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. 0382 _____ Самулевич В.А.

Преподаватель _____ Ефремов М.А.

Санкт-Петербург
2022

Цель работы.

Исследование возможности построения загрузочного модуля оверлейной структуры, а также изучение структуры оверлейного сегмента и способа загрузки и выполнения оверлейных сегментов.

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 размер памяти, достаточный для его загрузки.
3. Файл оверлейного сегмента загружается и выполняется.
4. Освобождается память, отведённая 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 аварийное.

Шаг 6:

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

Выполнение работы.

Шаг 1:

В процессе написания lab7.asm были реализованы следующие функции:

- free_memory – очищает неиспользуемую память из-под lab7.exe.Освобожденная таким образом память в дальнейшем используется для хранения оверлейных сегментов.
- Request_memory – С помощью прерывания 4eh int 21h получает размер оверлея, после чего запрашивает у операционной системы требуемое количество параграфов (48h int 21h).
- Init_file_path – инициализирует строку, содержащую путь до оверлея (путь до директории включительно берется из PSP, а имя – из строки, смещение которой хранится в BX).
- Load_overlay – с помощью прерывания 4B03 int 21h выполняет загрузку оверлейного сегмента в память, ранее выделенную процедурой Request_memory.
- Execute_overlay – Используя Load_overlay, загружает оверлейный сегмент в память, после чего передает ему управление.

Шаг 2:

Было создано для оверлея– first.asm и second.asm.Оба они печатают свой сегментный адрес среды, с уведомлением о том, кто именно из них был запущен.lab7 сначала загружает и выполняет first.ovl, а затем - second.ovl.

Шаг 3:

Результат работы отлаженного приложения представлен на рисунке 1.

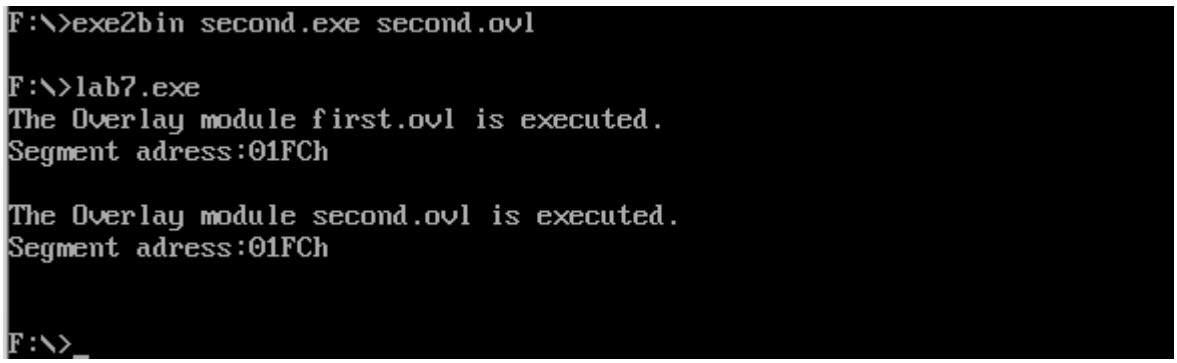

Рисунок 1- результат запуска lab7.exe.

Шаг 4:

lab7.exe была запущена из каталога, отличного от того, где располагаются все созданные модули. Результат представлен на рисунке 2.

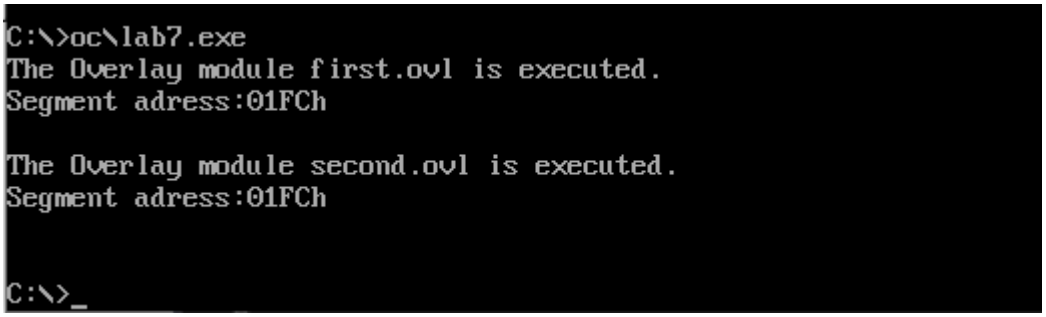

Рисунок 2 – результат запуска lab7.exe из другого каталога.

Как можно заметить, вывод программы не изменился.

Шаг 5:

Программа была запущена в случае, когда одного оверлея не было в каталоге. Результат запуска представлен на рисунке 3.

