

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

Студент гр. 0382

Шангичев В. А.

Преподаватель

Ефремов М.А.

Санкт-Петербург

2022

Цель работы.

Исследование возможности построения загрузочного модуля оверлейной структуры. Исследуется структура оверлейного сегмента и способ загрузки и выполнения оверлейных сегментов.

Задание.

1. Написать и отладить программный модуль типа .EXE, который выполняет следующие 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

2. Написать и отладить оверлейные сегменты. Оверлейный сегмент выводит адрес сегмента, в который он загружен.

3. Запустите отлаженное приложение. Оверлейные сегменты должны загружаться с одного и того же адреса, перекрывая друг друга.

4. Запустите приложение из другого каталога. Приложение должно быть выполнено успешно.

5. Запустите приложение в случае, когда одного оверлея нет в каталоге. Приложение должно закончиться аварийно.

Выполнение работы.

Шаг 1. В ходе выполнения задания по лабораторной работе был написан файл `main.asm`. Описание процедур:

`writemessage` – печатает сообщение в консоль

`memory_alloc` – процедура освобождения памяти

`free_mem` - процедура, освобождающая память с помощью процедуры выше и выводящая сообщение об ошибке в случае необходимости

`get_params` - процедура, подготавливающая блок параметров для оверлея.

`get_filesize` - определяет размер файла оверлея.

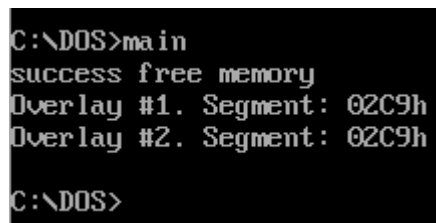
`memory_for_overlay` - запрашивает память под оверлей.

`load_overlay` - выполняет загрузку оверлея.

`main` - главная процедура.

Шаг 2. Далее были написаны исходные файлы `lb17`, `lb27`, скомпилированные в оверлеи `o1` и `o2` соответственно. Каждый оверлей выводит адрес сегмента, в который он был загружен.

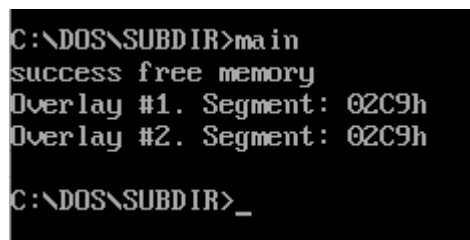
Шаг 3.



```
C:\DOS>main
success free memory
Overlay #1. Segment: 02C9h
Overlay #2. Segment: 02C9h
C:\DOS>
```

Рисунок 1 – запуск из одного каталога

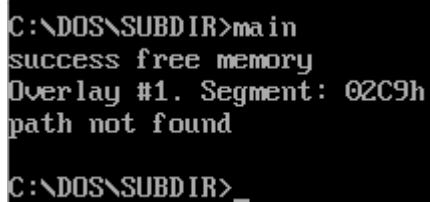
Шаг 4.



```
C:\DOS\SUBDIR>main
success free memory
Overlay #1. Segment: 02C9h
Overlay #2. Segment: 02C9h
C:\DOS\SUBDIR>_
```

Рисунок 2 – запуск из другого каталога

Шаг 5. Из текущего каталога был удален второй оверлей.



```
C:\DOS\SUBDIR>main
success free memory
Overlay #1. Segment: 02C9h
path not found
C:\DOS\SUBDIR>_
```

Рисунок 3 – удаление второго оверлея

Ответы на контрольные вопросы.

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

Основная программа должна быть устроена аналогично – в .COM модуле также только один сегмент, как и в оверлее. В коде модуля типа .COM задавать смещение директивой ORG 100h не нужно, т. к. при передаче управления оверлею PSP не создается.

Вывод.

Были исследованы возможности построения загрузочного модуля оверлейной структуры. Была написана программа, загружающая два оверлея.

ПРИЛОЖЕНИЕ А

КОД ПРОГРАММЫ

Файл main.asm

```
astack      segment  stack

              dw 512 dup(?)

astack      ends


data        segment


overlay1_name db "o1.ovl",0
overlay2_name db "o2.ovl",0


parametr_block dw 2 dup(0)

newline db 0dh,0ah,'$'

memory_destroyed_msg db 'the control memory block is destroyed', 0dh,
0ah,'$'

not_enough_msg db 'not enough memory to execute the function', 0dh,
0ah,'$'

inv_address_msg db 'invalid memory block address', 0dh, 0ah,'$'

success_mem_msg db 'success free memory', 0dh, 0ah,'$'

free_mem_flag db 0

path_to_file db 50 dup(0)

dta_buffer db 43 dup(0)

overlayaddress dd 0


pathnotfound db "path not found",0dh,0ah,'$'

nosuchfunc db "function doesn't exist",0dh,0ah,'$'
```

```

filenotfound db "file not found",0dh,0ah,'$'
toomanyopen db "too many open files",0dh,0ah,'$'
notenoughmemory db "not enought memory",0dh,0ah,'$'
noaccess db "no access",0dh,0ah,'$'
wrongenv db "wrong enviroment",0dh,0ah,'$'
keep_psp dw 0
end_data db 0
data ends

