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. 9382 _____ Павлов Р.В.

Преподаватель _____ Ефремов М.А.

Санкт-Петербург

2021

Постановка задачи.

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.

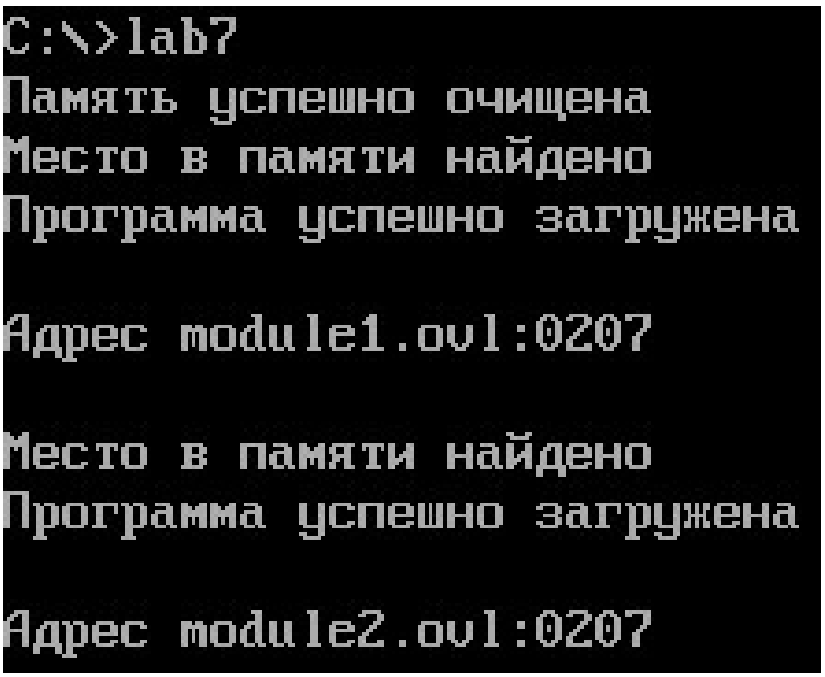
Функции и структуры данных:

Название процедуры	**Описание процедуры**
main	Вызов процедур выполнения оверлейных модулей
free	Освобождение занятой программой памяти для выделения её оверлейным модулям
run	Загрузка модуля в память и его вызов
find_path	Нахождение пути к указанному файлу
allocate	Нахождение участка доступной памяти
ovl_start	Запускает все необходимые для работы оверлейного модуля процедуры

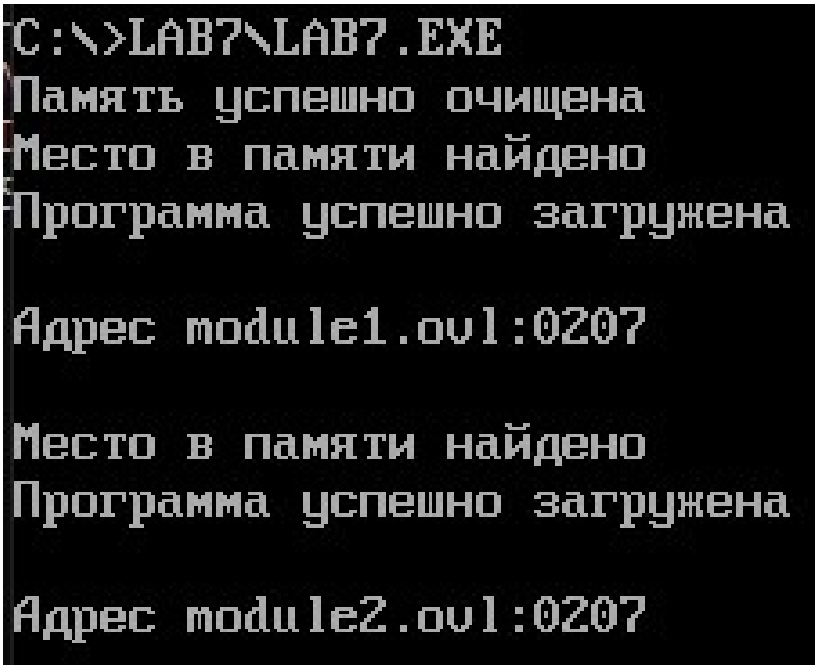
Ход работы.

Написан и отлажен программный модуль типа .EXE. Модуль осуществляет освобождение и поиск памяти, а также загрузку оверлейного модуля в память. После выполнения вызываемого модуля память, занимаемая им, очищается.

