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

Студентка гр. 9383 _____ Карпекина А.А.

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.

Постановка задачи.

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.

Шаг 1. Был написан и отлажен 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. Были написаны и отлажены оверлейные сегменты. Оверлейный сегмент выводит адрес сегмента, в который он загружен.

Шаг 3. Было запущено отлаженное приложение. Оверлейные сегменты загружаются с одного и того же адреса, перекрывая друг друга.

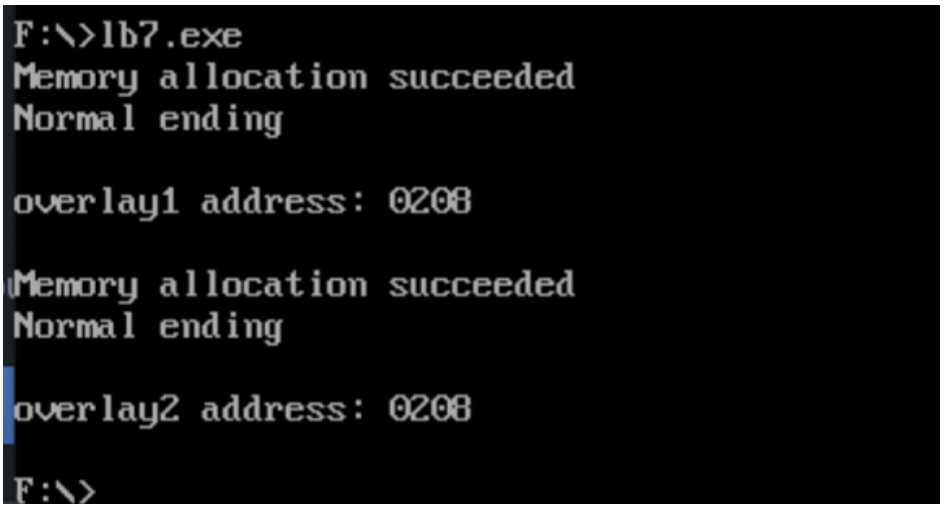

Рисунок 1 - Вывод программы lb7.asm

Шаг 4. Приложение было успешно выполнено из другого каталога.

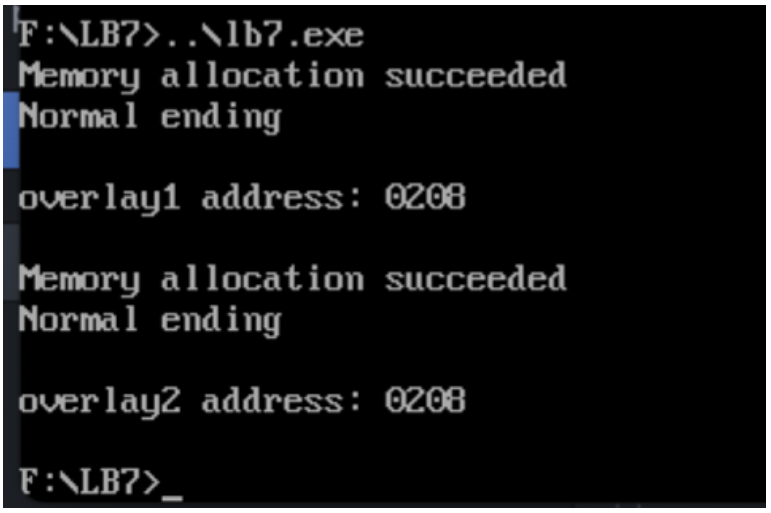

Рисунок 2 - Запуск программы из другого каталога

Шаг 5. Было запущено приложение, когда оверлея №2 нет в каталоге. Приложение закончилось аварийно.

