

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. 9383

Крейсманн К.В.

Преподаватель

Ефремов М.А.

Санкт-Петербург

2021

Цель работы.

Исследование возможности построения загрузочного модуля оверлейной структуры, структуры оверлейного сегмента и способа загрузки и выполнения оверлейных сегментов.

Задание.

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.

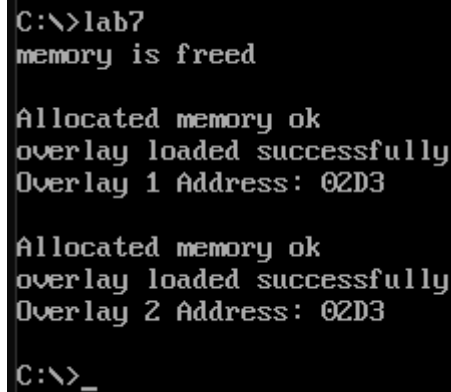
Исходный код.

Исходный код представлен в приложении А.

Выполнение работы

1. Был разработан и отлажен программный модуль типа .exe.
2. Программа запущена из директории с разработанным модулем.

Результат работы программа представлен на рисунке 1.



```
C:\>lab7
memory is freed

Allocated memory ok
overlay loaded successfully
Overlay 1 Address: 02D3

Allocated memory ok
overlay loaded successfully
Overlay 2 Address: 02D3

C:\>_
```

Рисунок 1

3. Программа была запущена из другой директории. Результат представлен на рисунке 2.



```
C:\>lab7\lab7.exe
memory is freed

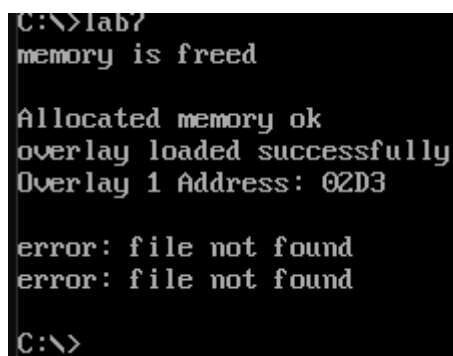
Allocated memory ok
overlay loaded successfully
Overlay 1 Address: 02D3

Allocated memory ok
overlay loaded successfully
Overlay 2 Address: 02D3

C:\>
```

Рисунок 2

4. Программа была запущена из каталога, в котором не присутствовал один оверлейный сегмент. Результат представлен на рисунке 3.



```
C:\>lab7
memory is freed

Allocated memory ok
overlay loaded successfully
Overlay 1 Address: 02D3

error: file not found
error: file not found

C:\>
```

Рисунок 3

ОТВЕТЫ НА КОНТРОЛЬНЫЕ ВОПРОСЫ ПО ЛАБОРАТОРНОЙ №6

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

В .com модуле изначально будет присутствовать смещение в 100h, но т.к. оверлейный модуль имеет точку входа по адресу 0, то нужно будет вычитать это смещение при обращении к данным.

Выводы.

Были исследована возможность построения загрузочного модуля оверлейной структуры.

Приложение А.

Исходный код программы lab7.asm:

```
STACK SEGMENT STACK
```

```
    DW 256 DUP(?)
```

```
STACK ENDS
```

```
DATA SEGMENT
```

```
string_free db 'memory is freed',0dh,0ah,'$'
```

```
string_mcb_des db 'error: mcb is destroyed',0dh,0ah,'$'
```

```
string_memory_n_engh db 'error: not enough memory', 0dh,0ah,'$'
```

```
string_mcb_addr db 'error: invalid mcb address',0dh,0ah,'$'
```

```
string_file_error db 'error: file not found',0dh,0ah,'$'
```

```
string_route_error db 'error: route not found',0dh,0ah,'$'
```

```
string_n_existent_error db 'error: non-existent function',0dh,0ah,'$'
```

```
string_many_files_error db 'error: too many open files',0dh,0ah,'$'
```

```
string_access_error db 'error: no access',0dh,0ah,'$'
```

```
string_environment_error db 'error: wrong environment',0dh,0ah,'$'
```

```
string_newline db 0dh,0ah,'$'
```

```
num db 0
```

```
string_ovl_load db 'overlay loaded successfully',0dh,0ah,'$'
```

```
string_path db 128 DUP(0)
```

```
string_alloc_mem_ok db 'Allocated memory ok',0dh,0ah,'$'
```

```
dta_mem db 43 dup(0)
```

```
err_free db 0
```

```
_psp dw 0
program dw 0
ovl_address dd 0
temp_sp dw 0
temp_ss dw 0
```

```
data_ db 0
DATA ENDS
```

```
CODE SEGMENT
```

```
    ASSUME CS:CODE, DS:DATA, SS:ASTACK
```

```
WRITE_STR proc near
```

```
    push ax
    mov ah,9h
    int 21h
    pop ax
    ret
```

```
WRITE_STR ENDP
```

```
FREE proc near
```

```
    push ax
    push bx
    push cx
    push dx
```

```
    xor dx,dx
```

```
    mov ax, offset data
    add bx,offset endofcode
```

add ax,bx

mov bx,10h

div bx

add ax,100h

mov bx,ax

xor ax,ax

mov ah,4ah

int 21h

jnc ok

mov err_free,1

cmp ax,7

jne m1

mov dx,offset string_mcb_des

call write_str

jmp endFree

m1:

