

**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

Морева Е.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.**

1. Для выполнения лабораторной работы был написан и отлажен 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;

Процедуры основного модуля:

FREE\_MEM – освобождение памяти для корректной работы программы.

DTA\_ — устанавливает адреса блока-DTA;

GET\_P— определение имени и пути до оверлей модуля.

ADD\_MEM\_OVL — выделение памяти для оверлейного сегмента;

LOAD\_OVL — загрузка вызываемого оверлей модуля.

CHECK — функция для проверки корректности работы программы

PRINT — вывод

2. Также были написаны и отлажены сами оверлейные сегменты, каждый из которых выводит адрес сегмента, в который он загружен.

Процедуры оверлейного сегмента:

PRINT — вывод

BYTE\_TO\_HEX, TETR\_TO\_HEX, WRD\_TO\_HEX — вспомогательные функции для перевода в 16-тиричную систему счисления.

3. Программа lab7.exe была запущена из корневого каталога, оба оверлейных сегмента находятся в нём:

```
C:\>LAB7.EXE  
  
Path: C:\ovl1.ovl Segment address: 1192h  
Path: C:\ovl2.ovl Segment address: 1192h
```

Рисунок 1 — Результат запуска lab7.exe из того же каталога, где находятся оверлеи.

4. Переместим приложение в другой каталог.

```
C:\>LAB7.EXE  
  
Path: C:\ovl1.ovl Segment address: 1192h  
Path: C:\ovl2.ovl Segment address: 1192h
```

Рисунок 2 — Результат запуска приложения из каталога, отличного от того, где само приложение

Рисунок 3 — Результат запуска приложения, когда одного оверлейного сегмента нет

Аналогично п.3 - приложения успешно загружены и выполнены.

5. Запустим программу, но одного из оверлеев не будет в папке.

```
C:\>LAB7.EXE  
  
Path: C:\ovl1.ovl Segment address: 1192h  
Path: C:\ovl2.ovl File is not found!
```

Рисунок 3 — Результат загрузки lab7.exe в папке с лишь одним из оверлеев

Видно, что второе приложение, в данном случае, завершается аварийно.

Исходный код программ см. в приложении А

### **Контрольные вопросы.**

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

Чтобы использовать в качестве оверлейного сегмента .COM модуль, нужно в начале выделенной памяти сформировать блок PSP размером 100h и выделить память под стек. При вызове оверлея необходимо сместить точку входа на 100h.

### **Выводы.**

В ходе работы были исследованы возможности построения загрузочного модуля оверлейной структуры и структура оверлейного сегмента, а также способ их загрузки и выполнения.

## ПРИЛОЖЕНИЕ А

### ИСХОДНЫЙ КОД ПРОГРАММ

Название файла: lb7.asm

```
Stack SEGMENT STACK
```

```
DW 100 DUP(?)
```

```
Stack ENDS
```

```
DATA SEGMENT
```

```
ERR_PATH db 'Path is not found!', 0DH,0AH,'$'
```

```
ERR_NUM db 'Wrong number!', 0DH,0AH,'$'
```

```
ERR_FILE db 'File is not found!', 0DH,0AH,'$'
```

```
ERR_DISK db 'Disk error!', 0DH,0AH,'$'
```

```
NO_MEM db 'Deficiency memory!', 0DH,0AH,'$'
```

```
ERR_ENV db 'Wrong environment!', 0DH,0AH,'$'
```

```
ERR_MCB db 'MCB is destroyed!', 0DH,0AH,'$'
```

```
ERR_ADR db 'Invalid MCB adress!', 0DH,0AH,'$'
```

```
ERR_ADD_MEM db 'Error by adding memory!', 0DH,0AH,'$'
```

```
END_S db 0DH,0AH,'$'
```

```
NAME_ db 64 DUP(0)
```

```
DTA_BLOCK db 43 DUP(0)
```

```
SEG_OVL dw 0
```

```
ADDRESS_OVL dd 0
```

```
KEEP_PSP dw 0
```

```
PATH db 'Path: $'
```

```
    OVL1 db 'ovl1.ovl', 0
```

```
    OVL2 db 'ovl2.ovl', 0
```

```
DATA ENDS
```

```
CODE SEGMENT
```

```
    ASSUME CS:CODE, DS:DATA, ES:DATA, SS:Stack
```

```
PRINT PROC NEAR
```

```
    push AX
```

```
    mov AH, 09h
```

```
    int 21h
```

```
    pop AX
```

```
    ret
```

```
PRINT ENDP
```

```
DTA_ PROC
```

```
    push DX
```

```

        mov DX, offset DTA_BLOCK
        mov AH, 1Ah
        int 21h
        pop DX
DTA_ ENDP

FREE_MEMORY PROC
        mov BX, offset LAST_BYTE
        mov AX, ES
        sub BX, AX
        mov CL, 4
        shr BX, CL
        mov AH, 4Ah
        int 21h
        jnc good
        cmp AX, 7
        mov DX, offset ERR_MCB
        je PRINT_MEM_ERR
        cmp AX, 8
        mov DX, offset ERR_ADR
        je PRINT_MEM_ERR
        cmp AX, 9
        mov DX, offset ERR_ADR
PRINT_MEM_ERR:
        call PRINT
        xor AL, AL
        mov AH, 4Ch
        int 21h
good:
        ret
FREE_MEMORY ENDP

GET_P PROC
        push ES
        mov ES, ES:[2Ch]
        xor SI, SI
        mov DI, offset NAME_
step_1:
        add SI, 1
        cmp word ptr ES:[SI], 0000h
        jne step_1
        add SI, 4
step_2:

