es:[2ch]
mov si,0
cmp byte ptr es:[si],00h
je go
mov ah,02h
do2:
    mov bl,es:[si]
    inc si
    cmp byte ptr es:[si],0
    jne do2
    cmp byte ptr es:[si + 1],0
    jne do2
```

```
go:
lea di, PATH
add si,4
cmp byte ptr es:[si],00h
je path_end
do3:
    mov bl,es:[si]
    mov [di],bl
    inc si
    inc di
    cmp byte ptr es:[si],00h
    jne do3
```

```
path_end:
    sub di,7
    mov si, dx
    name_loop:
        mov bl, [si]
        mov [di], bl
        inc si
        inc di
        cmp bl,00h
        jne name_loop
```

```
mov [di],bl
```

```

    path_ret:
    pop es
    pop si
    pop di
    pop bx
    pop ax
    ret
prepare_path endp

set_new_memory_size proc
    push ax
    push bx
    push cx
    push dx

    mov cx,0
    mov dx,offset PATH
    mov ax,4e00h
    int 21h
    jnc loading_ok

    cmp ax,2
    je error2

    cmp ax,3
    je error3

error2:
    mov dx,offset message4
    jmp print_error

error3:
    mov dx,offset message5
    jmp print_error

print_error:
    call print_message
    mov ax, 4C00h
    int 21h

loading_ok:
    mov bx, offset dta

```



```
mov ax, [bx+1ch]
mov bx, [bx+1ah]
mov cl, 12
shl ax, cl
mov cl, 4
shr bx, cl
add bx, ax
inc bx
mov ah, 48h
int 21h
```

```
mov overlay_seg, ax
pop dx
pop cx
pop bx
pop ax
ret
```

```
set_new_memory_size endp
```

```
run proc
```

```
push ax
push bx
push dx
push es
```

```
push es
mov keep_ss, ss
mov keep_sp, sp
mov keep_ds, ds
mov dx, offset path
mov bx, seg overlay_seg
mov es, bx
mov bx, offset overlay_seg
```

```
mov ax, 4b03h
int 21h
```

```
mov ds, keep_ds
mov ss, keep_ss
mov sp, keep_sp
pop es
jnc run_ok
```

```
cmp ax,1
je run_error_1
```

```
cmp ax,2
je run_error_2
```

```
cmp ax,3
je run_error_3
```

```
cmp ax,4
je run_error_4
```

```
cmp ax,5
je run_error_5
```

```
cmp ax,8
je run_error_8
```

```
cmp ax,10
je run_error_10
```

```
run_error_1:
mov dx,offset message6
jmp print_run_error
```

```
run_error_2:
mov dx,offset message7
jmp print_run_error
```

```
run_error_3:
mov dx,offset message8
jmp print_run_error
```

```
run_error_4:
mov dx,offset message9
jmp print_run_error
```

```
run_error_5:
mov dx,offset message10
jmp print_run_error
```

```
run_error_8:
mov dx,offset message11
jmp print_run_error
```

```
run_error_10:
mov dx,offset message12
jmp print_run_error
```

```
print_run_error:
call print_message
mov ax, 4C00h
int 21h
```

```
run_ok:
mov ax, overlay_seg
mov word ptr overlay_address+2, ax
call overlay_address
mov ax, overlay_seg
mov es, ax
mov ah, 49h
int 21h
```

```
pop es
pop dx
pop bx
pop ax
ret
```

run endp

```
Main  PROC  FAR
mov ax, data
mov ds, ax
call set_DTA
call free_memory
mov dx, offset name1
call prepare_path
call set_new_memory_size
call run
xor bx,bx
```

```
mov dx, offset name2
call prepare_path
call set_new_memory_size
call run
mov ax, 4c00h
int 21h
ret
```

Main ENDP

end\_of\_program:

CODE ENDS

END Main

## Приложение В

### Исходный код файла overlay1.asm

OVERLAY1 SEGMENT

ASSUME CS:OVERLAY1 DS:OVERLAY1

Main PROC FAR

```
    push ax
    push bx
    push dx
    push ds
    push di
    mov ax, cs
    mov ds, ax
    mov bx, offset address
    add bx, 14
    mov di, bx
    mov ax, cs
    call WRD_TO_HEX
    mov dx, offset address
    call print_message
    pop di
    pop ds
    pop dx
    pop bx
    pop ax
    retf
```

Main ENDP

address db 'address: ',0dh,0ah,'\$'

print\_message proc

```
    push ax
    mov ah, 09h
    int 21h
    pop ax
    ret
```

print\_message 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
                    xor     ah, ah
                    call    BYTE_TO_HEX
                    mov     [di], ah
                    dec     di
                    mov     [di], al
                    pop     bx
                    ret
WRD_TO_HEX       ENDP

```

OVERLAY1 ENDS

END Main

## Приложение С

### Исходный код файла overlay2.asm

OVERLAY2 SEGMENT

ASSUME CS:OVERLAY2 DS:OVERLAY2

Main PROC FAR

```
    push ax
    push bx
    push dx
    push ds
    push di
    mov ax, cs
    mov ds, ax
    mov bx, offset address
    add bx, 14
    mov di, bx
    mov ax, cs
    call WRD_TO_HEX
    mov dx, offset address
    call print_message
    pop di
    pop ds
    pop dx
    pop bx
    pop ax
    retf
```

Main ENDP

address db 'address: ',0dh,0ah,'\$'

print\_message proc

```
    push ax
    mov ah, 09h
    int 21h
    pop ax
    ret
```

print\_message 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
    xor          ah, ah
    call         BYTE_TO_HEX
    mov          [di], ah
    dec          di
    mov          [di], al
    pop          bx
    ret
WRD_TO_HEX       ENDP

```

OVERLAY2 ENDS

END Main