

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

Санников В.А.

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.

Шаг 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;

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.

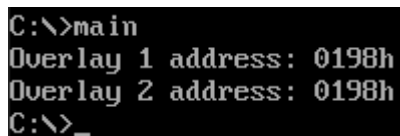
Ход работы.

Для выполнения лабораторной работы был написан .EXE модуль, который содержит следующие процедуры:

- 1)FREE_MEMORY — процедура проверки и очистки памяти.
- 2)LOAD — процедура загрузки файла и проверки на возможные ошибки при загрузке.
- 3)GETPATH — процедура получения пути до вызываемого каталога.
- 4)FILESIZE — процедура чтения размера файла и запроса объема памяти для его загрузки.
- 5)MALLOC — процедура выделения памяти.

Выполнение пунктов:

- 1) Был написан и отлажен программный модуль типа .EXE, который выполняет требуемые по заданию функции.
- 2) Были написаны оверлейные модули overlay1 и overlay2. Они выводят адреса сегментов, куда они загружены
- 3) На данном шаге был запущен модуль .EXE. Результат работы модуля см на рисунке 1.



```
C:\>main
Overlay 1 address: 0198h
Overlay 2 address: 0198h
C:\>_
```

Рисунок 1 — Запуск модуля с двумя оверлеями в каталоге.

- 4) На четвертом шаге модуль был запущен из другого каталога someDir. Результаты работы программы см на рисунке 2.



```
C:\>cd someDir
C:\SOMEDIR>main
Overlay 1 address: 0198h
Overlay 2 address: 0198h
C:\SOMEDIR>_
```

Рисунок 2 — Запуск модуля при другом каталоге.

5) На данном шаге программа запускается без одного оверлейного модуля в каталоге. Результаты работы модуля см на рисунке 3.



```
C:\SOMEDIR>main
Overlay 1 address: 0198h
File doesn't exist!!!
C:\SOMEDIR>
```

Рисунок 3 — Запуск программы, когда в каталоге нет одного оверлейного модуля.

Как видно из данного рисунка, программа выводит ошибку, так как одного из оверлеев нет в каталоге.

Исходный код программы см в приложении А.

Ответы на контрольные 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.

Вывод.

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

Файл main.asm

```
AStack SEGMENT STACK
    DW 128 DUP(?)
AStack ENDS
;-----
DATA SEGMENT
    OVERLAY1_NAME db 'overlay1.ovl', 0
    OVERLAY2_NAME db 'overlay2.ovl', 0
    CURRENT_PATH db 50 dup (0)

    ILLEGAL_FUNCTION_ERROR db "Illegal function!!!$"
    ILLEGAL_FILE_ERROR db "File doesn't exist!!!$"
    ILLEGAL_PATH_ERROR db "Path doesn't exist!!!$"
    OPLIMIT_ERROR db "Too many open files!!!$"
    NO_ACCESS_ERROR db "Access denied!!!$"
    MEMORY_ERROR db "Insufficient memory!!!$"
    ENVIROMENT_ERROR db "Incorrect enviroment!!!$"
    ADDRESS_ERROR db "Wrong block address!!!$"

    OVERLAY_ADDRESS dd 0
    DTA db 43 dup(0)
    MCB_CRASH db "MCB destroyed!!!$"
    END_LINE db 0dh, 0ah, '$'
    EPB dw 2 dup(0)
    ERROR db 0
DATA ENDS
;-----
CODE SEGMENT
    ASSUME CS:CODE, DS:DATA, SS:AStack
;-----
FREE_MEMORY PROC NEAR
    push ax
    push bx
    push cx
    push dx
    mov bx, offset exit_prog
    mov ax, es
    sub bx, ax
    mov cl, 4
    shr bx, cl
    mov ah, 4ah
    int 21h
    jnc free_memory_end
    cmp ax, 7
    je free_memory_7
    cmp ax, 8
    je free_memory_8
```

```

    cmp ax, 9
    je free_memory_9
    jmp free_memory_end

free_memory_7:
    mov dx, offset MCB_CRASH
    jmp free_memory_print

free_memory_8:
    mov dx, offset memory_error
    jmp free_memory_print

free_memory_9:
    mov dx, offset address_error

free_memory_print:
    mov ah, 09h
    int 21h
    mov error, 1

free_memory_end:
    pop dx
    pop cx
    pop bx
    pop ax
    ret
FREE_MEMORY ENDP
;-----
GETPATH PROC NEAR
    push si
    mov si, 0
    mov es, es:[2ch]
    env_cycle:
        mov al, es:[si]
        inc si
        cmp al, 0
        jne env_cycle
        mov al, es:[si]
        cmp al, 0
        jne env_cycle

    add si, 3
    push si

    search_sl:
        cmp byte ptr es:[si], '\'
        jne next_search
    mov ax, si

    next_search:
        inc si
        cmp byte ptr es:[si], 0

