

**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. 0382

\_\_\_\_\_

Корсунов А.А.

Преподаватель

\_\_\_\_\_

Ефремов М.А.

Санкт-Петербург

2022

## Постановка задачи

### Цель работы.

Исследование возможности построения загрузочного модуля оверлейной структуры.

**Сведения о функциях и структурах данных управляющей программы.**

Процедура	Описание
MODULE_PATH	Получение пути до вызываемого модуля .OVL
GET_PATH	Получение пути до вызываемого каталога .OVL
FREE	Освобождение памяти выделенную под программу
LOAD	Загрузка вызываемого модуля .OVL
OVERLAY_MEMORY	Освобождение и определение памяти для модуля .OVL
MAIN	Главная функция программы

### 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.

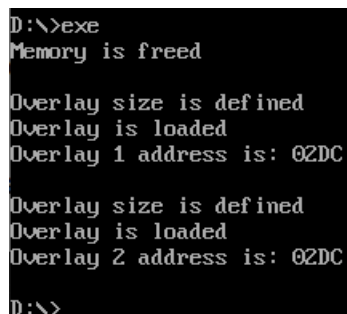
**Шаг 5.** Запустите приложение в случае, когда одного оверлея нет в каталоге. Приложение должно закончиться аварийно.

### Выполнение работы

**Шаг 1.** Для выполнения лабораторной работы был написан и отлажен программный модуль типа .EXE, который выполняет требуемые по заданию функции.

**Шаг 2.** Также были написаны и отлажены оверлейные сегменты. Оверлейный сегмент выводит адрес сегмента, в который он загружен.

**Шаг 3.** Были запущенно отлаженное приложение (текущий каталог — каталог с разработанными модулями).



```
D:\>exe
Memory is freed

Overlay size is defined
Overlay is loaded
Overlay 1 address is: 02DC

Overlay size is defined
Overlay is loaded
Overlay 2 address is: 02DC

D:\>
```

Рисунок 1 — Иллюстрация работы программы (текущий каталог — каталог с разработанными модулями). Оверлейные сегменты загружены из одного адреса.

**Шаг 4.** Приложение было запущено из другого каталога.

```
D:\KORSUNOV>exe
Memory is freed

Overlay is loaded
Overlay 1 address is: 0000

Overlay is loaded
Overlay 2 address is: 0000

D:\KORSUNOV>
```

Рисунок 2 — Иллюстрация работы программы (текущий каталог — «другой» каталог).

**Шаг 5.** Приложение было запущено, когда одного оверлея нет в каталоге. Приложение соответствующе реагирует на его отсутствие.

```
D:\KORSUNOV\ANTON>exe
Memory is freed

Overlay size is defined
Overlay is loaded
Overlay 1 address is: 02DC

Error(load): file is not found

D:\KORSUNOV\ANTON>_
```

Рисунок 3 — Иллюстрация работы программы (второй оверлейный сегмент отсутствует)

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

Ответ. В силу того, что в .COM модуле есть смещение 100h, при обращении к данным нужно будет вычитать его из их адреса.

## **Вывод.**

Было произведено исследование возможности построения загрузочного модуля оверлейной структуры.

## ПРИЛОЖЕНИЕ А

### ИСХОДНЫЙ КОД ПРОГРАММЫ

файл **exe.asm**:

*MYSTACK SEGMENT STACK*

*DW 256 DUP(?)*

*MYSTACK ENDS*

*DATA SEGMENT*

*adr\_overlay dd 0*

*file\_name\_overlay\_1 db 'overlay1.ovl', 0h*

*file\_name\_overlay\_2 db 'overlay2.ovl', 0h*

*file\_path\_overlay db 128 DUP(0)*

*keep\_SS dw 0*

*keep\_SP dw 0*

*mem\_error db 0*

*free\_memory\_mcb\_str db 'Error: MCB crashed', 0DH, 0AH, '\$'*

*free\_memory\_need\_more\_str db 'Error: It needs more memory', 0DH, 0AH, '\$'*

*free\_memory\_address\_str db 'Error: Wrong address', 0DH, 0AH, '\$'*

*free\_memory\_success\_str db 'Memory is freed', 0DH, 0AH, '\$'*

*overlay\_load\_function\_str db 'Error: function is not exist', 0DH, 0AH, '\$'*

*overlay\_load\_file\_not\_found\_str db 'Error(load): file is not found', 0DH, 0AH, '\$'*

*overlay\_load\_route\_not\_found\_str db 'Error(load): route is not found', 0DH, 0AH, '\$'*

*overlay\_load\_too\_many\_files\_str db 'Error: too many files are opened', 0DH, 0AH, '\$'*

*overlay\_load\_no\_access\_str db 'Error: no access', 0DH, 0AH, '\$'*

*overlay\_load\_need\_more\_str db 'Error(load): It needs more memory', 0DH, 0AH, '\$'*

