**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 |  | Литягин С.М. |
| 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:

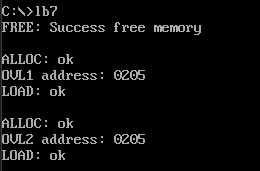


Рисунок 1 – Результат запуска приложения из того же каталога, где само приложение

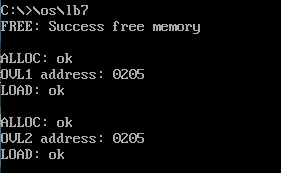


Рисунок 2 – Результат запуска приложения из каталога, отличного от того, где само приложение



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