

**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. 9383

\_\_\_\_\_

Арутюнян С.Н.

Преподаватель

\_\_\_\_\_

Ефремов М.А.

Санкт-Петербург

2021

## Цель работы

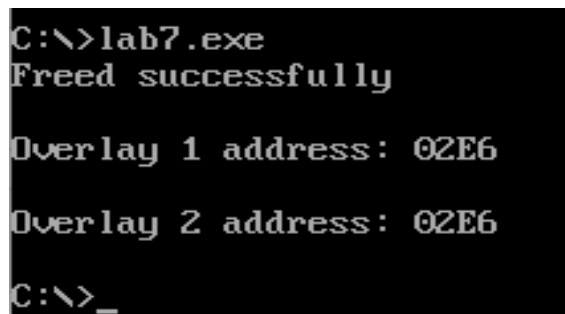
Исследовать возможности построения загрузочного модуля оверлейной структуры, структуру оверлейного сегмента и способ загрузки и выполнения оверлейных сегментов.

## Выполнение работы

**Шаг 1.** Разработал и отладил программный модуль типа .EXE, который выполняет следующие 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) Действия 1-4 выполняются повторно для второго сегмента.

**Шаг 2.** Написал и отладил оверлейные сегменты. Оверлейные сегменты выводят сообщения о том, что они были успешно загружены с одного и того же адреса.



```
C:\>lab7.exe
Freed successfully

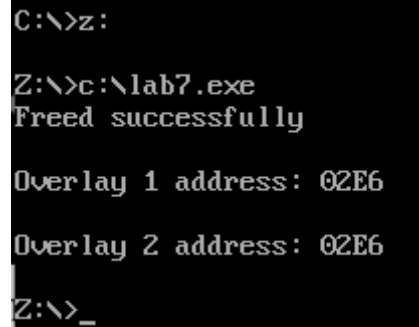
Overlay 1 address: 02E6

Overlay 2 address: 02E6

C:\>_
```

Рисунок 1. Вывод головной программы и оверлеев из корневого каталога

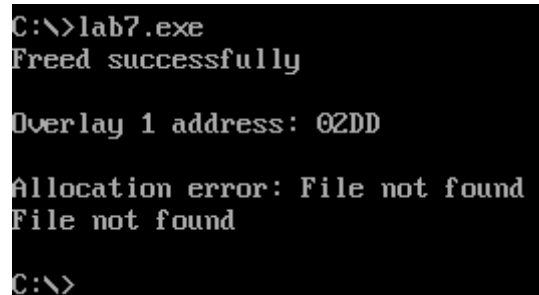
**Шаг 3.** Запустил программу из другого каталога:



```
C:\>z:  
  
Z:\>c:\lab7.exe  
Freed successfully  
  
Overlay 1 address: 02E6  
  
Overlay 2 address: 02E6  
  
Z:\>_
```

Рисунок 2. Вывод головной программы и оверлеев из другого каталога

**Шаг 4.** Запустил программу с отсутствующим оверлеем overlay2.ovl:



```
C:\>lab7.exe  
Freed successfully  
  
Overlay 1 address: 02DD  
  
Allocation error: File not found  
File not found  
  
C:\>
```

Рисунок 3. Вывод головной программы и оверлеев из каталога с отсутствующим оверлеем overlay2.ovl

## **Контрольные вопросы**

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

.СOM модули всегда используют смещение 100h, поэтому при обращении к данным необходимо будет учитывать это смещение и вычитать его из адреса данных.

## **Заключение**

В процессе выполнения лабораторной работы был изучен механизм работы модулей оверлейной структуры и способ загрузки и выполнения оверлейных сегментов.

## Приложение А

ASTACK SEGMENT STACK

DW 256 DUP(?)

ASTACK ENDS

DATA SEGMENT

NEWLINE db 0dh, 0ah, '\$'

DTA db 43 DUP(0)

OVERLAY\_ADDRESS dd 0

OVERLAY\_NAME1 db 'overlay1.ovl', 0h

OVERLAY\_NAME2 db 'overlay2.ovl', 0h

OVERLAY\_PATH db 128 DUP(0)

KEEP\_SS dw 0

KEEP\_SP dw 0

ERROR\_MEM\_FREE db 0

MCB\_ERROR\_STRING db 'Memory Free Error: Memory Control Block has crashed',  
0DH, 0AH, '\$'

OUT\_OF\_MEM\_ERROR\_STR db 'Memory Free Error: Not Enough Memory', 0DH, 0AH,  
'\$'

WRONG\_ADDRESS\_ERROR\_STRING db 'Memory Free Error: Wrong Address', 0DH,  
0AH, '\$'

