

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.0382

Андрющенко К.С.

Преподаватель

Ефремов М.А.

Санкт-Петербург

2022

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 оверлейные модули находятся в одном каталоге.

Задание.

1. Для выполнения лабораторной работы необходимо написать и отладить программный модуль типа .EXE, который выполняет функции:

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

2. Также необходимо написать и отладить оверлейные сегменты. Оверлейный сегмент выводит адрес сегмента, в который он загружен.

3. Запустите отлаженное приложение. Оверлейные сегменты должны загружаться с одного и того же адреса, перекрывая друг друга.

4. Запустите приложение из другого каталога. Приложение должно быть выполнено успешно.

5. Запустите приложение в случае, когда одного оверлея нет в каталоге. Приложение должно закончиться аварийно.

6. Занесите полученные результаты в виде скриншотов в отчет. Оформите отчет в соответствии с требованиями.

Выполнение работы.

1. Был написан программный модуль типа .EXE, состоящий из трех процедур:

- FREE_MEM – подготавливает место в памяти, необходимое для программы;
- PATH – подготавливает путь и имя вызываемого оверлейного модуля.
- LOAD – загружает вызываемый оверлейный модуль.
- ALLOC_MEM – определение и отведение памяти для оверлейного модуля

2. Результаты выполнения шагов задания представлены на рисунках 1-5:

```
C:\>lb7
FREE: Success free memory

ALLOC: ok
OVL1 address: 0205
LOAD: ok

ALLOC: ok
OVL2 address: 0205
LOAD: ok
```

Рисунок 1 – Результат запуска приложения из того же каталога, где само приложение

```
C:\>os\lb7
FREE: Success free memory

ALLOC: ok
OVL1 address: 0205
LOAD: ok

ALLOC: ok
OVL2 address: 0205
LOAD: ok
```

Рисунок 2 – Результат запуска приложения из каталога, отличного от того, где само приложение

```
C:\>lb7
FREE: Success free memory

ALLOC: Path not found
LOAD: File not found

ALLOC: ok
OVL2 address: 0205
LOAD: ok
```

Рисунок 3 – Результат запуска приложения, когда одного оверлейного сегмента нет

Исходный код программы см. в приложении А.

Ответы на вопросы.

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

Чтобы использовать в качестве оверлейного сегмента .COM модуль, нужно учитывать смещение 100h, т.к. в начале .COM модуля присутствует PSP.

Выводы.

В ходе работы были исследованы возможности построения загрузочного модуля оверлейной структуры и структура оверлейного сегмента, а также способ их загрузки и выполнения.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: lb7.asm

```
AStack SEGMENT STACK
```

```
    DW 32 DUP(?)
```

```
AStack ENDS
```

```
DATA SEGMENT
```

```
    FILE_NAME db 'OVL1.OVL', 0
```

```
    FILE_NAME2 db 'OVL2.OVL', 0
```

```
    CMD_L db 1h, 0dh
```

```
    FILE_PATH db 128 DUP (?)
```

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

```
    NUM_OVL db 0
```

```
    FREE_MEM_1 db 'FREE: The control memory block is  
destroyed', 0DH, 0AH,'$'
```

```
    FREE_MEM_2 db 'FREE: Not enough memory to execute the  
function', 0DH, 0AH,'$'
```

```
    FREE_MEM_3 db 'FREE: Invalid memory block address', 0DH,  
0AH,'$'
```

```
    FREE_MEM_4 db 'FREE: Success free memory', 0DH, 0AH,'$'
```

```
    FREE_MEM_FLAG db 0
```

```
    LOAD_1 db 'LOAD: Function doesnt exist', 0DH, 0AH,'$'
```

```
    LOAD_2 db 'LOAD: File not found', 0DH, 0AH,'$'
```

```
    LOAD_3 db 'LOAD: Path not found', 0DH, 0AH,'$'
```

```
    LOAD_4 db 'LOAD: Too many open files', 0DH, 0AH,'$'
```

```
    LOAD_5 db 'LOAD: No assecc', 0DH, 0AH,'$'
```

```
    LOAD_6 db 'LOAD: Not enough memory', 0DH, 0AH,'$'
```

```
    LOAD_7 db 'LOAD: Incorrect environment', 0DH, 0AH,'$'
```

```
    GOOD_OVL_LOAD db 'LOAD: ok', 0DH, 0AH,'$'
```

```
    ALLOC_1 db 'ALLOC: File not found', 0DH, 0AH,'$'
```

```
    ALLOC_2 db 'ALLOC: Path not found', 0DH, 0AH,'$'
```

```
    GOOD_ALLOC db 'ALLOC: ok', 0DH, 0AH,'$'
```

```
    DTA db 43 dup(?)
```

```
    OVL_ADDRESS dd 0
```

```
    KEEP_SS dw 0
```

```
    KEEP_SP dw 0
```

```
    KEEP_PSP dw 0
```

```
    END_DATA db 0
```

```
DATA ENDS
```

```
TESTPC SEGMENT
```

```

        ASSUME CS:TESTPC, DS:DATA, SS:AStack

; ПРОЦЕДУРЫ
;-----
PRINT PROC near
    push AX
    mov AH, 09h
    int 21h
    pop AX
    ret
PRINT ENDP
;-----
FREE_MEM PROC near
    push AX
    push BX
    push CX
    push DX

    lea BX, end_programm
    lea AX, END_DATA
    add BX, AX
    mov CL, 4
    shr BX, CL
    add BX, 2Bh
    mov AH, 4Ah
    int 21h

    jnc free_mem_suc

    mov FREE_MEM_FLAG, 0
    cmp AX, 7
    jne low_mem
    lea DX, FREE_MEM_1
    jmp free_mem_print
low_mem:
    cmp AX, 8
    jne inv_addr
    lea DX, FREE_MEM_2
    jmp free_mem_print
inv_addr:
    cmp AX, 9
    lea DX, FREE_MEM_3
    jmp free_mem_print

free_mem_suc:
    mov FREE_MEM_FLAG, 1
    lea DX, FREE_MEM_4

free_mem_print:
    call PRINT

