

**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

\_\_\_\_\_

Дейнега В.Е.

Преподаватель

\_\_\_\_\_

Ефремов М.А.

Санкт-Петербург

2020

### **Цель работы.**

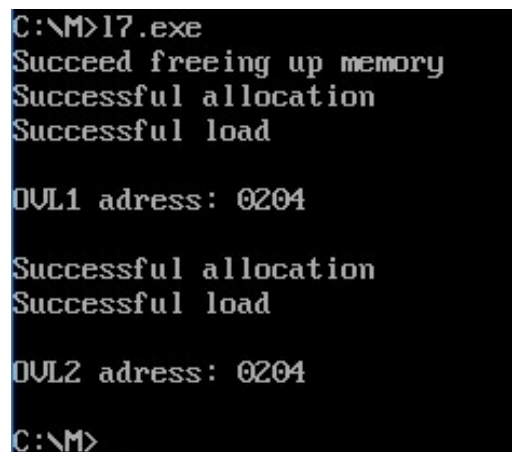
Исследование возможности построения загрузочного модуля оверлейной структуры.

### **Ход работы.**

Был написан 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.



```
C:\M>17.exe
Succeed freeing up memory
Successful allocation
Successful load

OUL1 adress: 0204

Successful allocation
Successful load

OUL2 adress: 0204

C:\M>
```

Рисунок 1 – Результат выполнения с оверлеями

Результат запуска отлаженной программы в другой директории, представлен на рис. 2.

```
C:\N>I7.exe
Succeed freeing up memory
Successful allocation
Successful load

OVL1 adress: 0204

Successful allocation
Successful load

OVL2 adress: 0204

C:\N>
```

Рисунок 2 – Результат выполнения в другой директории

Результаты запуска отлаженной программы, когда один из оверлеев не находится в директории, представлен на рис. 3, 4 и 5 соответственно.

```
C:\M>I7.exe
Succeed freeing up memory
ERROR 2 - file not found

Successful allocation
Successful load

OVL2 adress: 0204

C:\M>
```

Рисунок 3 – Результат выполнения без 1-го оверлея

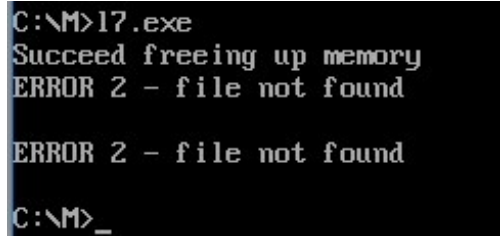
```
C:\M>I7.exe
Succeed freeing up memory
Successful allocation
Successful load

OVL1 adress: 0204

ERROR 2 - file not found

C:\M>
```

Рисунок 4 – Результат выполнения без 2-го оверлея



```
C:\M>17.exe
Succeed freeing up memory
ERROR 2 - file not found

ERROR 2 - file not found

C:\M>_
```

Рисунок 5 – Результат выполнения без обоих оверлеев

### **Контрольные вопросы.**

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

Так как в .COM модулях код располагается с адреса 100h, то оверлейный сегмент необходимо вызывать по смещению 100h, иначе PSP не будет сформирован.

