

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. 9382

Дерюгин Д.А.

Преподаватель

Ефремов М.А.

Санкт-Петербург

2021

Цель работы.

Исследование возможности построения загрузочного модуля оверлейной структуры.

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 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.

Запустим приложение. Оверлейные сегменты находятся в одной папке с приложением.

```
D:\>LAB7.EXE
load was successful
OVERLAY 1 ADDRESS:01EE

load was successful
OVERLAY 2 ADDRESS:01EE
```

Рис. 1 Результат работы программы

Запустим программу с другой директории. Как видно из скриншота ниже, программа работает. Оверлеи загружаются

```
D:\A>LAB7.EXE
load was successful
OVERLAY 1 ADDRESS:01EE

load was successful
OVERLAY 2 ADDRESS:01EE
```

Рис. 2 Запуск программы с другой директории

Теперь запустим программу, когда нет первого оверлея. Как видно на рисунке ниже один оверлей загружен, а другой нет

```
D:\A>LAB7.EXE
Load error code: 2

load was successful
OVERLAY 2 ADDRESS:01EE
```

Рис 3. Результат работы, когда один оверлей отсутствует

Ответы на вопросы.

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

После записи значений в стек нужно положить значения регистра CS в регистр DS(адреса сегментов данных и кода совпадают). При этом оверлейный сегмент надо вызвать по смещению 100h(занимает PSP).

Выводы.

В ходе данной лабораторной работы были получены навыки разработки загрузочных модулей оверлейной структуры.

ПРИЛОЖЕНИЕ А.

ИСХОДНЫЙ КОД ПРОГРАММЫ

```
AStack SEGMENT  STACK
```

```
    DW 100 DUP(?)
```

```
AStack ENDS
```

```
DATA SEGMENT
```

```
    KEEP_PSP dw 0
```

```
    OVL1 db "obl1.ovl", 0
```

```
    OVL2 db "obl2.ovl", 0
```

```
    PROGRAM dw 0
```

```
    DTA db 43 dup(0)
```

```
    MEMORY_ERROR db 0
```

```
    POS db 128 dup(0)
```

```
    OVERLAY dd 0
```

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

```
    BLOCKED_DESTROYED db 'Block destroyed' , 13, 10, '$'
```

```
    LOW_MEMORY db 'Low memory', 13, 10, '$'
```

```
    INVALID_ADRESS_OF_BLOCK db 'Invalid adress of block', 13, 10, '$'
```

```
    FILE_NOT_FOUNT db 'File not found', 13, 10, '$'
```

```
    ROUTE_ERROR db 'Route not found', 0dh, 0ah, '$'
```

```
    LOAD_ERROR db 'Load error code:      ', 0dh, 0ah, '$'
```

```
    SUCCESS_LOAD db 'load was successful', 0dh, 0ah, '$'
```

```
    END_DATA db 0
```

```
DATA ENDS
```

```
CODE SEGMENT
```

```
    ASSUME CS:CODE, DS:DATA, SS:ASTACK
```

```
BYTE_TO_DEC PROC near
```

```
;Перевод в 10чную с/с, SI - адрес младшей цифры
```

```
    push CX
```

```
    push DX
```

```
    xor AH,AH
```

```
    xor DX,DX
```

```
    mov CX,10
```

```
loop_bd:    div CX
```

```
    or DL,30h
```

```
    mov [SI],DL
```

```

                                dec SI
                                xor DX,DX
                                cmp AX,10
                                jae loop_bd
                                cmp AL,00h
                                je end_1
                                or AL,30h
                                mov [SI],AL
end_1:                          pop DX
                                pop CX
                                ret

```

BYTE_TO_DEC ENDP

```

print PROC near
    push ax
    mov ah, 09h
    int 21h
    pop ax
    ret
print ENDP

