

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

Студент гр. 9382

Кузьмин Д. И.

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.

Выполнение работы.

1) Сначала было реализовано освобождение памяти, занимаемой основной программой.

2) Затем была создана функция, вычисляющая размер оверлейного сегмента и устанавливала его при помощи функции 4b03h прерывания int 21h

3) Была реализована обработка случаев некорректного выделения памяти или определения размера оверлейного сегмента.

Результаты работы программы см. в приложении А

Исходный код см. в приложении Б

Контрольные вопросы.

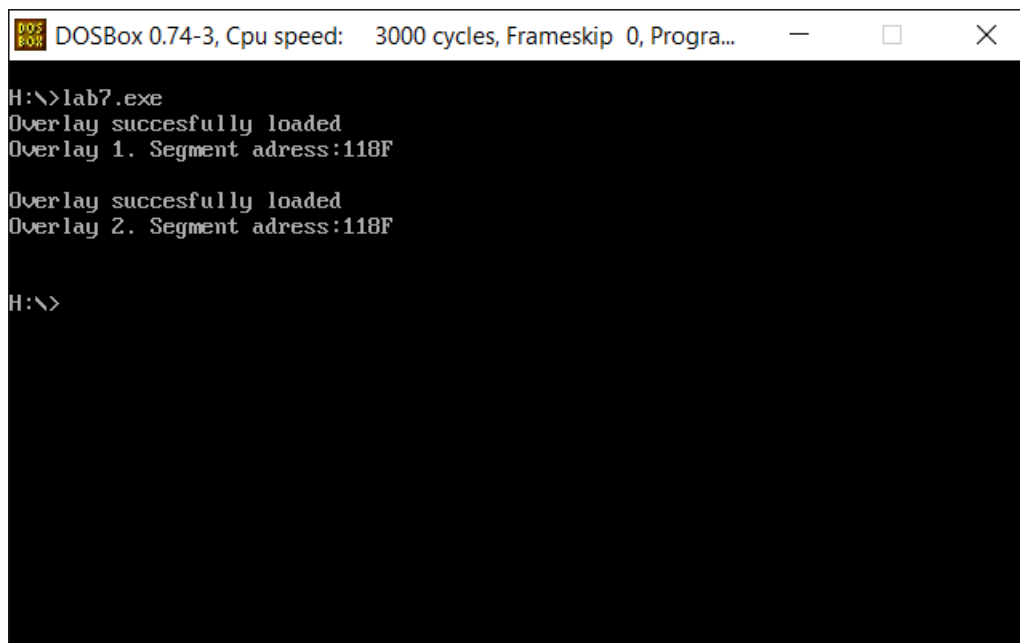
1) Как должна быть устроена программа, если в качестве оверлейного сегмента использовать .COM модули?

Нужно добавить смещение в 100h при обращении к оверлейному сегменту.

Выводы.

Были изучены возможности построения загрузочного модуля оверлейной структуры. Исследованы структуры оверлейного сегмента и способы загрузки и выполнения оверлейных сегментов.

ПРИЛОЖЕНИЕ А. ДЕМОНСТРАЦИЯ РАБОТЫ ПРОГРАММЫ

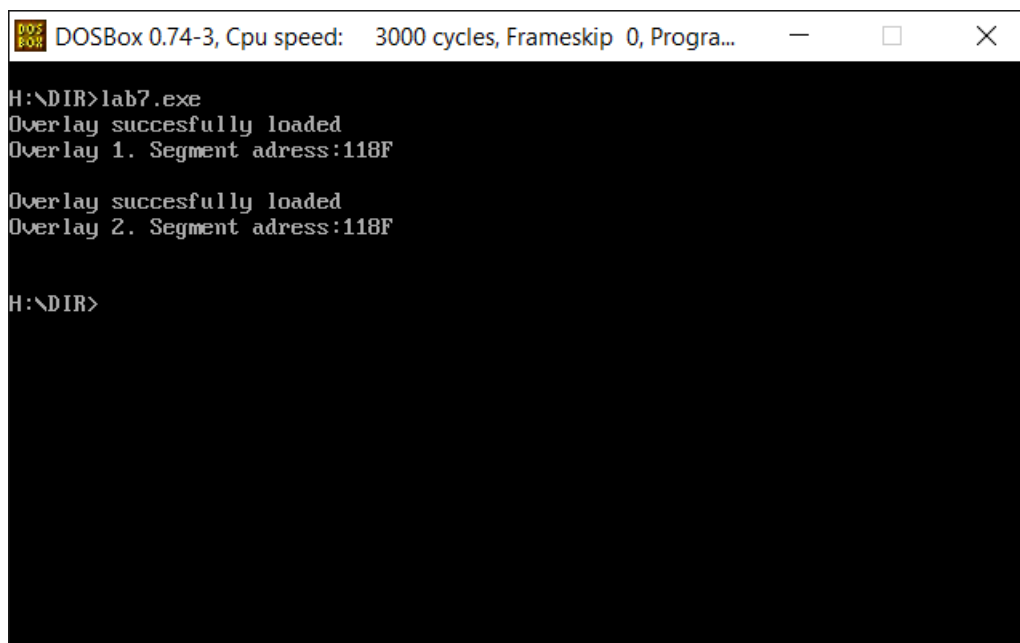


```
DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip 0, Progra...
H:\>lab7.exe
Overlay succesfully loaded
Overlay 1. Segment adress:118F

Overlay succesfully loaded
Overlay 2. Segment adress:118F

H:\>
```

