

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. 8383

Федоров И.А.

Преподаватель

Ефремов М.А.

Санкт-Петербург

2020

Цель работы.

Исследование возможности построения загрузочного модуля оверлейной структуры. Исследование структуры оверлейного сегмента и способа загрузки и выполнения оверлейных сегментов.

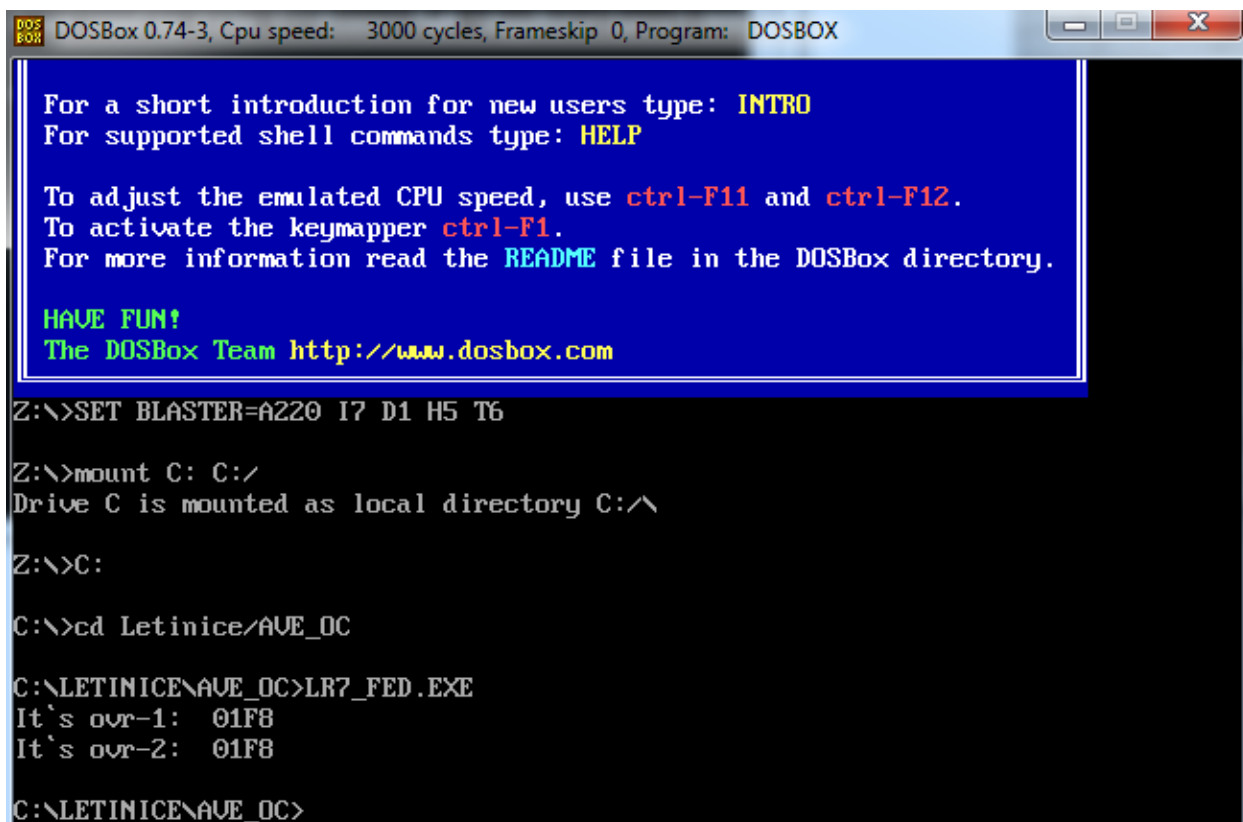
Ход работы.

Был написан и отлажен программный модуль типа **.EXE**, который выполняет следующие 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.

Были написаны и отлажены оверлейные сегменты, которые выводят адрес сегмента, в которые они загружены. Исходный код оверлейных сегментов приведен в приложении Б. Исходный код **.EXE** модуля приведен в приложении А.

