

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. 9383

Моисейченко К. А.

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.

Основные теоретические положения.

Для организации программы, имеющей оверлейную структуру, используется функция 4B03h прерывания int 21h. Эта функция позволяет в отведенную область памяти, начинающуюся с адреса сегмента, загрузить программу, находящуюся в файле на диске. Передача управления загруженной программе этой функцией не осуществляется и префикс сегмента программы (PSP) не создается. Обращение к функции 4B03h:

AX=4B03h - код функции;

DS:DX - указывает на строку ASCIIZ, содержащую путь к оверлею;

ES:BX - указатель на блок параметров, который представляет собой два слова памяти, содержащих сегментный адрес загрузки программы.

Если флаг переноса CF=1 после выполнения функции, то произошли ошибки и регистр AX содержит код ошибки. Значение регистра AX характеризует следующие ситуации:

- 1 - несуществующая функция;
- 2 - файл не найден;
- 3 - маршрут не найден;
- 4 - слишком много открытых файлов;
- 5 - нет доступа;
- 8 - мало памяти;
- 10 - неправильная среда.

Если флаг переноса CF=0, то оверлей загружен в память.

Перед загрузкой оверлея вызывающая программа должна освободить память по функции 4Ah прерывания int 21h. Затем определить размер оверлея. Это можно сделать с помощью функции 4Eh прерывания 21h. Перед обращением к функции необходимо определить область памяти размером в 43 байта под буфер DTA, которую функция заполнит, если файл будет найден.

Функция использует следующие параметры:

CX - значение байта атрибутов, которое для файла имеет значение 0;

DS:DX - указатель на путь к файлу, который записывается в формате строки ASCIIZ.

Если флаг переноса CF=1 после выполнения функции, то произошли ошибки и регистр AX содержит код ошибки. Значение регистра AX характеризует следующие ситуации:

- 2 - файл не найден;
- 3 - маршрут не найден.

Если CF=0, то в области памяти буфера DTA со смещением 1Ah будет находится младшее слово размера файла, а в слове со смещением 1Ch - старшее слово размера памяти в байтах.

Полученный размер файла следует перевести в параграфы, причем следует взять большее целое числа параграфов. Затем необходимо отвести память с помощью функции 48h прерывания 21h. После этого необходимо сформировать параметры для функции 4B03h и выполнить ее.

После отработки оверлея необходимо освободить память с помощью функции 49h прерывания int 21h. Обращение к этой функции содержит следующие параметры:

АН=49h - код функции;

ES - сегментный адрес освобождаемой памяти.

Оверлейный сегмент не является загрузочным модулем типов .COM или .EXE. Он представляет собой кодовый сегмент, который оформляется в ассемблере как функция с точкой входа по адресу 0 и возврат осуществляется командой RETF. Это необходимо сделать, потому что возврат управления должен быть осуществлен в программу, выполняющую оверлейный сегмент. Если использовать функции выхода 4Ch прерывания int 21h, то программа закончит свою работу.

Контрольные вопросы по лабораторной работе.

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

Выполнение работы.

Шаг 1.

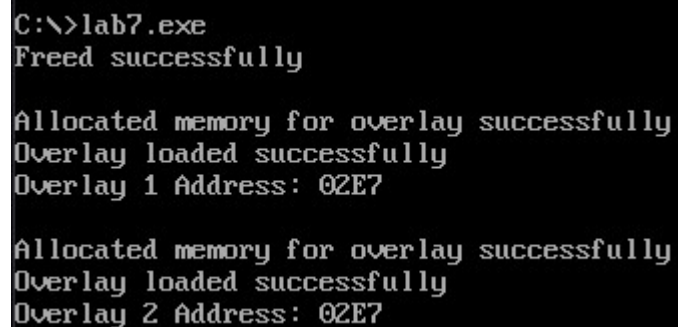
Был написан и отлажен программный модуль типа .EXE, который выполняет поставленные в задании функции.

Шаг 2.

Были написаны и отлажены два оверлейных сегмента. Оверлейный сегмент выводит адрес сегмента, в который он загружен.

Шаг 3.

Отлаженное приложение было запущено. Оверлейные сегменты загружаются с одного и того же адреса, перекрывая друг друга.



```
C:\>lab7.exe
Freed successfully

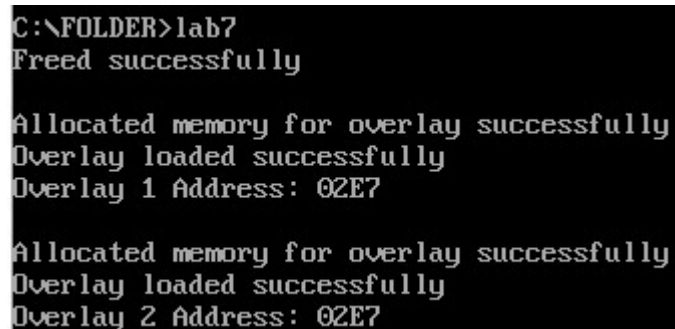
Allocated memory for overlay successfully
Overlay loaded successfully
Overlay 1 Address: 02E7

Allocated memory for overlay successfully
Overlay loaded successfully
Overlay 2 Address: 02E7
```

Рисунок 1 - Иллюстрация работы программы.

