МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МОЭВМ

ОТЧЕТ

по лабораторной работе №7

по дисциплине «Операционные системы»

Тема: Построение модуля оверлейной структуры

Студент гр. 9383 _____ Звега А.Р.

Преподаватель _____ Ефремов М.А.

Санкт-Петербург

2021

Постановка задачи.

Цель работы.

Исследование структуры оверлейного сегмента и способа загрузки и выполнения оверлейных сегментов. Написание программы, состоящей из нескольких модулей.

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.

Выполнение работы.

Был написан и отлажен программный модуль типа .EXE, который выполняет, прописанные в задании функции. Были написаны и отлажены оверлейные сегменты, они выводят адрес сегмента, в который они загружены. Программа была запущена для того, чтобы убедиться, что оверлейные сегменты загружаются с одного и того же адреса, перекрывая друг друга.

Рисунок 1 – Демонстрация корректной работы программы

Программа была запущена из другого каталога, чтобы убедиться в ее работоспособности.

Рисунок 2 – Демонстрация корректной работы программы при запуске из другого каталога.

Программа была запущена, после того, как из каталога был перемешен файл ovl2.ovl, для того, чтобы убедиться, что программа корректно обрабатывает ошибки.

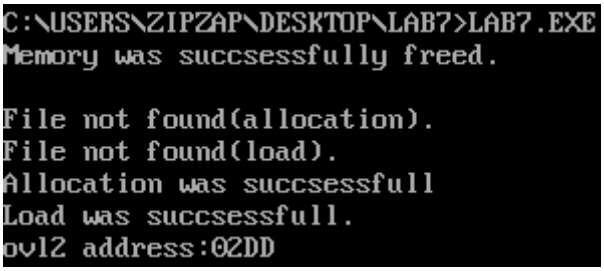

Рисунок 3 – Демонстрация корректной обработки ошибок программы, если оверлейный файл находится в другом каталоге.

Ответы на контрольные вопросы:

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

После того как произойдет запись значений регистров в стек, надо положить значение регистра CS в DS, так как адрес сегмента данных совпадает с сегментом кода. А также, по причине того, что сегменты настроены на PSP, следует добавить 100h.

Выводы.