Отлаженная программа была запущена, когда текущим каталогом является каталог с разработанным модулем и оверлейными сегментами, которые выводят свой сегментный адрес. Результат работы программы представлен на рис. 1.

A screenshot of a DOSBox 0.74-3 window. The title bar shows 'DOS BOX' and 'DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip 0, Program: DOSBOX'. The main window has a blue background with white text. It displays instructions for new users, including typing 'INTRO' for an introduction and 'HELP' for shell commands. It also mentions adjusting CPU speed with 'ctrl-F11' and 'ctrl-F12', and activating the keymapper with 'ctrl-F1'. A green box highlights the text 'HAVE FUN! The DOSBox Team http://www.dosbox.com'. Below this, the command prompt shows the following sequence: 'Z:\>SET BLASTER=A220 I7 D1 H5 T6', 'Z:\>mount C: C:/', 'Drive C is mounted as local directory C:/\', 'Z:\>C:', 'C:\>cd Letinice\AUE_OC', 'C:\LETINICE\AUE_OC>LR7_FED.EXE', 'It's ovr-1: 01F8', 'It's ovr-2: 01F8', and 'C:\LETINICE\AUE_OC>'.

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

For a short introduction for new users type: INTRO
For supported shell commands type: HELP

To adjust the emulated CPU speed, use ctrl-F11 and ctrl-F12.
To activate the keymapper ctrl-F1.
For more information read the README file in the DOSBox directory.

HAVE FUN!
The DOSBox Team http://www.dosbox.com

Z:\>SET BLASTER=A220 I7 D1 H5 T6

Z:\>mount C: C:/
Drive C is mounted as local directory C:/\

Z:\>C:

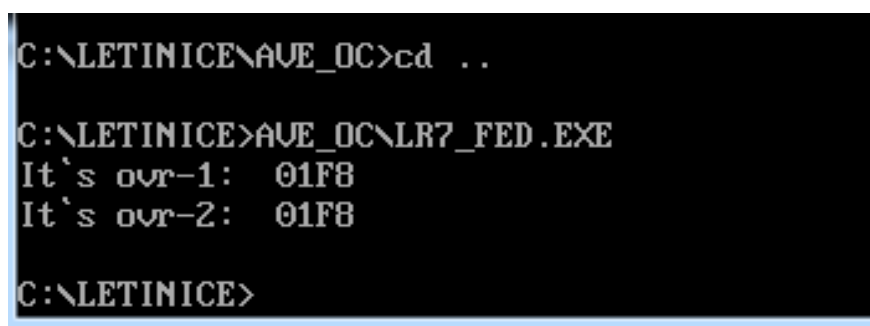
C:\>cd Letinice\AUE_OC

C:\LETINICE\AUE_OC>LR7_FED.EXE
It's ovr-1: 01F8
It's ovr-2: 01F8

C:\LETINICE\AUE_OC>
```

Рисунок 1 – Результат работы программы

Программа была запущена при условии, что текущим является другой каталог. Программа успешно выполняется. Результат работы программы представлен на рис. 2.

A screenshot of a DOSBox command prompt window. It shows the following sequence of commands and output: 'C:\LETINICE\AUE_OC>cd ..', 'C:\LETINICE>AUE_OC\LR7_FED.EXE', 'It's ovr-1: 01F8', 'It's ovr-2: 01F8', and 'C:\LETINICE>'.

```
C:\LETINICE\AUE_OC>cd ..

C:\LETINICE>AUE_OC\LR7_FED.EXE
It's ovr-1: 01F8
It's ovr-2: 01F8

C:\LETINICE>
```

Рисунок 2 – Результат при вызове из другого каталога

Программа была запущена при условии, что одного из оверлеев нет в каталоге. Приложение выводит предупреждающее сообщение. Результат работы приведен на рис. 3.

