

**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. 8382

Мирончик П.Д.

Преподаватель

Ефремов М.А.

Санкт-Петербург

2020

## **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 и оверлейные модули находятся в одном каталоге.

В этой работе также рассматривается приложение, состоящее из нескольких модулей, поэтому все модули помещаются в один каталог и вызываются с использованием полного пути.

## **Ход работы.**

Была реализована программа, которая загружает и выполняет оверлейные сегменты. Так как при первоначальном запуске программы ей отводится вся доступная память, для загрузки оверлеев осуществляется освобождение памяти, не используемой программой, при помощи функции 4Ah прерывания int 21h. При возникновении ошибки при освобождении памяти, выводится соответствующее сообщение и осуществляется выход из программы.

Затем формируется путь до оверлея. Для этого в специально выделенную строку записывается путь до вызываемой программы, который расположен после переменных среды, после чего имя вызываемого модуля заменяется именем загружаемого оверлея.

После этого определяется размер оверлея при помощи функции 4Eh прерывания int 21h. Перед обращением к функции 1Ah прерывания int 21h устанавливается адрес для буфера DTA, под который в сегменте данных предварительно была выделена область памяти размеров в 43 байта. В регистре CX устанавливается атрибут, который для файла имеет значение 0. После запуска функции проверяется значение флага переноса CF и в случае ошибки

выводится соответствующее сообщение и осуществляется выход из программы. Если функция была выполнена успешно, то в области памяти буфера DTA со смещением 1Ah будет находиться младшее слово размера файла в байтах, а со 3

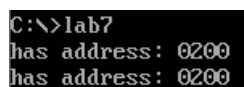
смещением 1Ch – старшее. Размер файла переводится в параграфы. Затем с помощью функции 48h прерывания int 21h под оверлей выделяется блок памяти. Сегментный адрес выделенного блока сохраняется в переменную.

Для загрузки оверлея используется функция 4B03h прерывания int 21h. Перед вызовом функции ей передаются параметры: в DS:DX – указатель на строку, содержащую путь к оверлею, в ES:BX – указатель на сегментный адрес загрузки программы. Если при выполнении функции произошла ошибка, выводится сообщение, соответствующее коду ошибки, и осуществляется выход из программы. Если ошибки не было, оверлей загружен в память. Он вызывается по своему сегментному адресу. После вызова оверлея происходит освобождение занимаемой им памяти.

Программа выполняет перечисленные действия для двух оверлейных сегментов.

Оверлейный сегмент представляет собой кодовый сегмент, который оформлен как функция и выводит на экран адрес сегмента, в который он загружен.

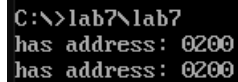
Полученная программа была запущена. Результат ее работы представлен на рисунке 1. Можно заметить, что сегменты загружаются с одного и того же адреса, перекрывая друг друга.



```
C:\>lab7
has address: 0200
has address: 0200
```

Рисунок 1 - Результат запуска приложения

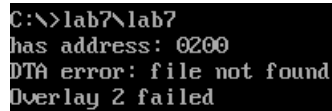
Приложение было запущено из другого каталога и было выполнено успешно. Результат работы представлен на рисунке 2.



```
C:\>lab7\lab7
has address: 0200
has address: 0200
```

Рисунок 2 - Результат запуска приложения из другого каталога

Из каталога был удален второй оверлей. При запуске приложения был выполнен первый оверлей, при попытке загрузки второго программа аварийно завершилась. Результат ее работы представлен на рисунке 3.



```
C:\>lab7\lab7
has address: 0200
DTA error: file not found
Overlay 2 failed
```

Рисунок 3 - Результат запуска приложения после удаления одного из оверлеев

## Ответы на вопросы.

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

Оверлейный сегмент является не загрузочным модулем, а кодовым сегментом, оформленным как функция с точкой входа по адресу 0, поэтому при использовании модуля .COM в качестве оверлейного сегмента необходимо учитывать, что кодовый сегмент модуля предполагает смещение на 100h байт, занимаемых PSP, относительно его начала.

## Выводы

Было реализовано приложение, состоящее из вызывающей программы и двух оверлейных сегментов, расположенных в одном каталоге. В ходе

выполнения работы исследована структура оверлейного сегмента и способ загрузки и выполнения оверлейных сегментов.

