**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

|  |  |  |
| --- | --- | --- |
| Студентка гр. 6381 |  | Лопатина А.С. |
| Преподаватель |  | Губкин А.Ф. |

Санкт-Петербург

2018

**Постановка задачи**

**Цель работы.**

Исследование возможности построение 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 загруженные 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.

**Сведения о функциях и структурах данных управляющей программы.**

Сведения о функциях:

|  |  |
| --- | --- |
| Название | Назначение |
| OUTPUT\_PROC | Вывод на экран |
| OVL\_SIZE | Определение размера оверлея |
| FIND\_PATH | Нахождение пути до программы |
| CALL\_OVL | Вызов оверлейной программы |

Сведения о структурах данных:

|  |  |  |
| --- | --- | --- |
| Название | Тип | Назначение |
| err\_1\_7 | db | Разрушен управляющий блок памяти |
| err\_1\_8 | db | Недостаточно памяти для выполнения функции |
| err\_1\_9 | db | Неверный адрес блока памяти |
| err\_2 | db | Оверлейная программа не загружена |
| err\_2\_1 | db | Несуществующая функция |
| err\_2\_2 | db | Файл не найден |
| err\_2\_3 | db | Маршрут не найден |
| err\_2\_4 | db | Слишком много открытых файлов |
| err\_2\_5 | db | Нет доступа |
| err\_2\_8 | db | Мало памяти |
| err\_2\_10 | db | Неправильная среда |
| err\_3 | db | Размер оверлея не определен |
| err\_3\_2 | db | Файл не найден |
| err\_3\_3 | db | Маршрут не найден |
| name1 | db | Имя первого оверлея |
| name2 | db | Имя второго оверлея |
| adr | dd | Адрес, по которому вызывается оверлейная программа |
| keep\_PSP | dw | Переменная для хранения PSP |
| ovl\_adr | dw | Блок параметров |
| DTA | db | Буфер DTA |
| DTA\_path | db | Путь к оверлею |

**Последовательность действий, выполняемых программой.**

Модуль типа .EXE выполняет следующие 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. Действия 1-4 выполняются до следующего оверлейного сегмента.

**Результат работы программы.**

1. Написали и отладили программный модуль .EXE. Запустили отлаженную программу.

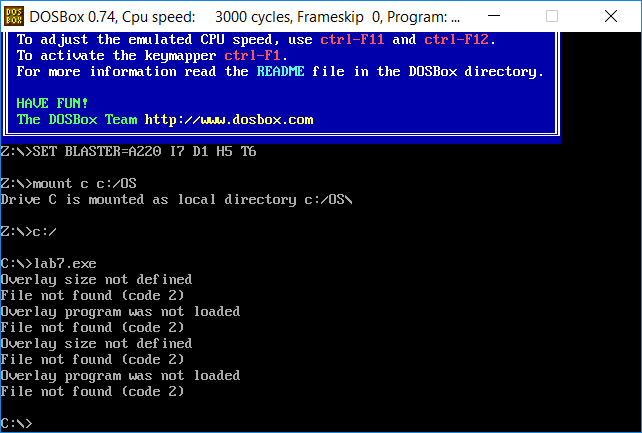


рис.1

1. Написали и отладили оверлейные сегменты. Оверлейный сегмент выводит адрес сегмента, в который он загружен. Запустили отлаженное приложение. Оверлейные сегменты загружаются с одного и того же адреса, перекрывая друг друга.

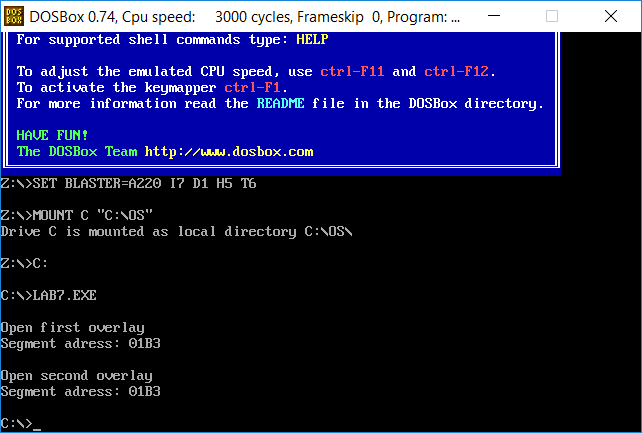


рис. 2

1. Запустили приложение из другого каталога. Приложение выполнено успешно.

