

**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

Щемель Д.А.

Преподаватель

Ефремов М.А.

Санкт-Петербург

2020

## Цель работы

Исследование возможности построения загрузочного модуля оверлейной структуры. Исследуется структура оверлейного сегмента и способ его 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.

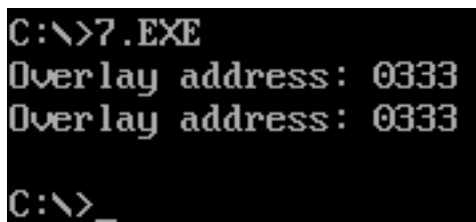
В этой работе также рассматривается приложение, состоящее из нескольких модулей, поэтому все модули помещаются в один каталог и вызываются с использованием полного пути.

## Ход выполнения работы

Был написан исходный код для .EXE-модуля, выполняющий следующие действия:

1. Вычисляет путь до запускаемой программы и выделяет память для оверлея
2. Вызывает оверлейный модуль, который выводит на экран адрес, в который он загружен
3. В случае ошибок на любом из этапов подготовки запуска выводится соответствующий этой ошибке код и завершается выполнение программы
4. п.1-3 повторяются для второго модуля оверлейной структуры

Результат работы программы при запуске программы, когда вызываемая программа находится в том же каталоге:



```
C:\>7.EXE
Overlay address: 0333
Overlay address: 0333
C:\>_
```

Рис. 1: Результат работы

Результат работы программы при запуске программы, когда вызываемая программа находится в том же каталоге, но уже другом:

Результат работы программы, когда в каталоге с вызывающей программой нет второго оверлея:

## Контрольные вопросы

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

```
C:\>mkdir lol

C:\>cd lol

C:\LOL>dir
Directory of C:\LOL\
.                <DIR>                28-05-2020   1:00
..              <DIR>                28-05-2020   1:00
    0 File(s)                0 Bytes.
    2 Dir(s)                262,111,744 Bytes free.

C:\LOL>..\7.EXE
Overlay address: 0333
Overlay address: 0333

C:\LOL>_
```

Рис. 2: Результат работы

```
C:\LOL>delete ..\2.OVL

C:\LOL>..\7.EXE
Overlay address: 0333
Find file error: 0012

C:\LOL>_
```

Рис. 3: Результат работы

Поскольку .COM модуль отличается от .OVL модуля тем, что в первом выставляется смещение - это смещение необходимо учитывать в вызывающей программе(вызывать с функции с этим смещением).

## **Вывод**

В ходе выполнения лабораторной работы были получены практические навыки по построению загрузочного модуля оверлейной структуры.

## ПРИЛОЖЕНИЕ А. ИСХОДНЫЙ КОД ПРОГРАММЫ 7.ASM

```
_STACK SEGMENT STACK
```

```
    DW 100h DUP(0)
```

```
_STACK ENDS
```

```
DATA SEGMENT
```

```
    PROGRAM_PATH db 80 dup(0)
```

```
    EXEC_ERROR db 'Exec error: 0000', 13, 10, '$'
```

```
    FIND_FILE_ERROR db 'Find file error: 0000', 13, 10, '$'
```

```
    ALLOC_MEMORY_ERROR db 'Allocation memory error: 0000', 13, 10, '$'
```

```
    FREE_SEGMENT dw 0
```

```
    OVERLAY_ADDRESS dd 0
```

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

```
    OVERLAY_1_NAME db '1.OVL$'
```

```
    OVERLAY_2_NAME db '2.OVL$'
```

```
DATA ENDS
```

```
CODE SEGMENT
```

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

```
EXIT PROC near
```

```
    xor AL, AL
```

```
    mov AH, 4ch
```

```
    int 21h
```

```
    ret
```

```
EXIT ENDP
```

```
PRINT PROC near
```

```
    push ax
```

```
    mov ah, 09h
```

```
    int 21h
```

```
    pop ax
```

```
    ret
```

```
PRINT ENDP
```

```
ADJUST_SIZE PROC near
```

```
    push ax
```

```
    push bx
```

```
    mov ah, 4ah
```

```
    mov bx, offset END_LABEL
```

```
    int 21h
```

```
    pop bx
```

```
    pop ax
```

```
    ret
```

```
ADJUST_SIZE ENDP
```

```
FIND_START_PATH PROC near
```

```
; OUT: si - offset to end of path var
```

```
    push es
```

```
    push ax
```

```
    push dx
```

```
    mov es, es:[2ch]
```

```
    mov si, 0
```

```
CMP_WITH_0000_AND_INC:
```

```
    mov al, es:[si]
```

```
    mov ah, es:[si+1]
```

```
    cmp ax, 0000h
```

```
    je FINISH_FIND_END_PATH
```

```
    inc si
```

```
    jmp CMP_WITH_0000_AND_INC
```

```
FINISH_FIND_END_PATH:
```

```
    add si, 4
```

```
    pop dx
```

```

    pop ax
    pop es
    ret
FIND_START_PATH ENDP

COPY_PATH PROC near
; IN si - start path IN PSP
; OUT di - end of path IN DATA
    push ax
    push es
    push si

    mov es, es:[2ch]
    mov di, offset PROGRAM_PATH
    CMP_WITH_COPY_AND_INC:
    mov al, es:[si]
    mov [di], al
    cmp al, 0
    je FINISH_COPY_PATH
    inc si
    inc di
    jmp CMP_WITH_COPY_AND_INC

    FINISH_COPY_PATH:
    pop si
    pop es
    pop ax
    ret
COPY_PATH ENDP

SET_DTA PROC near
    push ax
    push dx