*overlay\_load\_wrong\_enviroment\_str db 'Error: wrong environment', 0DH, 0AH, '\$'*

*overlay\_load\_success\_str db 'Overlay is loaded', 0DH, 0AH, '\$'*

*size\_overlay\_file\_not\_found\_str db 'Error: file is not found', 0DH, 0AH, '\$'*

*size\_overlay\_route\_not\_found\_str db 'Error: route is not found', 0DH, 0AH, '\$'*

*size\_overlay\_success\_str db 'Overlay size is defined', 0DH, 0AH, '\$'*

*DTA\_buffer db 43 DUP(0)*

*end\_of\_data db 0*

*DATA ENDS*

*CODE SEGMENT*

*ASSUME CS:CODE, DS:DATA, SS:MYSTACK*

*MODULE\_PATH PROC near*

*push AX*

*push BX*

*push BP*

*push DX*

*push ES*

*push DI*

*mov BX, offset file\_path\_overlay*

*add DI, 3*

*loop1:*

```
mov DL, ES:[DI]
mov [BX], DL
cmp DL, '.'
je slash
inc DI
inc BX
jmp loop1
```

*slash:*

```
mov DL, [BX]
cmp DL, '\'
je module_name
mov DL, 0h
mov [BX], DL
dec BX
jmp slash
```

*module\_name:*

```
mov DI, SI
inc BX
```

*add\_name:*

```
mov DL, [DI]
cmp DL, 0h
je module_path_end
mov [BX], DL
inc BX
inc DI
jmp add_name
```

```

    module_path_end:
        mov [BX], DL
        pop DI
        pop ES
        pop DX
        pop BP
        pop BX
        pop AX

    ret
MODULE_PATH ENDP

GET_PATH PROC near; имя файла находится в SI
    push AX
    push DX
    push ES
    push DI

    xor DI, DI
    mov AX, ES:[2ch]
    mov ES, AX

    loop2:
        mov DL, ES:[DI]
        cmp DL, 0
        je endl
        inc DI
        jmp loop2

    endl:
        inc DI

```



```
mov DL, ES:[DI]
cmp DL, 0
jne loop2
```

```
call MODULE_PATH
```

```
pop DI
pop ES
pop DX
pop AX
ret
```

```
GET_PATH ENDP
```

```
OVERLAY_MEMORY PROC far
push AX
push BX
push CX
push DX
push DI
```

```
mov DX, offset DTA_buffer
mov AH, 1ah
int 21h
mov DX, offset file_path_overlay
mov CX, 0
mov AH, 4eh
int 21h
jnc size_success
```

```
cmp AX, 2
```

```
jne route_2
mov DX, offset size_overlay_file_not_found_str
call WRITE_MESSAGE_WORD
jmp overlay_memory_end
```

*route\_2:*

```
    cmp AX, 3
    jne overlay_memory_end
    mov DX, offset size_overlay_route_not_found_str
    call WRITE_MESSAGE_WORD
    jmp overlay_memory_end
```

*size\_success:*

```
    mov DI, offset DTA_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 adr_overlay
    mov CX, 0000h
    mov [BX], AX
    mov [BX+2], CX

    mov DX, offset size_overlay_success_str
    call WRITE_MESSAGE_WORD
```

```

        overlay_memory_end:
            pop DI
            pop DX
            pop CX
            pop BX
            pop AX

        ret
OVERLAY_MEMORY ENDP

```

```

FREE PROC far

```

```

    push AX
    push BX
    push CX
    push DX
    push ES

```

```

    xor DX, DX

```

```

    mov mem_error, 0h

```

```

    mov AX, offset end_of_data

```

```

    mov BX, offset main_fin

```

```

    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\_success*

*mov mem\_error, 1h*

*cmp AX, 7*

*jne free\_memory\_need\_more*

*mov DX, offset free\_memory\_mcb\_str*

*call WRITE\_MESSAGE\_WORD*

*jmp free\_end*

*free\_memory\_need\_more:*

*cmp AX, 8*

*jne free\_memory\_address*

*mov DX, offset free\_memory\_need\_more\_str*

*call WRITE\_MESSAGE\_WORD*

*jmp free\_end*

*free\_memory\_address:*

*cmp AX, 9*

*jne free\_end*

*mov DX, offset free\_memory\_address\_str*

*call WRITE\_MESSAGE\_WORD*

*jmp free\_end*

*free\_memory\_success:*

*mov DX, offset free\_memory\_success\_str*

*call WRITE\_MESSAGE\_WORD*