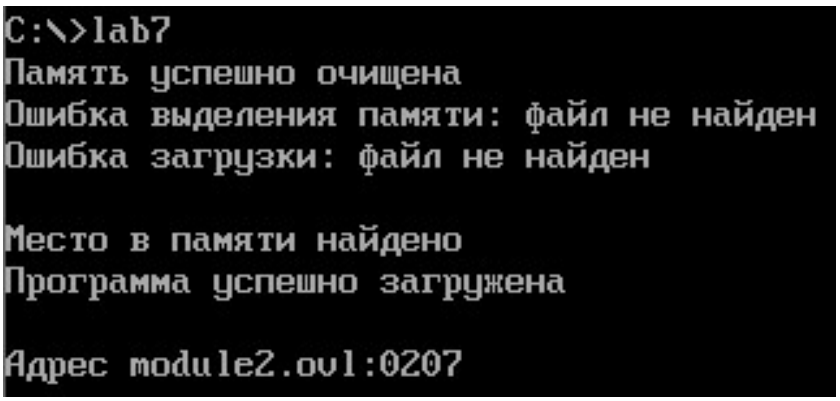
Оверлейные сегменты загружаются с одного и того же адреса:



При запуске из другого каталога программа работает корректно:



В случае, когда одного из модулей нет в каталоге, выводятся сообщения об ошибках:



Ответы на контрольные вопросы.

1. Необходимо использовать директиву org 100h, так как изначально значения сегментных регистров указывают на PSP.

Выводы.

В результате выполнения лабораторной работы освоена работа с оверлейными модулями, создан модуль, загружающий их в память по одному адресу, реализованы процедуру, обеспечивающие корректную работу этих модулей.

ПРИЛОЖЕНИЕ А. ИСХОДНЫЙ КОД

- имя файла : lab7.asm

```
data segment
	module1 db "module1.ovl", 0
	module2 db "module2.ovl", 0
	child dw 0
	dta_buffer db 43 dup(0)
	mem_allocated db 0
	cline db 128 dup(0)
	ovl_address dd 0
	psp dw 0

	newline db 0dh, 0ah, '$'
	mcb_error db 'Повреждён блок управления памятью', 0dh, 0ah, '$'
	little_memory db 'Недостаточно памяти для выполнения функции', 0dh, 0ah, '$'
	invalid_address db 'Некорректный адрес памяти', 0dh, 0ah, '$'
	free_msg db 'Память успешно очищена' , 0dh, 0ah, '$'

	executed_msg db  'Программа успешно загружена' , 0dh, 0ah, '$'
	alloc_success_msg db  'Место в памяти найдено' , 0dh, 0ah, '$'

	func_missing db 'Несуществующая функция', 0dh, 0ah, '$'
	file_not_found db 'Ошибка загрузки: файл не найден', 0dh, 0ah, '$'
	route_missing db 'Ошибка загрузки: путь не найден', 0dh, 0ah, '$'
	files_err db 'Слишком много открытых файлов', 0dh, 0ah, '$'
	inaccessible db 'Нет доступа', 0dh, 0ah, '$'
	not_enough db 'Недостаточно памяти', 0dh, 0ah, '$'
	env_failure db 'Неправильная среда', 0dh, 0ah, '$'
	alloc_file_not_found db  'Ошибка выделения памяти: файл не найден' , 0dh, 0ah, '$'
	alloc_route_missing db  'Ошибка выделения памяти: путь не найден' , 0dh, 0ah, '$'
```

```
      data_end db 0
data ends

Astack segment stack
      dw 64 dup(?)
Astack ends

code segment

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

print_string proc near
      push ax
      mov ah, 09h
      int 21h
      pop ax
      retn
print_string endp

free proc near
      push ax
      push bx
      push cx
      push dx

      mov ax, offset data_end
      mov bx, offset code_end
      add bx, ax

      mov cl, 4
      shr bx, cl
      add bx, 2bh
      mov ah, 4ah
      int 21h
      jnc free_success
```

```
        free_err1:
                cmp ax, 7
                jne free_err2
                mov dx, offset mcb_error
                call print_string
                jmp free_exit
        free_err2:
                cmp ax, 8
                jne free_err3
                mov dx, offset little_memory
                call print_string
                jmp free_exit
        free_err3:
                cmp ax, 9
                mov dx, offset invalid_address
                call print_string
                jmp free_exit
        free_success:
                mov mem_allocated, 1
                mov dx, offset free_msg
                call print_string

        free_exit:
                pop dx
                pop cx
                pop bx
                pop ax
                retn
free endp

run proc near
        push ax
        push bx
        push cx
```

```
push dx
push ds
push es

mov ax, data
mov es, ax
mov bx, offset ovl_address
mov dx, offset cline
mov ax, 4b03h
int 21h

jnc run_success

run_err1:
      cmp ax, 1
      jne run_err2
      mov dx, offset newline
      call print_string
      mov dx, offset func_missing
      call print_string
      jmp run_exit
run_err2:
      cmp ax, 2
      jne run_err3
      mov dx, offset file_not_found
      call print_string
      jmp run_exit
run_err3:
      cmp ax, 3
      jne run_err4
      mov dx, offset newline
      call print_string
      mov dx, offset route_missing
      call print_string
      jmp run_exit
```

```
run_err4:
      cmp ax, 4
      jne run_err5
      mov dx, offset files_err
      call print_string
      jmp run_exit
run_err5:
      cmp ax, 5
      jne run_err6
      mov dx, offset inaccessible
      call print_string
      jmp run_exit
run_err6:
      cmp ax, 8
      jne run_err7
      mov dx, offset not_enough
      call print_string
      jmp run_exit
run_err7:
      cmp ax, 10
      mov dx, offset env_failure
      call print_string
      jmp run_exit

run_success:
      mov dx, offset executed_msg
      call print_string

      mov ax, word ptr ovl_address
      mov es, ax
      mov word ptr ovl_address, 0
      mov word ptr ovl_address + 2, ax

      call ovl_address
      mov es, ax
```

```
            mov ah, 49h
            int 21h

      run_exit:
            pop es
            pop ds
            pop dx
            pop cx
            pop bx
            pop ax
            retn
run endp

find_path proc near
      push ax
      push bx
      push cx
      push dx
      push di
      push si
      push es

      mov child, dx

      mov ax, psp
      mov es, ax
      mov es, es:[2ch]
      mov bx, 0

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

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