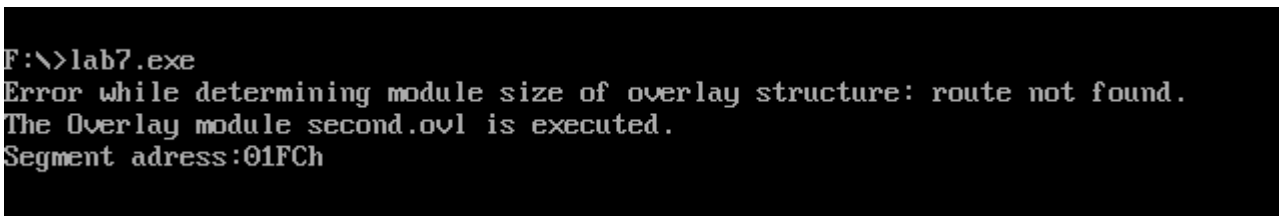

Рисунок 3 – результат запуска lab7.exe без first.ovl.

Ответы на контрольные вопросы.

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

   В .COM файле код начинается со смещения 100h(т.к. первые 256 байт занимает PSP), следовательно при передаче управления .COM модулю нужно указать соответствующее смещение.

Выводы.

Была исследована возможность построения загрузочного модуля оверлейной структуры.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: lab7.asm

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

DATA SEGMENT
    DTA_buffer DB 45 DUP (?)
    first_clear_error DB "Error clearing memory: control block
destroyed.",0Dh,0Ah,'$'
    second_clear_error DB "Error while clearing memory: Not enough
memory to execute function.",0Dh,0Ah,'$'
    fird_clear_error DB "Error clearing memory: invalid memory
block address.",0Dh,0Ah,'$'


    size_calculation_error_header DB "Error while determining
module size of overlay structure: ",'$'
    size_calculation_error_1 DB "file not found.",0Dh,0Ah,'$'
    size_calculation_error_2 DB "route not found.",0Dh,0Ah,'$'
    allocation_error DB "Error while allocating memory for overlay
structure module.",0Dh,0Ah,'$'

    load_error_header DB "Error loading overlay into memory:",'$'
    load_error_1 DB "non-existent function.",0Dh,0Ah,'$'
    load_error_4 DB "too many open files.",0Dh,0Ah,'$'
    load_error_5 DB "no access",0Dh,0Ah,'$'
    load_error_6 DB "not enough memory",0Dh,0Ah,'$'
    load_error_7 DB "wrong environment",0Dh,0Ah,'$'



    overlay_name_1 DB "first.ovl",'$'
    overlay_name_2 DB "second.ovl",'$'
    overlay_path DB 128 DUP (0)

    overlay_segment DW 0
    overlay_offset DW 0

end_data DB 0

DATA ENDS


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

Main PROC FAR
    push DS
    sub AX,AX
    push AX
    mov AX,DATA
```

```
    mov DS,AX

    mov AH,1ah
    mov DX,offset DTA_buffer
    int 21h

    call free_memory
    jb Main_end
    mov BX,offset overlay_name_1
    call execute_overlay
    mov BX,offset overlay_name_2
    call execute_overlay


Main_end:
    mov AX,0
    mov AH,4ch
    int 21h

Main ENDP

execute_overlay PROC NEAR; BX- offset имени модуля
    push AX
    push ES
    call init_file_path
    call Request_memory
    jb execute_end
    call Load_overlay
    jb execute_end

    mov AX,overlay_segment
    mov ES,AX
    xchg AX,overlay_offset
    xchg AX,overlay_segment

    call DWORD PTR overlay_segment
    mov ES,overlay_offset
    mov AH,49h
    int 21h
    mov overlay_segment,0
    mov overlay_offset,0

execute_end:
    pop ES
    pop AX
    ret
execute_overlay ENDP



Load_overlay PROC NEAR
    push AX
    push BX
    push DX
    push ES

    mov DX,offset overlay_path
    push DS
```

```
    pop ES
    mov BX,offset overlay_segment
    mov AX,4B03h
    int 21h
    jae successful_upload

    cmp AX,1
    jne load_error_check2
    mov DX,offset load_error_1
    jmp print_load_error

load_error_check2:
    cmp AX,2
    jne load_error_check3
    mov DX,offset size_calculation_error_1
    jmp print_load_error

load_error_check3:
    cmp AX,3
    jne load_error_check4
    mov DX,offset size_calculation_error_2
    jmp print_load_error

load_error_check4:
    cmp AX,4
    jne load_error_check5
    mov DX,offset load_error_4
    jmp print_load_error


load_error_check5:
    cmp AX,5
    jne load_error_check6
    mov DX,offset load_error_5
    jmp print_load_error


load_error_check6:
    cmp AX,8
    jne load_error_check7
    mov DX,offset load_error_6
    jmp print_load_error


load_error_check7:
    mov DX,offset load_error_7

print_load_error:
    call print
    STC
    jmp load_end
successful_upload:
    CLC

