

**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. 9382

\_\_\_\_\_

Субботин М.О.

Преподаватель

\_\_\_\_\_

Ефремов М.А.

Санкт-Петербург

2021

## **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.

### Ход выполнения:

Работа программы в директории с двумя оверлеями:

```
D:\LAB7>lab7.exe
memory allocated successfully
Normal ending

OVL1 address:      0200

memory allocated successfully
Normal ending

OVL2 address:      0200

D:\LAB7>_
```

Работа программы при запуске из другой директории:

```
D:\>lab7\lab7.exe
memory allocated successfully
Normal ending

OVL1 address:      0200

memory allocated successfully
Normal ending

OVL2 address:      0200

D:\>_
```

Работа программы при запуске только со вторым оверлеем в директории:

```
D:\LAB7>lab7.exe
File not found
File was not found

memory allocated successfully
Normal ending

OVL2 address:      0200

D:\LAB7>_
```

Работа программы при запуске только с первым оверлеем в директории:

```
D:\LAB7>lab7.exe
memory allocated successfully
Normal ending

OVL1 address:      0200

File not found
File was not found

D:\LAB7>_
```

Работа программы при запуске без оверлеев в директории:

```
D:\LAB7>lab7.exe
File not found
File was not found

File not found
File was not found

D:\LAB7>_
```

### Ответы на контрольные вопросы.

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

После записи значений регистров в стек, надо в регистр DS вставить значение регистра CS, т.к. адрес сегмента данных совпадает с сегментом кода. Также следует добавить 100h, т.к. сегменты настроены на PSP.

### Выводы.

Была исследована возможность построения загрузочного модуля оверлейной структуры.

## ПРИЛОЖЕНИЕ А

### КОД ПРОГРАММ

#### Lab7.asm:

```
AStack SEGMENT STACK
        DW 128 DUP(?)
AStack ENDS
```

#### DATA SEGMENT

```
FILE_1 db 'FILE1.OVL', 0
FILE_2 db 'FILE2.OVL', 0
prog dw 0
data_mem db 43 dup(0)
POS_CL db 128 dup(0)
ovls_addr dd 0
KEEP_PSP dw 0
```

```
eof db 13, 10, '$'
MEMORY_N7 db 'Destroyed memory block',13,10,'$'
MEMORY_N8 db 'Not enough memory for running function',13,10,'$'
MEMORY_N9 db 'Incorrect memorys address',13,10,'$'
```

```
ERROR_N1 db 'Wrong functions number',13,10,'$'
ERROR_N2 db 'File was not found',13,10,'$'
ERROR_N5 db 'Disk error',13,10,'$'
ERROR_N8 db 'Disk has not enough free memory space',13,10,'$'
ERROR_N10 db 'Wrong string enviroment',13,10,'$'
ERROR_N11 db 'Incorrect format',13,10,'$'
```

```
END_N0 db 'Normal ending',13,10,'$'
END_N1 db 'Ending by ctrl-break',13,10,'$'
END_N2 db 'Ending by device error',13,10,'$'
END_N3 db 'Ending by 31h function',13,10,'$'
```

```
ALLOCATE_SUC_STR db 'memory allocated successfully', 13, 10, '$'
FILE_ERROR_STR db 'File not found', 13, 10, '$'
ROUTE_ERROR_STR db 'Route not found', 13, 10, '$'
```

```
end_data db 0
DATA ENDS
```

#### CODE SEGMENT

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

WRITE\_STRING PROC

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

WRITE\_STRING ENDP

FREE\_MEMORY PROC

```
push ax
push bx
push cx
push dx

mov ax, offset end_data
mov bx, offset END_APP
add bx, ax
shr bx, 1
shr bx, 1
shr bx, 1
shr bx, 1
add bx, 2bh
mov ah, 4ah
int 21h
```

jnc END\_FREE\_MEMORY

```
lea dx, MEMORY_N7
cmp ax, 7
je WRITE_MEMORY_COMMENT
lea dx, MEMORY_N8
cmp ax, 8
je WRITE_MEMORY_COMMENT
lea dx, MEMORY_N9
cmp ax, 9
je WRITE_MEMORY_COMMENT
jmp END_FREE_MEMORY
```

WRITE\_MEMORY\_COMMENT:

```
mov ah, 09h
int 21h
```

END\_FREE\_MEMORY:

```
pop dx
pop cx
pop bx
pop ax
ret
FREE_MEMORY ENDP
```

```
SET_FULL_FILENAME PROC NEAR
```

```
push ax
push bx
push cx
push dx
push di
push si
push es
```

```
mov prog, dx
```

```
mov ax, KEEP_PSP
mov es, ax
mov es, es:[2ch]
mov bx, 0
```

```
FIND_SMTH:
```

```
inc bx
cmp byte ptr es:[bx-1], 0
jne FIND_SMTH
cmp byte ptr es:[bx+1], 0
jne FIND_SMTH
```

```
add bx, 2
mov di, 0
```

```
FIND_LOOP:
```

```
mov dl, es:[bx]
mov byte ptr [POS_CL + di], dl
inc di
inc bx
cmp dl, 0
je END_LOOP
cmp dl, '\'
jne FIND_LOOP
mov cx, di
jmp FIND_LOOP
```

