**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра математического обеспечения и применения ЭВМ**

**ОТЧЕТ  
по лабораторной работе № 7**

**по дисциплине «Операционные системы»**

**Тема: «Построение модуля оверлейной структуры»**

|  |  |  |
| --- | --- | --- |
| Студент гр. 8304 |  | Сергеев А.Д. |
| Преподаватель |  | Губкин А.Ф. |

Санкт-Петербург

2020

**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 .ЕХЕ, который выполняет функции:

1. Освобождает память для загрузки оверлеев.
2. Читает размер файла оверлея и запрашивает объём 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 аварийно.

**Шаг 6.** Занесите полученные результаты в виде скриншотов в отчёт. Оформите отчёт в соответствии с требованиями.

**Необходимые сведения для составления программы.**

Для организации программы, имеющей оверлейную структуру, используется функция 4B03h прерывания int 21h. Эта функция позволяет в отведённую область памяти, начинающуюся с адреса сегмента, загрузить программу, находящуюся в файле на диске. Передача управления загруженной программе этой функцией не осуществляется и префикс сегмента программы (PSP) не создаётся.

Перед загрузкой оверлея вызывающая программа должна освободить память по функции 4Ah прерывания int 21h. Затем определить размер оверлея. Это можно сделать с помощью функции 4Eh прерывания 21h. Перед обращением к функции необходимо определить область памяти размером в 43 байта под буфер DTA, которую функция заполнит, если файл не будет найден.

Оверлейные сегмент не является загрузочным модулем типов .COM или .EXE. Он представляет собой кодовый сегмент, который оформляется в ассемблере как функция с точкой входа по адресу 0 и возврат осуществляется командой RETF. Это необходимо сделать, потому что возврат управления должен быть осуществлён в программу, выполняющую оверлейный сегмент. Если использовать функции выхода 4Ch прерывания int 21h, то программа закончит свою работу.

**Описание программы.**

В результате выполнения лабораторной работы была написана программа, описание функций которой представлено ниже.

- STRONG\_OVERLOAD - загружает полный путь к файлу, адрес названия которого передан в AX в отдельную область памяти;

- OVER\_EXEC - выделяет память, загружает туда модуль оверлейной структуры и выполняет его;

- MEMORY\_FREE - освобождение незанятого программой места в памяти;

- PRINT\_STRING - вывод сообщения на экран;

Оверлейные модули содержат следующие функции:

- PRINT - вывод сообщения на экран;

- TETR\_TO\_HEX - перевод символа в 16 системе счисления в ASCII-код этого символа;

- BYTE\_TO\_HEX - перевод байта в 16 системе счисления в ASCII-строку;

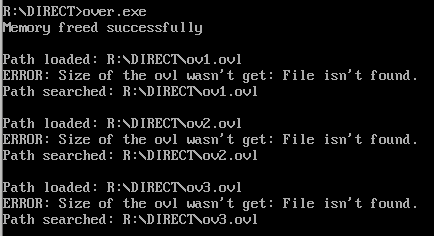
- WRD\_TO\_HEX - перевод регистра в 16 системе счисления в ASCII-строку и запись её по адресу DI;

**Ход работы**

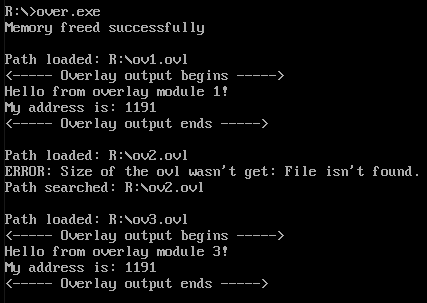
Написание исходного кода производилось в редакторе Atom на базе операционной системы Windows 10, сборка и отладка производились в эмуляторе DOSBox.

Рисунок 1 — Выполнение программы в одном каталоге с оверлейными модулями

Как видно из рисунка, программа завершилась в штатном режиме.

Рисунок 2 – Выполнение программы в другом каталоге

Как видно из рисунка, оверлейные модули в другом каталоге найдены не были.

Рисунок 3 — Одного из модулей нет в каталоге

Как видно из рисунка, программа выполнила первый и третий модуль, выведя сообщение об ошибке во втором.

**Вывод.**

В результате выполнения данной лабораторной работы была изучена возможность построения загрузочного модуля оверлейной структуры.

**Контрольные вопросы.**

**Как должна быть устроена программа, если в качестве оверлейного сегмента использовать .COM модули:**

В начале выделенной для оверлейного сегмента памяти необходимо сформировать и поместить PSP, соответственно сместив точку входа на 100h байт. Также необходимо будет сохранить, а затем восстановить регистры.