Load_end:
    pop ES
    pop DX
    pop BX
```

```
    pop AX
    ret
Load_overlay ENDP

Request_memory PROC NEAR
    push CX
    push DX
    push AX
    push SI

    mov CX,0
    mov DX,offset overlay_path
    mov AX,0
    mov AH,4eh
    int 21h
    jae start_allocating_memory

    mov DX,offset size_calculation_error_header
    call print

    cmp AX,2
    jne second_size_calculation_error
    mov DX,offset overlay_name_1
    jmp print_size_calculation_error

second_size_calculation_error:
    mov DX,offset size_calculation_error_2

print_size_calculation_error:
    call print
    STC
    jmp Request_end

start_allocating_memory:
    mov SI,offset DTA_buffer
    mov DX,[SI+1Ch]
    mov AX, [SI+1Ah]
    mov CX,16
    div CX
    inc AX
    mov BX,AX
    mov AH,48h
    int 21h
    jae successful_memory_allocation

    mov DX,offset allocation_error
    call print
    STC
    jmp Request_end

successful_memory_allocation:
    mov overlay_segment,AX
    CLC

Request_end:
    pop SI
    pop AX
    pop DX
```

```
    pop CX
    ret
Request_memory ENDP



Free_memory PROC NEAR
    push AX
    push BX
    push DX

    mov DX,0
    mov AX,offset end_code
    add AX,offset end_data
    inc AX
    add AX,228h
    mov BX,16
    div BX
    inc AX
    mov BX,AX
    mov AX,0
    mov AH,4ah
    int 21h
    jae free_without_errors
    cmp AX,7
    jne second_free_check
    mov DX,offset first_clear_error
    jmp handle_free_error

second_free_check:
    cmp AX,8
    jne fird_free_check
    mov DX,offset second_clear_error
    jmp handle_free_error

fird_free_check:
     mov DX,offset fird_clear_error

handle_free_error:
    call print
    STC ; флаг того,что очистка памяти не удалась
    jmp free_end

free_without_errors:
   CLC
free_end:
    pop DX
    pop BX
    pop AX
    ret
Free_memory ENDP

init_file_path PROC NEAR ;BX - смещение имени файла
    push AX
    push ES
    push CX
    push SI
    push DI
```

```
    mov AX, ES:[2ch]
    mov ES,AX
    mov CX,0
    mov SI,0
find_file_path_cycle:
    mov AL,ES:[SI]
    cmp AL,0
    je two_zeros_checking
    mov CX,0
    inc SI
    jmp find_file_path_cycle
two_zeros_checking:
    inc CX
    cmp CX,2
    je stop_find_file_loop
    inc SI
    jmp find_file_path_cycle

stop_find_file_loop:
    add SI,3
    mov DI,offset overlay_path
file_path_copy_loop:
    cmp  BYTE PTR ES:[SI],0
    je stop_file_path_copy_loop
    mov AL,ES:[SI]
    mov DS:[DI],AL
    inc SI
    inc DI
    cmp AL,'\'
    je update_last_dir
    jmp file_path_copy_loop

update_last_dir:
    mov CX,DI
    jmp file_path_copy_loop


stop_file_path_copy_loop:
    mov DI,CX
    mov SI,BX
copy_file_name_loop:
    mov AL,[SI]
    cmp AL,'$'
    je stop_copy_file_name_loop
    mov [DI],AL
    inc SI
    inc DI
    jmp copy_file_name_loop

stop_copy_file_name_loop:
    mov BYTE PTR [DI],0

    pop DI
    pop SI
    pop CX
    pop ES
    pop AX
```

```
    ret
init_file_path ENDP


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

end_code:

CODE ENDS
END Main
```

Название файла: first.asm

```
CODE SEGMENT
ASSUME CS:CODE

Main PROC FAR
    push DX
    push AX
    push DI
    push DS

    mov AX,CS
    mov DS,AX

    mov DX,offset message
    call Print_msg
    mov DI,offset segment_adress
    add DI,18
    call WRD_TO_HEX
    mov DX,offset segment_adress
    call Print_msg
    pop DS
    pop DI
    pop AX
    pop DX
    retf
Main ENDP

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


   message DB "The Overlay module first.ovl is executed.",0DH,0Ah,'$'
   segment_adress DB "Segment adress:    h",0DH,0Ah,0DH,0Ah,'$'

CODE ENDS
END Main
```

Название файла: second.asm

```
CODE SEGMENT
ASSUME CS:CODE

Main PROC FAR
    push DX
    push AX
    push DI
    push DS

    mov AX,CS
    mov DS,AX

    mov DX,offset message
```

```
    call Print_msg
    mov DI,offset segment_adress
    add DI,18
    call WRD_TO_HEX
    mov DX,offset segment_adress
    call Print_msg
    pop DS
    pop DI
    pop AX
    pop DX
    retf
Main ENDP

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

         message     DB     "The     Overlay     module     second.ovl     is
executed.",0DH,0Ah,'$'
         segment_adress DB "Segment adress:    h",0DH,0Ah,0Dh,0Ah,'$'

      CODE ENDS
      END Main
```