

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

Мололкин К.А.

Преподаватель

Ефремов М.А.

Санкт-Петербург

2020

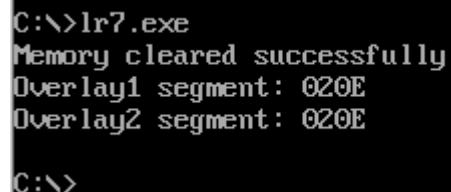
Цель работы

Исследовать возможность построения загрузочного модуля оверлейной структуры.

Ход работы

Первым шагом был написан и отлажен программный модуль типа .EXE, который освобождает память для загрузки оверлеев, читает размер файла оверлея, запрашивает объем памяти, достаточный для его загрузки. Затем загружается и выполняется файл оверлейного сегмента, освобождается память, отведенная для оверлейного сегмента. Таким же образом загружается второй оверлейный модуль. Также были написаны и отлажены оверлейные сегменты, каждый из которых выводит адрес сегмента, в который он загружен.

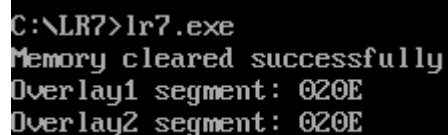
Следующим шагом была запущена отлаженная программа, когда текущим каталогом является каталог с разработанным модулем. Пример выполнения представлен на рис. 1.



```
C:\>lr7.exe
Memory cleared successfully
Overlay1 segment: 020E
Overlay2 segment: 020E
C:\>
```

Рисунок 1 – пример работы программы

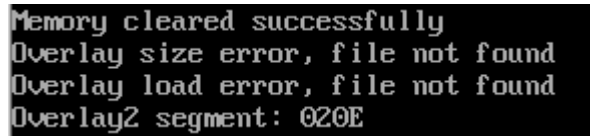
Следующим шагом данная программа была запущена, когда текущим каталогом является какой-либо другой каталог, отличный от того в котором содержатся разработанные программные модули. Результат запуска представлен на рис. 2.



```
C:\LR7>lr7.exe
Memory cleared successfully
Overlay1 segment: 020E
Overlay2 segment: 020E
```

Рисунок 2 – запуск из другого каталога

На завершающем шаге. Программа была запущена, когда один из оверлейных модулей отсутствует в каталоге. Результат на рис. 3. Код программы LR7.ASM представлен в приложении А.



```
Memory cleared successfully
Overlay size error, file not found
Overlay load error, file not found
Overlay2 segment: 020E
```

Рисунок 3 – работа программы с отсутствующим оверлейным модулем.

Контрольные вопросы

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

В данном случае следует обращаться к модулю со смещение 100h.

Вывод

Во время выполнения лабораторной работы была исследована возможность построения загрузочного модуля оверлейной структуры.

Приложение А

```
ASTACK SEGMENT STACK
        dw 128 dup(0)
ASTACK ENDS

DATA SEGMENT
        CLEAR_ERROR_7 db "Memory clear error, control block
destroyed", 13, 10, "$"
        CLEAR_ERROR_8 db "Memory clear error, not enough memory
to execute func", 13, 10, "$"
        CLEAR_ERROR_9 db "Memory clear error, incorrect memory
block address", 13, 10, "$"
        OVERLAY_1_PATH db 'OVERLAY1.OVL$'
        OVERLAY_2_PATH db 'OVERLAY2.OVL$'
        CALLED_PROG_PATH db 50 dup(0)
        OFFSET_OVL_NAME dw 0
        BUFFER db 43 dup(0)
        KEEP_SEG dw 0
        OVERLAY_PATH dd 0
        IS_SUCCESS_CLEAR db 0
        SUCCESS_CLEAR_MEM db "Memory cleared successfully", 13,
10, "$"
        NOT_COMPLETED_OVERLAY_SIZE db "Overlay size wasn't
detected", 13, 10, "$"
        OVERLAY_SIZE_ERROR db "Overlay size error", 13, 10, "$"
        OVERLAY_SIZE_ERROR_2 db "Overlay size error, file not
found", 13, 10, "$"
        OVERLAY_SIZE_ERROR_3 db "Overlay size error, rout not
found", 13, 10, "$"
        LOAD_ERROR db "Overlay load error", 13, 10, "$"
        LOAD_ERROR_1 db "Overlay load error, non-existent
function", 13, 10, "$"
        LOAD_ERROR_2 db "Overlay load error, file not found", 13,
10, "$"
        LOAD_ERROR_3 db "Overlay load error, rout not found", 13,
10, "$"
        LOAD_ERROR_4 db "Overlay load error, too many open
files", 13, 10, "$"
        LOAD_ERROR_5 db "Overlay load error, no access", 13, 10,
"$"
        LOAD_ERROR_8 db "Overlay load error, not enough memory",
13, 10, "$"
        LOAD_ERROR_10 db "Overlay load error, wrong environment",
13, 10, "$"
        DATA_ENDD db 0
DATA ENDS

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

CLEAR_MEMORY PROC near
        push ax
```

```

    push bx
    push cx
    push dx

    mov bx, offset END_PART
    mov ax, offset DATA_ENDD
    add bx, ax
    add bx, 30Fh
    mov cl, 4
    shr bx, cl
    mov ax, 4A00h
    int 21h
    jnc GOOD_END

    mov IS_SUCCESS_CLEAR, 0
    cmp ax, 7
    je C_ERROR_7
    cmp ax, 8
    je C_ERROR_8
    cmp ax, 9
    je C_ERROR_9

C_ERROR_7:
    mov dx, offset CLEAR_ERROR_7
    call PRINT_STRING
    jmp END_CLEAR
C_ERROR_8:
    mov dx, offset CLEAR_ERROR_8
    call PRINT_STRING
    jmp END_CLEAR
C_ERROR_9:
    mov dx, offset CLEAR_ERROR_9
    call PRINT_STRING
    jmp END_CLEAR

GOOD_END:
    mov dx, offset SUCCESS_CLEAR_MEM
    call PRINT_STRING
    mov IS_SUCCESS_CLEAR, 1

END_CLEAR:
    pop dx
    pop cx
    pop bx
    pop ax
    ret
CLEAR_MEMORY ENDP

PRINT_STRING PROC near
    push ax
    mov ah, 09h
    int 21h
    pop ax

