

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. 0381

Захаров Ф.С.

Преподаватель

Ефремов М.А.

Санкт-Петербург

2022

Цель работы

Исследовать возможности построения загрузочного модуля оверлейной структуры. Исследовать структуру оверлейного сегмента и способ загрузки и выполнения оверлейных сегментов.

В лабораторной работе рассматривается приложение, состоящее из нескольких модулей, поэтому все модули помещаются в один каталог и вызываются с использованием полного пути.

Порядок выполнения работы

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.

Выполнение работы.

Шаг 1.

Был написан и отлажен программный модуль типа .EXE, который выполняет следующие функции:

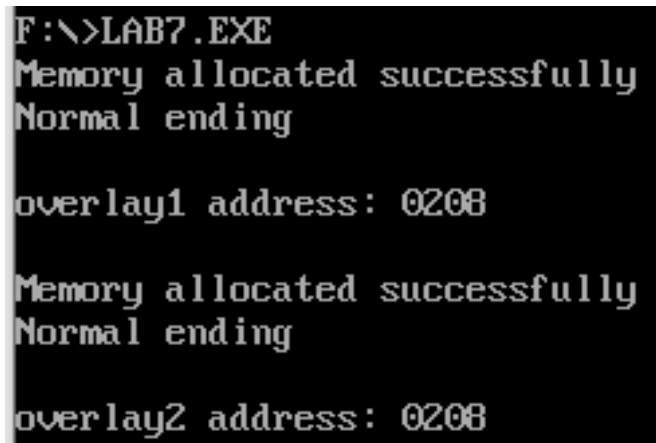
- 1) Освобождает память для загрузки оверлеев.
- 2) Читает размер файла оверлея и запрашивает объем памяти, достаточный для его загрузки.
- 3) Файл оверлейного сегмента загружается и выполняется.
- 4) Освобождается память, отведенная для оверлейного сегмента.
- 5) Затем действия 1)-4) выполняются для следующего оверлейного сегмента.

Шаг 2.

Были написаны и отлажены оверлейные сегменты.

Шаг 3.

Было запущено отлаженное приложение. Оверлейные сегменты загружаются с одного и того же адреса, не перекрывая друг друга.



```
F:\>LAB7.EXE
Memory allocated successfully
Normal ending

overlay1 address: 0208

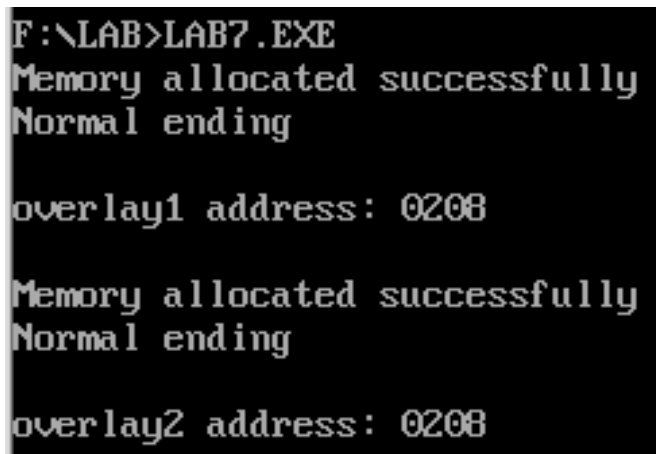
Memory allocated successfully
Normal ending

overlay2 address: 0208
```

Рисунок 1 - результат работы приложения на шаге 3

Шаг 4.

Приложение было запущено из другого каталога. Выполнено успешно.



```
F:\LAB>LAB7.EXE
Memory allocated successfully
Normal ending

overlay1 address: 0208

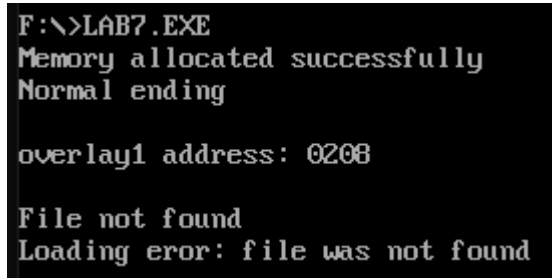
Memory allocated successfully
Normal ending

overlay2 address: 0208
```

Рисунок 2 - результат работы приложения на шаге 4

Шаг 5.

Приложение было запущено, когда одного оверлея нет в каталоге.



```
F:\>LAB7.EXE
Memory allocated successfully
Normal ending

overlay1 address: 0208

File not found
Loading error: file was not found
```

Рисунок 3 - результат работы приложения на шаге 5

Контрольные вопросы.

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

В таком случае, при обращении к оверлейному сегменту необходимо учитывать смещение 100h, потому что в .COM модуле присутствует PSP.

Вывод.

