

**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**

Студентка гр. 8382

\_\_\_\_\_

Рочева А.К.

Преподаватель

\_\_\_\_\_

Ефремов М.А.

Санкт-Петербург

2020

### **Цель работы.**

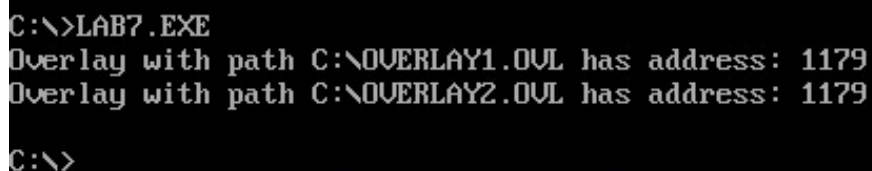
Исследование возможности построения загрузочного модуля оверлейной структуры.

### **Ход выполнения.**

Для выполнения лабораторной работы была написана программа (Приложение А), которая освобождает память для загрузки оверлеев, читает размер файла оверлея и запрашивает объем памяти, достаточный для его загрузки, загружает и выполняет файл оверлейного сегмента и освобождает память, отведенную для него.

Так же были написаны программы (Приложение В и С) с оверлейными сегментами, выводящими адрес сегмента, в который они загружены.

Результат работы программы, когда оба оверлея находятся в одном каталоге с вызывающей программой, представлен на рис.1.

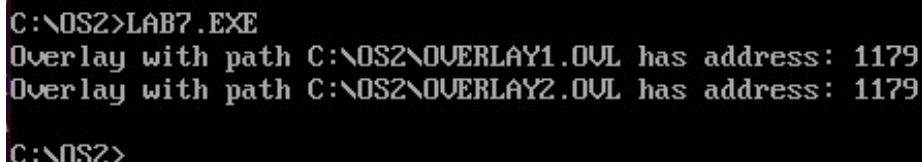


```
C:\>LAB7.EXE
Overlay with path C:\OVERLAY1.OVL has address: 1179
Overlay with path C:\OVERLAY2.OVL has address: 1179
C:\>
```

Рис. 1 — работа программы, когда оба оверлея находятся в одном каталоге с ней

Как видно, оверлейные сегменты загружаются с одного и того же адреса, перекрывая друг друга.

Результат работы программы, когда приложение запускается из другого каталога, представлен на рис.2.



```
C:\OS2>LAB7.EXE
Overlay with path C:\OS2\OVERLAY1.OVL has address: 1179
Overlay with path C:\OS2\OVERLAY2.OVL has address: 1179
C:\OS2>
```

Рис. 2 — запуск программы из другого каталога

Результат работы программы, когда в каталоге отсутствует первый оверлей, представлен на рис. 3.

```
C:\>del OVERLAY1.OVL  
  
C:\>LAB7.EXE  
File not found.  
  
C:\>_
```

Рис. 3 — результат работы программы при отсутствии в каталоге первого оверлея

Результат работы программы, когда в каталоге отсутствует второй оверлей, представлен на рис. 4.

```
C:\>LAB7.EXE  
Overlay with path C:\OVERLAY1.OVL has address: 1179  
File not found.  
  
C:\>
```

Рис. 4 - результат работы программы при отсутствии в каталоге второго оверлея (но первый оверлей присутствует)

### Ответы на вопросы:

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

Оверлейный сегмент представляет собой кодовый сегмент, который оформляется в ассемблере как функции с точкой входа по адресу 0 и возврат осуществляется командой RETF, т. к. возврат управления должен быть осуществлен в программу, выполняющую оверлейный сегмент (т. е. нельзя использовать функции выхода 4Ch прерывания 21h). Получается, что при использовании оверлейного сегмента com модуля нужно вызывать его по смещению 100h, т. к. в com файлах код

располагается именно с этого адреса (в ином случае не будет сформирован psp).

### **Выводы**

В ходе выполнения работы была исследована возможность построения загрузочного модуля оверлейной структуры.