```

        free_end:
            pop ES
            pop DX
            pop CX
            pop BX
            pop AX

        ret

FREE ENDP

LOAD 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 adr_overlay
    mov DX, offset file_path_overlay

    mov AX, 4b03h
    int 21h

    mov SS, keep_SS

```

*mov SP, keep\_SP*

*pop ES*

*pop DS*

*jnc success\_load*

*cmp AX, 1*

*jne load\_file\_not\_found*

*mov DX, offset overlay\_load\_function\_str*

*call WRITE\_MESSAGE\_WORD*

*jmp load\_end*

*load\_file\_not\_found:*

*cmp AX, 2*

*jne load\_route*

*mov DX, offset overlay\_load\_file\_not\_found\_str*

*call WRITE\_MESSAGE\_WORD*

*jmp load\_end*

*load\_route:*

*cmp AX, 3*

*jne load\_too\_many\_files*

*mov DX, offset overlay\_load\_route\_not\_found\_str*

*call WRITE\_MESSAGE\_WORD*

*jmp load\_end*

*load\_too\_many\_files:*

*cmp AX, 4*

*jne load\_no\_access*

```
mov DX, offset overlay_load_too_many_files_str
call WRITE_MESSAGE_WORD
jmp load_end
```

```
load_no_access:
    cmp AX, 5
    jne load_need_more
    mov DX, offset overlay_load_no_access_str
    call WRITE_MESSAGE_WORD
    jmp load_end
```

```
load_need_more:
    cmp AX, 8
    jne load_wrong_enviroment
    mov DX, offset overlay_load_need_more_str
    call WRITE_MESSAGE_WORD
    jmp load_end
```

```
load_wrong_enviroment:
    cmp AX, 10
    jne load_end
    mov DX, offset overlay_load_wrong_enviroment_str
    call WRITE_MESSAGE_WORD
    jmp load_end
```

```
success_load:
    mov DX, offset overlay_load_success_str
    call WRITE_MESSAGE_WORD
```

```
mov BX, offset adr_overlay
```

```
mov AX, [BX]
mov CX, [BX+2]
mov [BX], CX
mov [BX+2], AX
call adr_overlay
mov ES, AX
mov AH, 49h
int 21h
```

```
load_end:
```

```
    pop ES
    pop DX
    pop CX
    pop BX
    pop AX
```

```
ret
```

```
LOAD ENDP
```

```
    NEXT_LINE PROC near
```

```
    push AX
```

```
    push DX
```

```
    mov DL, 0DH
```

```
    call WRITE_MESSAGE_BYTE
```

```
    mov DL, 0AH
```

```
    call WRITE_MESSAGE_BYTE
```

```
    pop DX
```

```
    pop AX
```



*ret*

*NEXT\_LINE ENDP*

*WRITE\_MESSAGE\_WORD PROC near*  
*push AX*

*mov AH, 9*

*int 21h*

*pop AX*

*ret*

*WRITE\_MESSAGE\_WORD ENDP*

*WRITE\_MESSAGE\_BYTE PROC near*  
*push AX*

*mov AH, 02h*

*int 21h*

*pop AX*

*ret*

*WRITE\_MESSAGE\_BYTE ENDP*

*MAIN PROC far*

*mov AX, data*

*mov DS, AX*

*call FREE*

*cmp mem\_error, 0h*

*jne main\_end*

```
call NEXT_LINE  
mov SI, offset file_name_overlay_1  
call GET_PATH  
call OVERLAY_MEMORY  
call LOAD
```

```
call NEXT_LINE  
mov SI, offset file_name_overlay_2  
call GET_PATH  
call OVERLAY_MEMORY  
call LOAD
```

```
main_end:  
xor AL, AL  
mov AH, 4ch  
int 21h
```

```
MAIN ENDP
```

```
main_fin:  
CODE ENDS
```

```
END MAIN
```

```
файл overlay1.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 overlay1\_address\_str*

*add DI, 25*

*call WRD\_TO\_HEX*

*mov DX, offset overlay1\_address\_str*

*call WRITE\_MESSAGE\_WORD*

*pop DI*

*pop DS*

*pop DX*

*pop AX*

*retf*

*MAIN 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*

```
WRITE_MESSAGE_WORD PROC near  
push AX
```

```
mov AH, 9
```

```
int 21h
```

```
pop AX
```

```
ret
```

```
WRITE_MESSAGE_WORD ENDP
```

```
overlay1_address_str db 'Overlay 1 address is: ', 0DH, 0AH, '$'
```

```
CODE ENDS
```

```
END MAIN
```

файл overlay2.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 overlay2_address_str
```

```
add DI, 25
```

```
call WRD_TO_HEX
```

```
mov DX, offset overlay2_address_str
call WRITE_MESSAGE_WORD
```

```
pop DI
pop DS
pop DX
pop AX
retf
```

```
MAIN 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
```

```
WRITE_MESSAGE_WORD PROC near
    push AX

    mov AH, 9
    int 21h

    pop AX
    ret
WRITE_MESSAGE_WORD ENDP
```

```
overlay2_address_str db 'Overlay 2 address is:    ', 0DH, 0AH, '$'
```

```
CODE ENDS
```

```
END MAIN
```