```
        jne seek_path

        add bx, 2
        mov di, 0

write_dir:
        mov dl, es:[bx]
        mov byte ptr [cline+di], dl
        inc di
        inc bx
        cmp dl, 0
        je quit
        cmp dl, '\'
        jne write_dir
        mov cx, di
        jmp write_dir
quit:
        mov di, cx
        mov si, child

child_append:
        mov dl, byte ptr [si]
        mov byte ptr [cline+di], dl
        inc di
        inc si
        cmp dl, 0
        jne child_append


pop es
pop si
pop di
pop dx
pop cx
pop bx
```

```
        pop ax
        retn
find_path endp

allocate proc near
        push ax
        push bx
        push cx
        push dx

        push dx
        mov dx, offset dta_buffer
        mov ah, 1ah
        int 21h
        pop dx
        mov cx, 0
        mov ah, 4eh
        int 21h

        jnc alloc_success

        alloc_err1:
                cmp ax, 2
                je alloc_err2
                mov dx, offset alloc_file_not_found
                call print_string
                jmp alloc_exit
        alloc_err2:
                cmp ax, 3
                mov dx, offset alloc_route_missing
                call print_string
                jmp alloc_exit

        alloc_success:
                push di
```

```
            mov di, offset dta_buffer
            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 ovl_address, ax
            mov dx, offset alloc_success_msg
            call print_string

      alloc_exit:
            pop dx
            pop cx
            pop bx
            pop ax
            retn
allocate endp

ovl_start proc near
      push dx
      call find_path
      mov dx, offset cline
      call allocate
      call run
      pop dx
      retn
ovl_start endp
```

```
main proc far
	push ds
	xor ax, ax
	push ax
	mov ax, data
	mov ds, ax
	mov psp, es

	call free
	cmp mem_allocated, 0
	je exit
	mov dx, offset module1
	call ovl_start
	mov dx, offset newline
	call print_string
	mov dx, offset module2
	call ovl_start
	exit:
		xor al, al
		mov ah, 4ch
		int 21h

main endp

code_end:
code ends
end main
```

- имя файла : module1.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
	mov di, offset ovl
	add di, 23
	call wrd_to_hex
	mov dx, offset ovl
	call print

	pop di
	pop ds
	pop dx
	pop ax
	retf
main endp

ovl db 13, 10, "Адрес module1.ovl:          ", 13, 10, '$'

print proc
	push dx
	push ax

	mov ah, 09h
	int 21h

	pop ax
	pop dx
	ret
print endp


tetr_to_hex proc
	and al,0fh
```

```
      cmp al,09
      jbe next
      add al,07
next:
      add al,30h
      ret
tetr_to_hex endp


byte_to_hex proc
      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
      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

code ends
end main
```

- имя файла : module2.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
            mov di, offset ovl
            add di, 23
            call wrd_to_hex
            mov dx, offset ovl
            call print

            pop di
            pop ds
            pop dx
            pop ax
            retf
      main endp

      ovl db 13, 10, "Адрес module2.ovl:          ", 13, 10, '$'

      print proc
```

```
        push dx
        push ax

        mov ah, 09h
        int 21h

        pop ax
        pop dx
        ret
print endp


tetr_to_hex proc
        and al,0fh
        cmp al,09
        jbe next
        add al,07
next:
        add al,30h
        ret
tetr_to_hex endp


byte_to_hex proc
        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
            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
code ends
end main
```