## ПРИЛОЖЕНИЕ А

### ИСХОДНЫЙ КОД LAB7.ASM

```
AStack SEGMENT STACK
    dw 100h dup(?)
AStack ENDS

DATA SEGMENT
    MESSAGE_DAMAGE_CONTROL db 'The control block of memory is destroyed.',
0DH,0AH, '$'
    MESSAGE_NO_MEMORY db 'Not enough memory to complete the function.',
0DH,0AH, '$'
    MESSAGE_INVALID_ADDRESS db 'Invalid memory block address.', 0DH,0AH, '$'
    KEEP_PSP dw 0
    PATH db 64 dup (0), '$'
    DTA db 43 dup (?)
    NAME_OVERLAY1 db 'OVERLAY1.OVL', 0
    NAME_OVERLAY2 db 'OVERLAY2.OVL', 0
    MESSAGE_FILE_NOT_FOUND db 'File not found.', 0DH,0AH, '$'
    MESSAGE_PATH_NOT_FOUND db 'Path not found.', 0DH,0AH, '$'
    MESSAGE_MEMORY_ERROR db 'Memory allocation error.', 0DH,0AH, '$'
    OVERLAY_SEG dw 0
    CALL_ADDR dd 0
    MESSAGE_NO_FUNCTION db 'Function doesnt exist.', 0DH,0AH, '$'
    MESSAGE_TOO_MANY_FILES db 'Too many open files.', 0DH,0AH, '$'
    MESSAGE_NO_ACCESS db 'No access.', 0DH,0AH, '$'
    MESSAGE_LITTLE_MEMORY db 'Little memory.', 0DH,0AH, '$'
    MESSAGE_INVALID_ENV db 'Invalid environment.', 0DH,0AH, '$'
    MESSAGE_PRINT_OVL db 'Overlay with path $'

DATA ENDS

CODE SEGMENT
    ASSUME CS:CODE, DS:DATA, SS:AStack

FREE_MEMORY PROC near
    mov bx, offset LAST_SEGMENT
    mov ax, es
    sub bx, ax
    mov cl, 4h
    shr bx, cl
    mov ah, 4Ah
    int 21h
    jc Err
    jmp EndProcNormal

Err:
    cmp ax, 7
    je DamageControl
    cmp ax, 8
    je NoMemory
    cmp ax, 9
    je InvalidAddress
    jmp EndProcError
DamageControl:
    mov dx, offset MESSAGE_DAMAGE_CONTROL
    jmp EndProcError
NoMemory:
    mov dx, offset MESSAGE_NO_MEMORY
    jmp EndProcError
InvalidAddress:
    mov dx, offset MESSAGE_INVALID_ADDRESS
    jmp EndProcError
EndProcError:
    call WRITE
    xor al, al
    mov ah, 4Ch
    int 21H

EndProcNormal:
```

```

        ret
FREE_MEMORY ENDP

PREPARE_PATH PROC near
    push ds
    push dx
    mov dx, seg DTA
    mov ds, dx
    mov dx, offset DTA
    mov ah, 1Ah
    int 21h
    pop dx
    pop ds

    push es
    push dx
    push di
    push cx
    push si

    mov es, KEEP_PSP
    mov es, es:[2Ch]
    mov di, offset PATH
    mov si, 0
CopyEnv:
    mov dl, es:[si]
    cmp dl, 0
    je EndCopyEnv
    inc si
    jmp CopyEnv
EndCopyEnv:
    inc si
    mov dl, es:[si]
    cmp dl, 0
    jne CopyEnv
    add si, 3
ChangePath:
    mov dl, es:[si]
    cmp dl, 0
    je EndChange
    mov [di], dl
    inc si
    inc di
    jmp ChangePath
EndChange:
    mov si, bp
EntryPath:
    mov dl, byte ptr [si]
    mov byte ptr [di-8], dl
    inc di
    inc si
    cmp dl, 0
    jne EntryPath

    mov dl, '$'
    mov byte ptr [di-8], dl

    pop si
    pop cx
    pop di
    pop dx
    pop es

    ret
PREPARE_PATH ENDP

READ_SIZE PROC near
    push cx
    push dx
    push ds

```

```

        push ax
        push es
        mov cx, 0
        mov dx, seg PATH
        mov ds, dx
        mov dx, offset PATH
        mov ah, 4Eh
        int 21h
        jnc FileFound
        cmp ax, 2
        je FileNotFound
        cmp ax, 3
        je PathNotFound
FileNotFound:
        mov dx, offset MESSAGE_FILE_NOT_FOUND
        jmp ErrorEnd
PathNotFound:
        mov dx, offset MESSAGE_PATH_NOT_FOUND
        jmp ErrorEnd
ErrorEnd:
        call WRITE
        pop es
        pop ax
        pop ds
        pop dx
        pop cx
        mov al, 0
        mov ah, 4Ch
        int 21h
FileFound:
        mov si, offset DTA
        mov ax, [si+1Ah]
        mov bx, [si+1Ch]
        mov cl, 4
        shr ax, cl
        mov cl, 12
        shr bx, cl
        add ax, bx
        add ax, 2
        mov bx, ax

        mov ah, 48h
        int 21h
        jc MemoryError
        mov OVERLAY_SEG, ax
        pop es
        pop ax
        pop ds
        pop dx
        pop cx
        ret
MemoryError:
        mov dx, offset MESSAGE_MEMORY_ERROR
        call WRITE
        pop es
        pop ax
        pop ds
        pop dx
        pop cx
        mov al, 0
        mov ah, 4Ch
READ_SIZE ENDP

LOAD_OVERLAY PROC near
        push bx
        push ax
        push dx
        push ds
        push ss
        push sp
        mov bx, seg OVERLAY_SEG