END\_LOOP:

mov di, cx  
mov si, prog

LOOP\_2:

mov dl, byte ptr[si]  
mov byte ptr [POS\_CL + di], dl  
inc di  
inc si  
cmp dl, 0  
jne LOOP\_2

pop es  
pop si  
pop di  
pop dx  
pop cx  
pop bx  
pop ax  
ret

SET\_FULL\_FILENAME ENDP

DEPLOY\_ANOTHER\_PROGRAM PROC NEAR

push ax  
push bx  
push cx  
push dx  
push ds  
push es

mov ax, DATA  
mov es, ax  
mov bx, offset ovls\_addr  
mov dx, offset POS\_CL  
mov ax, 4b03h  
int 21h

jnc COME\_OVER

err\_1:  
cmp ax, 1  
jne err\_2  
mov dx, offset ERROR\_N1  
call WRITE\_STRING



jmp DEPLOY\_END

err\_2:  
cmp ax, 2  
jne err\_5  
mov dx, offset ERROR\_N2  
call WRITE\_STRING  
jmp DEPLOY\_END

err\_5:  
cmp ax, 5  
jne err\_8  
mov dx, offset ERROR\_N5  
call WRITE\_STRING  
jmp DEPLOY\_END

err\_8:  
cmp ax, 8  
jne err\_10  
mov dx, offset ERROR\_N8  
call WRITE\_STRING  
jmp DEPLOY\_END

err\_10:  
cmp ax, 10  
jne err\_11  
mov dx, offset ERROR\_N10  
call WRITE\_STRING  
jmp DEPLOY\_END

err\_11:  
cmp ax, 11  
mov dx, offset ERROR\_N11  
call WRITE\_STRING  
jmp DEPLOY\_END

COME\_OVER:  
mov dx, offset END\_N0  
call WRITE\_STRING

mov ax, word ptr ovls\_addr  
mov es, ax  
mov word ptr ovls\_addr, 0  
mov word ptr ovls\_addr + 2, ax

```

call ovls_addr
mov es, ax
mov ah, 49h
int 21h

DEPLOY_END:
pop es
pop ds
pop dx
pop cx
pop bx
pop ax
ret
DEPLOY_ANOTHER_PROGRAM ENDP

ALLOCATE_MEMORY PROC
push ax
push bx
push cx
push dx

push dx
mov dx, offset data_mem
mov ah, 1ah
int 21h
pop dx
mov cx, 0
mov ah, 4eh
int 21h

jnc ALLOCATE_SUCCESS

cmp ax, 2
je ROUTE_ERR
mov dx, offset FILE_ERROR_STR
call WRITE_STRING
jmp ALLOCATE_END

ROUTE_ERR:
cmp ax, 3
mov dx, offset ROUTE_ERROR_STR
call WRITE_STRING
jmp ALLOCATE_END

```

ALLOCATE\_SUCCESS:

```
    push di
    mov di, offset data_mem
    mov bx, [di + 1ah]
    mov ax, [di + 1ch]
    pop di
    push cx
    mov cl, 4
    shr bx, cl
    mov cl, 12
    shl ax, cl
    pop cx
    add bx, ax
    add bx, 1
    mov ah, 48h
    int 21h
    mov word ptr ovls_addr, ax
    mov dx, offset ALLOCATE_SUC_STR
    call WRITE_STRING
```

ALLOCATE\_END:

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

ALLOCATE\_MEMORY ENDP

START\_OVL PROC

```
    push dx
    call SET_FULL_FILENAME
    mov dx, offset POS_CL
    call ALLOCATE_MEMORY
    call DEPLOY_ANOTHER_PROGRAM
    pop dx
    ret
```

START\_OVL ENDP

Main PROC FAR

```
    push ds
    xor ax, ax
    push ax
```

```

mov ax, DATA
mov ds, ax
mov KEEP_PSP, es
call FREE_MEMORY
mov dx, offset FILE_1
call START_OVL
mov dx, offset eof
call WRITE_STRING
mov dx, offset FILE_2
call START_OVL

VERY_END:
xor al,al
mov ah,4ch
int 21h
Main ENDP
END_APP:
CODE ENDS
    END Main

```

## **FILE1.asm**

```

ovl1 SEGMENT
ASSUME CS:ovl1, DS:NOTHING, SS:NOTHING
MAIN PROC FAR
    push ax
    push dx
    push ds
    push di

    mov ax, cs
    mov ds, ax
    mov di, offset ovl
    add di, 23
    call WRD_TO_HEX
    mov dx, offset ovl
    call WRITE_STRING

    pop di
    pop ds
    pop dx
    pop ax
    retf
main endp

```

ovl db 13, 10, "OVL1 address: ", 13, 10, '\$'

WRITE\_STRING PROC

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

WRITE\_STRING ENDP

TETR\_TO\_HEX PROC

```
and al,0fh
cmp al,09
jbe next
add al,07
```

next:

```
add al,30h
ret
```

TETR\_TO\_HEX ENDP

BYTE\_TO\_HEX PROC

```
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

```
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

```

```

ovl1 ENDS
END MAIN

```

## FILE2.ASM

```

OVL2 SEGMENT
    ASSUME CS:OVL2, DS:NOTHING, SS:NOTHING
    MAIN PROC FAR
        PUSH AX
        PUSH DX
        PUSH DS
        PUSH DI

        MOV AX, CS
        MOV DS, AX
        MOV DI, OFFSET OVL
        ADD DI, 23
        CALL WRD_TO_HEX
        MOV DX, OFFSET OVL
        CALL WRITE_STRING

        POP DI
        POP DS
        POP DX
        POP AX
        RETF
    MAIN ENDP

    OVL DB 13, 10, "OVL2 ADDRESS:      ", 13, 10, '$'

    WRITE_STRING PROC
        PUSH DX

```

```

    PUSH AX
    MOV AH, 09H
    INT 21H
    POP AX
    POP DX
    RET
WRITE_STRING ENDP

```

```

TETR_TO_HEX PROC
    AND AL, 0FH
    CMP AL, 09
    JBE NEXT
    ADD AL, 07
NEXT:
    ADD AL, 30H
    RET
TETR_TO_HEX ENDP

```

```

BYTE_TO_HEX PROC
    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
    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
OVL2 ENDS
END MAIN
```