

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. 9383

Соседков К.С.

Преподаватель

Ефремов М.А.

Санкт-Петербург

2021

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

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.

Выполнение работы.

При выполнении работы 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.

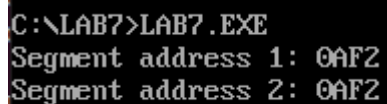
Также были написаны и отлажены оверлейные сегменты. Результат работы программы представлен на Рисунке 1.



```
C:\>LAB7.EXE
Segment address 1: 0AF2
Segment address 2: 0AF2
```

Рисунок 1: Результат работы программы

Результат запуска программы из другого каталога представлен на Рисунке 2.



```
C:\>LAB7>LAB7.EXE
Segment address 1: 0AF2
Segment address 2: 0AF2
```

Рисунок 2: Результат работы программы в другом каталоге

Ниже, на Рисунках 3,4 и 5 представлены результаты работы программы когда одного оверлея, или всех, нет в каталоге.

```
C:\LAB7>LAB7.EXE
Error 2: File not found
Segment address 2: 0AF2
```

Рисунок 3: Без первого оверлея

```
C:\LAB7>LAB7.EXE
Segment address 1: 0AF2
Error 2: File not found
```

Рисунок 4: Без второго оверлея

```
C:\LAB7>LAB7.EXE
Error 2: File not found
Error 2: File not found
```

Рисунок 5: Оверлеев нет в каталоге

Контрольные вопросы.

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

При обращении к оверлейному сегменту необходимо учитывать смещение на 100h, так как в COM модуле есть PSP.

Выводы.

При выполнении лабораторной работы были изучены возможности построения загрузочного модуля оверлейной структуры

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММ

Название файла: lab7.asm

ASTACK segment stack

dw 256 dup(?)

ASTACK ends

DATA segment

DTA db 43 dup (0), '\$'

file_path dw 50 dup(0)

current_ovl dw 0

ovl1 db 'OVL1.OVL', 0

ovl2 db 'OVL2.OVL', 0

ovl_address dd 0

KEEP_PSP DW 0

ovl_error_1 db 'Error 1: Non-existent function ',0dh,0ah,'\$'

ovl_error_2 db 'Error 2: File not found ',0dh,0ah,'\$'

ovl_error_3 db 'Error 3: Route not found ',0dh,0ah,'\$'

ovl_error_4 db 'Error 4: Too many open files ',0dh,0ah,'\$'

ovl_error_5 db 'Error 5: No access',0dh,0ah,'\$'

ovl_error_8 db 'Error 8: Not enough memory',0dh,0ah,'\$'

EXEC_Parameter_Block dw 0

db 0

db 0

db 0

free_memory_error_msg db 'Free memory error',0dh,0ah,'\$'

DATA ends

CODE segment

assume cs:CODE, ds:DATA, ss:ASTACK

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

```

        pop    cx
        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
    xor     AH,AH
    call    BYTE_TO_HEX
    mov     [DI],AH
    dec     DI
    mov     [DI],AL
    pop     BX
    ret
WRD_TO_HEX      ENDP

```

```

OVL_SIZE PROC NEAR
    push    ax
    push    bx

```

push cx

push dx

push dx

mov dx, offset DTA

mov ah, 1ah

int 21h

pop dx

mov cx, 0

mov ah, 4Eh

int 21h

push di

mov di, offset DTA

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

pop dx


```
    pop cx
    pop bx
    pop ax
    ret
OVL_SIZE ENDP
```

```
PRINT proc near
    push ax
    mov ah, 09h
    int 21h
    pop ax
    ret
PRINT endp
```

```
FREE_MEMORY_ERROR proc near
    push dx
    mov dx, offset free_memory_error_msg
    call PRINT
    pop dx
    ret
FREE_MEMORY_ERROR endp
```

```

FREE_MEMORY proc far
    mov bx, offset _end
    mov ax, es
    sub bx, ax
    add bx, 950h
    mov ah, 4Ah
    int 21h
    jc error
    jmp fm_exit
error:
    call FREE_MEMORY_ERROR
fm_exit:
    ret
FREE_MEMORY endp

```

```

READ_PATH PROC
    push ax
    push bx
    push cx
    push dx

```

```

    push di
    push si
    push es

    mov current_ovl, dx

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

find_path_loop:
    cmp byte ptr es:[bx],00h
    jne continue
    cmp byte ptr es:[bx+1],00h
    jne continue
    jmp find_path_loop_end
continue:
    inc bx
    jmp find_path_loop

find_path_loop_end:
    add bx, 4
    xor si, si
    lea di, file_path

read_path_loop:
    mov dl, es:[bx+si]
    cmp byte ptr es:[bx+si],0
    je copy_program_name
    mov [di], dl
    inc di
    inc si
    jmp read_path_loop

copy_program_name:

```

```
sub di, 8
mov cx, 8
mov si, current_ovl
copy_loop:
    mov al, [si]
    mov [di], al
    inc si
    inc di
loop copy_loop
mov [di], byte ptr '$'
```

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

READ_PATH ENDP

RUN_OVL 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 file_path
mov ax, 4b03h
int 21h

jnc run_ovl_ok
jmp run_ovl_error

```

```

run_ovl_ok:
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
jmp run_ovl_exit

```

```

run_ovl_error:
cmp ax, 1
je run_ovl_error_1
cmp ax, 2
je run_ovl_error_2
cmp ax, 3
je run_ovl_error_3
cmp ax, 4

```

```

        je run_ovl_error_4
        cmp ax, 5
        je run_ovl_error_5
        cmp ax, 8
        je run_ovl_error_8

run_ovl_error_1:
        mov dx, offset ovl_error_1
        jmp run_ovl_print_error
run_ovl_error_2:
        mov dx, offset ovl_error_2
        jmp run_ovl_print_error
run_ovl_error_3:
        mov dx, offset ovl_error_3
        jmp run_ovl_print_error
run_ovl_error_4:
        mov dx, offset ovl_error_4
        jmp run_ovl_print_error
run_ovl_error_5:
        mov dx, offset ovl_error_5
        jmp run_ovl_print_error
run_ovl_error_8:
        mov dx, offset ovl_error_8
        jmp run_ovl_print_error

run_ovl_print_error:
        call PRINT

run_ovl_exit:
        pop es
        pop ds
        pop dx
        pop cx
        pop bx

```

```
        pop ax
        ret
RUN_OVL ENDP
```

```
MAIN proc far
        mov ax, DATA
        mov ds, ax
        mov KEEP_PSP, es
```

```
        call FREE_MEMORY
```

```
        mov dx, offset ovl1
        call READ_PATH
        call OVL_SIZE
        call RUN_OVL
```

```
        mov dx, offset ovl2
        call READ_PATH
        call OVL_SIZE
        call RUN_OVL
```

```
exit:
        xor al,al
        mov ah,4ch
        int 21h
```

```
MAIN endp
```

```
_end:
```

```
CODE ends
```

```
end main
```