**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. 9381 |  | Прибылов Н.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 использованием полного пути.

**Ход работы.**

1) Написан и отлажен программный модуль типа .EXE, который выполняет функции:

• Проверяет, установлено ли пользовательское прерывание с вектором 09h.

• Если прерывание не установлено, то устанавливает резидентную функцию для обработки прерывания и настраивает вектор прерываний. Адрес точки входа в стандартный обработчик прерывания находится в теле пользовательского обработчика. Осуществляется выход по функции 4Ch прерывания int 21h.

• Если прерывание установлено, то выводится соответствующее сообщение и осуществляется выход по функции 4Ch прерывания int 21h.

2) Написаны и отлажены оверлейные сегменты, которые выводят адрес сегмента, в который они загружены.

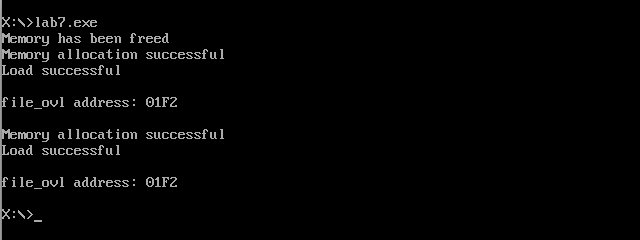
3) Запущено отлаженное приложение. Оверлейные сегменты загружаются с одного и того же адреса, перекрывая друг друга.

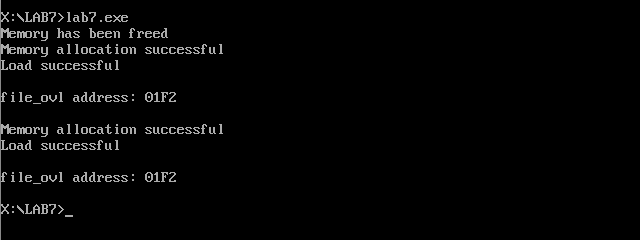
4) Запущено приложение из другого каталога.

5) Запущено приложение в том случае, когда одного оверлея нет в каталоге.

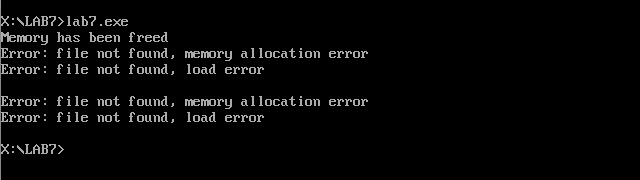
6) Получившиеся результаты показаны в виде скриншотов.

## **Результат работы программы**.

Программа запущена

Программа запущена из другого каталога

Программа запущена, первый оверлей в другой директории

Программа запущена, оба оверлея в другой директории

**Контрольные вопросы.**

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

В COM-модуле необходимо поместить значение регистра CS в регистр DS после записи значений в стек (адрес сегмента данных = адрес сегмента кода). Также при обращении к оверлейному сегменту необходимо обращаться к сегменту, смещённому на 100h.

## Выводы.

Была исследована структура, способ загрузки и выполнение оверлейных сегментов.

# Приложение А Исходный код программы

Название файла: lab7.asm

data segment

file1 db "ovl1.ovl", 0

file2 db "ovl2.ovl", 0

program dw 0

dta\_mem db 43 dup(0)

mem\_flag db 0

cl\_pos db 128 dup(0)

ovls\_addr dd 0

keep\_psp dw 0

eof db 0dh, 0ah, '$'

str\_free\_memory db 'Memory has been freed' , 0dh, 0ah, '$'

str\_file\_error db 'Error: file not found, load error', 0dh, 0ah, '$'

str\_route\_error db 'Error: route not found, load error', 0dh, 0ah, '$'

str\_normal\_end db 'Load successful', 0dh, 0ah, '$'

str\_allocation\_mem\_end db 'Memory allocation successful', 0dh, 0ah, '$'

str\_all\_file\_error db 'Error: file not found, memory allocation error' , 0dh, 0ah, '$'

str\_all\_route\_error db 'Error: route not found, memory allocation error' , 0dh, 0ah, '$'

end\_data db 0

data ends

stacks segment stack

dw 128 dup(?)

