

**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

\_\_\_\_\_

Гладких А.А.

Преподаватель

\_\_\_\_\_

Ефремов М.А.

Санкт-Петербург

2021

## ПОСТАНОВКА ЗАДАЧИ

### Цель работы.

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

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

Процедура	Описание
WRITEWRD	Функция печати строки на экран
WRITEBYTE	Функция печати символа на экран
ENDLINE	Функция печати символов переноса строки
FREE_UNUSED_MEMORY	Функция очистки выделенную под программу память
LOAD_OVERLAY	Функция загрузки оверлея
ALLOCATE_FOR_OVERLAY	Функция выделения места под оверлей
GET_PATH	Функция получения пути до каталога с вызывающим модулем
PARSE_PATH	Функция получения пути до вызываемого модуля
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.

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

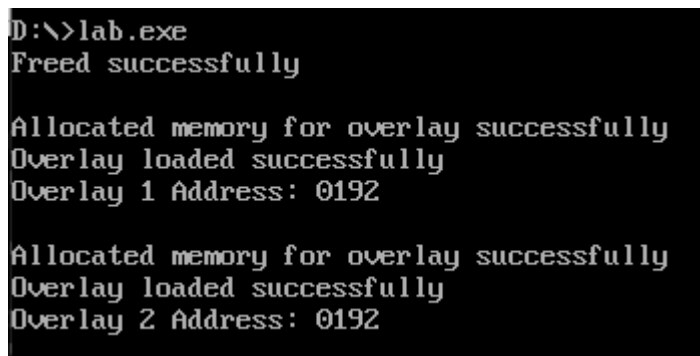
**Исходный код.**

Исходный код представлен в приложении А.

## РЕЗУЛЬТАТЫ ИССЛЕДОВАНИЯ ПРОБЛЕМ

**Шаг 1.** Был написан и отлажен программный модуль типа .EXE, который выполняет поставленные в задании функции. Также были написаны и отлажены два оверлейных сегмента.

**Шаг 2.** Программа была запущена из каталога с разработанными оверлейными сегментами. Программа успешно вызвала оба оверлея. Оба оверлейных сегмента были загружены с одного и того же адреса.



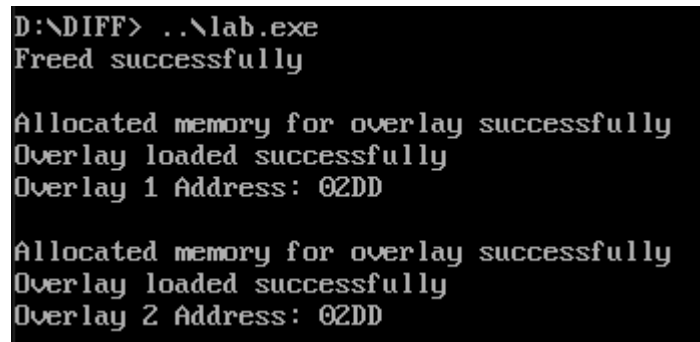
```
D:\>lab.exe
Freed successfully

Allocated memory for overlay successfully
Overlay loaded successfully
Overlay 1 Address: 0192

Allocated memory for overlay successfully
Overlay loaded successfully
Overlay 2 Address: 0192
```

Рисунок 1 - Иллюстрация работы программы

**Шаг 3.** Программа была запущена из другого каталога. Программа снова успешно вызвала оба оверлейных сегмента.



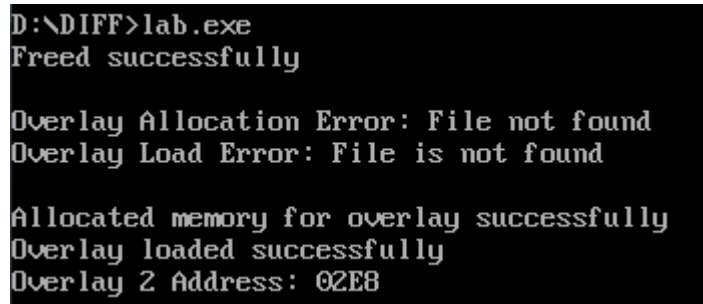
```
D:\DIFF> ..\lab.exe
Freed successfully

Allocated memory for overlay successfully
Overlay loaded successfully
Overlay 1 Address: 02DD

Allocated memory for overlay successfully
Overlay loaded successfully
Overlay 2 Address: 02DD
```

Рисунок 2 - Иллюстрация работы программы, запущенной из другого каталога

**Шаг 4.** Программа была запущена из каталога, в котором не присутствовал один оверлейный сегмент. Программа успешно обработала отсутствие оверлейного сегмента, что отражено на рисунке 3.



```
D:\DIFF>lab.exe
Freed successfully

Overlay Allocation Error: File not found
Overlay Load Error: File is not found

Allocated memory for overlay successfully
Overlay loaded successfully
Overlay 2 Address: 02E8
```

Рисунок 3 - Иллюстрация работы программы, запущенной из каталога, в котором нет первого оверлейного сегмента

**Шаг 5.** Была произведена оценка результатов и были отвечены контрольные вопросы:

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

Ответ: в .COM модуле изначально будет присутствовать смещение в 100h, а так как оверлейный модуль имеет точку входа по адресу 0, то нужно будет вычитать это смещение при обращении к данным.

## **Выводы.**

Были исследованы возможности построения загрузочного модуля оверлейной структуры, структура оверлейного сегмента и способ загрузки и выполнения оверлейных сегментов. Была написана программа, которая подготавливает место под оверлейные сегменты и успешно вызывает оверлейные сегменты, а также обрабатывает исключительные ситуации, когда один или оба оверлейных сегмента отсутствуют.

