

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. 9381

Давыдов Д.С.

Преподаватель

Ефремов М.А.

Санкт-Петербург

2021

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.

Задание.

Шаг 1. Для выполнения лабораторной работы необходимо написать и отладить программный модуль типа .EXE, который выполняет функции:

- 1) Освобождает память для загрузки оверлеев.
- 2) Читает размер файла оверлея и запрашивает объем памяти, достаточный для его загрузки.
- 3) Файл оверлейного сегмента загружается и выполняется.
- 4) Освобождается память, отведенная для оверлейного сегмента.
- 5) 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.

Шаг 1.

Был написан и отлажен программный модуль типа .EXE, который выполняет функции:

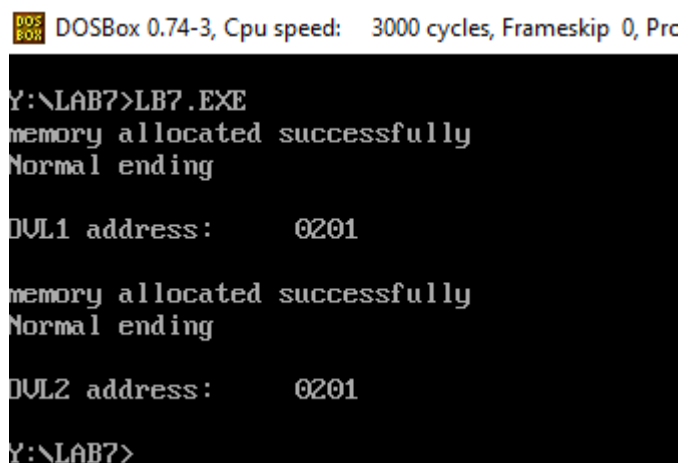
- 1) Освобождает память для загрузки оверлеев.
- 2) Читает размер файла оверлея и запрашивает объем памяти, достаточный для его загрузки.
- 3) Файл оверлейного сегмента загружается и выполняется.
- 4) Освобождается память, отведенная для оверлейного сегмента.
- 5) Затем действия 1)-4) выполняются для следующего оверлейного сегмента.

Шаг 2.

Были написаны и отлажены оверлейные сегменты. Оверлейный сегмент выводит адрес сегмента, в который он загружен.

Шаг 3.

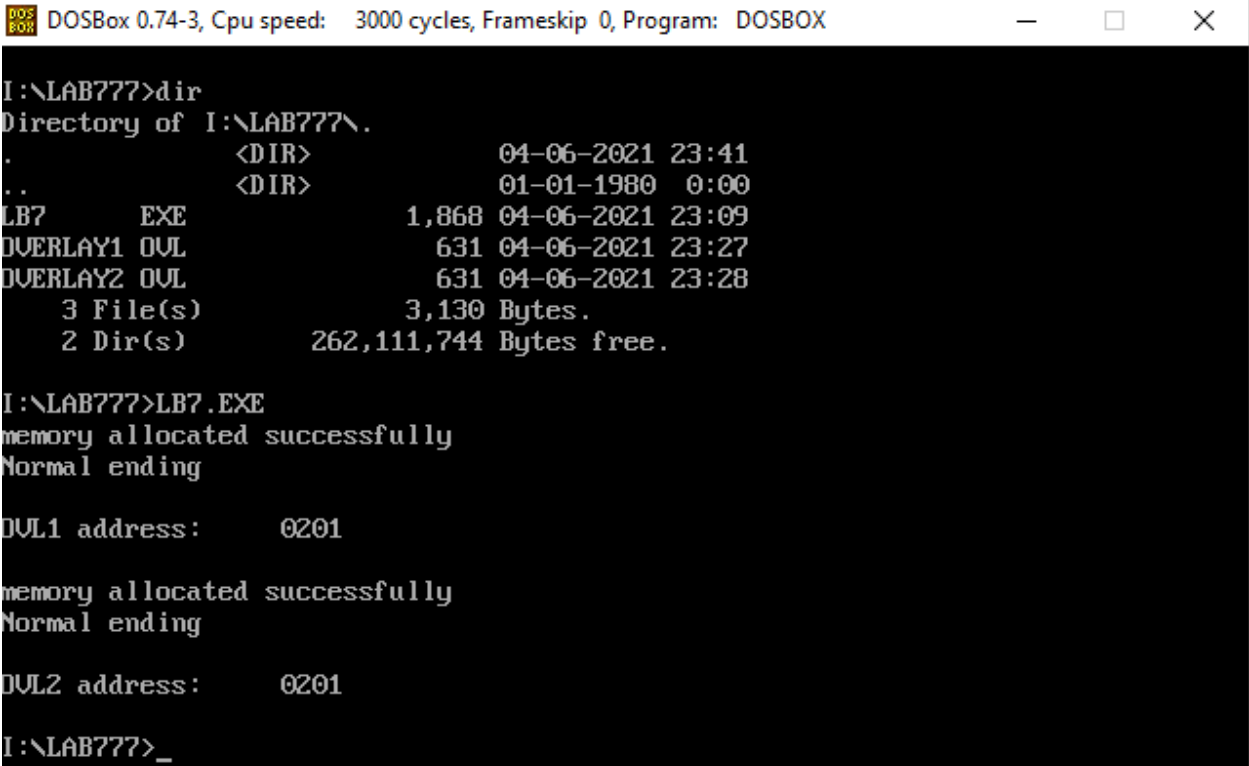
Был запущен lb7.exe. Оверлейные сегменты загружаются с одного и того же адреса, перекрывая друг друга.



```
DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip 0, Prc
Y:\LAB7>LB7.EXE
memory allocated successfully
Normal ending
DUL1 address: 0201
memory allocated successfully
Normal ending
DUL2 address: 0201
Y:\LAB7>
```

