

**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**

Студентка гр. 0382

Кривенцова Л.С.

Преподаватель

Ефремов М.А.

Санкт-Петербург

2022

### **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 оверлейные модули находятся в одном каталоге.

### **Задание.**

1. Для выполнения лабораторной работы необходимо написать и отладить программный модуль типа .EXE, который выполняет функции:

- Освобождает память для загрузки оверлеев;
- Читает размер файла оверлея и запрашивает объем памяти, достаточный для его загрузки;
- Файл оверлейного сегмента загружается и выполняется;
- Освобождается память, отведенная для оверлейного сегмента;
- Затем действия 1)-4) выполняются для оверлейного сегмента;

2. Также необходимо написать и отладить оверлейные сегменты. Оверлейный сегмент выводит адрес сегмента, в который он загружен.

3. Запустите отлаженное приложение. Оверлейные сегменты должны загружаться с одного и того же адреса, перекрывая друг друга.

4. Запустите приложение из другого каталога. Приложение должно быть выполнено успешно.

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

6. Занесите полученные результаты в виде скриншотов в отчет. Оформите отчет в соответствии с требованиями.

### **Выполнение работы.**

Для выполнения лабораторной работы были написаны и отлажены программный модуль типа .EXE, оверлейные сегменты, которые выводят адрес сегмента, в который они загружены.

Были реализованы:

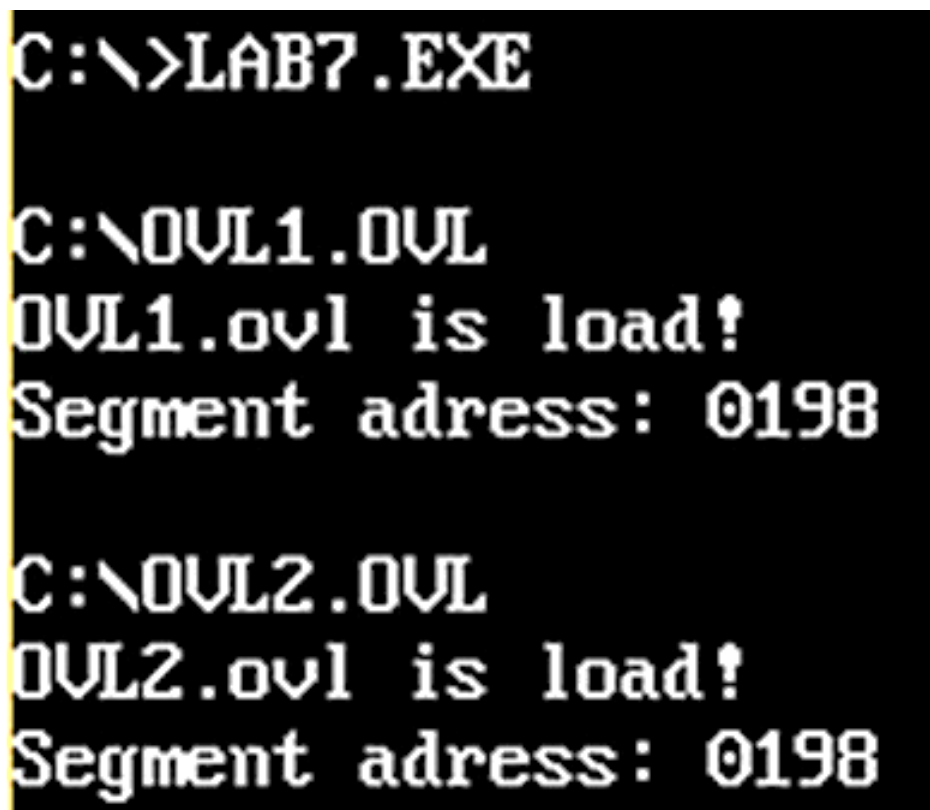
Название	Комментарий
DATA SEGMENT	
Структуры данных	
PATH	Путь к загружаемому модулю
LAUNCHING_SETTINGS	Настройки загрузки
ADDRESS_LAUNCH	Загрузочный адрес
DTA	Буфер для загрузки функции
Строки	
NAME_OVL1	Имя первого ovl модуля
NAME_OVL2	Имя второго ovl модуля
ERROR_FREEMEM	Сообщение для вывода: ошибка освобождения памяти 'Memory release error'
ERROR_DF	Сообщение для вывода: ошибка при запуске дочерней программы 'Defunct function'
ERROR_WF	Сообщение для вывода: ошибка при запуске дочерней программы 'File not found'
ERROR_WR	Сообщение для вывода: ошибка при запуске дочерней программы 'Route not found'
ERROR_MF	Сообщение для вывода: ошибка при запуске дочерней программы 'Too many files open'
ERROR_NOT	Сообщение для вывода: ошибка при запуске дочерней программы 'Not available'