## ПРИЛОЖЕНИЕ А

### ИСХОДНЫЙ КОД ПРОГРАММЫ

```
CODE    SEGMENT
        ASSUME DS:CODE, CS:CODE, SS:CODE, ES:CODE
        org 100h

START:
        jmp MAIN
        OverlayName1      db      'over1.ovl',0,'$'
        OverlayName2      db      'over2.ovl',0,'$'

        OverlayLoaded     db      'Overlay loaded',10,13,'$'
        OverlayFailed1    db      'Overlay 1 failed',10,13,'$'
        OverlayFailed2    db      'Overlay 2 failed',10,13,'$'
        CallAddr          dd      0

        MemErr7           db      'Memory error: memory control block has been destroyed',13,10,'$'
        MemErr8           db      'Memory error: lack of memory',13,10,'$'
        MemErr9           db      'Memory error: incorrect block address',13,10,'$'

        ProgErr1          db      'Loading error: incorrect function number',13,10,'$'
        ProgErr2          db      'Loading error: file not found',13,10,'$'
        ProgErr3          db      'Loading error: route not found',13,10,'$'
        ProgErr4          db      'Loading error: too much opened files',13,10,'$'
        ProgErr5          db      'Loading error: no access',13,10,'$'
        ProgErr8          db      'Loading error: lack of memory',13,10,'$'
        ProgErr10         db      'Loading error: incorrect environment string',13,10,'$'

        DtaErr2           db      'DTA error: file not found',10,13,'$'
        DtaErr3           db      'DTA error: route not found',10,13,'$'

        OverAllocErr      db      'Failed alloc memory for overlay',10,13,'$'

        MStack            db      100h dup(0)
        PathFileIndex     dw      0
        OverlayParams      dw      0
        OverlayOffset      dw      0
        Path               db      128 DUP('$')
        Endl               db      13,10,'$'
        DTA                db      43 dup(0)

;-----освобождение памяти-----
MEMFREE PROC NEAR
        push AX
        push BX
        push DX
        push CX
```

```

    mov BX,offset LAST_BYTE
    mov cl, 4
    shr BX,CL
    inc BX
    mov AH,4Ah
    int 21h

    jnc MEMFREE_success
    cmp AX,7
    je MEMFREE_err7
    cmp AX,8
    je MEMFREE_err8
    cmp AX,9
    je MEMFREE_err9

MEMFREE_err7:
    mov DX,offset MemErr7
    jmp MEMFREE_err_write
MEMFREE_err8:
    mov DX,offset MemErr8
    jmp MEMFREE_err_write
MEMFREE_err9:
    mov DX,offset MemErr9
    jmp MEMFREE_err_write
MEMFREE_err_write:
    mov AH,09h
    int 21h
    mov AH,4Ch
    int 21h

MEMFREE_success:
    pop CX
    pop DX
    pop BX
    pop AX
    ret
MEMFREE      ENDP

PREPARE_PATH PROC NEAR
    push ES
    push SI
    push DI
    push DX

    mov ES,DS:[2Ch]          ;извлекаем сегментный адрес среды
    mov SI,0                 ;инициализируем счетчик
PREPARE_PATH_env_loop:
    mov DL,ES:[SI]
    inc SI
    cmp DL,0h
    jne PREPARE_PATH_env_loop

    mov DL,ES:[SI]
    inc SI
    cmp DL,0h
    jne PREPARE_PATH_env_loop

```

```

        add SI,2

        mov DI,offset Path
PREPARE_PATH_path_loop:
        mov DL,ES:[SI]

        cmp dl, '\\'
        jne PREPARE_PATH_path_check_end
        inc di
        mov PathFileIndex, di
        dec di

        PREPARE_PATH_path_check_end:
            cmp DL,00h
            je PREPARE_PATH_finish

        mov [DI],DL
        inc DI
        inc SI
        jmp PREPARE_PATH_path_loop

PREPARE_PATH_finish:
        pop DX
        pop DI
        pop SI
        pop ES
        ret
PREPARE_PATH ENDP

SET_PATH PROC NEAR
        push dx
        push di
        push si

        mov di, PathFileIndex

        SET_PATH_loop:
            mov dl, [si]
            mov [di], dl
            inc di
            inc si
            cmp dl,0
            jne SET_PATH_loop

        pop si
        pop di
        pop dx
        ret
SET_PATH ENDP

FILL_DTA PROC NEAR
        push AX
        push dx
        push cx

        mov ah, 1Ah
        mov dx, offset DTA
        int 21h

```



```

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

    jnc FILL_DTA_success
    cmp ax, 2
    je FILL_DTA_err2
    cmp ax, 3
    je FILL_DTA_err3

FILL_DTA_err2:
    mov dx, offset DtaErr2
    jmp FILL_DTA_err_print
FILL_DTA_err3:
    mov dx, offset DtaErr3
    jmp FILL_DTA_err_print
FILL_DTA_err_print:
    call PRINT
    stc
    jmp FILL_DTA_finish

FILL_DTA_success:
    clc

FILL_DTA_finish:
    pop cx
    pop dx
    pop AX
    ret
FILL_DTA ENDP

OVER_ALLOC PROC NEAR
    push dx
    push ax
    push bx
    push di

    mov di, offset DTA
    mov bx, di[1Ch]
    sub bx, di[1Ah]
    shr bx, 4
    inc bx
    mov ah, 48h
    int 21h

    jnc OVER_ALLOC_success

    mov dx, offset OverAllocErr
    call PRINT
    stc
    jmp OVER_ALLOC_finish

OVER_ALLOC_success:
    mov OverlayParams, ax
    clc