Шаг 4.

Было запущено приложение из другого каталога. Результат останется прежним — программа успешно завершится.



DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip 0, Program: DOSBOX

```
I:\LAB777>dir
Directory of I:\LAB777\
.                <DIR>                04-06-2021 23:41
..               <DIR>                01-01-1980  0:00
LB7              EXE                  1,868 04-06-2021 23:09
OVERLAY1 OVL     631 04-06-2021 23:27
OVERLAY2 OVL     631 04-06-2021 23:28
    3 File(s)      3,130 Bytes.
    2 Dir(s)       262,111,744 Bytes free.

I:\LAB777>LB7.EXE
memory allocated successfully
Normal ending

OVL1 address:      0201

memory allocated successfully
Normal ending

OVL2 address:      0201

I:\LAB777>_
```

Шаг 5.

Было запущено приложение в случае, когда одного оверлея нет в каталоге.

```
DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip 0, Program: DOSBOX

I:\LAB777>dir
Directory of I:\LAB777\
.                <DIR>                04-06-2021 23:40
..               <DIR>                01-01-1980  0:00
LB7              EXE                  1,868 04-06-2021 23:09
OVERLAY1 OVL     631 04-06-2021 23:27
      2 File(s)                2,499 Bytes.
      2 Dir(s)                262,111,744 Bytes free.

I:\LAB777>LB7.EXE
memory allocated successfully
Normal ending

OVL1 address:      0201

File not found
File was not found

I:\LAB777>
```

Разработанный программный код смотреть в приложении А.

Ответы на контрольные вопросы.

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

Ответ: необходимо будет учитывать смещение 100h, потому что в .COM модуле присутствует PSP.

Вывод.

Был написан загрузочный модуль оверлейной структуры, а также написаны и отлажены сами оверлеи. Изучены дополнительные функции работы с памятью и способы её загрузки.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММ

Название файла: lb7.asm

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

```
LAB_STACK      SEGMENT  STACK
```

```
                DW 128 DUP(?)
```

```
LAB_STACK      ENDS
```

```
DATA SEGMENT
```

```
    nameOverlay1 db 'overlay1.OVL', 0
```

```
    nameOverlay2 db 'overlay2.OVL', 0
```

```
    prog dw 0
```

```
    dataMem db 43 dup(0)
```

```
    pos db 128 dup(0)
```

```
    ovlsAddr dd 0
```

```
    pspKeep dw 0
```

```
    eof db 13, 10, '$'
```

```
    memoryDestroyed db 'Destroyed memory block',13,10,'$' ;7
```

```
        memoryLowMemFunc db 'Not enough memory for running  
function',13,10,'$' ;8
```

```
    memoryWrongMemAdr db 'Incorrect memorys address',13,10,'$' ;9
```

```
    errorWrongFuncNum db 'Wrong functions number',13,10,'$' ;1
```

```
    errorMissFile db 'File was not found',13,10,'$' ;2
```

```
    errorDisk db 'Disk error',13,10,'$' ;5
```

```
    errorNotEnoughMem db 'Not enough free disk memory space',13,10,'$'  
;8
```

```
    errorWrongStrFormat db 'Wrong string enviroment',13,10,'$' ;10
```

```
    errorWrongFormat db 'Wrong format',13,10,'$' ;11
```

```
    endSuccess db 'Normal ending',13,10,'$' ;0
```

```
    endCtrlBreak db 'Ending by ctrl-break',13,10,'$' ;1
```

```
        endDeviceError db 'Program was ended with device  
error',13,10,'$' ;2
```

```
        endInterruption    db    'Program    was    ended    by    int    31h  
interruption',13,10,'$' ;3
```

```
allocateSuccessStr db 'memory allocated successfully', 13, 10, '$'  
errorMissStrFile db 'File not found', 13, 10, '$'  
ERROR_ROUTE db 'Route not found', 13, 10, '$'
```

```
    end_data db 0  
DATA ENDS
```

```
CODE SEGMENT
```

```
PRINT PROC
```

```
    push ax  
    mov ah, 09h  
    int 21h  
    pop ax  
    ret
```

```
PRINT ENDP
```

```
FREE_MEMORY PROC
```

```
    push ax  
    push bx  
    push cx  
    push dx
```

```
    mov ax, offset end_data  
    mov bx, offset END_APP  
    add bx, ax  
    shr bx, 1  
    shr bx, 1  
    shr bx, 1  
    shr bx, 1  
    add bx, 2bh  
    mov ah, 4ah  
    int 21h
```

```
    jnc END_FREE_MEMORY
```

```
    lea dx, memoryDestroyed
    cmp ax, 7
    je WRITE_MEMORY_COMMENT
    lea dx, memoryLowMemFunc
    cmp ax, 8
    je WRITE_MEMORY_COMMENT
    lea dx, memoryWrongMemAdr
    cmp ax, 9
    je WRITE_MEMORY_COMMENT
    jmp END_FREE_MEMORY
```

WRITE_MEMORY_COMMENT:

```
    mov ah, 09h
    int 21h
```

END_FREE_MEMORY:

```
    pop dx
    pop cx
    pop bx
    pop ax
    ret
```

FREE_MEMORY ENDP

SET_FULL_FILENAME PROC NEAR

```
    push ax
    push bx
    push cx
    push dx
    push di
    push si
    push es
```

```
    mov prog, dx
```

```
    mov ax, pspKeep
    mov es, ax
    mov es, es:[2ch]
    mov bx, 0
```


FIND_SMTH:

```
    inc bx
    cmp byte ptr es:[bx-1], 0
    jne FIND_SMTH
    cmp byte ptr es:[bx+1], 0
    jne FIND_SMTH
```

```
    add bx, 2
```

```
    mov di, 0
```

FIND_LOOP:

```
    mov dl, es:[bx]
    mov byte ptr [pos + di], dl
    inc di
    inc bx
    cmp dl, 0
    je END_LOOP
    cmp dl, '\'
    jne FIND_LOOP
    mov cx, di
    jmp FIND_LOOP
```

END_LOOP:

```
    mov di, cx
    mov si, prog
```

LOOP_2:

```
    mov dl, byte ptr[si]
    mov byte ptr [pos + di], dl
    inc di
    inc si
    cmp dl, 0
    jne LOOP_2
```

```
    pop es
```

```
    pop si
```

```
    pop di
```

```
    pop dx
```

```
    pop cx
```

```
    pop bx
```

```
    pop ax
    ret
SET_FULL_FILENAME ENDP
```

```
DEPLOY_ANOTHER_PROGRAM PROC NEAR
```

```
    push ax
    push bx
    push cx
    push dx
    push ds
    push es
```

```
    mov ax, DATA
    mov es, ax
    mov bx, offset ovlsAddr
    mov dx, offset pos
    mov ax, 4b03h
    int 21h
```

```
    jnc END_SUCCESS
```

```
ERROR_1:
```

```
    cmp ax, 1
    jne ERROR_2
    mov dx, offset errorWrongFuncNum
    call PRINT
    jmp DEPLOY_END
```

```
ERROR_2:
```

```
    cmp ax, 2
    jne ERROR_5
    mov dx, offset errorMissFile
    call PRINT
    jmp DEPLOY_END
```

```
ERROR_5:
```

```
    cmp ax, 5
    jne ERROR_8
    mov dx, offset errorDisk
```

```
call PRINT
jmp DEPLOY_END
```

ERROR_8:

```
cmp ax, 8
jne ERROR_10
mov dx, offset errorNotEnoughMem
call PRINT
jmp DEPLOY_END
```

ERROR_10:

```
cmp ax, 10
jne ERROR_11
mov dx, offset errorWrongStrFormat
call PRINT
jmp DEPLOY_END
```

ERROR_11:

```
cmp ax, 11
mov dx, offset errorWrongFormat
call PRINT
jmp DEPLOY_END
```