CNTRL_MEMORY_BL	Сообщение для вывода: ошибка памяти 'Breakdown Control memory block'
NO_MEMORY	Сообщение для вывода: ошибка памяти 'Not enough free memory'
WRONG_MEMORY	Сообщение для вывода: ошибка памяти 'Wrong memory address'
DATA ENDS	

CODE SEGMENT	
Процедуры	
PRINT	Процедура для вывода на экран
FREEMEM	Процедура для освобождения памяти с учётом возможного возникновения ошибок
SET_FULL_FILE_NAME	Процедура считывания пути файла для построения всего пути
OVL_SIZE	Процедура получения размера ovl модуля
LOAD_OVERLAY	Процедура для загрузки ovl модуля, и его запуска с учётом возможного возникновения ошибок
MAKE_FILE_NAME	Вспомогательная процедура построения всего пути
OVL_LOADING	Вспомогательная процедура для загрузки ovl модуля

LINE	Процедура переносит указатель на следующую строку
Main	Функция (основная) для вызова процедур
CODE ENDS	

Запущено отлаженное приложение. Оверлейные сегменты загружаются с одного и того же адреса, перекрывая друг друга.



```

C:\>LAB7.EXE

C:\>OVL1.OVL
OVL1.ovl is load!
Segment adress: 0198

C:\>OVL2.OVL
OVL2.ovl is load!
Segment adress: 0198

```

Рис. 1 - Результат запуска отлаженного приложения.

Приложение запущено из другого каталога. Приложение было выполнено успешно.

```
C:\>\TEMP\LAB7.EXE

C:\TEMP\OVL1.OVL
OVL1.ovl is load!
Segment adress: 0198

C:\TEMP\OVL2.OVL
OVL2.ovl is load!
Segment adress: 0198
```

Рис. 2 - Результат успешного запуска отлаженного приложения из другого каталога.

Был произведён запуск приложения, когда одного оверлея нет в каталоге. Приложение завершилось аварийно.

```
C:\>\TEMP\LAB7.EXE

C:\TEMP\OVL1.OVL
Route not found
File not found

C:\TEMP\OVL2.OVL
OVL2.ovl is load!
Segment adress: 0198
```

Рис. 3 - Результат аварийно завершеного запуска отложенного приложения при отсутствии оверлея в каталоге.

### **Ответы на контрольные вопросы.**

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

- Чтобы использовать .COM модули в качестве оверлейного сегмента нужно также в начале выделенной памяти выделить память под стек и организовать блок PSP, при обращении к которому учитывать смещение 100h. Кроме того, необходимо сохранять регистры, чтобы восстановить их в конце модуля.

### **Выводы.**

Были исследованы возможности построения загрузочного модуля оверлейной структуры, структура оверлейного сегмента и способ загрузки и выполнения оверлейных сегментов.