**ПРИЛОЖЕНИЕ А**

**ИСХОДНЫЙ КОД ПРОГРАММЫ. OVER.ASM**

ASSUME CS:CODE, DS:DATA, SS:STACKK

DATA SEGMENT

MEM\_FREE\_SUCCESS db "Memory freed successfully", 13, 10, "$"

MEM\_FREE\_ERROR db "Error freeing memory$", 13, 10, "$"

DELIM\_START db "<----- Overlay output begins ----->", 13, 10, "$"

DELIM\_END db 13, 10, "<----- Overlay output ends ----->", 13, 10, "$"

LOADED\_PATH db "Path loaded: $"

END\_L db 13, 10, "$"

LOADING\_ERROR db "ERROR: Size of the ovl wasn't get: $"

LOADING\_NO\_FILE db "File isn't found.", 13, 10, "$"

LOADING\_SEARCHED db "Path searched: $"

EXEC\_MEMORY db "Not enought memory!", 13, 10, "$"

EXEC\_LOADING db "Overlay isn't loaded!", 13, 10, "$"

PARAMS dw 0

ENTRY\_ADRESS dd 0

OVERLAY\_1 db "ov1.ovl", 0

OVERLAY\_2 db "ov2.ovl", 0

OVERLAY\_3 db "ov3.ovl", 0

MODULE\_PATH db 100 dup(0)

DATA ENDS

STACKK SEGMENT STACK

dw 100h dup (0)

STACKK ENDS

CODE SEGMENT

PRINT\_STRING PROC near

push ax

mov ah, 09h

int 21h

pop ax

ret

PRINT\_STRING ENDP

MEMORY\_FREE PROC near

push ax

push bx

push cx

push dx

mov bx, offset PROGRAM\_END

mov ax, es

sub bx, ax

mov cl, 4

shr bx, cl

inc bx

xor ax, ax

mov ah, 4Ah

int 21h

jc M\_ERR

mov dx, offset MEM\_FREE\_SUCCESS

call PRINT\_STRING

jmp M\_DEF

M\_ERR:

mov dx, offset MEM\_FREE\_ERROR

call PRINT\_STRING

stc

M\_DEF:

pop dx

pop cx

pop bx

pop ax

ret

MEMORY\_FREE ENDP

STRONG\_OVERLOAD PROC

push ax

push es

push si

push di

push dx

push ax

mov es, es:[2Ch]

mov si, 0

SKIP:

mov ax, es:[si]

inc si

cmp ax, 0

jne SKIP

add si, 3

mov di, offset MODULE\_PATH

EMPATH:

mov al, es:[si]

mov BYTE PTR [di], al

cmp al, '\'

je RECALL

inc di

inc si

jmp EMPATH

RECALL:

pop si

APPEND:

inc di

mov al, [si]

cmp al, 0

je PROCEED

mov BYTE PTR [di], al

inc si

jmp APPEND

PROCEED:

mov BYTE PTR [di], '$'

mov dx, offset LOADED\_PATH

call PRINT\_STRING

mov dx, offset MODULE\_PATH

call PRINT\_STRING

mov dx, offset END\_L

call PRINT\_STRING

pop dx

pop di

pop si

pop es

pop ax

ret

STRONG\_OVERLOAD ENDP

OVER\_EXEC PROC

push ax

push bx

push cx

push dx

push si

push es

mov ah, 4Eh

mov cx, 0

mov dx, offset MODULE\_PATH

int 21h

jnc NO\_ERROR

mov dx, offset LOADING\_ERROR

call PRINT\_STRING

cmp ax, 2

je NO\_FILE

cmp ax, 3

je NO\_FILE

cmp ax, 12h

je NO\_FILE

jmp EXEC\_DEF

NO\_FILE:

mov dx, offset LOADING\_NO\_FILE

call PRINT\_STRING

mov dx, offset LOADING\_SEARCHED

call PRINT\_STRING

mov dx, offset MODULE\_PATH

call PRINT\_STRING

mov dx, offset END\_L

call PRINT\_STRING

jmp EXEC\_DEF

NO\_ERROR:

mov si, 0080h

add si, 1Ah

mov bx, [si]

mov ax, [si + 2]

mov cl, 4

shr bx, cl

mov cl, 12

shl ax, cl

add bx, ax

add bx, 2

mov ah, 48h

int 21h

jnc PREPARE\_DATA

mov dx, offset EXEC\_MEMORY

call PRINT\_STRING

jmp EXEC\_DEF

PREPARE\_DATA:

mov PARAMS, ax

mov WORD PTR ENTRY\_ADRESS + 2, ax

mov dx, offset MODULE\_PATH

push ds

pop es

mov bx, offset PARAMS

mov ax, 4B03h

int 21h

jnc LOADED

mov dx, offset EXEC\_LOADING

call PRINT\_STRING

jmp EXEC\_DEF

LOADED:

mov ax, PARAMS

mov es, ax

mov dx, offset DELIM\_START

call PRINT\_STRING

call ENTRY\_ADRESS

mov dx, offset DELIM\_END

call PRINT\_STRING

mov es, ax

mov ah, 49h

int 21h

EXEC\_DEF:

pop es

pop si

pop dx

pop cx

pop bx

pop ax

ret

OVER\_EXEC ENDP

MAIN PROC

BEGIN:

mov ax, DATA

mov ds, ax

call MEMORY\_FREE

jc MAIN\_END

mov dx, offset END\_L

call PRINT\_STRING

mov ax, offset OVERLAY\_1

call STRONG\_OVERLOAD

call OVER\_EXEC

mov dx, offset END\_L

call PRINT\_STRING

mov ax, offset OVERLAY\_2

call STRONG\_OVERLOAD

call OVER\_EXEC

mov dx, offset END\_L

call PRINT\_STRING

mov ax, offset OVERLAY\_3

call STRONG\_OVERLOAD

call OVER\_EXEC

MAIN\_END:

mov ah, 4Ch

int 21h

MAIN ENDP

PROGRAM\_END:

CODE ENDS

END BEGIN

**ПРИЛОЖЕНИЕ Б**

**ИСХОДНЫЙ КОД ПРОГРАММЫ. OV1.COM**

ASSUME CS:OVL1, DS:OVL1, SS:NOTHING, ES:NOTHING

OVL1 SEGMENT

MAIN1 PROC FAR

push ds

push dx

push di

push ax

push bx

mov ax, cs

mov ds, ax

mov bx, offset MESSAGE

add bx, 48

mov di, bx

mov ax, cs

call WRD\_TO\_HEX

mov dx, offset MESSAGE

call PRINT

pop bx

pop ax

pop di

pop dx

pop ds

retf

MAIN1 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

xor ah, ah

call BYTE\_TO\_HEX

mov [di], ah

dec di

mov [di], al

pop bx

ret

WRD\_TO\_HEX ENDP

MESSAGE DB 'Hello from overlay module 1!', 13, 10, 'My address is: $'

OVL1 ENDS

END

**ПРИЛОЖЕНИЕ В**

**ИСХОДНЫЙ КОД ПРОГРАММЫ. OV2.COM**

ASSUME CS:OVL2, DS:OVL2, SS:NOTHING, ES:NOTHING

OVL2 SEGMENT

MAIN1 PROC FAR

push ds

push dx

push di

push ax

push bx

mov ax, cs

mov ds, ax

mov bx, offset MESSAGE

add bx, 48

mov di, bx

mov ax, cs

call WRD\_TO\_HEX

mov dx, offset MESSAGE

call PRINT

pop bx

pop ax

pop di

pop dx

pop ds

retf

MAIN1 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

xor ah, ah

call BYTE\_TO\_HEX

mov [di], ah

dec di

mov [di], al

pop bx

ret

WRD\_TO\_HEX ENDP

MESSAGE DB 'Hello from overlay module 2!', 13, 10, 'My address is: $'

OVL2 ENDS

END

**ПРИЛОЖЕНИЕ Г**

**ИСХОДНЫЙ КОД ПРОГРАММЫ. OV3.COM**

ASSUME CS:OVL3, DS:OVL3, SS:NOTHING, ES:NOTHING

OVL3 SEGMENT

MAIN1 PROC FAR

push ds

push dx

push di

push ax

push bx

mov ax, cs

mov ds, ax

mov bx, offset MESSAGE

add bx, 48

mov di, bx

mov ax, cs

call WRD\_TO\_HEX

mov dx, offset MESSAGE

call PRINT

pop bx

pop ax

pop di

pop dx

pop ds

retf

MAIN1 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

xor ah, ah

call BYTE\_TO\_HEX

mov [di], ah

dec di

mov [di], al

pop bx

ret

WRD\_TO\_HEX ENDP

MESSAGE DB 'Hello from overlay module 3!', 13, 10, 'My address is: $'

OVL3 ENDS

END