```

```

    mov es, bx
    mov bx, offset OVERLAY_SEG
    mov dx, seg PATH
    mov ds, dx
    mov dx, offset PATH
    mov ax, 4B03h
    int 21h
    jnc OvlLoad
    cmp ax, 1
    je NoFunction
    cmp ax, 2
    je NoFile
    cmp ax, 3
    je NoPath
    cmp ax, 4
    je TooManyFiles
    cmp ax, 5
    je NoAccess
    cmp ax, 8
    je LittleMemory
NoFunction:
    mov dx, offset MESSAGE_NO_FUNCTION
    jmp OvlNotLoad
NoFile:
    mov dx, offset MESSAGE_FILE_NOT_FOUND
    jmp OvlNotLoad
NoPath:
    mov dx, offset MESSAGE_PATH_NOT_FOUND
    jmp OvlNotLoad
TooManyFiles:
    mov dx, offset MESSAGE_TOO_MANY_FILES
    jmp OvlNotLoad
NoAccess:
    mov dx, offset MESSAGE_NO_ACCESS
    jmp OvlNotLoad
LittleMemory:
    mov dx, offset MESSAGE_LITTLE_MEMORY
    jmp OvlNotLoad
OvlNotLoad:
    call WRITE
    jmp EndOvl
OvlLoad:
    mov dx, offset MESSAGE_PRINT_OVL
    call WRITE
    mov dx, offset PATH
    call WRITE

    mov ax, DATA
    mov ds, ax
    mov ax, OVERLAY_SEG
    mov word ptr CALL_ADDR+2, ax
    call CALL_ADDR
    mov ax, OVERLAY_SEG
    mov es, ax
    mov ax, 4900h
    int 21h
    mov ax, DATA
    mov ds, ax

EndOvl:
    pop sp
    pop ss
    pop ds
    pop dx
    pop ax
    pop bx
    mov es, KEEP_PSP
    ret
LOAD_OVERLAY ENDP

```

```
WRITE PROC near
```



```

    push ax
    mov ah, 09h
    int 21h
    pop ax
    ret
WRITE ENDP

MAIN PROC FAR
    mov ax, DATA
    mov ds, ax

    mov KEEP_PSP, es
    call FREE_MEMORY
    mov bp, offset NAME_OVERLAY1
    call PREPARE_PATH
    call READ_SIZE
    call LOAD_OVERLAY

    xor bx, bx
    mov bp, offset NAME_OVERLAY2
    call PREPARE_PATH
    call READ_SIZE
    call LOAD_OVERLAY

    xor ax, ax
    mov ah, 4Ch
    int 21h
    ret
MAIN ENDP
CODE ENDS
LAST_SEGMENT SEGMENT
LAST_SEGMENT ENDS
END MAIN

```

## ПРИЛОЖЕНИЕ В

### ИСХОДНЫЙ КОД OVERLAY1.ASM

```
OVERLAY1 SEGMENT
ASSUME CS:OVERLAY1, DS:OVERLAY1

START: jmp START_PROC

PRINT_ADDRESS DB 'has address:          ', 0DH, 0AH, '$'

START_PROC PROC FAR
    push ax
    push bx
    push dx
    push ds
    mov ax, cs
    mov ds, ax
    mov bx, offset PRINT_ADDRESS
    add bx, 16
    mov di, bx
    mov ax, cs
    call WRD_TO_HEX
    mov dx, offset PRINT_ADDRESS
    call WRITE
    pop ds
    pop dx
    pop bx
    pop ax
    retf
START_PROC ENDP

WRITE PROC NEAR
    push ax
    mov ah, 09h
    int 21h
    pop ax
    ret
WRITE 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

OVERLAY1 ENDS
END START

```

## ПРИЛОЖЕНИЕ С

### ИСХОДНЫЙ КОД OVERLAY2.ASM

```
OVERLAY2 SEGMENT
ASSUME CS:OVERLAY2, DS:OVERLAY2

START: jmp START_PROC

PRINT_ADDRESS DB 'has address:          ', 0DH, 0AH, '$'

START_PROC PROC FAR
    push ax
    push bx
    push dx
    push ds
    mov ax, cs
    mov ds, ax
    mov bx, offset PRINT_ADDRESS
    add bx, 16
    mov di, bx
    mov ax, cs
    call WRD_TO_HEX
    mov dx, offset PRINT_ADDRESS
    call WRITE
    pop ds
    pop dx
    pop bx
    pop ax
    retf
START_PROC ENDP

WRITE PROC NEAR
    push ax
    mov ah, 09h
    int 21h
    pop ax
    ret
WRITE 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

OVERLAY2 ENDS
END START

```