```

```

FREE_MEMORY_PROC PROC
    push ax
    push bx
    push cx
    push dx

    mov ax, offset END_DATA
    mov bx, offset END_ALL
    add bx, ax

    mov cl, 4
    shr bx, cl
    add bx, 43
    mov ah, 4ah
    int 21h

    jnc ending
    ; error
    mov MEMORY_ERROR, 10
    cmp ax, 7
    je block_destroyed_1

```

```

    cmp ax, 8
    je low_memory_1

    cmp ax, 9
    je invalid_adress_of_block_1
    ;jmp ending

block_destroyed_1:
    mov dx, offset BLOCKED_DESTROYED
    call print
    jmp ending

low_memory_1:
    mov dx, offset LOW_MEMORY
    call print
    jmp ending

invalid_adress_of_block_1:
    mov dx, offset INVALID_ADRESS_OF_BLOCK
    call print
    jmp ending

ending:
    pop dx
    pop cx
    pop bx
    pop ax
    ret
FREE_MEMORY_PROC ENDP

LOAD_OVL PROC
    push ax
    push bx
    push cx
    push dx
    push ds
    push es

    mov ax, data
    mov es, ax

```

```

    mov bx, offset OVERLAY
    mov dx, offset POS
    mov ax, 4b03h
    int 21h

    jnc great

    mov si, offset LOAD_ERROR
    add si, 17
    call BYTE_TO_DEC
    mov dx, offset LOAD_ERROR
    call print

    jmp finish_load

great:
    mov dx, offset SUCCESS_LOAD
    call print
    mov ax, word ptr OVERLAY
    mov es, ax
    mov word ptr OVERLAY, 0
    mov word ptr OVERLAY + 2, ax

    call OVERLAY
    mov es, ax
    mov ah, 49h
    int 21h

finish_load:
    pop es
    pop ds
    pop dx
    pop cx
    pop bx
    pop ax
    ret
LOAD_OVL ENDP

FINDING PROC
    push ax
    push bx
    push cx

```

```

push dx
push di
push si
push es

mov PROGRAM, dx

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

find:
inc bx
cmp byte ptr es:[bx-1], 0
jne find

cmp byte ptr es:[bx+1], 0
jne find

add bx, 2
mov di, 0

looping:
mov dl, es:[bx]
mov byte ptr [POS+di], dl
inc di
inc bx
cmp dl, 0
je end_loop
cmp dl, '\\'
jne looping
mov cx, di
jmp looping
end_loop:
mov di, cx
mov si, PROGRAM

fn:
mov dl, byte ptr [si]
mov byte ptr [POS+di], dl
inc di

```



```

    inc si
    cmp dl, 0
    jne fn

    pop es
    pop si
    pop di
    pop dx
    pop cx
    pop bx
    pop ax
    ret
FINDING ENDP

ALLOCATION PROC

    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

    jnc successful
    cmp ax, 2
    je not_found

    cmp ax, 3
    je route_not_found

    jmp restore

not_found:
    mov dx, offset FILE_NOT_FOUNT
    call print
    jmp restore

```

```

route_not_found:
    mov dx, offset ROUTE_ERROR
    call print
    jmp restore

successful:
    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 OVERLAY, ax

restore:
    pop dx
    pop cx
    pop bx
    pop ax
    ret

ALLOCATION ENDP

READING_OVL PROC
    call FINDING
    mov dx, offset POS
    call ALLOCATION
    call LOAD_OVL
    ret
READING_OVL ENDP

MAIN PROC FAR
    mov ax, DATA

```

```

    mov ds, ax
    mov KEEP_PSP, es
    call FREE_MEMORY_PROC

    cmp MEMORY_ERROR, 10
    je finish

    mov dx, offset OVL1
    call READING_OVL
    mov dx, offset EOF
    call print
    mov dx, offset OVL2
    call READING_OVL

finish:
    xor al, al
    mov ah, 4ch
    int 21h

MAIN    ENDP

END_ALL:
CODE    ENDS
        END MAIN

```