```

```

OVER_ALLOC_finish:
    pop di
    pop bx
    pop ax
    pop dx
    ret
OVER_ALLOC ENDP

OVER_FREE PROC NEAR
    push ax
    push es

    mov es, OverlayParams
    mov ah, 49h
    int 21h

    pop es
    pop ax
    ret
OVER_FREE ENDP

PRINT PROC NEAR
    push ax

    mov ah, 09h
    int 21h

    pop ax
    ret
PRINT ENDP

OVER_LOAD PROC NEAR
    push ax
    push es
    push bx
    push cx
    push dx

    mov AX, DS
    mov ES, AX
    mov BX, offset OverlayParams
    mov DX, offset Path
    mov AX, 4B03h
    int 21h

    jnc OVER_LOAD_loaded
    cmp AX, 1
    je OVER_LOAD_err1
    cmp AX, 2
    je OVER_LOAD_err2
    cmp AX, 3
    je OVER_LOAD_err3
    cmp AX, 4
    je OVER_LOAD_err4
    cmp AX, 5
    je OVER_LOAD_err5
    cmp AX, 8
    je OVER_LOAD_err8

```

```

    cmp AX,10
    je OVER_LOAD_err10

OVER_LOAD_err1:
    mov DX,offset ProgErr1
    jmp OVER_LOAD_err_write
OVER_LOAD_err2:
    mov DX,offset ProgErr2
    jmp OVER_LOAD_err_write
OVER_LOAD_err3:
    mov DX,offset ProgErr3
    jmp OVER_LOAD_err_write
OVER_LOAD_err4:
    mov DX,offset ProgErr4
    jmp OVER_LOAD_err_write
OVER_LOAD_err5:
    mov DX,offset ProgErr5
    jmp OVER_LOAD_err_write
OVER_LOAD_err8:
    mov DX,offset ProgErr8
    jmp OVER_LOAD_err_write
OVER_LOAD_err10:
    mov DX,offset ProgErr10
    jmp OVER_LOAD_err_write
OVER_LOAD_err_write:
    mov AH,09h
    int 21h
    stc
    jmp OVER_LOAD_end

OVER_LOAD_loaded:
    mov ax, OverlayParams
    mov bx, offset CallAddr
    mov [bx+2], ax
    call CallAddr

    clc

OVER_LOAD_end:
    pop dx
    pop cx
    pop bx
    pop es
    pop ax
    ret
OVER_LOAD ENDP

MAIN:
    mov SP,offset MStack
    add SP,100h

    call MEMFREE
    call PREPARE_PATH

    mov si, offset OverlayName1
    call SET_PATH

    call FILL_DTA

```

```

        jc OVER1_failed

        call OVER_ALLOC
        jc OVER1_failed

        call OVER_LOAD
        jc OVER1_failed

        call OVER_FREE
        jc OVER1_failed

        jmp OVER2

OVER1_failed:
        mov dx, offset OverlayFailed1
        call PRINT

OVER2:
        mov si, offset OverlayName2
        call SET_PATH
        jc OVER2_failed

        call FILL_DTA
        jc OVER2_failed

        call OVER_ALLOC
        jc OVER2_failed

        call OVER_LOAD
        jc OVER2_failed

        call OVER_FREE
        jc OVER2_failed

        jmp EXIT

OVER2_failed:
        mov dx, offset OverlayFailed2
        call PRINT

EXIT:
        mov al, 0
        mov AH, 4Ch
        int 21h

LAST_BYTE:
CODE     ENDS
END START

```

## ПРИЛОЖЕНИЕ Б

### ИСХОДНЫЙ КОД ОВЕРЛЕЯ

```
overlay segment
    assume cs:overlay, ds:overlay
    start: jmp main
    msg_address db "has address:      ", 13, 10, "$"

main proc far
    push ax
    push ds
    push di
    mov ax, cs
    mov ds, ax
    mov di, offset msg_address
    push di
    add di, 10h
    call WRD_TO_HEX
    pop di
    call print
    pop di
    pop ds
    pop ax
    retf
main endp

print proc near
    ; prints di content
    push dx
    push ax
    mov ah, 9h
    mov dx, di
    int 21h
    pop ax
    pop dx
    ret
print endp

WRD_TO_HEX PROC near
;перевод в 16 с/с 16-ти разрядного числа
; в AX - число, 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
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 шест. 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

overlay ends
end start

```