# ПРИЛОЖЕНИЕ А

## ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: lab.asm

```

STACK SEGMENT STACK
    DW 256 DUP(?)
STACK ENDS

DATA SEGMENT
    dta_buffer db 43 DUP(0)

    overlay_addr dd 0
    overlay_name1 db 'overlay1.com', 0h
    overlay_name2 db 'overlay2.com', 0h
    overlay_path db 128 DUP(0)

    keep_ss dw 0
    keep_sp dw 0

    error_mem_free db 0
    mcb_crash_string db 'Memory Free Error: Memory Control Block has
crashed', 0DH, 0AH, '$'
    not_enough_memory_string db 'Memory Free Error: Not Enough Memory',
0DH, 0AH, '$'
    wrong_address_string db 'Memory Free Error: Wrong Address', 0DH, 0AH,
'$'
    free_without_error_string db 'Freed successfully', 0DH, 0AH, '$'

    file_not_found_error_string db 'Overlay 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_string db 'Allocated memory for overlay
successfully', 0DH, 0AH, '$'

    overlay_function_not_exist db 'Overlay Load Error: Function does not
exist', 0DH, 0AH, '$'
    overlay_file_not_found db 'Overlay Load Error: File is not found',
0DH, 0AH, '$'
```

```

        overlay_route_not_found db 'Overlay Load Error: Route not found', 0DH,
0AH, '$'

        overlay_too_many_files_opened db 'Overlay Load Error: Too many files
opened', 0DH, 0AH, '$'

        overlay_no_access db 'Overlay Load Error: No access', 0DH, 0AH, '$'
        overlay_not_enough_memory db 'Overlay Load Error: Not enough memory',
0DH, 0AH, '$'

        overlay_wrong_env db 'Overlay Load Error: Wrong environment', 0DH,
0AH, '$'

        overlay_load_success db 'Overlay loaded successfully', 0DH, 0AH, '$'


        data_end db 0
DATA ENDS


CODE SEGMENT

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


WRITEWRD  PROC  NEAR
        push ax
        mov ah, 9
        int 21h
        pop ax
        ret
WRITEWRD  ENDP


WRITEBYTE  PROC  NEAR
        push ax
        mov ah, 02h
        int 21h
        pop ax
        ret
WRITEBYTE  ENDP


ENDLINE  PROC  NEAR
        push ax
        push dx

        mov dl, 0dh
        call WRITEBYTE

        mov dl, 0ah
        call WRITEBYTE

```



```

    pop dx
    pop ax
    ret
ENDLINE 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 lafin
    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

; mcb crash
    cmp ax, 7
    jne not_enough_memory

    mov dx, offset mcb_crash_string

```

```

        call WRITEWRD
        jmp free_unused_memory_end

not_enough_memory:
        cmp ax, 8
        jne wrong_address

        mov dx, offset not_enough_memory_string
        call WRITEWRD
        jmp free_unused_memory_end

wrong_address:
        cmp ax, 9
        jne free_unused_memory_end

        mov dx, offset wrong_address_string
        call WRITEWRD
        jmp free_unused_memory_end

free_without_error:
        mov dx, offset free_without_error_string
        call WRITEWRD

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_addr

    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
    call WRITEWRD
    jmp load_module_end

load_file_not_found:
    cmp ax, 2
    jne load_route_error
    mov dx, offset overlay_file_not_found
    call WRITEWRD
    jmp load_module_end

load_route_error:
    cmp ax, 3
    jne load_too_many_files_opened
    mov dx, offset overlay_route_not_found
    call WRITEWRD
    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
    call WRITEWRD
    jmp load_module_end

load_no_access_error:
    cmp ax, 5
    jne load_not_enough_memory
    mov dx, offset overlay_no_access
    call WRITEWRD
    jmp load_module_end

load_not_enough_memory:
    cmp ax, 8
    jne load_wrong_env
    mov dx, offset overlay_not_enough_memory
    call WRITEWRD
    jmp load_module_end

load_wrong_env:
    cmp ax, 10
    jne load_module_end
    mov dx, offset overlay_wrong_env
    call WRITEWRD
    jmp load_module_end

loaded_successfully:
    mov dx, offset overlay_load_success
    call WRITEWRD

    mov bx, offset overlay_addr

    mov ax, [bx]
    mov cx, [bx + 2]
    mov [bx], cx
    mov [bx + 2], ax

    call overlay_addr

```

```

        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_buffer
    mov ah, 1ah
    int 21h

    mov dx, offset overlay_path

```

```

        mov cx, 0
        mov ah, 4eh
        int 21h

        jnc got_size_succesfully

;file not found error
        cmp ax, 12h
        jne route_error
        mov dx, offset file_not_found_error_string
        call WRITEWRD
        jmp allocate_for_overlay_end

route_error:
        cmp ax, 3
        jne allocate_for_overlay_end
        mov dx, offset route_not_found_error_string
        call WRITEWRD
        jmp allocate_for_overlay_end

got_size_succesfully:
        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 overlay_addr
        mov cx, 0000h
        mov [bx], ax
        mov [bx + 2], cx

        mov dx, offset allocated_mem_for_overlay_string
        call WRITEWRD

```



```

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 ENDLINE

    mov si, offset overlay_name1
    call GET_PATH
    call ALLOCATE_FOR_OVERLAY

    call LOAD_OVERLAY

    call ENDLINE

    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

```

```
labin:
CODE ENDS

END MAIN
```

## Название файла: 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
    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

```

## Название файла: 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 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

```