Рис. 1- запуск программы, когда программа и оверлейные модули находятся в одном каталоге



```
DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip 0, Progra...
H:\DIR>lab7.exe
Overlay succesfully loaded
Overlay 1. Segment adress:118F

Overlay succesfully loaded
Overlay 2. Segment adress:118F

H:\DIR>
```

Рис. 2 - запуск программы, когда программа и оверлейные модули находятся в одном каталоге, отличном от первого

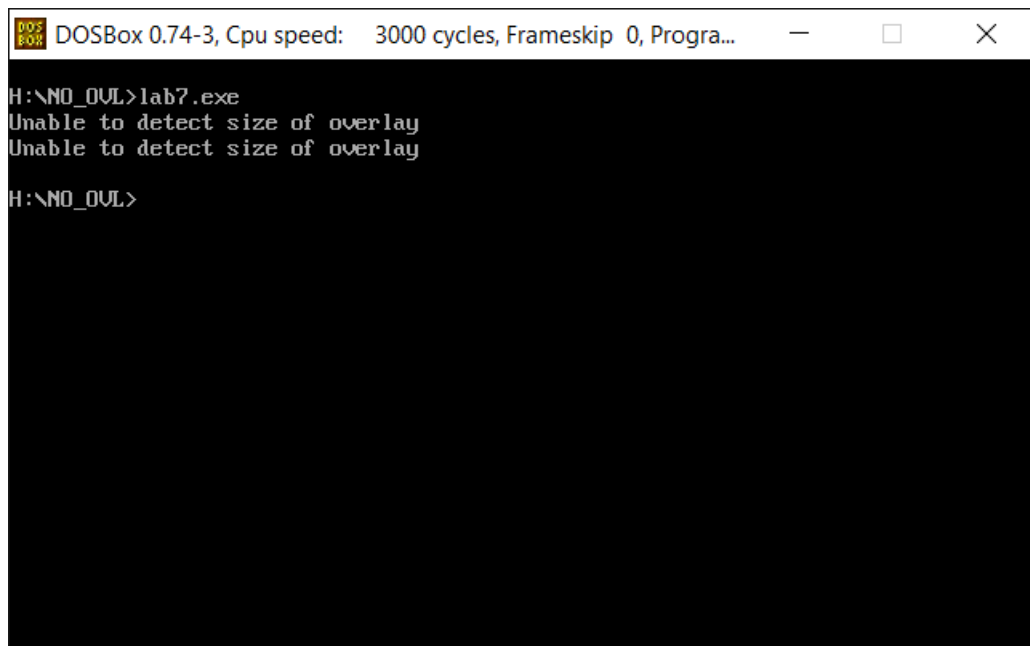


Рис. 3 - запуск программы, когда программа и оверлейные модули находятся в разных каталогах.

ПРИЛОЖЕНИЕ Б. ИСХОДНЫЙ КОД

Файл lab7.asm

```
ASTACK      SEGMENT      STACK
            DW 200 DUP (?)
ASTACK      ENDS
DATA SEGMENT
overlay_loaded db 'Overlay succesfully loaded' ,13, 10, '$'
overlay_name1 db 'overlay1.ovl', 0
overlay_name2 db 'overlay2.ovl', 0
overlay_name_addr dw 0
setflag db 1
path db 50 dup(0)
mem_error db 'Memory error. Code      ', 13, 10, '$'
load_error db 'Overlay loading error. Code  ', 13, 10, '$'
size_detect_error db 'Unable to detect size of overlay', 13, 10, '$'
epb dw 0
      dw 0
dta db 43 dup(0)
overlay_entry_point  dd  0
keep_sp dw 0
keep_ss dw 0
psp dw 0
DATA ENDS
CODE SEGMENT
      .386
      ASSUME CS:CODE, DS:DATA, SS:ASTACK
; Процедуры
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
;-----
WRITE_MSG PROC near
      mov AH, 09h
      int 21h
      ret
WRITE_MSG ENDP
;-----
BYTE_TO_HEX PROC near
; Байт в AL переводится в два символа шестн. числа в AX
      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, в AH - младшая
      pop CX
      ret
BYTE_TO_HEX ENDP
```

```

;-----
WRD_TO_HEX PROC near
;перевод в 16 с/с 16-ти разрядного числа
;в AX - число, DI - адрес последнего символа
    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
;-----
BYTE_TO_DEC PROC near
;Перевод в 10чную с/с, SI - адрес младшей цифры
    push CX
    push DX
    xor AH,AH
    xor DX,DX
    mov CX,10
loop_bd:    div CX
            or DL,30h
            mov [SI],DL
            dec SI
            xor DX,DX
            cmp AX,10
            jae loop_bd
            cmp AL,00h
            je end_1
            or AL,30h
            mov [SI],AL
end_1:      pop DX
            pop CX
            ret

BYTE_TO_DEC ENDP
;-----
FREE_MEMORY PROC NEAR
    push ax
    push bx
    push si
    push dx
    mov bx, offset end_point
    mov ax, cs
    add bx, ax
    shr bx,4h
    add bx, 1Fh

    mov ah,4ah
    int 21h