```
DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip 0, Program: DOSBOX
Directory of C:\LETINICE\AUE_OC\PAKKA2\
.                <DIR>                01-05-2020 20:20
..               <DIR>                01-05-2020 20:20
EDIT            COM                   69,886 21-03-2013 19:10
LR7_FED / EXE    1,803 01-05-2020 19:18
LR_2_6          COM                   436 21-04-2020 19:31
OUR2 ] COM      111 01-05-2020 19:47
    4 File(s)          72,236 Bytes.
    2 Dir(s)          262,111,744 Bytes free.

C:\LETINICE\AUE_OC\PAKKA2>cd ..

C:\LETINICE\AUE_OC>cd PAKKA

C:\LETINICE\AUE_OC\PAKKA>dir
Directory of C:\LETINICE\AUE_OC\PAKKA\
.                <DIR>                01-05-2020 20:18
..               <DIR>                01-05-2020 20:20
FK              ASM                   8,231 23-04-2020 10:42
LR_6_FED EXE     1,518 21-04-2020 20:46
OUR1 ] COM      111 01-05-2020 19:47
    3 File(s)          9,860 Bytes.
    2 Dir(s)          262,111,744 Bytes free.

C:\LETINICE\AUE_OC\PAKKA2>LR7_FED.EXE
Path not found
Path not found
It's ovr-2: 01F8

C:\LETINICE\AUE_OC\PAKKA2>_
```

Рисунок 3 – Вызов программы при отсутствии одного из оверлеев

Ответы на контрольные вопросы.

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

Т.к. **.COM** модули включают в себе PSP, то обращение к оверлею должно происходить со смещением 100h. (В данной работе в качестве оверлеев использовались файлы формата **.COM**, однако они не являются **.COM** модулями по структуре).

Выводы.

В ходе выполнения работы была исследована возможность построения модуля оверлейной структуры. Были разобраны структура оверлейного сегмента и способ загрузки и выполнения оверлейных сегментов.

ПРИЛОЖЕНИЕ А

КОД ПРОГРАММЫ

DATA SEGMENT

```
CODE_ db ' ', 13,10,'$'
OVERLAY_SEG DW 0
OVERLAY_OFFSET DW 0
CODE_SEG DW 0
BLOCK DD 0
KEEP_SS DW 0
KEEP_SP DW 0
KEEP_PSP DW 0

FILENAME1 DB 'OVR1.COM',0
FILENAME2 DB 'OVR2.COM',0
FILEPATH DB 128 DUP(0)
PARAMETERS DW 7 DUP(0)
CURRENT_FILENAME DW 0
MEM_FLAG DB 1

; Load programm errors (CF = 1)
LOAD_ERR_1 db 'Nonexistent function',13,10,'$'
LOAD_ERR_2 db 'File not found',13,10,'$'
LOAD_ERR_3 db 'Path not found',13,10,'$'
LOAD_ERR_4 db 'Too many open files',13,10,'$'
LOAD_ERR_5 db 'No access',13,10,'$'
LOAD_ERR_8 db 'Out of memory',13,10,'$'
LOAD_ERR_10 db 'Invalid enviroment',13,10,'$'

; Succes load (CF = 0)
NORMAL_CODE_0 db 'Normal end: ',13,10,'$'
NORMAL_CODE_1 db 'Ctrl-Breal end',13,10,'$'
NORMAL_CODE_2 db 'Device error',13,10,'$'
NORMAL_CODE_3 db '31H End',13,10,'$'

; Memory error
MEMORY_ERR_7 db 'Memory block was destroyed',13,10,'$'
MEMORY_ERR_8 db 'Out of memory to function',13,10,'$'
MEMORY_ERR_9 db 'Invalid memory block`s address',13,10,'$'

DETERM_FILE_SIZE_ERR2 db 'File not found',13,10,'$'
DETERM_FILE_SIZE_ERR3 db 'Path not found',13,10,'$'