```

```

end_free_mem:
    pop DX
    pop CX
    pop BX
    pop AX
    ret
FREE_MEM ENDP
;-----
PATH PROC near
    push AX
    push BX
    push CX
    push DX
    push DI
    push SI
    push ES

    mov AX, KEEP_PSP
    mov ES, AX
    mov ES, ES:[2Ch]
    mov BX, 0

find_zero:
    inc BX
    cmp byte ptr ES:[BX-1], 0
    jne find_zero

    cmp byte ptr ES:[BX+1], 0
    jne find_zero

    add BX, 2
    mov DI, 0

path_loop:
    mov DL, ES:[BX]
    mov byte ptr [FILE_PATH+DI], DL
    inc DI
    inc BX
    cmp DL, 0
    je path_end_loop
    cmp DL, '\\'
    jne path_loop
    mov CX, DI
    jmp path_loop
path_end_loop:
    mov DI, CX
    mov SI, 0

_file_name:
    cmp NUM_OVL, 0
    jne num2

```

```

        mov DL, byte ptr [FILE_NAME+SI]
        jmp num1
num2:
        mov DL, byte ptr [FILE_NAME2+SI]
num1:
        mov byte ptr [FILE_PATH+DI], DL
        inc DI
        inc SI
        cmp DL, 0
        jne _file_name

        pop ES
        pop SI
        pop DI
        pop DX
        pop CX
        pop BX
        pop AX
        ret
PATH ENDP
;-----
ALLOC_MEM PROC near
        push AX
        push BX
        push CX
        push DX

        lea DX, DTA
        mov AH, 1Ah
        int 21h

        lea DX, FILE_PATH
        mov CX, 0
        mov AH, 4Eh
        int 21h

        jnc alloc_suc

        cmp AX, 3
        jne path_not_found
        lea DX, ALLOC_1
        jmp end_alloc

path_not_found:
        lea DX, ALLOC_2
        jmp end_alloc

alloc_suc:
        lea DI, DTA
        mov DX, [DI+1Ch]
        mov AX, [DI+1Ah]

```



```

        mov BX, 10h
        div BX
        inc AX
        mov BX, AX
        mov AH, 48h
        int 21h

        lea BX, OVL_ADDRESS
        mov CX, 0h
        mov [BX], AX
        mov [BX+2], CX

        lea DX, GOOD_ALLOC

end_alloc:
        call PRINT
        pop DX
        pop CX
        pop BX
        pop AX
        ret
ALLOC_MEM ENDP
;-----
LOAD PROC near
        push AX
        push BX
        push CX
        push DX

        push DS
        push ES
        mov AX, SS
        mov KEEP_SS, AX
        mov KEEP_SP, SP

        mov AX, DATA
        mov ES, AX

        lea BX, OVL_ADDRESS
        lea DX, FILE_PATH
        mov AX, 4B03h
        int 21h

        mov SP, KEEP_SP
        mov BX, KEEP_SS
        mov SS, BX
        pop ES
        pop DS

        jnc load_suc

