

МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МОЭВМ

ОТЧЕТ
по практической работе № 7
по дисциплине «Операционные системы»
Тема: Построение модуля оверлейной структуры

Студент гр. 8383

Степанов В.Д.

Преподаватель

Губкин А.Ф.

Санкт-Петербург

2020

Цель работы.

Исследование возможности построения загрузочного модуля оверлейной структуры.

Выполнение работы.

1. Был написан программный модуль типа .EXE, который выполняет следующий функции:

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

Данные функции выполняются для двух оверлейных сегментов OVER1 и OVER2.

2. Были написаны оверлейные сегменты, которые выводят адрес сегмента, в который он загружен.

3. Был запущен программный модуль. Результат запуска представлен на рисунке 1.



```
DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip 0, Program: DOSBOX
C:\>lab7
Memory was freed
OVER1: 04C5
OVER2: 04C5
C:\>
```

Рисунок 1 – Запуск программы

4. Модуль и оверлеи были перемещены в каталог “dir”, после программный модуль был запущен. Результат запуска представлен на рисунке 2.



```
DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip 0, Program: DOSBOX  
C:\>DIR>lab7  
Memory was freed  
OVER1: 04C5  
OVER2: 04C5  
C:\>DIR>
```

Рисунок 2 – Запуск программы из другого каталога

5. Из каталога “dir” был удален второй оверлейный сегмент “OVER2”, после программный модуль был запущен. Результат запуска представлен на рисунке 3.



```
DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip 0, Program: DOSBOX  
C:\>DIR>lab7  
Memory was freed  
OVER1: 04C5  
Error get overlay size  
Overlay was not loaded  
file not founded  
C:\>DIR>_
```

Рисунок 3 – Запуск программы без одного оверлейного сегмента

Контрольные вопросы.

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

Так как в первые 256 байт этого сегмента записывается PSP, то нужно обращаться к оверлейному сегменту со смещением 100h (256 байт).

Выводы.

В ходе лабораторной работы была исследована возможность построения загрузочного модуля оверлейной структуры.

ПРИЛОЖЕНИЕ

КОД ПРОГРАММЫ

```
DATA SEGMENT
    PSP_SEG dw 0
    FREE_FLAGG db 0
    SAVE_SS dw 0
    SAVE_SP dw 0

    OVER1 db "OVER1.OVL", 0
    OVER2 db "OVER2.OVL", 0

    ERROR_FREE_MEM db 13, 10, "Memory was not freed$", 13, 10
    ERRIR_FREE_MEM_7 db 13, 10, "The control memory block was
destroyed$"
    ERRIR_FREE_MEM_8 db 13, 10, "Not enough memory to execute$"
    ERRIR_FREE_MEM_9 db 13, 10, "Invalid address of the memory
block$"
    SUCCES_FREED db 13, 10, "Memory was freed$"

    ERROR_SIZE db 13, 10, "Error get overlay size$"
    ERROR_SIZE_FILE db 13, 10, "File not founded$"
    ERROR_SIZE_PATH db 13, 10, "Path not founded$"

    ERROR_LOAD db 13, 10, "Overlay was not loaded$"
    ERROR_LOAD_1 db 13, 10, "non-existent function$"
    ERROR_LOAD_2 db 13, 10, "file not founded$"
    ERROR_LOAD_3 db 13, 10, "route not found$"
    ERROR_LOAD_4 db 13, 10, "route not found$"
    ERROR_LOAD_5 db 13, 10, "too many open files$"
    ERROR_LOAD_8 db 13, 10, "no access$"
    ERROR_LOAD_10 db 13, 10, "wrong environment$"

    OFFSET_OVER dw 0
    PROG_PATH db 100h dup(0)
    POS_OF_LINE dw 0
    BUFF db 43 dup(0)
    SEG_OVER dw 0
    ADRES_OVER dd 0
    END_DATA db 0
DATA ENDS

LAB6STACK SEGMENT STACK
```

```

    dw 100h dup(0)
LAB6STACK ENDS

CODE SEGMENT
    ASSUME CS:CODE, DS:DATA, SS:LAB6STACK
;-----
MEM_FREE PROC NEAR
    push ax
    push bx
    push cx
    push dx

    mov bx, offset END_PROG
    mov ax, offset END_DATA
    add bx, ax
    add bx, 40Fh

    mov cl, 4
    shr bx, cl

    xor ax, ax
    mov ah, 4Ah
    int 21h

    jnc FREE_MEM_OK
    mov FREE_FLAGG, 0
    mov dx, offset ERROR_FREE_MEM
    call PRINT

    cmp ax, 7
    je ERR_7
    cmp ax, 8
    je ERR_8
    cmp ax, 9
    je ERR_9

ERR_7:
    mov dx, offset ERRIR_FREE_MEM_7
    jmp END_FREE
ERR_8:
    mov dx, offset ERRIR_FREE_MEM_8
    jmp END_FREE
ERR_9:

```

```

    mov dx, offset ERRIR_FREE_MEM_9
    jmp END_FREE
FREE_MEM_OK:
    mov FREE_FLAGG, 1
    mov dx, offset SUCCES_FREED

END_FREE:
    call PRINT
    pop dx
    pop cx
    pop bx
    pop ax
    ret
MEM_FREE ENDP
;-----

SET_PROG PROC NEAR
    push ax
    push si
    push di
    push es

    mov OFFSET_OVER, ax
    mov ax, PSP_SEG
    mov es, ax
    mov es, es:[2Ch]
    mov si, 0

FIND00:
    mov ax, es:[si]
    inc si
    cmp ax, 0
    jne FIND00
    add si, 3
    mov di, 0

WRITE:
    mov al, es:[si]
    cmp al, 0
    je WRITE_PROG
    cmp al, '\'
    jne ADD_SYM
    mov POS_OF_LINE, di

```

```

ADD_SYM:
    mov BYTE PTR [PROG_PATH + di], AL
    inc di
    inc si
    jmp WRITE

WRITE_PROG:
    cld
    mov di, POS_OF_LINE
    inc di
    add di, offset PROG_PATH
    mov si, OFFSET_OVER
    mov ax, ds
    mov es, ax

REWRITE_NAME_SYMB:
    lodsb
    stosb
    cmp AL, 0
    jne REWRITE_NAME_SYMB

    pop es
    pop di
    pop si
    pop ax
    ret

SET_PROG ENDP
;-----
LOAD_PROG PROC NEAR
    push ax
    push bx
    push dx
    push es

    mov dx, offset PROG_PATH
    push ds
    pop es
    mov bx, offset SEG_OVER
    mov ax, 4B03h
    int 21h

    jnc LOAD_OK
    mov dx, offset ERROR_LOAD

```

```

call PRINT
cmp ax, 1
je L_ERR_1
cmp ax, 2
je L_ERR_2
cmp ax, 3
je L_ERR_3
cmp ax, 4
je L_ERR_4
cmp ax, 5
je L_ERR_5
cmp ax, 8
je L_ERR_8
cmp ax, 10
je L_ERR_10
L_ERR_1:
    mov dx, offset ERROR_LOAD_1
    jmp LOAD_P
L_ERR_2:
    mov dx, offset ERROR_LOAD_2
    jmp LOAD_P
L_ERR_3:
    mov dx, offset ERROR_LOAD_3
    jmp LOAD_P
L_ERR_4:
    mov dx, offset ERROR_LOAD_4
    jmp LOAD_P
L_ERR_5:
    mov dx, offset ERROR_LOAD_5
    jmp LOAD_P
L_ERR_8:
    mov dx, offset ERROR_LOAD_8
    jmp LOAD_P
L_ERR_10:
    mov dx, offset ERROR_LOAD_10
    jmp LOAD_P

LOAD_P:
    call PRINT
    jmp LOAD_END

LOAD_OK:

```



```

mov ax, SEG_OVER
mov es, ax
mov WORD PTR ADRES_OVER + 2, ax
call ADRES_OVER
mov es, ax
mov ah, 49h
int 21h
LOAD_END:

pop es
pop dx
pop bx
pop ax
ret
LOAD_PROG ENDP
;-----

SIZE_P PROC
push ax
push bx
push cx
push dx
push di

xor ax, ax
mov ah, 1Ah
mov dx, offset BUFF
int 21h

mov ah, 4Eh
mov cx, 0
mov dx, offset PROG_PATH
int 21h

jnc SIZE_OK
mov dx, offset ERROR_SIZE
call PRINT
cmp ax, 2
je ERR_2
cmp ax, 3
je ERR_3
jmp SIZE_END

```

```

ERR_2:
    mov dx, offset ERROR_SIZE_FILE
    call PRINT
    jmp SIZE_END

```

```

ERR_3:
    mov dx, offset ERROR_SIZE_PATH
    call PRINT
    jmp SIZE_END

```

```

SIZE_OK:
    mov si, offset BUFF
    add si, 1Ah
    mov bx, [si]
    mov ax, [si+2]
    shr bx, 4
    shl ax, 12
    add bx, ax
    add bx, 2
    xor ax, ax
    mov ah, 48h
    int 21h

```

```

    jnc SAVE_SEG
    mov dx, offset ERROR_FREE_MEM
    call PRINT
    jmp SIZE_END

```

```

SAVE_SEG:
    mov SEG_OVER, ax

```

```

SIZE_END:
    pop si
    pop dx
    pop cx
    pop bx
    pop ax
    ret

```

```

SIZE_P ENDP

```

```

;-----
PRINT PROC near

    push ax

```

```

    sub ax, ax
    mov ah, 9h
    int 21h
    pop ax

    ret
PRINT ENDP
;-----
MAIN:

    mov bx, ds
    mov ax, DATA
    mov ds, ax
    mov PSP_SEG, bx

    call MEM_FREE

    cmp FREE_FLAGG, 1
    jne END_MAIN

    mov ax, offset OVER1
    call SET_PROG
    call SIZE_P
    call LOAD_PROG

    mov ax, offset OVER2
    call SET_PROG
    call SIZE_P
    call LOAD_PROG

END_MAIN:
    xor ax, ax
    mov ah, 4Ch
    int 21h
END_PROG:
CODE ENDS
END MAIN

```