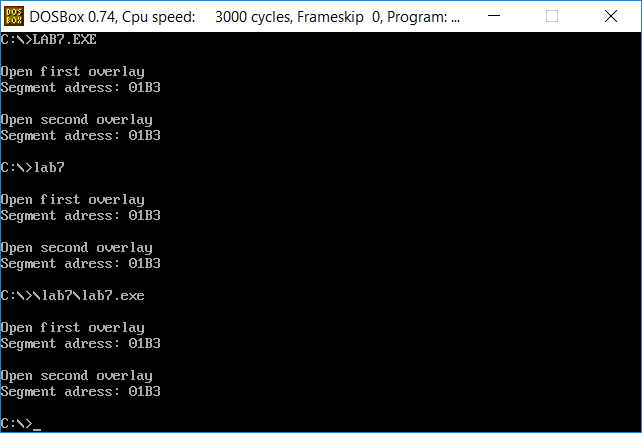


рис. 3

1. Запустили приложение в случае, когда одного оверлея нет в каталоге. Приложение заканчивается аварийно.

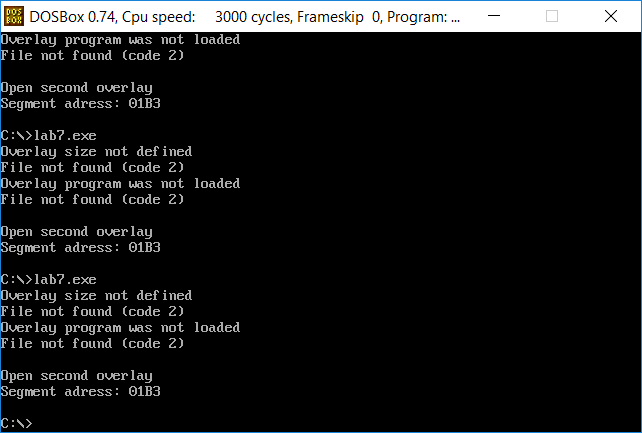


рис.4

**Описание результатов исследования проблем, поставленных в лабораторной работе**

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

Ответ: если в качестве оверлейного сегмента использовать .COM модули, то должно присутствовать смещение адресации 100h, так как в .COM файлах код располагается с адреса 100h. Необходимо в .COM модуле переписать содержимое PSP из вызывающей программы.

**Заключение**

В результате выполнения лабораторной работы были изучены возможности построения загрузочного модуля оверлейной структуры и исследована структура оверлейного сегмента и способ загрузки и выполнения оверлейных сегментов.

Приложения

**Код программы** **lab7**.**asm**

ASSUME CS:CODE, ds:DATA, ss:AStack

AStack SEGMENT STACK

DW 64 DUP(?)

AStack ENDS

DATA SEGMENT

err\_1\_7 db 'The memory block is destroyed (code 7)', 0dh, 0ah, '$'

err\_1\_8 db 'Not enought memory to perform the function (code 8)', 0dh, 0ah, '$'

err\_1\_9 db 'Wrong adress of memory block (code 9)', 0dh, 0ah, '$'

err\_2 db 'Overlay program was not loaded', 0dh, 0ah, '$'

err\_2\_1 db 'Non-existen function (code 1)', 0dh, 0ah, '$'

err\_2\_2 db 'File not found (code 2)', 0dh, 0ah, '$'

err\_2\_3 db 'Way not found (code 3)', 0dh, 0ah, '$'

err\_2\_4 db 'Too many open files (code 4)', 0dh, 0ah, '$'