## ПРИЛОЖЕНИЕ А

### ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: laba7.asm

```
AStack      SEGMENT  STACK
             DW 64 DUP(?)
AStack      ENDS

DATA        SEGMENT
NAME_OVL1 db 'OVL1.OVL$'
NAME_OVL2 db 'OVL2.OVL$'
PATH db 50 dup (0)
LAUNCHING_SETTINGS dw 0,0
ADDRESS_LAUNCH dd 0
DTA db 43 dup(0)
ERROR_FREEMEM db 'Memory release error',13,10,'$'
ERROR_DF db 'Defunct function', 13,10,'$'
ERROR_WF db 'File not found',13,10,'$'
ERROR_WR db 'Route not found',13,10,'$'
ERROR_MF db 'Too many files open',13,10,'$'
ERROR_NOT db 'Not available',13,10,'$'
CNTRL_MEMORY_BL db 'Breakdown Control memory block',13,10,'$'
NO_MEMORY db 'Not enough free memory',13,10,'$'
WRONG_MEMORY db 'Wrong memory address',13,10,'$'
DATA ENDS

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

PRINT PROC
push AX
mov AH,09h
int 21h
pop AX
ret
PRINT ENDP

FREEMEM PROC
push ax
push bx
lea bx, end_program
mov ax,es
sub bx,ax
mov ax,bx
shr bx,4
inc bx
mov ah,4ah
int 21h
jnc end_memory
lea dx, CNTRL_MEMORY_BL
cmp ax,7
je memory_write
lea dx, NO_MEMORY
cmp ax,7
je memory_write
lea dx,WRONG_MEMORY
```



```

cmp ax,7
je memory_write
jmp end_memory

memory_write:
call PRINT
jmp end_error_memory

end_memory:
pop bx
pop ax
ret

end_error_memory:
pop bx
mov AH,4Ch
int 21H
FREEMEM ENDP

SET_FULL_FILE_NAME PROC NEAR
    push dx
    push di
    push si
    push es

    xor di,di
    mov es,es:[2ch]

skip_content:
    mov dl,es:[di]
    cmp dl,0h
    je last_content
    inc di
    jmp skip_content

last_content:
    inc di
    mov dl,es:[di]
    cmp dl,0h
    jne skip_content
    add di,3h
    mov si,0

write_patch:
    mov dl,es:[di]
    cmp dl,0h
    je delete_file_name
    mov PATCH[si],dl
    inc di
    inc si
    jmp write_patch

delete_file_name:
    dec si
    cmp PATCH[si],'\ '
    je ready_add_file_name
    jmp delete_file_name

```

```

ready_add_file_name:
    mov di,-1

add_file_name:
    inc si
    inc di
    mov dl,bx[di]
    cmp dl,'$'
    je set_patch_end
    mov PATCH[si],dl
    jmp add_file_name

set_patch_end:
    mov PATCH[si],'$'
    pop es
    pop si
    pop di
    pop dx
    ret
SET_FULL_FILE_NAME ENDP

OVL_SIZE PROC NEAR
push ax
push bx
push cx
push dx
push bp
mov ah,1Ah
lea dx,DTA
    int 21h
mov ah,4Eh
    lea dx, PATCH
mov cx,0
int 21h
jnc memory_allocation
lea dx,ERROR_WF
cmp ax,2
je write_error_overlay_size
lea dx,ERROR_WR
cmp ax,3
je write_error_overlay_size

write_error_overlay_size:
call PRINT
jmp end_get_overlay_size

memory_allocation:
mov si, offset DTA
add si, 1Ah
mov bx, [si]
shr bx, 4
mov ax, [si+2]
shl ax, 12
add bx, ax
add bx, 2
    mov ah,48h
    int 21h