```

```

        jne search_sl
inc ax
pop si
mov di,0

dir_path:
    mov bl,es:[si]
    mov CURRENT_PATH[di], bl
    inc si
    inc di
    cmp si,ax
    jne dir_path
pop si

fileName:
    mov bl,[si]
    mov CURRENT_PATH[di], bl
    inc si
    inc di
    cmp bl, 0
    jne fileName

ret
GETPATH ENDP
;-----
FILESIZE PROC NEAR
    push cx
    push dx
    mov dx, offset DTA
    mov ah, 1ah
    int 21h

    mov cx, 0
    mov dx, offset CURRENT_PATH
    mov ah, 4eh
    int 21h

    jnc fileSize_correct
    cmp ax, 2
    je fileSize_err2
    cmp ax, 3
    je fileSize_err3

fileSize_err2:
    mov dx,offset ILLEGAL_FILE_ERROR
    jmp fileSize_print

fileSize_err3:
    mov dx, offset ILLEGAL_PATH_ERROR

fileSize_print:
    mov error, 1
    mov ah, 09h

```

```

        int 21h

fileSize_correct:
        mov ax, word ptr dta[1ah]
        mov dx, word ptr dta[1ah+2]
        mov cl, 4
        shr ax, cl
        mov cl, 12
        shl dx, cl
        add ax, dx
        add ax, 1
    pop dx
    pop cx
    ret
FILESIZE ENDP
;-----
MALLOC PROC NEAR
    push bx
    push dx
    mov bx, ax
    mov ah, 48h
    int 21h
    jnc malloc_correct
    mov dx, offset MEMORY_ERROR
    mov ah, 09h
    int 21h
    mov error, 1
    jmp p_end_malloc

    malloc_correct:
        mov epb[0], ax
        mov epb[2], ax

    p_end_malloc:
        pop dx
        pop bx
        ret
MALLOC ENDP
;-----
LOAD PROC NEAR
    push ax
    push es
    push bx
    push dx
    mov dx, offset CURRENT_PATH
    mov ax, ds
    mov es, ax
    mov bx, offset EPB
    mov ax, 4b03h
    int 21h

    jnc load_correct

```



```

cmp ax, 1
je load_err1
cmp ax, 2
je load_err2
cmp ax, 3
je load_err3
cmp ax, 4
je load_err4
cmp ax, 5
je load_err5
cmp ax, 8
je load_err8
cmp ax, 10
je load_err10

load_err1:
    mov dx, offset ILLEGAL_FUNCTION_ERROR
    jmp load_print
load_err2:
    mov dx, offset ILLEGAL_FILE_ERROR
    jmp load_print
load_err3:
    mov dx, offset ILLEGAL_PATH_ERROR
    jmp load_print
load_err4:
    mov dx, offset OPLIMIT_ERROR
    jmp load_print
load_err5:
    mov dx, offset NO_ACCESS_ERROR
    jmp load_print
load_err8:
    mov dx, offset MEMORY_ERROR
    jmp load_print
load_err10:
    mov dx, offset ENVIROMENT_ERROR
load_print:
    mov ah, 09h
    mov error, 1
    int 21
load_correct:
    mov ax,ebp[2]
    mov word ptr OVERLAY_ADDRESS+2, ax
    call OVERLAY_ADDRESS

    mov es, ax
    mov ah, 49h
    int 21h
load_end:
    pop dx
    pop bx
    pop es
    pop ax

```

```

        ret
LOAD ENDP
;-----
MAIN PROC FAR
    mov ax, DATA
    mov ds, ax
    push es
    call FREE_MEMORY
    cmp error, 0
    jne p_exit

    mov si, offset OVERLAY1_NAME
    call GETPATH
    call FILESIZE
    cmp error, 0
    jne p_ovl1
    call MALLOC
    cmp error, 0
    jne p_ovl1
    call LOAD

p_ovl1:
    mov dx, offset END_LINE
    mov ah, 09h
    int 21h
    pop es
    mov error, 0
    mov si, offset OVERLAY2_NAME
    call GETPATH
    call FILESIZE
    cmp error, 0
    jne p_exit
    call MALLOC
    cmp error, 0
    jne p_exit
    call LOAD

p_exit:
    xor al, al
    mov ah, 4ch
    int 21h
MAIN ENDP
exit_prog:
CODE ENDS
        END MAIN

```

Файл overlay1.asm

```

ASSUME CS:CODE
CODE SEGMENT
    MAIN PROC FAR
        push ax

```

```

    push dx
    push ds
    push di

    mov ax, cs
    mov ds, ax
    mov di, offset OVERLAY_ADDR
    add di, 22
    call WRD_TO_HEX
    mov dx, offset OVERLAY_ADDR
    call PRINT

    pop di
    pop ds
    pop dx
    pop ax
    retf
MAIN ENDP

OVERLAY_ADDR db "Overlay 1 address: 0000h$", 0dh, 0ah, '$'

PRINT PROC
    push ax
    mov ah, 9h
    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
    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

CODE ENDS
END MAIN

```

Файл overlay2.asm

```

ASSUME CS:CODE
CODE SEGMENT
    MAIN PROC FAR
        push ax
        push dx
        push ds
        push di

        mov ax, cs
        mov ds, ax
        mov di, offset OVERLAY_ADDR
        add di, 22
        call WRD_TO_HEX
        mov dx, offset OVERLAY_ADDR
        call PRINT

        pop di
        pop ds
        pop dx
        pop ax
        retf
    MAIN ENDP

OVERLAY_ADDR db "Overlay 2 address: 0000h$", 0dh,0ah,'$'

PRINT PROC
    push ax
    mov ah, 9h

```

```

        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
    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

CODE ENDS
END MAIN

```