

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.0382

Кондратов Ю.А.

Преподаватель

Ефремов М.А.

Санкт-Петербург

2022

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 оверлейные модули находятся в одном каталоге.

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 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.

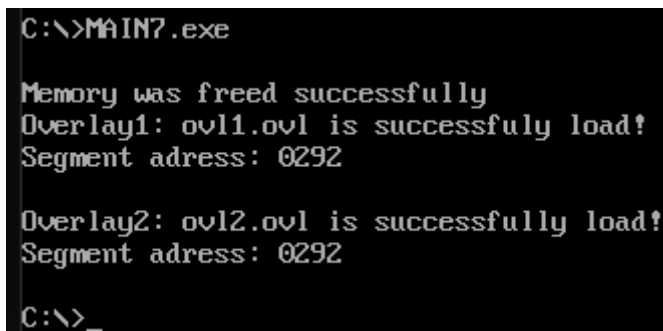
Шаг 5. Запустите приложение в случае, когда одного оверлея нет в каталоге. Приложение должно закончиться аварийно.

Шаг 6. Занесите полученные результаты в виде скриншотов в отчет. Оформите отчет в соответствии с требованиями.

Выполнение работы.

Шаги 1-2. Был написан программный модуль типа .EXE, выполняющий все необходимые функции. Также были написаны и отлажены оверлейные сегменты, которые выводят адрес сегмента, в который они загружены.

Шаг 3. Отлаженное приложение было запущено. Результат запуска представлен на рисунке 1.



```
C:\>MAIN7.exe

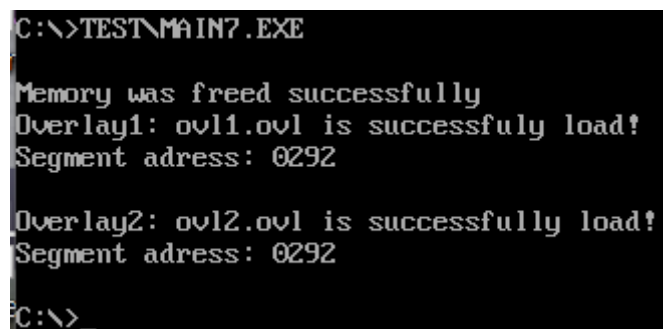
Memory was freed successfully
Overlay1: ovl1.ovl is successfully load!
Segment address: 0292

Overlay2: ovl2.ovl is successfully load!
Segment address: 0292

C:\>_
```

Рисунок 1 – Результат запуска отлаженного приложения

Шаг 4. Отлаженное приложение было запущено из другого каталога. Результат запуска представлен на рисунке 2.



```
C:\>TEST\MAIN7.EXE

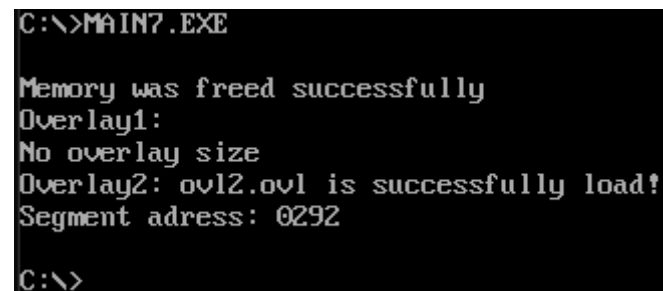
Memory was freed successfully
Overlay1: ovl1.ovl is successfully load!
Segment address: 0292

Overlay2: ovl2.ovl is successfully load!
Segment address: 0292

C:\>_
```

Рисунок 2 – Результат запуска приложения из другого каталога

Шаг 5. Приложение было запущено в случае, когда одного оверлея нет в каталоге. Результат запуска представлен на рисунке 3.



```
C:\>MAIN7.EXE

Memory was freed successfully
Overlay1:
No overlay size
Overlay2: ovl2.ovl is successfully load!
Segment address: 0292

C:\>
```

Рисунок 3 – Результат запуска при отсутствии одного оверлейного сегмента

Ответы на вопросы.

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

Нужно учитывать, что в начале .COM модуля находится PSP, поэтому необходимо использовать смещение 100h.

Выводы.