```

```

        jnc save_seg
        lea dx,ERROR_FREEMEM
        call PRINT
        jmp end_get_overlay_size

save_seg:
        mov LAUNCHING_SETTINGS,ax
        mov LAUNCHING_SETTINGS+2,ax

end_get_overlay_size:
pop bp
pop dx
pop cx
pop bx
pop ax
ret
OVL_SIZE ENDP

LOAD_OVERLAY PROC NEAR
push ax
push dx
push es
lea dx,PATCH
push ds
pop es
lea bx, LAUNCHING_SETTINGS
mov ax,4B03h
        int 21h
jnc success_load
lea dx, ERROR_DF
cmp ax,1
je write_error_load_overlay
lea dx, ERROR_WF
cmp ax,2
je write_error_load_overlay
lea dx, ERROR_WR
cmp ax,3
je write_error_load_overlay
lea dx, ERROR_MF
cmp ax,4
je write_error_load_overlay
lea dx, ERROR_NOT
cmp ax,5
je write_error_load_overlay
lea dx, NO_MEMORY
cmp ax,8
je write_error_load_overlay

write_error_load_overlay:
call PRINT
jmp end_overlay

success_load:
mov ax,LAUNCHING_SETTINGS
        mov word ptr ADDRESS_LAUNCH + 2, ax
        call ADDRESS_LAUNCH
        mov es,ax
mov ah, 49h

```

```

int 21h

end_overlay:
pop es
pop dx
pop ax
ret
LOAD_OVERLAY ENDP

MAKE_FILE_NAME MACRO OVERLAY_NAME
    push bx
    lea bx,OVERLAY_NAME
    call SET_FULL_FILE_NAME
    pop bx
ENDM

OVL_LOADING MACRO OVERLAY_NAME
push dx
MAKE_FILE_NAME OVERLAY_NAME
lea dx, PATCH
call PRINT
call LINE
call OVL_SIZE
call LOAD_OVERLAY
call LINE
pop dx
ENDM

LINE PROC
    push dx
    push ax
    mov dl,10
    mov ah,02h
    int 21h
    mov dl,13
    mov ah,02h
    int 21h
    pop ax
    pop dx
    ret
LINE ENDP

Main PROC FAR
sub    AX,AX
push  AX
mov    AX,DATA
mov    DS,AX
call  FREEMEM
call  LINE
OVL_LOADING NAME_OVL1
OVL_LOADING NAME_OVL2
xor    AL,AL
mov    AH,4Ch
int    21H
Main ENDP
end_program:
CODE ENDS

```

END Main

## Название файла: ovl1.asm

```
OVL1 SEGMENT
ASSUME CS:OVL1, DS:NOTHING, SS:NOTHING, ES:NOTHING
```

```
MAIN PROC FAR
push ax
push dx
push ds
push di
mov ax,cs
mov ds,ax
lea dx, LOADING
call PRINT
lea di, ADRESS
add di, 19
mov ax, cs
call WRD_TO_HEX
lea dx, ADRESS
call PRINT
pop di
pop ds
pop dx
pop ax
RETF
MAIN 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 ;в AL старшая цифра
    pop CX ;в AH младшая
    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
    call BYTE_TO_HEX
    mov [DI],AH
    dec DI
    mov [DI],AL
    pop BX
    ret
WRD_TO_HEX ENDP

PRINT PROC
    push AX
    mov AH,09h
    int 21h
    pop AX
    ret
PRINT ENDP

LOADING db 'OVL1.ovl is load! ',13,10,'$'
ADRESS db 'Segment adress:      ',13,10,'$'

OVL1 ENDS
END MAIN

```

## Название файла: ovl2.asm

```

OVL2 SEGMENT
ASSUME CS:OVL2, DS:NOTHING, SS:NOTHING, ES:NOTHING

MAIN PROC FAR
    push ax
    push dx
    push ds
    push di
    mov ax,cs
    mov ds,ax
    lea dx, LOADING
    call PRINT
    lea di, ADRESS
    add di, 19
    mov ax, cs
    call WRD_TO_HEX
    lea dx, ADRESS
    call PRINT
    pop di
    pop ds
    pop dx
    pop ax
    RETF
MAIN 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 ;в AL старшая цифра
    pop CX ;в AH младшая
    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

PRINT PROC
    push AX
    mov AH,09h
    int 21h
    pop AX
    ret
PRINT ENDP

LOADING db 'OVL2.ovl is load! ',13,10,'$'
ADRESS db 'Segment adress:      ',13,10,'$'

OVL2 ENDS
END MAIN

```