STR_GETSIZE_ERROR db 13, 10, "Size of the ovl wasn't get$"
STR_GS_ERROR2 db 13, 10, "File wasn't found$"
STR_GS_ERROR3 db 13, 10, "Path wasn't found$"
OVL1_NAME db "ovl1.ovl", 0
OVL2_NAME db "ovl2.ovl", 0
DTA db 43 dup(0)
END_STR EQU '$'
```

```

        dsize=$-CODE_           ;размер сегмента данных
DATA ENDS

```

```

        astack segment stack
        dw 64 dup(?)
        astack ends

```

```

        code segment
        assume CS:CODE, DS:DATA, ss:astack

```

```

CODES:

```

```

;-----
WriteMsg PROC NEAR
    push ax
    mov ah,09h
    int 21h
    pop ax
    ret
WriteMsg ENDP
;-----

```

```

;-----
BYTE_TO_DEC PROC near
; перевод в 10с/с, SI - адрес поля 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
;-----

```

```

;-----
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 шестн. 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

```

```

;-----
PREPARE_MEMORY_SPACE PROC NEAR
    PUSH AX
    PUSH BX
    PUSH DX
    MOV CODE_SEG, CS
    MOV BX,((csize/16)+1)+256/16+((dsize/16)+1)+200/16 ; перевод
в параграфы /16
    MOV AH,4Ah ; psp +
64*2 stack
    int 21h
    JC MEMORY_ERROR ;проверяем
на ошибку CF = 1
    MOV MEM_FLAG, 0
    JMP END_MEM_SPACE
MEMORY_ERROR:
    cmp ax, 7

```

```

        Jne else_if_1
        mov dx, offset MEMORY_ERR_7
        jmp print_err
else_if_1:
        cmp ax, 8
        jne else_if_2
        mov dx, offset MEMORY_ERR_8
        jmp print_err
else_if_2:
        mov dx, offset MEMORY_ERR_9
        jmp print_err
print_err:
        call WriteMsg
        mov MEM_FLAG, -1

END_MEM_SPACE:
        POP DX
        POP BX
        POP AX
        ret
PREPARE_MEMORY_SPACE ENDP
;-----

```

```

;-----
GET_FILEPATH PROC NEAR
        PUSH DX
        PUSH AX
        PUSH DI
        PUSH SI
        PUSH ES

        mov KEEP_PSP, es
        mov es, es:[2CH]
        xor si, si
while_not_path:
        mov ax, es:[si]
        inc si
        cmp ax, 0
        jne while_not_path
        inc si
        inc si
        inc si
        xor di, di

while_path:
        mov dl, es:[si]
        cmp dl, 0
        je rewrite_name

```



```

        mov FILEPATH[di], dl
        inc SI
        inc di
        jmp while_path

rewrite_name:
        dec di
        cmp FILEPATH[di], '\'
        je filename__
        jmp rewrite_name

filename__:
        inc di
        xor si, si
        mov bx, CURRENT_FILENAME
while_filename:
        mov dl, [bx+si]
        mov FILEPATH[di], dl
        cmp dl, 0
        je end_filepath
        inc si
        inc di
        jmp while_filename

end_filepath:
        POP ES
        POP SI
        POP DI
        POP AX
        POP DX
        ret
GET_FILEPATH ENDP
;-----

```

```

;-----
LOAD_OVERLAY PROC NEAR
        PUSH DX
        PUSH AX
        PUSH DI
        PUSH SI
        PUSH ES
        MOV KEEP_PSP, ES
        MOV CODE_SEG, CS
        MOV AH, 1Ah
        MOV DX, offset DTA
;установить адрес DTA

```