Была написана программа, состоящая из нескольких модулей. Исследованы структуры оверлейного сегмента и способ загрузки и выполнения оверлейных сегментов.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: lab7.asm:

```
AStack    SEGMENT  STACK
          DW 64 DUP(?)
AStack    ENDS

DATA  SEGMENT
    parametr_block dw 0
                   db 0
                   db 0
                   db 0
    new_command_line db 1h,0DH
    path db 128 dup(0), '$'
    ovl_address dd 0
    ovl1_file db "ovl1.ovl", 0
    ovl2_file db "ovl2.ovl", 0
    current_file dw 0
    keep_ss dw 0
    keep_sp dw 0
    dta_buf db 43 dup(0)

    KEEP_PSP dw 0
    MEMORY_BLOCK_ERROR db "Memory control block destroyed. ", 0DH, 0AH,
'$'
    LOW_MEMORY db "Not enough memory to execute the function. ", 0DH,
0AH, '$'
    WRONG_PTR db "Invalid memory block address. ", 0DH, 0AH, '$'
    MEMORY_FREE_SUCCESS db "Memory was succsessfully freed. ", 0DH, 0AH,
0AH, '$'

    WRONG_FUNC_NUMBER db "Wrong function number.", 0DH, 0AH, '$'
    FILE_NOT_FOUND_LOAD db "File not found(load).", 0DH, 0AH, '$'
    ROUTE_NOT_FOUND_LOAD db "Route not found(load).", 0DH, 0AH, '$'
    TOO_MUCH_FILES db "Too much files opened.", 0DH, 0AH, '$'
    ACCESS_ERROR db "Access error.", 0DH, 0AH, '$'
    NOT_ENOUGH_MEMORY db "Not enough memory.", 0DH, 0AH, '$'
    WRONG_ENVIRONMENT db "Wrong environment string.", 0DH, 0AH, '$'

    FILE_NOT_FOUND_ALLOCATION db "File not found(allocation).", 0DH, 0AH,
'$'
    ROUTE_NOT_FOUND_ALLOCATION db "Route not found(allocation).", 0DH,
0AH, '$'

    LOAD_SUCCESSFUL db "Load was succsessfull.", 0DH, 0AH, '$'
    ALLOCATION_SUCCESSFUL db "Allocation was succsessfull", 0DH, 0AH, '$'

    carry_flag_value db 0
    last_data_byte db 0
DATA  ENDS

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

PRINT_BUF proc near
```

```
    push ax
    mov ah, 9h
    int 21h
    pop ax
    ret
PRINT_BUF endp


FREE_MEMORY proc near
    push ax
    push bx
    push dx

    mov ax, offset end_of_proc
    mov bx, offset last_data_byte
    add ax, bx
    mov bx, 10h
    xor dx, dx
    div bx
    mov bx, ax
    add bx, dx
    add bx, 100h

    mov ah, 4ah
    int 21h

    jnc memory_free_success_label
    mov carry_flag_value, 1

    cmp ax, 7
    je memory_block_destroyed_label
    cmp ax, 8
    je low_memory_label
    cmp ax, 9
    je invalid_address_label

memory_block_destroyed_label:
    mov dx, offset MEMORY_BLOCK_ERROR
    call PRINT_BUF
    jmp memory_free_end
low_memory_label:
    mov dx, offset LOW_MEMORY
    call PRINT_BUF
    jmp memory_free_end
invalid_address_label:
    mov dx, offset WRONG_PTR
    call PRINT_BUF
    jmp memory_free_end
memory_free_success_label:
    mov dx, offset MEMORY_FREE_SUCCESS
    call PRINT_BUF
memory_free_end:
    pop dx
    pop bx
    pop ax
    ret
FREE_MEMORY endp
```

```
PATH_FIND proc near
    push ax
    push si
    push es
    push bx
    push di
    push dx

    mov ax, KEEP_PSP
    mov es, ax
    mov ax, es:[2Ch]
    mov es, ax
    xor si, si
find_two_zeros:
    inc si
    mov dl, es:[si-1]
    cmp dl, 0
    jne find_two_zeros
    mov dl, es:[si]
    cmp dl, 0
    jne find_two_zeros

    add si, 3
    mov bx, offset path
while_not_point:
    mov dl, es:[si]
    mov [bx], dl
    cmp dl, '.'
    je loop_back

    inc bx
    inc si

    jmp while_not_point
loop_back:
    mov dl, [bx]
    cmp dl, '\'
    je break_loop
    mov dl, 0h
    mov [bx], dl
    dec bx
    jmp loop_back
break_loop:
    pop dx
    mov di, dx
    push dx
    inc bx
loop_new_file:
    mov dl, [di]
    cmp dl, 0
    je end_path_find
    mov [bx], dl
    inc di
    inc bx
    jmp loop_new_file
end_path_find:
```

```
    mov [bx], byte ptr '$'
    pop dx
    pop di
    pop bx
    pop es
    pop si
    pop ax
    ret
PATH_FIND endp

ALLOCATION_MEMORY proc near
    push ax
     push bx
     push cx
     push dx

     push dx
     mov dx, offset dta_buf
     mov ah, 1ah
     int 21h
     pop dx
     xor cx, cx