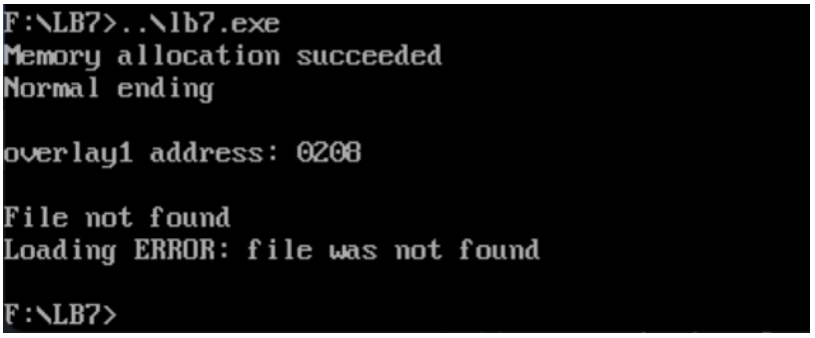

Рисунок 3 - Результат запуска программы на 5 шаге

Ответы на вопросы.

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

Необходимо учитывать смещение 100h, так как в .COM модуле присутствует PSP. После записи значений регистров в стек, нужно положить значение регистра CS в DS, так как адрес сегмента данных совпадает с сегментом кода.

Вывод.

В результате выполнения работы были исследованы возможности построения загрузочного модуля оверлейной структуры.

ПРИЛОЖЕНИЕ А

КОД ПРОГРАММЫ

lb7.asm

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


DATA SEGMENT
	FILE1 DB 'OVERLAY1.OVL', 0
	FILE2 DB 'OVERLAY2.OVL', 0
	PROG DW 0
	DATA_MEM DB 43 dup(0)
	POS_CL DB 128 dup(0)
	address DD 0
	KEEP_PSP DW 0
	NEW_STR DB 13, 10, '$'
	MEM_7 DB 'Memory ERROR: destroyed memory block', 13, 10, '$'
	MEM_8 DB 'Memory ERROR: not enough memory for running function', 13,
10, '$'
	MEM_9 DB 'Memory ERROR: incorrect memory address', 13, 10, '$'
	ERR_1_STR DB 'Loading ERROR: wrong function number', 13, 10, '$'
	ERR_2_STR DB 'Loading ERROR: file was not found', 13, 10, '$'
	ERR_5_STR DB 'Loading ERROR: disk ERROR', 13, 10, '$'
	ERR_8_STR DB 'Loading ERROR: disk has not enough free memory space',
13, 10, '$'
	ERR_10_STR DB 'Loading ERR: wrong string environment', 13, 10, '$'
	ERR_11_STR DB 'Loading ERROR: incorrect format', 13, 10, '$'
	END_0 DB 'Normal ending', 13, 10, '$'
	END_1 DB 'Ending by ctrl-break', 13, 10, '$'
	END_2 DB 'Ending by device ERROR', 13, 10, '$'
	END_3 DB 'Ending by 31h function', 13, 10, '$'
	ALLOCATE_SUCCESS_STR DB 'Memory allocation succeeded', 13, 10, '$'
	FILE_ERR_STR DB 'File not found', 13, 10, '$'
	ROUTE_ERR_STR DB 'Route not found', 13, 10, '$'
	END_DATA DB 0
DATA ENDS

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

PRINT_STR PROC
	push ax
```

```
        mov ah, 09h
        int 21h
        pop ax
        ret
PRINT_STR ENDP


FREE_MEM PROC
        push ax
        push bx
        push cx
        push dx
        mov ax, offset END_DATA
        mov bx, offset END_PROG
        add bx, ax
        shr bx, 1
        shr bx, 1
        shr bx, 1
        shr bx, 1
        add bx, 2bh
        mov ah, 4ah
        int 21h

        jnc end_mem

        lea dx, MEM_7
        cmp ax, 7
        je print
        lea dx, MEM_8
        cmp ax, 8
        je print
        lea dx, MEM_9
        cmp ax, 9
        je print
        jmp end_mem

print:
        mov ah, 09h
        int 21h

end_mem:
        pop dx
        pop cx
        pop bx
```

```
        pop ax
        ret

FREE_MEM ENDP


SET_NAME PROC NEAR
        push ax
        push bx
        push cx
        push dx
        push di
        push si
        push es
        mov PROG, dx
        mov ax, KEEP_PSP
        mov es, ax
        mov es, es:[2ch]
        mov bx, 0

find:
        inc bx
        cmp byte ptr es:[bx-1], 0
        jne find
        cmp byte ptr es:[bx+1], 0
        jne find
        add bx, 2
        mov di, 0