Шаг 4.

Приложение было запущено из другого каталога. Приложение было выполнено успешно.



```
C:\FOLDER>lab7
Freed successfully

Allocated memory for overlay successfully
Overlay loaded successfully
Overlay 1 Address: 02E7

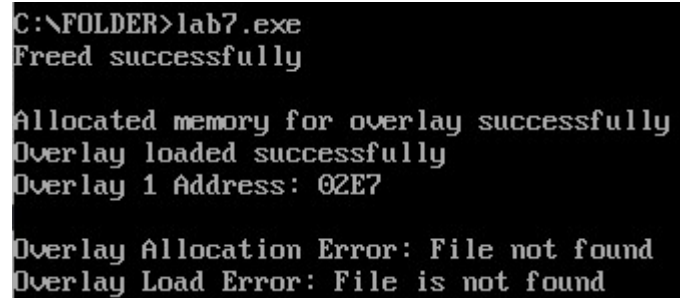
Allocated memory for overlay successfully
Overlay loaded successfully
Overlay 2 Address: 02E7
```

Рисунок 2 - Иллюстрация работы программы, запущенной из другого каталога.

Шаг 5.

Приложение было запущено в случае, когда одного оверлея нет в каталоге.

Приложение закончилось аварийно.



```
C:\FOLDER>lab7.exe
Freed successfully

Allocated memory for overlay successfully
Overlay loaded successfully
Overlay 1 Address: 02E7

Overlay Allocation Error: File not found
Overlay Load Error: File is not found
```

Рисунок 3 - Иллюстрация работы программы, запущенной из каталога, в котором нет одного оверлейного сегмента.

Ответы на контрольные вопросы.

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

В .COM модуле присутствует смещение в 100h, а т.к. оверлейный модуль имеет точку входа по адресу 0, то, при обращении к данным, нужно вычитать это смещение.

Выводы.

Были исследованы возможности построения загрузочного модуля оверлейной структуры, структура оверлейного сегмента и способ загрузки и выполнения оверлейных сегментов. Была написана программа, которая подготавливает место под оверлейные сегменты и успешно вызывает оверлейные сегменты и обрабатывает исключительные ситуации, когда один или оба оверлейных сегмента отсутствуют.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: lab7.asm

```
ASTACK SEGMENT STACK
```

```
    DW 256 DUP(?)
```

```
ASTACK ENDS
```

```
DATA SEGMENT
```

```
    dta_buffer db 43 DUP(0)
```

```
    overlay_addr dd 0
```

```
    overlay_name1 db 'ovl1.ovl', 0h
```

```
    overlay_name2 db 'ovl2.ovl', 0h
```

```
    overlay_path db 128 DUP(0)
```

```
    keep_ss dw 0
```

```
    keep_sp dw 0
```

```
    error_mem_free db 0
```

```
    mcb_crash_string db 'Memory Free Error: Memory Control Block has  
crashed', 0DH, 0AH, '$'
```

```
    not_enough_memory_string db 'Memory Free Error: Not Enough  
Memory', 0DH, 0AH, '$'
```

```
    wrong_address_string db 'Memory Free Error: Wrong Address', 0DH,  
0AH, '$'
```

```
    free_without_error_string db 'Freed successfully', 0DH, 0AH, '$'
```

```
    file_not_found_error_string db 'Overlay Allocation Error: File  
not found', 0DH, 0AH, '$'
```

```
    route_not_found_error_string db 'Overlay Allocation Error: Route  
not found', 0DH, 0AH, '$'
```

```
    allocated_mem_for_overlay_string db 'Allocated memory for  
overlay successfully', 0DH, 0AH, '$'
```

```
    overlay_function_not_exist db 'Overlay Load Error: Function does  
not exist', 0DH, 0AH, '$'
```

```
    overlay_file_not_found db 'Overlay Load Error: File is not  
found', 0DH, 0AH, '$'
```

```
    overlay_route_not_found db 'Overlay Load Error: Route not found',  
0DH, 0AH, '$'
```

```
    overlay_too_many_files_opened db 'Overlay Load Error: Too many  
files opened', 0DH, 0AH, '$'
```

```
    overlay_no_access db 'Overlay Load Error: No access', 0DH, 0AH,  
'$'
```

```
    overlay_not_enough_memory db 'Overlay Load Error: Not enough  
memory', 0DH, 0AH, '$'
```

```
    overlay_wrong_env db 'Overlay Load Error: Wrong environment',  
0DH, 0AH, '$'
```

```
    overlay_load_success db 'Overlay loaded successfully', 0DH, 0AH,  
'$'
```

```
    data_end db 0
```

```
DATA ENDS
```

```
CODE SEGMENT
```