```

```

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

    pop dx
    pop ax
    ret
SET_DTA ENDP

PREPARE_DATA PROC near
; IN dx - programm path
; OUT dx - programm path
    push bx
    push di
    push cx
    push si
    push es

    call SET_DTA
    call FIND_START_PATH
    call COPY_PATH

    sub di, 5
    mov cx, 5
    mov si, 0
    mov bx, dx

COPY_SYBOL:
    mov al, [bx+si]
    mov [di], al
    inc si
    inc di
    loop COPY_SYBOL

```



```

FINISH_PREPARE_DATA:
    mov dx, offset PROGRAM_PATH

    pop es
    pop si
    pop cx
    pop di
    pop bx
    ret
PREPARE_DATA 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
; IN: al
; OUT: ax
    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
```

```
RUN_PROGRAM PROC near
```

```
    push ax
```

```
    push es
```

```
    push bx
```

```
    push dx
```

```
    push si
```

```
    mov dx, offset PROGRAM_PATH
```

```
    mov ax, seg FREE_SEGMENT
```

```
    mov es, ax
```

```
    mov bx, offset FREE_SEGMENT
```

```
    mov ax, 4b03h
```

```
    int 21h
```

```
    jnc USE_OVERLAY
```

```
    mov bl, al
```

```
    xchg al, ah
```

```
    call BYTE_TO_HEX
```

```
    mov si, offset EXEC_ERROR
```

```
    add si, 12
```

```
    mov [si], ax
```

```
    mov al, bh
```

```
    call BYTE_TO_HEX
```

```
    add si, 2
```

```
    mov [si], ax
```

```
    mov dx, offset EXEC_ERROR
```

```
    call PRINT
```

```
    jmp FINISH_RUN_PROGRAM
```

```
USE_OVERLAY:
```

```

    mov ax, FREE_SEGMENT
    mov word ptr OVERLAY_ADDRESS + 2, ax
    call OVERLAY_ADDRESS

FINISH_RUN_PROGRAM:
    mov ax, FREE_SEGMENT
    mov es, ax
    mov ah, 49h
    int 21h

    pop si
    pop dx
    pop bx
    pop es
    pop ax
    ret
RUN_PROGRAM ENDP

FIND_OVERLAY PROC near
    push ax
    push cx
    push bx
    push dx
    push si

    mov ah, 4eh
    mov dx, offset PROGRAM_PATH
    xor cx, cx
    int 21h
    jnc FINISH_FIND_OVERLAY

    mov bx, ax
    xchg al, ah

```

```

    call BYTE_TO_HEX
    mov si, offset FIND_FILE_ERROR
    add si, 17
    mov [si], ax
    mov al, bl
    call BYTE_TO_HEX
    add si, 2
    mov [si], ax
    mov dx, offset FIND_FILE_ERROR
    call PRINT
    call EXIT

FINISH_FIND_OVERLAY:
    pop si
    pop dx
    pop bx
    pop cx
    pop ax
    ret
FIND_OVERLAY ENDP

READ_OVERLAY_SIZE PROC near
; OUT dx - programm name
    push ax
    push cx

    call FIND_OVERLAY
    mov bx, offset DTA
    mov ax, [bx+1ah]
    mov bx, [bx+1ch]
    mov cl, 4
    shr ax, cl
    mov cl, 12
    shl bx, cl

```

```

    add bx, ax
    inc bx

    pop cx
    pop ax
    ret
READ_OVERLAY_SIZE ENDP

ALLOC_MEMORY PROC near
    push ax
    push bx
    push si

    call READ_OVERLAY_SIZE
    mov ah, 48h
    int 21h
    jnc FINISH_ALLOC_MEMORY

    mov bx, ax
    call BYTE_TO_HEX
    mov si, offset ALLOC_MEMORY_ERROR
    add si, 25
    mov [si], ax
    mov al, bh
    call BYTE_TO_HEX
    add si, 2
    mov [si], ax
    mov dx, offset ALLOC_MEMORY_ERROR
    call EXIT

FINISH_ALLOC_MEMORY:
    mov FREE_SEGMENT, ax

    pop si

```

```

        pop bx
        pop ax
        ret
ALLOC_MEMORY ENDP

LOAD_AND_EXEC_OVERLAY PROC near
;IN dx - program name
        call PREPARE_DATA
        call ALLOC_MEMORY
        call RUN_PROGRAM
        ret
LOAD_AND_EXEC_OVERLAY ENDP

MAIN:
        mov ax, DATA
        mov ds, ax

        call ADJUST_SIZE

        mov dx, offset OVERLAY_1_NAME
        call LOAD_AND_EXEC_OVERLAY
        mov dx, offset OVERLAY_2_NAME
        call LOAD_AND_EXEC_OVERLAY

        call EXIT
        END_LABEL:
CODE ENDS
END MAIN

```

## ПРИЛОЖЕНИЕ В. ИСХОДНЫЙ КОД ПРОГРАММЫ 1.ASM

```

OVERLAY SEGMENT
        ASSUME CS:OVERLAY, DS:OVERLAY, SS:NOTHING, ES:NOTHING
        MAIN PROC FAR

```

```

    push ax
    push dx
    push di
    push ds

    mov ax, cs
    mov ds, ax

    mov di, offset SEGMENT_STRING
    add di, 20
    call WRD_TO_HEX
    mov dx, offset SEGMENT_STRING
    call PRINT

    pop ds
    pop di
    pop dx
    pop ax
    retf
MAIN ENDP

SEGMENT_STRING DB 'Overlay address: 0000', 13, 10, '$'

TETR_TO_HEX PROC near
    and al, 0fh
    cmp al, 9
    jbe NEXT
    add al, 7
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
; IN ax - number
; OUT di - last symbol address
    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

PRINT PROC near
    push ax

    mov ah, 09h

```



```
int 21h

pop ax
ret
PRINT ENDP
OVERLAY ENDS
END MAIN
```