find_loop:
        mov dl, es:[bx]
        mov byte ptr [POS_CL + di], dl
        inc di
        inc bx
        cmp dl, 0
        je end_loop
        cmp dl, '\'
        jne find_loop
        mov cx, di
        jmp find_loop

end_loop:
        mov di, cx
        mov si, PROG
```

```
find_loop_2:
        mov dl, byte ptr[si]
        mov byte ptr [POS_CL + di], dl
        inc di
        inc si
        cmp dl, 0
        jne find_loop_2
        pop es
        pop si
        pop di
        pop dx
        pop cx
        pop bx
        pop ax
        ret

SET_NAME ENDP


DIF_PROG PROC NEAR
        push ax
        push bx
        push cx
        push dx
        push ds
        push es
        mov ax, DATA
        mov es, ax
        mov bx, offset address
        mov dx, offset POS_CL
        mov ax, 4b03h
        int 21h
        jnc transition

err_1:
        cmp ax, 1
        jne err_2
        mov dx, offset ERR_1_STR
        call PRINT_STR
        jmp dif_prog_end

err_2:
        cmp ax, 2
```

```
        jne err_5
        mov dx, offset ERR_2_STR
        call PRINT_STR
        jmp dif_prog_end

err_5:
        cmp ax, 5
        jne err_8
        mov dx, offset ERR_5_STR
        call PRINT_STR
        jmp dif_prog_end

err_8:
        cmp ax, 8
        jne err_10
        mov dx, offset ERR_8_STR
        call PRINT_STR
        jmp dif_prog_end

err_10:
        cmp ax, 10
        jne err_11
        mov dx, offset ERR_10_STR
        call PRINT_STR
        jmp dif_prog_end

err_11:
        cmp ax, 11
        mov dx, offset ERR_11_STR
        call PRINT_STR
        jmp dif_prog_end

transition:
        mov dx, offset END_0
        call PRINT_STR
        mov ax, word ptr address
        mov es, ax
        mov word ptr address, 0
        mov word ptr address + 2, ax
        call address
        mov es, ax
        mov ah, 49h
        int 21h
```

```
dif_prog_end:
        pop es
        pop ds
        pop dx
        pop cx
        pop bx
        pop ax
        ret

DIF_PROG ENDP


ALMEM PROC
        push ax
        push bx
        push cx
        push dx
        push dx
        mov dx, offset DATA_MEM
        mov ah, 1ah
        int 21h
        pop dx
        mov cx, 0
        mov ah, 4eh
        int 21h
        jnc allocate_success
        cmp ax, 2
        je route_err
        mov dx, offset FILE_ERR_STR
        call PRINT_STR
        jmp allocate_end


route_err:
        cmp ax, 3
        mov dx, offset ROUTE_ERR_STR
        call PRINT_STR
        jmp allocate_end

allocate_success:
        push di
        mov di, offset DATA_MEM
        mov bx, [di + 1ah]
        mov ax, [di + 1ch]
```

```
        pop di
        push cx
        mov cl, 4
        shr bx, cl
        mov cl, 12
        shl ax, cl
        pop cx
        add bx, ax
        add bx, 1
        mov ah, 48h
        int 21h
        mov word ptr address, ax
        mov dx, offset ALLOCATE_SUCCESS_STR
        call PRINT_STR

allocate_end:
        pop dx
        pop cx
        pop bx
        pop ax
        ret

ALMEM ENDP


START_OVERLAY PROC
        push dx
        call SET_NAME
        mov dx, offset POS_CL
        call ALMEM
        call DIF_PROG
        pop dx
        ret
START_OVERLAY ENDP


MAIN PROC FAR
        push ds
        xor ax, ax
        push ax
        mov ax, DATA
        mov ds, ax
        mov KEEP_PSP, es
        call FREE_MEM
```

```
        mov dx, offset FILE1
        call START_OVERLAY
        mov dx, offset NEW_STR
        call PRINT_STR
        mov dx, offset FILE2
        call START_OVERLAY

end_:
        xor al,al
        mov ah,4ch
        int 21h

Main ENDP

END_PROG:
CODE ENDS
END MAIN
```