### **Выводы.**

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

## ПРИЛОЖЕНИЕ А

### СОДЕРЖИМОЕ L7.ASM

```
DATA SEGMENT
    FILENAME_1 db "OVL1.OVL", 0
    FILENAME_2 db "OVL2.OVL", 0
    PROG_OFFSET dw 0
    FULL_PATH db 128 dup(0)
    STR_MEM_ERR7 db "ERROR 7 - MCB destroyed!",10,13,"$"
    STR_MEM_ERR8 db "ERROR 8 - not enough memory!",10,13,"$"
    STR_MEM_ERR9 db "ERROR 9 - wrong memory block address!",10,13,"$"
    STR_MEM_OK db "Succeed freeing up memory",10,13,"$"
    ENDL db 13, 10, "$"
    STR_SIZE_SUCCESS db "Successful allocation",10,13,"$"
    STR_SIZE_ERROR2 db "ERROR 2 - file not found",10,13,"$"
    STR_SIZE_ERROR3 db "ERROR 3 - route not found",10,13,"$"

    STR_LOAD_SUCCESS db "Successful load",10,13,"$"
    STR_LOAD_ERROR1 db "ERROR 1 - wrong function number",10,13,"$"
    STR_LOAD_ERROR2 db "ERROR 2 - file not found",10,13,"$"
    STR_LOAD_ERROR3 db "ERROR 3 - route not found",10,13,"$"
    STR_LOAD_ERROR4 db "ERROR 4 - too many open files",10,13,"$"
    STR_LOAD_ERROR5 db "ERROR 5 - access denied",10,13,"$"
    STR_LOAD_ERROR8 db "ERROR 8 - not enough memory",10,13,"$"
    STR_LOAD_ERROR10 db "ERROR 10 - wrong environment",10,13,"$"

    MEMORY_FOR_DTA db 43 dup(?)
    OVERLAY_SEG_ADDRESS dd 0

    KEEP_PSP dw 0

    END_OF_DATA db 0
DATA ENDS

AStack SEGMENT STACK
    DW 200 DUP(?)
AStack ENDS

CODE SEGMENT

    ASSUME CS:CODE, DS:DATA, ES:NOTHING, SS:AStack

    WRITE_STR PROC NEAR
        push DX
        push AX

        mov AH, 09h
        int 21h

        pop AX
        pop DX
        ret
    WRITE_STR ENDP

    FREE_MEMORY PROC NEAR
        push BX
```

```

push DX

mov BX, offset END_OF_PROG
mov AX, offset END_OF_DATA

add BX, AX
shr BX, 1
shr BX, 1
shr BX, 1
shr BX, 1
shr BX, 1
add BX, 2Bh
mov AH, 4Ah
int 21h
jnc MEM_OK
cmp AX, 7
je MEM_ERR7
cmp AX, 8
je MEM_ERR8
cmp AX, 9
je MEM_ERR9
MEM_ERR7:
    mov DX, offset STR_MEM_ERR7
    jmp MEM_FAIL
MEM_ERR8:
    mov DX, offset STR_MEM_ERR8
    jmp MEM_FAIL
MEM_ERR9:
    mov DX, offset STR_MEM_ERR9
    jmp MEM_FAIL

MEM_OK:
    mov AX, 1
    mov DX, offset STR_MEM_OK
    call WRITE_STR
    jmp FREE_MEMORY_END

MEM_FAIL:
    mov AX, 0
    call WRITE_STR

FREE_MEMORY_END:
    pop DX
    pop BX
    ret
FREE_MEMORY ENDP

PATH_MAKE PROC NEAR
    push AX
    push CX
    push BX
    push DI
    push SI
    push ES

    mov PROG_OFFSET, DX

```

```

mov AX, KEEP_PSP
mov ES, AX
mov ES, ES:[2Ch]
mov BX, 0

print_env_variable:
    cmp BYTE PTR ES:[BX], 0
    je variable_end
    inc BX
    jmp print_env_variable
variable_end:
    inc BX
    cmp BYTE PTR ES:[BX+1], 0
    jne print_env_variable

add BX, 2
mov DI, 0

MARK:
    mov DL, ES:[BX]
    mov BYTE PTR [FULL_PATH+DI], DL
    inc BX
    inc DI
    cmp DL, 0
    je loop_end
    cmp DL, '\\'
    jne MARK
    mov CX, DI
    jmp MARK
loop_end:
mov DI, CX
mov SI, PROG_OFFSET
filename_loop:
    mov DL, BYTE PTR [SI]
    mov BYTE PTR [FULL_PATH+DI], DL
    inc DI
    inc SI
    cmp DL, 0
    jne filename_loop

    pop ES
    pop SI
    pop DI
    pop BX
    pop CX
    pop AX

ret
PATH_MAKE ENDP

MEM_ALLOCATE PROC
    push BX
    push CX
    push DX

    push DX

```

```

mov DX, offset MEMORY_FOR_DTA
mov AH, 1Ah
int 21h

pop DX
mov CX, 0
mov AH, 4Eh
int 21h

jnc SIZE_SUCCES

cmp AX, 2
jmp SIZE_ERROR2
cmp AX, 3
jmp SIZE_ERROR3

SIZE_ERROR2:
    mov DX, offset STR_SIZE_ERROR2
    jmp SIZE_FAIL
SIZE_ERROR3:
    mov DX, offset STR_SIZE_ERROR3
    jmp SIZE_FAIL

SIZE_SUCCES:
    push DI
    mov DI, offset MEMORY_FOR_DTA
    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 OVERLAY_SEG_ADDRESS, AX
    mov DX, offset STR_SIZE_SUCCESS
    call WRITE_STR
    mov AX, 1
    jmp SIZE_END

SIZE_FAIL:
    mov AX, 0
    call WRITE_STR

SIZE_END:
    pop DX
    pop CX
    pop BX

ret
MEM_ALLOCATE ENDP