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

```
    WRITEWRD PROC NEAR
```

```
        push ax
```

```

        mov ah, 9
        int 21h
        pop ax
        ret
WRITEWRD ENDP

WRITEBYTE PROC NEAR
        push ax
        mov ah, 02h
        int 21h
        pop ax
        ret
WRITEBYTE ENDP

ENDLINE PROC NEAR
        push ax
        push dx
        mov dl, 0dh
        call WRITEBYTE
        mov dl, 0ah
        call WRITEBYTE
        pop dx
        pop ax
        ret
ENDLINE ENDP

FREE_UNUSED_MEMORY PROC FAR
        push ax
        push bx
        push cx
        push dx
        push es
        xor dx, dx
        mov error_mem_free, 0h
        mov ax, offset data_end
        mov bx, offset lafin
        add ax, bx
        mov bx, 10h
        div bx
        add ax, 100h
        mov bx, ax
        xor ax, ax
        mov ah, 4ah
        int 21h
        jnc free_without_error
mov error_mem_free, 1h
;mcb crash
        cmp ax, 7
        jne not_enough_memory
        mov dx, offset mcb_crash_string
        call WRITEWRD
        jmp free_unused_memory_end
not_enough_memory:
        cmp ax, 8
        jne wrong_address

```



```

        mov dx, offset not_enough_memory_string
        call WRITEWRD
        jmp free_unused_memory_end

wrong_address:
        cmp ax, 9
        jne free_unused_memory_end
        mov dx, offset wrong_address_string
        call WRITEWRD
        jmp free_unused_memory_end

free_without_error:
        mov dx, offset free_without_error_string
        call WRITEWRD

free_unused_memory_end:
        pop es
        pop dx
        pop cx
        pop bx
        pop ax
        ret

FREE_UNUSED_MEMORY ENDP

LOAD_OVERLAY PROC FAR
        push ax
        push bx
        push cx
        push dx
        push es
        push ds
        push es
        mov keep_sp, sp
        mov keep_ss, ss
        mov ax, data
        mov es, ax
        mov bx, offset overlay_addr
        mov dx, offset overlay_path
        mov ax, 4b03h
        int 21h
        mov ss, keep_ss
        mov sp, keep_sp
        pop es
        pop ds
        jnc loaded_successfully

;function does not exist error
        cmp ax, 1
        jne load_file_not_found
        mov dx, offset overlay_function_not_exist
        call WRITEWRD
        jmp load_module_end

```

```

load_file_not_found:
    cmp ax, 2
    jne load_route_error
    mov dx, offset overlay_file_not_found
    call WRITEWRD
    jmp load_module_end

load_route_error:
    cmp ax, 3
    jne load_too_many_files_opened
    mov dx, offset overlay_route_not_found
    call WRITEWRD
    jmp load_module_end

load_too_many_files_opened:
    cmp ax, 4
    jne load_no_access_error
    mov dx, offset overlay_too_many_files_opened
    call WRITEWRD
    jmp load_module_end

load_no_access_error:
    cmp ax, 5
    jne load_not_enough_memory
    mov dx, offset overlay_no_access
    call WRITEWRD
    jmp load_module_end

load_not_enough_memory:
    cmp ax, 8
    jne load_wrong_env
    mov dx, offset overlay_not_enough_memory
    call WRITEWRD
    jmp load_module_end

load_wrong_env:
    cmp ax, 10
    jne load_module_end
    mov dx, offset overlay_wrong_env
    call WRITEWRD
    jmp load_module_end

loaded_successfully:
    mov dx, offset overlay_load_success
    call WRITEWRD

    mov bx, offset overlay_addr
    mov ax, [bx]
    mov cx, [bx + 2]
    mov [bx], cx
    mov [bx + 2], ax
    call overlay_addr
    mov es, ax
    mov ah, 49h
    int 21h