     mov ah, 4eh
     int 21h

     jnc success_allocation

    cmp ax, 2
    je file_not_found_allocation_label
    cmp ax, 3
    je route_not_found_allocation_label

file_not_found_allocation_label:
     mov dx, offset FILE_NOT_FOUND_ALLOCATION
     call PRINT_BUF
     jmp allocation_end
route_not_found_allocation_label:
    mov dx, offset ROUTE_NOT_FOUND_ALLOCATION
    call PRINT_BUF
    jmp allocation_end
success_allocation:
     push di
     mov di, offset dta_buf
     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 ovl_address, ax
```

```
      mov dx, offset ALLOCATION_SUCCESSFUL
      call PRINT_BUF

allocation_end:
      pop dx
      pop cx
      pop bx
      pop ax
      ret
ALLOCATION_MEMORY endp

CALL_MODULE proc near
    push ax
      push bx
      push cx
      push dx
      push ds
      push es

      mov ax, data
      mov es, ax
      mov bx, offset ovl_address
      mov dx, offset path
      mov ax, 4b03h
      int 21h


    jnc module_was_loaded
    cmp ax, 1
    je wrong_func_number_label
    cmp ax, 2
    je file_not_found_label
    cmp ax, 3
    je route_not_found_load_label
    cmp ax, 4
    je too_much_files_label
    cmp ax, 6
    je access_error_label
    cmp ax, 8
    je not_enough_memory_load_label
    cmp ax, 10
    je wrong_environment_label
wrong_func_number_label:
    mov dx, offset WRONG_FUNC_NUMBER
    call PRINT_BUF
    jmp call_module_end
file_not_found_label:
    mov dx, offset FILE_NOT_FOUND_LOAD
    call PRINT_BUF
    jmp call_module_end
route_not_found_load_label:
    mov dx, offset ROUTE_NOT_FOUND_LOAD
    call PRINT_BUF
    jmp call_module_end
too_much_files_label:
    mov dx, offset TOO_MUCH_FILES
    call PRINT_BUF
```

```
    jmp call_module_end
access_error_label:
    mov dx, offset ACCESS_ERROR
    call PRINT_BUF
    jmp call_module_end
not_enough_memory_load_label:
    mov dx, offset NOT_ENOUGH_MEMORY
    call PRINT_BUF
    jmp call_module_end
wrong_environment_label:
    mov dx, offset WRONG_ENVIRONMENT
    call PRINT_BUF
    jmp call_module_end
module_was_loaded:
    mov dx, offset LOAD_SUCCESSFUL
     call PRINT_BUF

     mov ax, word ptr ovl_address
     mov es, ax
     mov word ptr ovl_address, 0
     mov word ptr ovl_address+2, ax

     call ovl_address
     mov es, ax
     mov ah, 49h
     int 21h
call_module_end:
     pop si
     pop di
     pop dx
     pop cx
     pop bx
     pop ax
    ret
CALL_MODULE endp

ONE_FILE_PROCESSING proc near
    push dx
    call PATH_FIND
    mov dx, offset path
    call ALLOCATION_MEMORY
    call CALL_MODULE
    pop dx

    ret
ONE_FILE_PROCESSING endp

MAIN proc far
    push  ds
    push  ax
    mov   ax,data
    mov   ds,ax

    mov KEEP_PSP, es
    call FREE_MEMORY
    cmp carry_flag_value, 1
    je end_error
```

```
    mov dx, offset ovl1_file
    call ONE_FILE_PROCESSING

    mov dx, offset ovl2_file
    call ONE_FILE_PROCESSING
end_error:
    mov ah, 4ch
     int 21h
end_of_proc:
MAIN endp
CODE ends
END Main



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 OVL_ADDRESS
    add di, 16
    call WRD_TO_HEX
    mov dx, offset OVL_ADDRESS
    call PRINT_BUF

    pop di
    pop ds
    pop dx
    pop ax
    retf
MAIN endp
```

Название файла ovl1.asm:

```
OVL_ADDRESS db "ovl1 address:    ", 0AH, 0DH, 0AH, '$'

PRINT_BUF proc near
    push dx
    push ax

    mov ah, 09h
    int 21h

    pop ax
    pop dx
    ret
PRINT_BUF 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

code ends
end main


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 OVL_ADDRESS
    add di, 16
    call WRD_TO_HEX
```

```
    mov dx, offset OVL_ADDRESS
    call PRINT_BUF

    pop di
    pop ds
    pop dx
    pop ax
    retf
MAIN endp
```

Название файла ovl2.asm:

```
OVL_ADDRESS db "ovl2 address:    ", 0DH, 0AH, '$'

PRINT_BUF proc near
    push dx
    push ax

    mov ah, 09h
    int 21h

    pop ax
    pop dx
    ret
PRINT_BUF 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

code ends
end main
```