```

```

        ret
PRINT_STRING ENDP

NEW_COMMAND_LINE PROC near
    push ax
    push di
    push si
    push es
    mov es, es:[2Ch]
    xor di, di

NEXT_0:
    mov al, es:[di]
    cmp al, 0
    je FIRST_0
    inc di
    jmp NEXT_0

FIRST_0:
    inc di
    mov al, es:[di]
    cmp al, 0
    jne NEXT_0
    add di, 3h
    mov si, 0

PRINT_NUMBER:
    mov al, es:[di]
    cmp al, 0
    je WRITE_PATH
    mov CALLED_PROG_PATH[si], al
    inc di
    inc si
    jmp PRINT_NUMBER

WRITE_PATH:
    dec si
    cmp CALLED_PROG_PATH[si], '\'
    je COMPLETE_PATH
    jmp WRITE_PATH

COMPLETE_PATH:
    mov di, -1

COMPLETE_NAME:
    inc si
    inc di
    mov al, bx[di]
    cmp al, "$"
    je END_NEW_COMMAND_LINE
    mov CALLED_PROG_PATH[si], al
    jmp COMPLETE_NAME

```

```

END_NEW_COMMAND_LINE:
    pop es
    pop si
    pop di
    pop ax
    ret
NEW_COMMAND_LINE ENDP

LOAD_OVERLAY PROC near
    push ax
    push bx
    push cx
    push dx
    push si
    push es

    mov ah, 1Ah
    mov dx, offset BUFFER
    int 21h
    mov ah, 4Eh
    mov dx, offset CALLED_PROG_PATH
    mov cx, 0
    int 21h
    jnc FINE_DET

    cmp ax, 2
    je S_ERROR_1
    cmp ax, 3
    je S_ERROR_2

S_ERROR_1:
    mov dx, offset OVERLAY_SIZE_ERROR_2
    call PRINT_STRING
    jmp END_SIZE

S_ERROR_2:
    mov dx, offset OVERLAY_SIZE_ERROR_3
    call PRINT_STRING
    jmp END_SIZE

FINE_DET:
    mov si, offset BUFFER
    add si, 1Ah
    mov bx, [si]
    shr bx, 4
    mov ax, [si + 2]
    shl ax, 12
    add bx, ax
    add bx, 2
    mov ah, 48h
    int 21h
    jnc SEG_KEEP
    mov dx, offset NOT_COMPLETED_OVERLAY_SIZE

```

```

        call PRINT_STRING
        jmp END_SIZE

SEG_KEEP:
        mov KEEP_SEG, ax

END_SIZE:

        mov dx, offset CALLED_PROG_PATH
        push ds
        pop es
        mov bx, offset KEEP_SEG
        mov ax, 4B03h
        int 21h
        jnc GOOD_LOAD

        cmp ax, 1
        je L_ERROR_1
        cmp ax, 2
        je L_ERROR_2
        cmp ax, 3
        je L_ERROR_3
        cmp ax, 4
        je L_ERROR_4
        cmp ax, 5
        je L_ERROR_5
        cmp ax, 8
        je L_ERROR_8
        cmp ax, 10
        je L_ERROR_10

L_ERROR_1:
        mov dx, offset LOAD_ERROR_1
        jmp PRINT_LOAD_ERR

L_ERROR_2:
        mov dx, offset LOAD_ERROR_2
        jmp PRINT_LOAD_ERR

L_ERROR_3:
        mov dx, offset LOAD_ERROR_3
        jmp PRINT_LOAD_ERR

L_ERROR_4:
        mov dx, offset LOAD_ERROR_4
        jmp PRINT_LOAD_ERR

L_ERROR_5:
        mov dx, offset LOAD_ERROR_5
        jmp PRINT_LOAD_ERR

L_ERROR_8:
        mov dx, offset LOAD_ERROR_8

```



```

        jmp PRINT_LOAD_ERR

L_ERROR_10:
        mov dx, offset LOAD_ERROR_10

PRINT_LOAD_ERR:
        call PRINT_STRING
        jmp END_LOAD_OVERLAY

GOOD_LOAD:
        mov ax, KEEP_SEG
        mov es, ax
        mov WORD PTR OVERLAY_PATH + 2, ax
        call OVERLAY_PATH
        mov ah, 49h
        int 21h

END_LOAD_OVERLAY:
        pop es
        pop si
        pop dx
        pop cx
        pop bx
        pop ax
        ret

LOAD_OVERLAY ENDP

MAIN PROC far
        xor ax, ax
        push ax
        mov ax, DATA
        mov ds, ax
        mov bx, ds
        call CLEAR_MEMORY
        cmp IS_SUCCESS_CLEAR, 1
        jne END_MAIN
        mov bx, offset OVERLAY_1_PATH
        call NEW_COMMAND_LINE
        call LOAD_OVERLAY
        mov bx, offset OVERLAY_2_PATH
        call NEW_COMMAND_LINE
        call LOAD_OVERLAY

END_MAIN:
        xor al, al
        mov ah, 4Ch
        int 21h

MAIN ENDP
END_PART:
CODE ENDS
END MAIN

```