В ходе работы были исследованы возможности построения, способы загрузки и выполнения оверлейного загрузочного модуля.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main7.asm

```
AStack SEGMENT STACK
    DW 128h DUP (0)
AStack ENDS

DATA SEGMENT
    psp DW 0

    ovl_seg DW 0
    ovl_addr DD 0

    path_err DB 13, 10, "No path$"
    load_err DB 13, 10, "No loaded overlay$"
    ovl1_str DB 13, 10, "Overlay1: $"
    ovl2_str DB 13, 10, "Overlay2: $"
    msg_free_mem DB 13, 10, "Memory was freed successfully$"
    size_err DB 13, 10, "No overlay size$"
    file_err DB 13, 10, "No file$"
    ovl1_name DB "ovl1.ovl", 0
    ovl2_name DB "ovl2.ovl", 0

    path DB 128h DUP(0)

    ovl_name_offset DW 0
    name_pos DW 0
    mem_err DW 0

    data_buf DB 43 DUP(0)
DATA ENDS

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

print PROC    near
    push     AX
    mov     AH, 09h
    int     21h
    pop     AX
    ret
print ENDP

free_mem PROC near
    lea     BX, PROGEND
    mov     AX, ES
    sub     BX, AX
    mov     CL, 8
    shr     BX, CL
```

```

        sub    AX, AX
        mov    AH, 4Ah
        int    21h
        jc     mem_err_catch
        mov    DX, offset msg_free_mem
        call   print
        jmp    plain_end
mem_err_catch:
        mov    mem_err, 1
plain_end:
        ret
free_mem ENDP

```

```

overlay PROC near
    push AX
    push BX
    push CX
    push DX
    push SI

    mov    ovl_name_offset, AX
    mov    AX, psp
    mov    ES, AX
    mov    ES, ES:[2Ch]
    mov    SI, 0
zero:
    mov    AX, ES:[SI]
    inc    SI
    cmp    AX, 0
    jne    zero
    add    SI, 3
    mov    DI, 0
path_writing:
    mov    AL, ES:[SI]
    cmp    AL, 0
    je     pathname_writing
    cmp    AL, '\'
    jne    next_char
    mov    name_pos, DI
next_char:
    mov    BYTE PTR [path + DI], AL
    inc    DI
    inc    SI
    jmp    path_writing
pathname_writing:
    cld
    mov    DI, name_pos
    inc    DI
    add    DI, offset path
    mov    SI, ovl_name_offset

```

```

        mov     AX, DS
        mov     ES, AX
_upd:
        lodsb
        stosb
        cmp     AL, 0
        jne     _upd
        mov     AX, 1A00h
        mov     DX, offset data_buf
        int     21h

        mov     AH, 4Eh
        mov     CX, 0
        mov     DX, offset path
        int     21h

        jnc     no_err
        mov     DX, offset size_err
        call    print
        cmp     AX, 2
        je      no_file
        cmp     AX, 3
        je      no_path
        jmp     path_end
no_file:
        mov     DX, offset file_err
        call    print
        jmp     path_end
no_path:
        mov     DX, offset path_err
        call    print
        jmp     path_end
no_err:
        mov     SI, offset data_buf
        add     SI, 1Ah
        mov     BX, [SI]
        mov     AX, [SI + 2]
        mov     CL, 4
        shr     BX, CL
        mov     CL, 12
        shl     AX, CL
        add     BX, AX
        add     BX, 2
        mov     AX, 4800h
        int     21h

        jnc     set_seg
        jmp     path_end
set_seg:
        mov     ovl_seg, AX
        mov     DX, offset path

```

```

        push     DS
        pop      ES
        mov     BX, offset ovl_seg
mov     AX, 4B03h
        int     21h

        jnc     load_done
        mov     DX, offset load_err
        call    print
        jmp     path_end

load_done:
        mov     AX, ovl_seg
        mov     ES, AX
        mov     WORD PTR ovl_addr + 2, AX
        call    ovl_addr
        mov     ES, AX
        mov     AH, 49h
        int     21h

path_end:
        pop     SI
        pop     DX
        pop     CX
        pop     BX
        pop     AX
        ret
overlay ENDP

main PROC
        mov     AX, DATA
        mov     DS, AX
        mov     psp, ES
        call    free_mem
        cmp     mem_err, 1
        je      main_end
        mov     DX, offset ovl1_str
        call    print
        mov     AX, offset ovl1_name
        call    overlay
        mov     DX, offset ovl2_str
        call    print
        mov     AX, offset ovl2_name
        call    overlay

main_end:
        mov     AX, 4C00h
        int     21h
PROGEND:

main ENDP

```



```
CODE ENDS  
END    main
```