Были исследованы возможности построения загрузочного модуля оверлейной структуры. Была исследована структура оверлейного сегмента и способ загрузки и выполнения оверлейных сегментов.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММ

lab7.asm

```
AStack SEGMENT  STACK
```

```
    DW 128 DUP(?)
```

```
AStack ENDS
```

```
DATA SEGMENT
```

```
    FILE1 DB 'OVERLAY1.OVL', 0
```

```
    FILE2 DB 'OVERLAY2.OVL', 0
```

```
    PROG DW 0
```

```
    DATA_MEMORY DB 43 dup(0)
```

```
    POS_CL DB 128 dup(0)
```

```
    address DD 0
```

```
    KEEP_PSP DW 0
```

```
    NEW_STRING DB 13, 10, '$'
```

```
    MEMORY_7 DB 'Memory error: destroyed memory block', 13, 10, '$'
```

```
    MEMORY_8 DB 'Memory error: not enough memory for running function',  
13, 10, '$'
```

```
    MEMORY_9 DB 'Memory error: incorrect memory address', 13, 10, '$'
```

```
    ERROR_1_STR DB 'Loading error: wrong function number', 13, 10, '$'
```

```
    ERROR_2_STR DB 'Loading error: file was not found', 13, 10, '$'
```

```
    ERROR_5_STR DB 'Loading error: disk error', 13, 10, '$'
```

```
    ERROR_8_STR DB 'Loading error: disk has not enough free memory space',  
13, 10, '$'
```

```
    ERROR_10_STR DB 'Loading error: wrong string enviroment', 13, 10, '$'
```

```
    ERROR_11_STR DB 'Loading error: incorrect format', 13, 10, '$'
```

```
    END_0 DB 'Normal ending', 13, 10, '$'
```

```
    END_1 DB 'Ending by ctrl-break', 13, 10, '$'
```

```
    END_2 DB 'Ending by device error', 13, 10, '$'
```

```
    END_3 DB 'Ending by 3lh function', 13, 10, '$'
```

```
    ALLOCATE_SUCCESS_STR DB 'Memory allocated successfully', 13, 10, '$'
```

```
    FILE_ERROR_STR DB 'File not found', 13, 10, '$'
```

```
    ROUTE_ERROR_STR DB 'Route not found', 13, 10, '$'
```

```
    END_DATA DB 0
```

```
DATA ENDS
```

```
CODE SEGMENT
    ASSUME CS:CODE,DS:DATA,SS:AStack
```

```
PRINT_STRING PROC
```

```
    push ax
    mov ah, 09h
    int 21h
    pop ax
    ret
```

```
PRINT_STRING ENDP
```

```
FREE_MEMORY PROC
```

```
    push ax
    push bx
    push cx
    push dx
    mov ax, offset END_DATA
    mov bx, offset END_PROG
    add bx, ax
    shr bx, 1
    shr bx, 1
    shr bx, 1
    shr bx, 1
    add bx, 2bh
    mov ah, 4ah
    int 21h
```

```
    jnc end_free_memory
```

```
    lea dx, MEMORY_7
    cmp ax, 7
    je print
    lea dx, MEMORY_8
    cmp ax, 8
    je print
    lea dx, MEMORY_9
    cmp ax, 9
```

```
        je print
        jmp end_free_memory
```

print:

```
    mov ah, 09h
    int 21h
```

end_free_memory:

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

FREE_MEMORY ENDP

SET_FULL_NAME PROC NEAR

```
    push ax
    push bx
    push cx
    push dx
    push di
    push si
    push es
    mov PROG, dx
    mov ax, KEEP_PSP
    mov es, ax
    mov es, es:[2ch]
    mov bx, 0
```

find:

```
    inc bx
    cmp byte ptr es:[bx-1], 0
    jne find
    cmp byte ptr es:[bx+1], 0
    jne find
    add bx, 2
```

```

        mov di, 0

find_loop:
        mov dl, es:[bx]
        mov byte ptr [POS_CL + di], dl
        inc di
        inc bx
        cmp dl, 0
        je end_loop
        cmp dl, '\\'
        jne find_loop
        mov cx, di
        jmp find_loop

end_loop:
        mov di, cx
        mov si, PROG

find_loop_2:
        mov dl, byte ptr[si]
        mov byte ptr [POS_CL + di], dl
        inc di
        inc si
        cmp dl, 0
        jne find_loop_2
        pop es
        pop si
        pop di
        pop dx
        pop cx
        pop bx
        pop ax
        ret

SET_FULL_NAME ENDP

ANOTHER_PROG PROC NEAR

```



```

push ax
push bx
push cx
push dx
push ds
push es
mov ax, DATA
mov es, ax
mov bx, offset address
mov dx, offset POS_CL
mov ax, 4b03h
int 21h
jnc transition