FREE\_OK\_MESSAGE db 'Freed successfully', 0DH, 0AH, '\$'

FILE\_NOT\_FOUND\_ERROR\_STRING db 'Allocation error: File not found', 0DH, 0AH,  
'\$'

ROUTE\_NOT\_FOUND\_ERROR\_STRING db 'Overlay Allocation Error: Route not found',  
0DH, 0AH, '\$'

ALLOCATED\_MEM\_FOR\_OVERLAY\_MESSAGE db 'Allocated memory for overlay  
successfully', 0DH, 0AH, '\$'

OVERLAY\_FUNCTION\_NOT\_EXIST\_ERROR db 'Function does not exist', 0DH, 0AH,  
'\$'

OVERLAY\_FILE\_NOT\_FOUND\_ERROR db 'File not found', 0DH, 0AH, '\$'

OVERLAY\_ROUTE\_NOT\_FOUND db 'Route not found', 0DH, 0AH, '\$'

OVERLAY\_TOO\_MANY\_FILES\_OPENED\_ERROR db 'Too many files opened', 0DH,  
0AH, '\$'

OVERLAY\_NO\_ACCESS\_ERROR db 'No access', 0DH, 0AH, '\$'

OVERLAY\_NOT\_ENOUGH\_MEMORY\_ERROR db 'Not enough memory', 0DH, 0AH,  
'\$'

OVERLAY\_WRONG\_ENV\_ERROR db 'Wrong environment', 0DH, 0AH, '\$'

OVERLAY\_LOAD\_SUCCESS\_ERROR db 'Overlay loaded successfully', 0DH, 0AH, '\$'

DATA\_END db 0

DATA ENDS

CODE SEGMENT

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

PRINT\_NEWLINE proc near

push ax

push dx

mov dx, offset NEWLINE

mov ah, 9h

int 21h

pop dx

pop ax

ret

PRINT\_NEWLINE endp

WRITE\_STRING proc near

push ax

mov ah, 9h

int 21h

pop ax

ret

WRITE\_STRING endp

FREE\_UNUSED\_MEMORY PROC FAR

push ax

push bx

push cx

push dx

push es

xor dx, dx

mov ERROR\_MEM\_FREE, 0h

mov ax, offset DATA\_END

mov bx, offset main\_proc\_end\_byte



add ax, bx

mov bx, 10h

div bx

add ax, 100h

mov bx, ax

xor ax, ax

mov ah, 4ah

int 21h

jnc free\_without\_error

mov ERROR\_MEM\_FREE, 1h

cmp ax, 7

jne not\_enough\_memory

mov dx, offset MCB\_ERROR\_STRING

call WRITE\_STRING

jmp free\_unused\_memory\_end

not\_enough\_memory:

cmp ax, 8

jne wrong\_address

mov dx, offset OUT\_OF\_MEM\_ERROR\_STR

call WRITE\_STRING

jmp free\_unused\_memory\_end

wrong\_address:

```
cmp ax, 9  
jne free_unused_memory_end
```

```
mov dx, offset WRONG_ADDRESS_ERROR_STRING  
call WRITE_STRING  
jmp free_unused_memory_end
```

```
free_without_error:
```

```
mov dx, offset FREE_OK_MESSAGE  
call WRITE_STRING
```

```
free_unused_memory_end:
```

```
pop es  
pop dx  
pop cx  
pop bx  
pop ax
```

```
ret
```

```
FREE_UNUSED_MEMORY 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 loaded_successfully
```

```
;function does not exist error
```

```
cmp ax, 1
```

```
    jne load_file_not_found
```

```
    mov dx, offset OVERLAY_FUNCTION_NOT_EXIST_ERROR
```

```
    call WRITE_STRING
```

```
    jmp load_module_end
```

```
load_file_not_found:
```

```
    cmp ax, 2
```

```
    jne load_route_error
```

```
    mov dx, offset OVERLAY_FILE_NOT_FOUND_ERROR
```

```
    call WRITE_STRING
```

```
jmp load_module_end
```

```
load_route_error:
```

```
    cmp ax, 3
    jne load_too_many_files_opened
    mov dx, offset OVERLAY_ROUTE_NOT_FOUND
    call WRITE_STRING
    jmp load_module_end
```

```
load_too_many_files_opened:
```

```
    cmp ax, 4
    jne load_no_access_error
    mov dx, offset OVERLAY_TOO_MANY_FILES_OPENED_ERROR
    call WRITE_STRING
    jmp load_module_end
```

```
load_no_access_error:
```

```
    cmp ax, 5
    jne load_not_enough_memory
    mov dx, offset OVERLAY_NO_ACCESS_ERROR
    call WRITE_STRING
    jmp load_module_end
```

```
load_not_enough_memory:
```

```
    cmp ax, 8
    jne load_wrong_env
    mov dx, offset OVERLAY_NOT_ENOUGH_MEMORY_ERROR
    call WRITE_STRING
    jmp load_module_end
```

```
load_wrong_env:
```

```
    cmp ax, 10
```

```
jne load_module_end
mov dx, offset OVERLAY_WRONG_ENV_ERROR
call WRITE_STRING
jmp load_module_end
```

loaded\_successfully:

```
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\_module\_end:

```
pop es
```

```
pop dx
```

```
pop cx
```

```
pop bx
```

```
pop ax
```

```
ret
```

LOAD\_OVERLAY ENDP

GET\_PATH PROC NEAR ;name in si

push ax

push dx

push es

push di

xor di, di

mov ax, es:[2ch]

mov es, ax

content\_loop:

mov dl, es:[di]

cmp dl, 0

je end\_string2

inc di

jmp content\_loop

end\_string2:

inc di

mov dl, es:[di]

cmp dl, 0

jne content\_loop

call PARSE\_PATH

pop di

pop es

pop dx

pop ax

ret

GET\_PATH ENDP

PARSE\_PATH PROC NEAR

push ax

push bx

push bp

push dx

push es

push di

mov bx, offset OVERLAY\_PATH

add di, 3

boot\_loop:

mov dl, es:[di]

mov [bx], dl

cmp dl, '.'

je parse\_to\_slash

inc di

inc bx

jmp boot\_loop

parse\_to\_slash:

mov dl, [bx]

cmp dl, '\'

je get\_overlay\_name

```
mov dl, 0h  
mov [bx], dl
```

```
dec bx  
jmp parse_to_slash
```

```
get_overlay_name:  
    mov di, si ; si - overlay_name  
    inc bx
```

```
add_overlay_name:  
    mov dl, [di]  
    cmp dl, 0h  
    je parse_path_end
```

```
    mov [bx], dl
```

```
    inc bx  
    inc di
```

```
    jmp add_overlay_name
```

```
parse_path_end:  
    mov [bx], dl
```

```
    pop di  
    pop es  
    pop dx  
    pop bp  
    pop bx  
    pop ax
```



ret

PARSE\_PATH ENDP

ALLOCATE\_FOR\_OVERLAY PROC FAR

push ax

push bx

push cx

push dx

push di

mov dx, offset DTA

mov ah, 1ah

int 21h

mov dx, offset OVERLAY\_PATH

mov cx, 0

mov ah, 4eh

int 21h

jnc got\_size\_successfully

;file not found error

cmp ax, 12h

jne route\_error

mov dx, offset FILE\_NOT\_FOUND\_ERROR\_STRING

call WRITE\_STRING

jmp allocate\_for\_overlay\_end

route\_error:

```
cmp ax, 3
jne allocate_for_overlay_end
mov dx, offset ROUTE_NOT_FOUND_ERROR_STRING
call WRITE_STRING
jmp allocate_for_overlay_end
```

got\_size\_successfully:

```
mov di, offset DTA
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
```

allocate\_for\_overlay\_end:

```
pop di
pop dx
pop cx
pop bx
pop ax
ret
```

ALLOCATE\_FOR\_OVERLAY ENDP

MAIN PROC FAR

mov ax, data

mov ds, ax

call FREE\_UNUSED\_MEMORY

cmp ERROR\_MEM\_FREE, 0h

jne main\_end

call PRINT\_NEWLINE

mov si, offset OVERLAY\_NAME1

call GET\_PATH

call ALLOCATE\_FOR\_OVERLAY

call LOAD\_OVERLAY

call PRINT\_NEWLINE

mov si, offset OVERLAY\_NAME2

call GET\_PATH

call ALLOCATE\_FOR\_OVERLAY

call LOAD\_OVERLAY

main\_end:

xor al, al

mov ah, 4ch

int 21h

MAIN ENDP

main\_proc\_end\_byte:

CODE ENDS

END MAIN

## Приложение Б

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 overlay1\_address

add di, 22

call WRD\_TO\_HEX

mov dx, offset overlay1\_address

call WRITEWRD

pop di

pop ds

pop dx

pop ax

retf

MAIN ENDP

overlay1\_address db 'Overlay 1 address: ', 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

## Приложение В

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 overlay1\_address

add di, 22

call WRD\_TO\_HEX

mov dx, offset overlay1\_address

call WRITEWRD

pop di

pop ds

pop dx

pop ax

retf

MAIN ENDP

overlay1\_address db 'Overlay 2 address: ', 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