```

```

        jc error_message2
        jmp exit_free
error_message2:
        mov si, offset mem_error
        add si, 19
        call BYTE_TO_DEC
        mov dx, offset mem_error
        call WRITE_MSG
exit_free:
        pop dx
        pop si
        pop bx
        pop ax
        ret
FREE_MEMORY ENDP
;-----
SET_OVERLAY PROC NEAR
        mov ah,1Ah
        mov dx, offset dta
        int 21h

        mov ah,4Eh
        mov dx, offset path
        mov cx,0
        int 21h
        jc error_message1

        mov si, offset dta
        mov bx, word ptr[si + 1ah]
        mov ax, word ptr[si + 1ch]
        shr bx, 4
        shl ax, 12
        add bx, ax
        add bx, 1
        mov ah,48h
        int 21h

        mov epb,ax
        mov epb + 2,ax
        mov word ptr overlay_entry_point + 2,ax
        jmp exit_set

error_message1:
        mov dx, offset size_detect_error
        call WRITE_MSG
        mov setflag, 0
exit_set:
        ret
SET_OVERLAY ENDP
RUN_OVERLAY PROC NEAR

        mov     ax,ds
        mov     es,ax

        mov     dx, offset path
        mov     bx,offset epb
        mov     ax,4b03h

```



```

        int      21h
        jc load_error_msg

        mov dx, offset overlay_loaded
        call WRITE_MSG

        push     ds
        call     overlay_entry_point
        pop      ds
        mov es, ebp
        mov ah, 49h
        int 21h
        jmp exit_run
load_error_msg:
        push si
        push dx
        mov si, offset load_error
        add si, 30
        call BYTE_TO_DEC
        mov dx, offset load_error
        call WRITE_MSG
        pop dx
        pop si
exit_run:
        ret
RUN_OVERLAY ENDP
LOAD_OVERLAY PROC NEAR

```

```

        push ax
        push bx
        push dx
        push es

        mov     keep_ss, ss
        mov     keep_sp, sp

        call SET_OVERLAY
        cmp setflag, 0
        je skiprun
        call RUN_OVERLAY

skiprun:
        mov setflag, 1
        mov     ss, keep_ss
        mov     sp, keep_sp

        pop es
        pop dx
        pop bx
        pop ax
        ret
LOAD_OVERLAY ENDP
GET_PATH_TO_FILE PROC NEAR

```

```

        push di

```

```

        push si
        push ax
        push bx
        push cx
        push dx
        push es

        mov ax, psp
        mov es, ax
        mov es, es:[2ch]
        mov bx, 0

continue:
        inc bx
        cmp byte ptr es:[bx-1], 0
        jne continue

        cmp byte ptr es:[bx+1], 0
        jne continue

        add bx, 2
        mov di, 0

check:
        mov dl, es:[bx]
        mov byte ptr [path+di], dl
        add di, 1
        add bx, 1
        cmp dl, 0
        je end_check
        cmp dl, '\\'
        jne check
        mov cx, di
        jmp check
end_check:
        mov di, cx
        mov si, overlay_name_addr

write_end:
        mov dl, byte ptr [si]
        mov byte ptr [path+di], dl
        inc di
        inc si
        cmp dl, 0
        jne write_end

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

GET_PATH_TO_FILE ENDP
MAIN      PROC FAR

```

```

        mov ax, DATA
        mov ds, ax
        mov esp, es
        xor ax, ax
        call FREE_MEMORY

        mov path, offset overlay_name1
        mov overlay_name_addr, offset overlay_name1
        call GET_PATH_TO_FILE
        call LOAD_OVERLAY

        mov path, offset overlay_name2
        mov overlay_name_addr, offset overlay_name2
        call GET_PATH_TO_FILE
        call LOAD_OVERLAY

        mov ah, 4ch
        int 21h
        end_point:

MAIN ENDP
CODE ENDS
        END MAIN

```