```



```

LOAD PROC
    push AX
    push BX
    push CX
    push DX
    push DS
    push ES

    mov AX, DATA
    mov ES, AX
    mov DX, offset FULL_PATH
    mov BX, offset OVERLAY_SEG_ADDRESS
    mov AX, 4B03h
    int 21h

    jnc LOAD_SUCCESS

    cmp AX, 1
    jmp LOAD_ERROR1
    cmp AX, 2
    jmp LOAD_ERROR2
    cmp AX, 3
    jmp LOAD_ERROR3
    cmp AX, 4
    jmp LOAD_ERROR4
    cmp AX, 5
    jmp LOAD_ERROR5
    cmp AX, 8
    jmp LOAD_ERROR8
    cmp AX, 10
    jmp LOAD_ERROR10

LOAD_ERROR1:
    mov DX, offset STR_LOAD_ERROR1
    jmp LOAD_FAIL
LOAD_ERROR2:
    mov DX, offset STR_LOAD_ERROR2
    jmp LOAD_FAIL
LOAD_ERROR3:
    mov DX, offset STR_LOAD_ERROR3
    jmp LOAD_FAIL
LOAD_ERROR4:
    mov DX, offset STR_LOAD_ERROR4
    jmp LOAD_FAIL
LOAD_ERROR5:
    mov DX, offset STR_LOAD_ERROR5
    jmp LOAD_FAIL
LOAD_ERROR8:
    mov DX, offset STR_LOAD_ERROR8
    jmp LOAD_FAIL
LOAD_ERROR10:
    mov DX, offset STR_LOAD_ERROR10
    jmp LOAD_FAIL

LOAD_SUCCESS:
    mov DX, offset STR_LOAD_SUCCESS

```

```

        call WRITE_STR

        mov AX, WORD PTR OVERLAY_SEG_ADDRESS
        mov ES, AX
        mov WORD PTR OVERLAY_SEG_ADDRESS, 0
        mov WORD PTR OVERLAY_SEG_ADDRESS+2, AX

        call OVERLAY_SEG_ADDRESS
        mov ES, AX
        mov AH, 49h
        int 21h
        jmp LOAD_END

LOAD_FAIL:
        call WRITE_STR

LOAD_END:

        pop ES
        pop DS
        pop DX
        pop CX
        pop BX
        pop AX

        ret
LOAD ENDP

RUN_OVL PROC
        push DX

        call PATH_MAKE

        mov DX, offset FULL_PATH
        call MEM_ALLOCATE
        cmp AX, 1
        jne RUN_OVL_END

        call LOAD

RUN_OVL_END:
        pop DX
        ret
RUN_OVL ENDP

MAIN PROC
        PUSH DS
        SUB AX, AX
        PUSH AX
        MOV AX, DATA
        MOV DS, AX
        mov KEEP_PSP, ES

        call FREE_MEMORY
        cmp AX, 1
        jne MAIN_END

```

```
mov DX, offset FILENAME_1
call RUN_OVL
mov dx, offset ENDL
call WRITE_STR
mov DX, offset FILENAME_2
call RUN_OVL

MAIN_END:
    xor AL, AL
    mov AH, 4Ch
    int 21h
MAIN ENDP
END_OF_PROG:
```

CODE ENDS

END MAIN

## ПРИЛОЖЕНИЕ А

### ОБЕРЛЕИ

#### №1

```
CODE SEGMENT
    ASSUME CS:CODE, DS:NOTHING, SS:NOTHING
    MAIN PROC FAR
        push AX
        push DX
        push DS
        push DI

        mov AX, CS
        mov DS, AX
        mov DI, offset ADDRESS
        add DI, 18
        call WRD_TO_HEX
        MOV DX, offset ADDRESS
        call WRITE_STR

        pop DI
        pop DS
        pop DX
        pop AX
        retf
    MAIN ENDP

    WRITE_STR PROC NEAR
        push DX
        push AX

        mov AH, 09h
        int 21h

        pop AX
        pop DX
        ret
    WRITE_STR 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

```

```
ADDRESS db 13, 10, "OVL1 adress:           ", 13, 10, '$'
```

```
CODE ENDS
END MAIN
```

## №2

```

CODE SEGMENT
    ASSUME CS:CODE, DS:NOTHING, SS:NOTHING
    MAIN PROC FAR
        push AX
        push DX
        push DS
        push DI

        mov AX, CS
        mov DS, AX
        mov DI, offset ADDRESS
        add DI, 18
        call WRD_TO_HEX
        MOV DX, offset ADDRESS
        call WRITE_STR

        pop DI
        pop DS
        pop DX
        pop AX
        retf
    MAIN ENDP

```

```

MAIN ENDP

WRITE_STR PROC NEAR
    push DX
    push AX

    mov AH, 09h
    int 21h

    pop AX
    pop DX
    ret
WRITE_STR 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

```

```
ADDRESS db 13, 10, "OVL2 adress:      ", 13, 10, '$'  
  
CODE ENDS  
END MAIN
```