```

```

        cmp AX, 1
        jne err2
        lea DX, LOAD_1
        jmp end_load
err2:
        cmp AX, 2
        jne err3
        lea DX, LOAD_2
        jmp end_load
err3:
        cmp AX, 3
        jne err4
        lea DX, LOAD_3
        jmp end_load
err4:
        cmp AX, 4
        jne err5
        lea DX, LOAD_4
        jmp end_load
err5:
        cmp AX, 5
        jne err8
        lea DX, LOAD_5
        jmp end_load
err8:
        cmp AX, 8
        jne err10
        lea DX, LOAD_6
        jmp end_load
err10:
        cmp AX, 10
        jne end_load
        lea DX, LOAD_7
        jmp end_load
load_suc:
        lea DX, GOOD_OVL_LOAD

        lea BX, 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:

```

```

        call PRINT
        pop DX
        pop CX
        pop BX
        pop AX
        ret
LOAD ENDP
;-----
; КОД
MAIN PROC far
    mov ax, data
    mov ds, ax
    mov KEEP_PSP, ES

    call FREE_MEM
    cmp FREE_MEM_FLAG, 0
    je main_end
    lea DX, NEWLINE
    call PRINT

    call PATH
    call ALLOC_MEM
    call LOAD
    inc NUM_OVL

    lea DX, NEWLINE
    call PRINT
    call PATH
    call ALLOC_MEM
    call LOAD
; Выход в DOS
main_end:
    xor AL, AL
    mov AH, 4Ch
    int 21h
MAIN ENDP
end_programm:
TESTPC ENDS
END MAIN

```

Название файла: ovl1.asm

```

TESTPC SEGMENT
    ASSUME CS:TESTPC, DS:NOTHING, SS:NOTHING
MAIN PROC far
    push AX
    push DX
    push DS
    push DI

    mov AX, CS

```

```

    mov DS, AX
    lea DI, OVL1_ADDRESS
    add DI, 17
    call WRD_TO_HEX
    lea DX, OVL1_ADDRESS
    call PRINT

    pop DI
    pop DS
    pop DX
    pop AX
    retf
MAIN ENDP

OVL1_ADDRESS db 'OVL1 address:      ', 0DH, 0AH, '$'
; ПРОЦЕДУРЫ
;-----
PRINT PROC near
    push AX
    mov AH, 09h
    int 21h
    pop AX
    ret
PRINT ENDP
;-----
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
; байт в AL переводится в два символа 16-го числа в AX
    push CX
    mov AH, AL
    call TETR_TO_HEX
    xchg AL, AH
    mov CL, 4
    shr AL, CL
    call TETR_TO_HEX ; в AL старшая цифра
    pop CX           ; в AH младшая
    ret
BYTE_TO_HEX ENDP
;-----
WRD_TO_HEX PROC near
; перевод в 16 с/с 16-ти разрядного числа
; в AX - число, в DI - адрес последнего символа
    push BX

```

```

        mov BH, AH
        call BYTE_TO_HEX
        mov [DI], AH
        dec DI
        mov [DI], AL
        dec DI
        mov AL, BH
        call BYTE_TO_HEX
        mov [DI], AH
        dec DI
        mov [DI], AL
        pop BX
        ret
WRD_TO_HEX ENDP
TESTPC ENDS
END MAIN

```

Название файла: ovl2.asm

```

TESTPC SEGMENT
    ASSUME CS:TESTPC, DS:NOTHING, SS:NOTHING
MAIN PROC far
    push AX
    push DX
    push DS
    push DI

    mov AX, CS
    mov DS, AX
    lea DI, OVL2_ADDRESS
    add DI, 17
    call WRD_TO_HEX
    lea DX, OVL2_ADDRESS
    call PRINT

    pop DI
    pop DS
    pop DX
    pop AX
    retf
MAIN ENDP

OVL2_ADDRESS db 'OVL2 address:      ', 0DH, 0AH, '$'
; ПРОЦЕДУРЫ
;-----
PRINT PROC near
    push AX
    mov AH, 09h
    int 21h
    pop AX
    ret

```

```

PRINT ENDP
;-----
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
; байт в AL переводится в два символа 16-го числа в AX
    push CX
    mov AH, AL
    call TETR_TO_HEX
    xchg AL, AH
    mov CL, 4
    shr AL, CL
    call TETR_TO_HEX ; в AL старшая цифра
    pop CX           ; в AH младшая
    ret
BYTE_TO_HEX ENDP
;-----
WRD_TO_HEX PROC near
; перевод в 16 с/с 16-ти разрядного числа
; в AX - число, в DI - адрес последнего символа
    push BX
    mov BH, AH
    call BYTE_TO_HEX
    mov [DI], AH
    dec DI
    mov [DI], AL
    dec DI
    mov AL, BH
    call BYTE_TO_HEX
    mov [DI], AH
    dec DI
    mov [DI], AL
    pop BX
    ret
WRD_TO_HEX ENDP
TESTPC ENDS
END MAIN

```