```

```

code      segment

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

```

```

;печать сообщения

```

```

writemessage proc near

```

```

push ax

```

```

mov ah,09h

```

```

int 21h

```

```

pop ax

```

```

ret

```

```

writemessage endp

```

```

memory_alloc proc near

```

```

    push ax

```

```

    push bx

```

```

    xor dx, dx

```

```

    mov ax, offset end_data

```

```

    mov bx, offset end_programm

    add ax, bx

    mov bx, 16

    div bx

    add ax, 100h

    mov bx, ax

    and ax, 0

    mov ah, 4ah

    int 21h

    pop bx

    pop ax

    ret

memory_alloc endp

```

```

free_mem proc near

    push dx

    push ax

    call memory_alloc

    jnc success_free

    cmp ax, 7

    je mem_destr

```

```

cmp ax, 8

je not_enough


cmp ax, 9

je inv_addr


mem_destr:

    mov dx, offset memory_destroyed_msg

    jmp finish_proc


not_enough:

    mov dx, offset not_enough_msg

    jmp finish_proc


inv_addr:

    mov dx, offset inv_address_msg

    jmp finish_proc


success_free:

    mov dx, offset success_mem_msg

    mov free_mem_flag, 1


finish_proc:

    call writemessage

    pop ax

    pop dx

```



```

                ret

free_mem endp


get_params proc near

push es

push ax

push si


sub si, si

mov es, es:[2ch]


find_loop:

        mov al, es:[si]

        inc si

        cmp al, 0

        jne find_loop

        mov al, es:[si]

        cmp al, 0

        jne find_loop


add si, 3

push si


find_slash:

        cmp byte ptr es:[si], '\'

        jne next_char

        mov ax, si

```

```

next_char:

        inc si

        cmp byte ptr es:[si], 0

        jne find_slash

        inc ax

        pop si

        mov di, 0

```

```

save_path:

        mov bl, es:[si]

        mov path_to_file[di], bl

        inc si

        inc di

        cmp si, ax

        jne save_path

        pop si

```

```

add_filename:

        mov bl, [si]

        mov path_to_file[di], bl

        inc si

        inc di

        cmp bl, 0

        jne add_filename

```

```

pop ax

pop es

ret

get_params endp


get_filesize proc near

push cx

push dx

mov dx, offset dta_buffer

mov ah, 1ah

int 21h


mov cx, 0

mov dx, offset path_to_file

mov ah, 4eh

int 21h


jnc no_err_get

cmp ax, 2

je get_error_2

lea dx, pathnotfound

jmp write_err_mess

get_error_2:

        lea dx, filenotfound

write_err_mess:

        call writemessage

        mov bx, 1

```

```

        jmp end_get

        ret

no_err_get:

        mov ax, word ptr dta_buffer[1ah]

        mov dx, word ptr dta_buffer[1ah+2]

        mov cl, 4

        shr ax, cl

        mov cl, 12

        shl dx, cl

        add ax, dx

        add ax, 1

        mov bx, 0

end_get:

        pop dx

        pop cx

        ret

get_filesize endp


memory_for_overlay proc near

push bx

push dx

mov bx, ax

mov ah, 48h

int 21h

jnc no_err_mem

lea dx, notenoughmemory

call writemessage

```

```

mov bx, 1

jmp mem_end

no_err_mem:

        mov parametr_block[0], ax

        mov parametr_block[2], ax

        mov bx, 0

mem_end:

pop dx

pop bx

ret

memory_for_overlay endp

```

```

load_overlay proc near

push ax

push es

push bx

push dx

lea dx, path_to_file

mov ax, ds

mov es, ax

lea bx, parametr_block

mov ax, 4b03h

int 21h

jnc no_err_load

cmp ax, 1

je load_nofunc

```

```

cmp ax, 2

je load_nofile

cmp ax, 3

je load_nopath

cmp ax, 4

je load_open

cmp ax, 5

je load_noacc

cmp ax, 8

je load_nomem

lea dx, wrongenv

jmp load_write_err_msg

load_nofunc:

    lea dx, nosuchfunc

    jmp load_write_err_msg

load_nofile:

    lea dx, filenotfound

    jmp load_write_err_msg

load_nopath:

    lea dx, pathnotfound

    jmp load_write_err_msg

load_open:

    lea dx, toomanyopen

    jmp load_write_err_msg

load_noacc:

    lea dx, noaccess

    jmp load_write_err_msg