cmp ax,8

jne m2

mov dx,offset string_memory_n_enh

call write_str

jmp endFree

m2:

cmp ax,9

mov dx,offset string_mcb_addr

call write_str

```
jmp endFree
```

```
ok:
```

```
mov dx,offset string_free
```

```
call WRITE_STR
```

```
endFree:
```

```
pop dx
```

```
pop cx
```

```
pop bx
```

```
pop ax
```

```
ret
```

```
FREE ENDP
```

```
path proc near
```

```
push ax
```

```
push bx
```

```
push cx
```

```
push dx
```

```
mov es,_psp
```

```
mov es,es:[2ch]
```

```
xor di,di
```

```
find_00:
```

```
mov al, es:[di]
```

```
inc di
```

```
cmp al,0
```

```
jne find_00
```



```

    mov al, es:[di]
    cmp al, 0
    jne find_00

    add di, 3
    mov si, offset string_path
n1:      ;смотрим путь
    mov cl, es:[di]
    mov byte ptr [si], cl
    cmp cl, 0h
    je n2
    inc si
    inc di
    jmp n1
n2:

    sub si, 8
    mov [si], byte ptr 'O'
    inc si
    mov [si], byte ptr 'V'
    inc si
    mov [si], byte ptr 'L'
    inc si

    mov al, num
    cmp al, 0
    jne n3
    mov [si], byte ptr '1'
    inc si
    jmp n4

```

```
n3:
    mov [si],byte ptr '2'
    inc si
```

```
n4:
    mov [si],byte ptr '.'
    inc si
    mov [si],byte ptr 'O'
    inc si
    mov [si],byte ptr 'V'
    inc si
    mov [si],byte ptr 'L'
    inc si
    mov [si],byte ptr 0
    inc si
    mov [si],byte ptr '$'
```

```
    pop dx
    pop cx
    pop bx
    pop ax
    ret
path endp
```

```
alloc_mem proc near
    push ax
    push bx
    push cx
    push dx
```

```
mov dx,offset dta_mem
mov ah,1ah
int 21h
```

```
mov dx,offset string_path
mov cx,0
mov ah,4eh
int 21h
```

```
jnc ok_alloc
```

```
cmp ax,12h
jne route_error
mov dx,offset string_file_error
call write_str
jmp alloc_end
```

```
route_error:
  cmp ax,3
  mov dx,offset string_route_error
  call write_str
  jmp alloc_end
```

```
ok_alloc:
  mov di,offset dta_mem
  mov dx,[di+1ch]
  mov ax,[di+1ah]
```

```
mov bx,10h
```

div bx

inc ax

mov bx,ax

mov ah,48h

int 21h

mov bx,offset ovl_address

mov cx,0h

mov [bx],ax

mov [bx+2],cx

mov dx,offset string_alloc_mem_ok

call write_str

alloc_end:

pop dx

pop cx

pop bx

pop ax

ret

alloc_mem endp

load_overlay proc near

push ax

push bx

push cx

push dx

```
push ds
push es
mov temp_ss,ss
mov temp_sp,sp
```

```
mov ax,data
mov es,ax
```

```
mov bx,offset ovl_address
mov dx,offset string_path
mov ax,4b03h
int 21h
```

```
mov sp ,temp_sp
mov ss, temp_ss
pop es
pop ds
```

```
jnc completion
```

```
err1:
```

```
cmp ax,1
jne err2
mov dx,offset string_n_existent_error
call write_str
jmp _end_load
```

```
err2:
```

```
cmp ax,2
jne err3
```

```

        mov dx,offset string_file_error
        call write_str
        jmp _end_load
err3:
        cmp ax,3
        jne err4
        mov dx,offset string_route_error
        call write_str
        jmp _end_load
err4:
        cmp ax,4
        jne err5
        mov dx,offset string_many_files_error
        call write_str
        jmp _end_load
err5:
        cmp ax,5
        jne err8
        mov dx,offset string_access_error
        call write_str
        jmp _end_load
err8:
        cmp ax,8
        jne err10
        mov dx,offset string_memory_n_enh
        call write_str
        jmp _end_load
err10:
        cmp ax,10
        jne _end_load

```

```
    mov dx,offset string_environment_error
    call write_str
    jmp _end_load
```

completion:

```
    mov dx,offset string_ovl_load
    call write_str
```

```
    mov bx,offset ovl_address
    mov ax,[bx]
    mov cx,[bx+2]
    mov [bx],cx
    mov [bx+2],ax
```

```
    call ovl_address
```

```
    mov es,ax
    mov ah,49h
    int 21h
```

_end_load:

```
    pop dx
    pop cx
    pop bx
    pop ax
    ret
```

```
load_overlay endp
```

MAIN PROC FAR

```

m:
    mov ax,data
    mov ds,ax
    mov _psp,es

    call free
    cmp err_free,0
    je _next
    jmp end_prog
_next:
    mov dx,offset string_newline
    call write_str

    call path
    call alloc_mem
    call load_overlay

    add num,1

    mov dx,offset string_newline
    call write_str

    call path
    call alloc_mem
    call load_overlay

end_prog:
    xor al,al

```



```
mov ah,4ch
```

```
int 21h
```

```
MAIN ENDP
```

```
endofcode:
```

```
code ends
```

```
end mainendp
```