err\_2\_5 db 'No acsess (code 5)', 0dh, 0ah, '$'

err\_2\_8 db 'Low memory(code 8)', 0dh, 0ah, '$'

err\_2\_10 db 'Incorrect environment (code 10)', 0dh, 0ah, '$'

err\_3 db 'Overlay size not defined', 0dh, 0ah, '$'

err\_3\_2 db 'File not found (code 2)', 0dh, 0ah, '$'

err\_3\_3 db 'Way not found (code 3)', 0dh, 0ah, '$'

name1 db 'ovl1.ovl',0

name2 db 'ovl2.ovl',0

adr dd 0

keep\_PSP dw 0

ovl\_adr dw 0

DTA db 43 dup (0), '$'

DTA\_path db 64 dup (0), '$'

DATA ENDS

CODE SEGMENT

;----------------------------

OVL\_SIZE PROC ;Определяем размер оверлея

push es

push bx

push si

push ds

push dx

mov dx, seg DTA

mov ds, dx

lea dx, DTA

mov ax, 1A00h ;Установка адреса для DTA

int 21h

pop dx

pop ds

push ds

push dx

mov cx, 0

mov dx, seg DTA\_path

mov ds, dx

lea dx, DTA\_path ;Указатель на строку, содержащую путь к оверлею

mov ax, 4E00h

int 21h

pop dx

pop ds

jnc without\_err

lea dx, err\_3

call OUTPUT\_PROC

cmp ax, 2

je err\_3\_2\_out

cmp ax, 3

je err\_3\_3\_out

err\_3\_2\_out:

lea dx, err\_3\_2

call OUTPUT\_PROC

jmp end\_1

err\_3\_3\_out:

lea dx, err\_3\_3

call OUTPUT\_PROC

jmp end\_1

without\_err:

push es

push bx

push si

lea si, DTA

add si, 1Ch ;В буфере старшее слово размера файла

mov bx, [si]

sub si, 2

mov bx, [si]

push cx

mov cl, 4

shr bx, cl ;Переводим в параграфы

pop cx

mov ax, [si+2]

push cx

mov cl, 12

sal ax, cl ;Переводим в байты, а затем в параграфы

pop cx

add bx, ax

inc bx

inc bx

mov ax, 4800h ;Выделение памяти

int 21h

mov ovl\_adr, ax

pop si

pop bx

pop es

end\_1:

pop si

pop bx

pop es

ret

OVL\_SIZE ENDP

;----------------------------

FIND\_PATH PROC

push ax

push bx

push cx

push dx

push si

push di

push es

mov es, keep\_PSP

mov ax, es:[2Ch]

mov es, ax

mov bx, 0

mov cx, 2

env\_loop:

inc cx

mov al, es:[bx]

inc bx

cmp al, 0

jz end\_env

loop env\_loop

end\_env:

cmp byte ptr es:[bx], 0

jnz env\_loop

add bx, 3

lea si, DTA\_path

path\_loop:

mov al, es:[bx]

mov [si], al

inc si

inc bx

cmp al, 0

jz end\_path

jmp path\_loop

end\_path:

sub si, 9

mov di, bp

replace\_loop:

mov ah, [di]

mov [si], ah

cmp ah, 0

jz end\_replace

inc di

inc si

jmp replace\_loop

end\_replace:

pop es

pop di

pop si

pop dx

pop cx

pop bx

pop ax

ret

FIND\_PATH ENDP

;----------------------------

CALL\_OVL PROC

push ax

push bx

push cx

push dx

push bp

mov bx, seg ovl\_adr

mov es, bx

lea bx, ovl\_adr

mov dx, seg DTA\_path

mov ds, dx

lea dx, DTA\_path

push ss

push sp

mov ax, 4B03h

int 21h

jnc without\_err\_2

lea dx, err\_2

call OUTPUT\_PROC

cmp ax, 1

je err\_2\_1\_out

cmp ax, 2

je err\_2\_2\_out

cmp ax, 3

je err\_2\_3\_out

cmp ax, 4

je err\_2\_4\_out

cmp ax, 5

je err\_2\_5\_out

cmp ax, 8

je err\_2\_8\_out

cmp ax, 10

je err\_2\_10\_out

jmp end\_2

err\_2\_1\_out:

lea dx, err\_2\_1

call OUTPUT\_PROC

jmp end\_2

err\_2\_2\_out:

lea dx, err\_2\_2

call OUTPUT\_PROC

jmp end\_2

err\_2\_3\_out:

lea dx, err\_2\_3

call OUTPUT\_PROC

jmp end\_2

err\_2\_4\_out:

lea dx, err\_2\_4

call OUTPUT\_PROC

jmp end\_2

err\_2\_5\_out:

lea dx, err\_2\_5

call OUTPUT\_PROC

jmp end\_2

err\_2\_8\_out:

lea dx, err\_2\_8

call OUTPUT\_PROC

jmp end\_2

err\_2\_10\_out:

lea dx, err\_2\_10

call OUTPUT\_PROC

jmp end\_2

without\_err\_2:

mov ax, seg DATA

mov ds, ax ;Восстанавливаем ds

mov ax, ovl\_adr

mov word ptr adr+2, ax

call adr

mov ax, ovl\_adr

mov es, ax

mov ax, 4900h

int 21h

mov ax, seg DATA

mov ds, ax

end\_2:

pop sp

pop ss

mov es, keep\_PSP

pop bp

pop dx

pop cx

pop bx

pop ax

ret

CALL\_OVL ENDP

;----------------------------

OUTPUT\_PROC PROC NEAR ;Вывод на экран сообщения

push ax

mov ah, 09h

int 21h

pop ax

ret

OUTPUT\_PROC ENDP

;----------------------------

Main PROC

mov ax, seg DATA

mov ds, ax

mov keep\_PSP, es

;Освобождение памяти

lea bx, ENDPROG

mov cl,4h

shr bx,cl

mov ah,4ah

int 21h

jnc success ; CF=0

cmp ax, 7

je err\_1\_7\_out

cmp ax, 8

je err\_1\_8\_out

cmp ax, 9

je err\_1\_8\_out

err\_1\_7\_out:

lea dx, err\_1\_7

call OUTPUT\_PROC

jmp quit

err\_1\_8\_out:

lea dx, err\_1\_8

call OUTPUT\_PROC

jmp quit

err\_1\_9\_out:

lea dx, err\_1\_9

call OUTPUT\_PROC

jmp quit

success:

lea bp, name1

call FIND\_PATH

call OVL\_SIZE

call CALL\_OVL

lea bp, name2

call FIND\_PATH

call OVL\_SIZE

call CALL\_OVL

quit:

xor al, al

mov ah, 4ch

int 21h

Main ENDP

ENDPROG:

CODE ENDS

END Main

**Код оверлейной программы ovl1.asm (ovl2.asm аналогичный)**

OVL\_1 segment

ASSUME cs:OVL\_1, ds:nothing, ss:nothing, es:nothing

MAIN PROC FAR

push ds

push ax

push di

push dx

push bx

mov ds, ax

lea dx, cs:string

call OUTPUT\_PROC

lea bx, cs:adr

add bx, 19

mov di, bx

mov ax, cs

call WRD\_TO\_HEX

lea dx, cs:adr

call OUTPUT\_PROC

pop bx

pop dx

pop di

pop ax

pop ds

retf

MAIN ENDP

;----------------------------

string db 10, 13, 'Open first overlay', 10, 13, '$'

adr db 'Segment adress: ', 10, 13, '$'

;----------------------------

OUTPUT\_PROC PROC ;Вывод на экран сообщения

push ax

mov ah, 09h

int 21h

pop ax

ret

OUTPUT\_PROC 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 ; в al - старший байт

pop cx ;в ah - младший

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

;----------------------------

OVL\_1 ENDS

END MAIN