```

```

load_module_end:
    pop es
    pop dx
    pop cx
    pop bx
    pop ax
    ret
LOAD_OVERLAY ENDP

GET_PATH PROC NEAR ;name in si

    push ax
    push dx
    push es
    push di
    xor di, di
    mov ax, es:[2ch]
    mov es, ax

content_loop:
    mov dl, es:[di]
    cmp dl, 0
    je end_string2
    inc di
    jmp content_loop

end_string2:
    inc di
    mov dl, es:[di]
    cmp dl, 0
    jne content_loop
    call PARSE_PATH

    pop di
    pop es
    pop dx
    pop ax
    ret

GET_PATH ENDP

PARSE_PATH PROC NEAR

    push ax
    push bx
    push bp
    push dx
    push es
    push di
    mov bx, offset overlay_path
    add di, 3

boot_loop:
    mov dl, es:[di]

```

```

        mov [bx], dl
        cmp dl, '.'
        je parse_to_slash
        inc di
        inc bx
        jmp boot_loop

parse_to_slash:
        mov dl, [bx]
        cmp dl, '\'
        je get_overlay_name
        mov dl, 0h
        mov [bx], dl
        dec bx
        jmp parse_to_slash

get_overlay_name:
        mov di, si ; si - overlay_name
        inc bx

add_overlay_name:
        mov dl, [di]
        cmp dl, 0h
        je parse_path_end
        mov [bx], dl
        inc bx
        inc di
        jmp add_overlay_name

parse_path_end:
        mov [bx], dl
        pop di
        pop es
        pop dx
        pop bp
        pop bx
        pop ax
        ret

PARSE_PATH ENDP

ALLOCATE_FOR_OVERLAY PROC FAR

        push ax
        push bx
        push cx
        push dx
        push di
        mov dx, offset dta_buffer
        mov ah, 1ah
        int 21h
        mov dx, offset overlay_path
        mov cx, 0
        mov ah, 4eh
        int 21h

```

```

        jnc got_size_successfully

;file not found error
        cmp ax, 12h
        jne route_error
        mov dx, offset file_not_found_error_string
        call WRITEWRD
        jmp allocate_for_overlay_end

route_error:
        cmp ax, 3
        jne allocate_for_overlay_end
        mov dx, offset route_not_found_error_string
        call WRITEWRD
        jmp allocate_for_overlay_end

got_size_successfully:
        mov di, offset dta_buffer
        mov dx, [di + 1ch]
        mov ax, [di + 1ah]
        mov bx, 10h
        div bx
        add ax, 1h
        mov bx, ax
        mov ah, 48h
        int 21h
        mov bx, offset overlay_addr
        mov cx, 0000h
        mov [bx], ax
        mov [bx + 2], cx
        mov dx, offset allocated_mem_for_overlay_string
        call WRITEWRD

allocate_for_overlay_end:
        pop di
        pop dx
        pop cx
        pop bx
        pop ax
        ret

ALLOCATE_FOR_OVERLAY ENDP

MAIN PROC FAR
        mov ax, data
        mov ds, ax
        call FREE_UNUSED_MEMORY
        cmp error_mem_free, 0h
        jne main_end
        call ENDLINE
        mov si, offset overlay_name1
        call GET_PATH
        call ALLOCATE_FOR_OVERLAY
        call LOAD_OVERLAY
        call ENDLINE

```

```

        mov si, offset overlay_name2
        call GET_PATH
        call ALLOCATE_FOR_OVERLAY
        call LOAD_OVERLAY

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

        MAIN ENDP

labin:
CODE ENDS

END MAIN

```

Название файла: olv1.asm

```

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 overlay1_address
    add di, 22
    call WRD_TO_HEX
    mov dx, offset overlay1_address
    call WRITEWRD
    pop di
    pop ds
    pop dx
    pop ax
    retf
MAIN ENDP

overlay1_address db 'Overlay 1 Address:      ', 0DH, 0AH, '$'

WRITEWRD PROC NEAR
    push ax
    mov ah, 9
    int 21h
    pop ax
    ret
WRITEWRD 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
    call BYTE_TO_HEX
    mov [di], ah
    dec di
    mov [di], al
    pop bx
    ret
WRD_TO_HEX endp

```

```

CODE ENDS
END MAIN

```

Название файла: olv2.asm

```

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 overlay1_address
    add di, 22
    call WRD_TO_HEX
    mov dx, offset overlay1_address
    call WRITEWRD
    pop di
    pop ds

```

```
pop dx
pop ax
retf
MAIN ENDP
```

```
overlay1_address db 'Overlay 2 Address:      ', 0DH, 0AH, '$'
```

```
WRITEWRD  PROC  NEAR
    push ax
    mov ah, 9
    int 21h
    pop ax
    ret
WRITEWRD  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
    call BYTE_TO_HEX
    mov [di], ah
    dec di
    mov [di], al
    pop bx
    ret
WRD_TO_HEX endp
```



```
CODE ENDS  
END MAIN
```