```



```

        cmp byte ptr ES:[SI],00h
        je step_3
        mov DL, ES:[SI]
        mov [DI], DL
        add SI, 1
        add DI, 1
        jmp step_2
step_3:
        sub SI, 1
        sub DI, 1
        cmp byte ptr ES:[SI],'\ '
        jne step_3
        add DI, 1
        mov SI, BX
        push DS
        pop ES
step_4:
        lodsb
        stosb
        cmp AL, 0
        jne step_4
        mov byte ptr [DI], '$'
        mov DX, offset PATH
        call PRINT
        mov DX, offset NAME_
        call PRINT
        pop ES
        ret
GET_P ENDP

```

```

ADD_MEM_OVL PROC
        push DS
        push DX
        push CX
        xor CX, CX
        mov DX, offset NAME_
        mov AH, 4Eh
        int 21h
        jnc V2
        cmp AX, 3
        mov DX, offset ERR_PATH
        je V1
        mov DX, offset ERR_FILE

```

```

V1:
    call PRINT
    pop CX
    pop DX
    pop DS
    xor AL, AL
    mov AH, 4Ch
    int 21h

V2:
    push ES
    push BX
    mov BX, offset DTA_BLOCK
    mov DX, [BX+1Ch]
    mov AX, [BX+1Ah]
    mov CL, 4h
    shr AX, CL
    mov CL, 12
    sal DX, CL
    add AX, DX
    add AX, 1
    mov BX, AX
    mov AH, 48h
    int 21h
    jc V3
    mov SEG_OVL, AX
    pop BX
    pop ES
    pop CX
    pop DX
    pop DS
    ret

V3:
    mov DX, offset ERR_ADD_MEM
    call PRINT
    mov AH, 4Ch
    int 21h
ADD_MEM_OVL ENDP

CHECK PROC
    cmp AX, 1
    mov DX, offset ERR_NUM
    je PRINT_ERR
    cmp AX, 2

```

```

    mov DX, offset ERR_FILE
    je PRINT_ERR
    cmp AX, 5
    mov DX, offset ERR_DISK
    je PRINT_ERR
    cmp AX, 8
    mov DX, offset NO_MEM
    je PRINT_ERR
    cmp AX, 10
    mov DX, offset ERR_ENV
PRINT_ERR:
    call PRINT
    ret
CHECK ENDP

LOAD_OVL PROC
    push DX
    push BX
    push AX
    mov BX, SEG SEG_OVL
    mov ES, BX
    mov BX, offset SEG_OVL
    mov DX, offset NAME_
    mov AX, 4B03h
    int 21h
    jnc LOAD
    call CHECK
    jmp OFF_OVL
LOAD:
    mov AX, DATA
    mov DS, AX
    mov AX, SEG_OVL
    mov word ptr ADDRESS_OVL+2, AX
    call ADDRESS_OVL
    mov AX, SEG_OVL
    mov ES, AX
    mov AX, 4900h
    int 21h
    mov AX, DATA
    mov DS, AX
OFF_OVL:
    mov ES, KEEP_PSP
    pop AX

```

```

        pop BX
        pop DX
        ret
LOAD_OVL ENDP

Main PROC FAR
    mov AX, DATA
    mov DS, AX
    mov KEEP_PSP, ES
    mov DX, offset END_S
    call PRINT
    call FREE_MEMORY
    call DTA_
    mov BX, offset OVL1
    call GET_P
    call ADD_MEM_OVL
    call LOAD_OVL
    mov BX, offset OVL2
    call GET_P
    call ADD_MEM_OVL
    call LOAD_OVL
    mov AH, 4Ch
    int 21h
Main ENDP

        LAST_BYTE:
CODE ENDS
        END Main

```

### **OVL1.asm :**

```

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

Main PROC FAR
    push DS
    push AX
    push DI
    push DX
    push BX
    mov DS, AX

```

```

    mov BX, offset SEG_ADDR
    add BX, 21
    mov DI, BX
    mov AX, CS
    call WRD_TO_HEX
    mov DX, offset SEG_ADDR
    call PRINT
    pop BX
    pop DX
    pop DI
    pop AX
    pop DS
    retf
Main ENDP

```

```

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

```
SEG_ADDR db ' Segment address:      h',0DH,0AH,'$'
```

```
OVL ENDS
```

```
    END Main
```

## **OVL2.asm:**

```
OVL SEGMENT
```

```
    ASSUME CS:OVL, DS:NOTHING, SS:NOTHING, ES:NOTHING
```

```
Main PROC FAR
```

```
    push DS
```

```
    push AX
```

```
    push DI
```

```
    push DX
```

```
    push BX
```

```
    mov DS, AX
```

```
    mov BX, offset SEG_ADDR
```

```
    add BX, 21
```

```
    mov DI, BX
```

```
    mov AX, CS
```

```
    call WRD_TO_HEX
```

```
    mov DX, offset SEG_ADDR
```

```
    call PRINT
```

```
    pop BX
```

```
    pop DX
```

```
    pop DI
```

```
    pop AX
```

```
    pop DS
```

```
    retf
```

```
Main ENDP
```

```
PRINT PROC NEAR
```

```
    push AX
```

```

        mov AH, 09h
        int 21h
        pop AX
        ret
PRINT 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

SEG_ADDR db ' Segment address:      h', 0DH, 0AH, '$'

OVL ENDS

        END Main

```