stacks ends

code segment

assume cs:code, ds:data, ss:stacks

print\_str proc

push ax

mov ah, 09h

int 21h

pop ax

ret

print\_str endp

free\_memory proc

push ax

push bx

push cx

push dx

mov ax, offset end\_data

mov bx, offset exit

add bx, ax

mov cl, 4

shr bx, cl

add bx, 2bh

mov ah, 4ah

int 21h

free\_mem\_end:

mov mem\_flag, 1

mov dx, offset str\_free\_memory

call print\_str

end\_free:

pop dx

pop cx

pop bx

pop ax

ret

free\_memory endp

load\_proc proc

push ax

push bx

push cx

push dx

push ds

push es

mov ax, data

mov es, ax

mov bx, offset ovls\_addr

mov dx, offset cl\_pos

mov ax, 4b03h

int 21h

jnc \_loads

file\_err:

cmp ax, 2

jne route\_err

mov dx, offset str\_file\_error

call print\_str

jmp \_loade

route\_err:

cmp ax, 3

jne \_loads

mov dx, offset eof

call print\_str

mov dx, offset str\_route\_error

call print\_str

jmp \_loade

\_loads:

mov dx, offset str\_normal\_end

call print\_str

mov ax, word ptr ovls\_addr

mov es, ax

mov word ptr ovls\_addr, 0

mov word ptr ovls\_addr+2, ax

call ovls\_addr

mov es, ax

mov ah, 49h

int 21h

\_loade:

pop es

pop ds

pop dx

pop cx

pop bx

pop ax

ret

load\_proc endp

path proc

push ax

push bx

push cx

push dx

push di

push si

push es

mov program, dx

mov ax, keep\_psp

mov es, ax

mov es, es:[2ch]

mov bx, 0

findz:

inc bx

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

jne findz

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

jne findz

add bx, 2

mov di, 0

\_loop:

mov dl, es:[bx]

mov byte ptr [cl\_pos+di], dl

inc di

inc bx

cmp dl, 0

je \_end\_loop

cmp dl, '\'

jne \_loop

mov cx, di

jmp \_loop

\_end\_loop:

mov di, cx

mov si, program

\_fn:

mov dl, byte ptr [si]

mov byte ptr [cl\_pos+di], dl

inc di

inc si

cmp dl, 0

jne \_fn

pop es

pop si

pop di

pop dx

pop cx

pop bx

pop ax

ret

path endp

allocation\_mem proc

push ax

push bx

push cx

push dx

push dx

mov dx, offset dta\_mem

mov ah, 1ah

int 21h

pop dx

mov cx, 0

mov ah, 4eh

int 21h

jnc \_all\_success

\_allfile\_err:

cmp ax, 2

je \_allroute\_err

mov dx, offset str\_all\_file\_error

call print\_str

jmp \_all\_end

\_allroute\_err:

cmp ax, 3

mov dx, offset str\_all\_route\_error

call print\_str

jmp \_all\_end

\_all\_success:

push di

mov di, offset dta\_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 ovls\_addr, ax

mov dx, offset str\_allocation\_mem\_end

call print\_str

\_all\_end:

pop dx

pop cx

pop bx

pop ax

ret

allocation\_mem endp

begin\_ovl proc

push dx

call path

mov dx, offset cl\_pos

call allocation\_mem

call load\_proc

pop dx

ret

begin\_ovl endp

begin proc far

push ds

xor ax, ax

push ax

mov ax, data

mov ds, ax

mov keep\_psp, es

call free\_memory

cmp mem\_flag, 0

je \_end

mov dx, offset file1

call begin\_ovl

mov dx, offset eof

call print\_str

mov dx, offset file2

call begin\_ovl

\_end:

xor al, al

mov ah, 4ch

int 21h

begin endp

exit:

code ends

end begin