```

INT 21h
    MOV CX, 0
    MOV AH, 4Eh                ;Найти 1-й совпадающий файл
    MOV DX, offset FILEPATH    ;AX = код ошибки если CF установлен
    INT 21h
    JC ERROR_1
    MOV SI, offset DTA
    MOV DX, word ptr[SI+1CH]
    MOV AX, word ptr[SI+1AH]
    MOV BX, 10h                ; 16
    DIV BX                      ;(dx ax)/num
    MOV BX, AX
    ADD BX, 1
    MOV AH, 48h
    INT 21h
    JC ERROR_2
    MOV OVERLAY_SEG, AX
    ;-----
    MOV AX, CODE_SEG
    MOV BX, OVERLAY_SEG
    SUB BX, AX
    MOV CL, 4
    SHL BX, CL
    MOV OVERLAY_OFFSET, BX
    ;-----
    PUSH ES
    PUSH DS
    POP ES
    MOV BX, OFFSET OVERLAY_SEG
    MOV DX, offset FILEPATH
    MOV AH, 4BH
    MOV AL, 3
    INT 21H
    POP ES
    JC LOAD_ERROR
    MOV ES, AX
    MOV AX, OVERLAY_SEG
    MOV ES, AX
    MOV WORD PTR [BLOCK+2], AX
    CALL DWORD PTR BLOCK
    MOV ES, AX
    MOV AH, 49h
    INT 21h
    JC ERROR_2
    JMP END_LOAD_OVERLAY

ERROR_1:
code_2:
    CMP AX, 2
    JNE code_3
    MOV DX, offset DETERM_FILE_SIZE_ERR2

```

```

        CALL WriteMsg
        JMP PRINT_MSG
code_3:
        MOV DX, offset DETERM_FILE_SIZE_ERR3
        CALL WriteMsg
        JMP PRINT_MSG
ERROR_2:
code_7:
        CMP AX, 7
        JNE code_8
        MOV DX, offset MEMORY_ERR_7
        JMP PRINT_MSG
code_8:
        CMP AX, 8
        JNE code_8
        MOV DX, offset MEMORY_ERR_8
        JMP PRINT_MSG
code_9:
        MOV DX, offset MEMORY_ERR_9
        JMP PRINT_MSG

LOAD_ERROR:
load_1:
        CMP AX, 1
        JNE load_2
        MOV DX, offset LOAD_ERR_1
        JMP PRINT_MSG
load_2:
        CMP AX, 2
        JNE load_3
        MOV DX, offset LOAD_ERR_2
        JMP PRINT_MSG
load_3:
        CMP AX, 3
        JNE load_4
        MOV DX, offset LOAD_ERR_3
        JMP PRINT_MSG
load_4:
        CMP AX, 4
        JNE load_5
        MOV DX, offset LOAD_ERR_4
        JMP PRINT_MSG
load_5:
        CMP AX, 5
        JNE load_8
        MOV DX, offset LOAD_ERR_5
        JMP PRINT_MSG
load_8:
        CMP AX, 8
        JNE load_10
        MOV DX, offset LOAD_ERR_8

```

```

        JMP PRINT_MSG
load_10:
        MOV DX, offset LOAD_ERR_10
        JMP PRINT_MSG

PRINT_MSG:
        CALL WriteMsg
        JMP END_LOAD_OVERLAY

END_LOAD_OVERLAY:
        POP ES
        POP SI
        POP DI
        POP AX
        POP DX
        ret
LOAD_OVERLAY ENDP
;-----

```

```

Main PROC FAR
        mov BX, DS
        mov AX, DATA
        mov DS, AX
        mov KEEP_PSP, BX
        call PREPARE_MEMORY_SPACE
        cmp MEM_FLAG, -1
        je END_MAIN
        mov ax, offset FILENAME1
        mov CURRENT_FILENAME, ax
        call GET_FILEPATH
        call LOAD_OVERLAY
        mov ax, offset FILENAME2
        mov CURRENT_FILENAME, ax
        call GET_FILEPATH
        call LOAD_OVERLAY

END_MAIN:
        mov AH, 4Ch
        int 21h
csize=$-CODES
Main ENDP
CODE ENDS
;ZSEG SEGMENT ;фиктивный сегмент
;ZSEG ENDS
END MAIN

```