END_SUCCESS:

```
mov dx, offset endSuccess
call PRINT
```

```
mov ax, word ptr ovlsAddr
mov es, ax
mov word ptr ovlsAddr, 0
mov word ptr ovlsAddr + 2, ax
```

```
call ovlsAddr
mov es, ax
mov ah, 49h
int 21h
```

DEPLOY_END:

```
pop es
```

```

    pop ds
    pop dx
    pop cx
    pop bx
    pop ax
    ret
DEPLOY_ANOTHER_PROGRAM ENDP

ALLOCATE_MEMORY PROC
    push ax
    push bx
    push cx
    push dx

    push dx
    mov dx, offset dataMem
    mov ah, 1ah
    int 21h
    pop dx
    mov cx, 0
    mov ah, 4eh
    int 21h

    jnc ALLOCATE_SUCCESS

    cmp ax, 2
    je ROUTE_ERR
    mov dx, offset errorMissStrFile
    call PRINT
    jmp ALLOCATE_END

ROUTE_ERR:
    cmp ax, 3
    mov dx, offset ERROR_ROUTE
    call PRINT
    jmp ALLOCATE_END

ALLOCATE_SUCCESS:

```

```

    push di
    mov di, offset dataMem
    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 ovlsAddr, ax
    mov dx, offset allocateSuccessStr
    call PRINT

ALLOCATE_END:
    pop dx
    pop cx
    pop bx
    pop ax
    ret
ALLOCATE_MEMORY ENDP

START_OVL PROC
    push dx
    call SET_FULL_FILENAME
    mov dx, offset pos
    call ALLOCATE_MEMORY
    call DEPLOY_ANOTHER_PROGRAM
    pop dx
    ret
START_OVL ENDP

MAIN PROC FAR
    push ds

```

```

    xor    ax, ax
    push  ax
    mov    ax, DATA
    mov    ds, ax
    mov    pspKeep, es
    call   FREE_MEMORY
    mov    dx, offset nameOverlay1
    call   START_OVL
    mov    dx, offset eof
    call   PRINT
    mov    dx, offset nameOverlay2
    call   START_OVL

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

MAIN ENDP

END_APP:

CODE ENDS
    END MAIN

```

Название файла: overlay1.asm

```

OVL1 SEGMENT
    ASSUME CS:OVL1, DS:NOTHING, SS:NOTHING
    MAIN PROC FAR
        push ax
        push dx
        push ds
        push di

        mov ax, cs
        mov ds, ax
        mov di, offset OvlAddr
        add di, 23
        call WRD_TO_HEX

```

```
    mov dx, offset OvlAddr
    call PRINT
```

```
    pop di
    pop ds
    pop dx
    pop ax
    retf
```

```
MAIN endp
```

```
OvlAddr db 13, 10, "OVL1 address:           ", 13, 10, '$'
```

```
PRINT PROC
    push dx
    push ax
    mov ah, 09h
    int 21h
    pop ax
    pop dx
    ret
```

```
PRINT ENDP
```

```
TETR_TO_HEX PROC
    and al, 0fh
    cmp al, 09
    jbe NEXT
    add al, 07
```

```
NEXT:
    add al, 30h
    ret
```

```
TETR_TO_HEX ENDP
```

```
BYTE_TO_HEX PROC
    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
        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

```

OVL1 ENDS

END MAIN

Название файла: overlay2.asm

```

OVL2 SEGMENT
        ASSUME CS:OVL2, DS:NOTHING, SS:NOTHING
        MAIN PROC FAR
                push ax
                push dx
                push ds
                push di

```



```
    mov ax, cs
    mov ds, ax
    mov di, offset ovlAddr
    add di, 23
    call WRD_TO_HEX
    mov dx, offset ovlAddr
    call PRINT
```

```
    pop di
    pop ds
    pop dx
    pop ax
    retf
```

```
MAIN ENDP
```

```
ovlAddr db 13, 10, "OVL2 address:           ", 13, 10, '$'
```

```
PRINT PROC
```

```
    push dx
    push ax
    mov ah, 09h
    int 21h
    pop ax
    pop dx
    ret
```

```
PRINT ENDP
```

```
TETR_TO_HEX PROC
```

```
    and al, 0fh
    cmp al, 09
    jbe NEXT
    add al, 07
NEXT:
    add al, 30h
    ret
```

```
TETR_TO_HEX ENDP
```

```

BYTE_TO_HEX PROC
    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
    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

```

```

OVL2 ENDS

```

```

END MAIN

```