```

```

error_1:
    cmp ax, 1
    jne error_2
    mov dx, offset ERROR_1_STR
    call PRINT_STRING
    jmp another_prog_end

```

```

error_2:
    cmp ax, 2
    jne error_5
    mov dx, offset ERROR_2_STR
    call PRINT_STRING
    jmp another_prog_end

```

```

error_5:
    cmp ax, 5
    jne error_8
    mov dx, offset ERROR_5_STR
    call PRINT_STRING
    jmp another_prog_end

```

```

error_8:
    cmp ax, 8
    jne error_10

```

```

        mov dx, offset ERROR_8_STR
        call PRINT_STRING
        jmp another_prog_end

error_10:
        cmp ax, 10
        jne error_11
        mov dx, offset ERROR_10_STR
        call PRINT_STRING
        jmp another_prog_end

error_11:
        cmp ax, 11
        mov dx, offset ERROR_11_STR
        call PRINT_STRING
        jmp another_prog_end

transition:
        mov dx, offset END_0
        call PRINT_STRING
        mov ax, word ptr address
        mov es, ax
        mov word ptr address, 0
        mov word ptr address + 2, ax
        call address
        mov es, ax
        mov ah, 49h
        int 21h

another_prog_end:
        pop es
        pop ds
        pop dx
        pop cx
        pop bx
        pop ax
        ret

```

```
ANOTHER_PROG ENDP
```

```
ALLOCATE_MEMORY PROC
```

```
    push ax
    push bx
    push cx
    push dx
    push dx
    mov dx, offset DATA_MEMORY
    mov ah, 1ah
    int 21h
    pop dx
    mov cx, 0
    mov ah, 4eh
    int 21h
    jnc allocate_success
    cmp ax, 2
    je route_error
    mov dx, offset FILE_ERROR_STR
    call PRINT_STRING
    jmp allocate_end
```

```
route_error:
```

```
    cmp ax, 3
    mov dx, offset ROUTE_ERROR_STR
    call PRINT_STRING
    jmp allocate_end
```

```
allocate_success:
```

```
    push di
    mov di, offset DATA_MEMORY
    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 address, ax
        mov dx, offset ALLOCATE_SUCCESS_STR
        call PRINT_STRING

allocate_end:
        pop dx
        pop cx
        pop bx
        pop ax
        ret

ALLOCATE_MEMORY ENDP

START_OVERLAY PROC
        push dx
        call SET_FULL_NAME
        mov dx, offset POS_CL
        call ALLOCATE_MEMORY
        call ANOTHER_PROG
        pop dx
        ret
START_OVERLAY ENDP

MAIN PROC FAR
        push ds
        xor ax, ax
        push ax
        mov ax, DATA

```

```

    mov ds, ax
    mov KEEP_PSP, es
    call FREE_MEMORY
    mov dx, offset FILE1
    call START_OVERLAY
    mov dx, offset NEW_STRING
    call PRINT_STRING
    mov dx, offset FILE2
    call START_OVERLAY

end_:
    xor al, al
    mov ah, 4ch
    int 21h

```

Main ENDP

END_PROG:

CODE ENDS

END MAIN

overlay1.asm

OVERLAY SEGMENT

ASSUME CS:OVERLAY, DS:NOTHING, SS:NOTHING

MAIN PROC FAR

```

    push ax
    push dx
    push ds
    push di
    mov ax, cs
    mov ds, ax
    mov di, offset message_add
    add di, 23
    call WRD_TO_HEX
    mov dx, offset message_add
    call PRINT_STRING
    pop di

```

```

        pop ds
        pop dx
        pop ax
        retf
MAIN endp

```

```

message_add db 13, 10, "overlay1 address:      ", 13, 10, '$'

```

```

PRINT_STRING PROC
        push dx
        push ax
        mov ah, 09h
        int 21h
        pop ax
        pop dx
        ret
PRINT_STRING 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

```

```

OVERLAY ENDS
END MAIN

```

overlay2.asm

```

OVERLAY SEGMENT
        ASSUME CS:OVERLAY, DS:NOTHING, SS:NOTHING

MAIN PROC FAR
        push ax
        push dx
        push ds
        push di

```

```

    mov ax, cs
    mov ds, ax
    mov di, offset message_add
    add di, 23
    call WRD_TO_HEX
    mov dx, offset message_add
    call PRINT_STRING
    pop di
    pop ds
    pop dx
    pop ax
    retf
MAIN endp

```

```

message_add db 13, 10, "overlay2 address:      ", 13, 10, '$'

```

```

PRINT_STRING PROC
    push dx
    push ax
    mov ah, 09h
    int 21h
    pop ax
    pop dx
    ret
PRINT_STRING 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

```

```

OVERLAY ENDS
END MAIN

```