```

```

load_nomem:

        lea dx, notenoughmemory

load_write_err_msg:

        call writemessage

        jmp load_end

no_err_load:

        mov ax, parametr_block[2]

        mov word ptr overlayaddress+2, ax

        call overlayaddress


        call free_mem

        cmp free_mem_flag, 0

        jne end_main


        mov si, offset overlay1_name

        call get_params

        call get_filesize

        cmp bx, 0


        mov es, ax

        mov ah, 49h

        int 21h

load_end:

        pop dx

        pop bx

        pop es

        pop ax

```

```

        ret

load_overlay endp


main      proc far

        push ds

        sub ax, ax

        push ax

        mov ax, data

        mov ds, ax

        mov keep_psp, es

        jne load_second

        call memory_for_overlay

        cmp bx, 0

        jne load_second

        call load_overlay


load_second:

        mov es, keep_psp

        lea si, overlay2_name

        call get_params

        call get_filesize

        cmp bx, 0

        jne end_main

        call memory_for_overlay

        cmp bx, 0

        jne end_main

        call load_overlay

```



```

        end_main:

        mov ah, 4ch

        int 21h

        end_programm:

        ret

main     endp

code     ends

        end main

```

Файл lb17.asm

```
CODE SEGMENT
```

```
        ASSUME CS:CODE,DS:NOTHING,SS:NOTHING
```

```
MAIN PROC Far
```

```
    push AX
```

```
    push DX
```

```
    push DS
```

```
    push DI
```

```
    mov AX, CS
```

```
    mov DS, AX
```

```
    lea DI, CONTROL_LINE
```

```
    add DI, 24
```

```
    call WRD_TO_HEX
```

```
lea DX, CONTROL_LINE
```

```
call WRITEMESSAGE
```

```
pop DI
```

```
pop DS
```

```
pop DX
```

```
pop AX
```

```
RETF
```

```
MAIN ENDP
```

```
CONTROL_LINE db "Overlay #1. Segment:      h",0DH,0AH,'$'
```

```
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
```

```
;байт в AL переводится в два символа шест. числа в AX
```

```
    push CX
```

```
    mov AH,AL
```

```
    call TETR_TO_HEX
```

```

    xchg AL,AH

    mov CL,4

    shr AL,CL

    call TETR_TO_HEX ;в AL старшая цифра

    pop CX ;в AH младшая

    ret

BYTE_TO_HEX ENDP

WRD_TO_HEX PROC near

;перевод в 16 с/с 16-ти разрядного числа

; в AX - число, DI - адрес последнего символа

    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

```

```
WRITEMESSAGE PROC Near
```

```
push AX
```

```
mov AH,09h
```

```
int 21h
```

```
pop AX
```

```
ret
```

```
WRITEMESSAGE ENDP
```

```
CODE ENDS
```

```
END MAIN
```

Файл lb27.asm

```
CODE SEGMENT
```

```
ASSUME CS:CODE,DS:NOTHING,SS:NOTHING
```

```
MAIN PROC Far
```

```
push AX
```

```
push DX
```

```
push DS
```

```
push DI
```

```
mov AX, CS
```

```
mov DS, AX
```

```
lea DI, CONTROL_LINE
```

```
add DI, 24
```

```
call WRD_TO_HEX
lea DX, CONTROL_LINE
call WRITEMESSAGE
```

```
pop DI
pop DS
pop DX
pop AX
RETF
MAIN ENDP
```

```
CONTROL_LINE db "Overlay #2. Segment:      h",0DH,0AH,'$'
```

```
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
```

```
;байт в AL переводится в два символа шест. числа в AX
```

```
    push CX
```

```
    mov AH,AL
```

```

    call TETR_TO_HEX

    xchg AL,AH

    mov CL,4

    shr AL,CL

    call TETR_TO_HEX ;в AL старшая цифра

    pop CX ;в AH младшая

    ret

BYTE_TO_HEX ENDP


WRD_TO_HEX PROC near

;перевод в 16 с/с 16-ти разрядного числа

; в AX - число, DI - адрес последнего символа

    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

```

```
WRITEMESSAGE PROC Near
```

```
push AX
```

```
mov AH,09h
```

```
int 21h
```

```
pop AX
```

```
ret
```

```
WRITEMESSAGE ENDP
```

```
CODE ENDS
```

```
END MAIN
```