

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

Студентка гр. 0382

Деткова А.С.

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 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.

Выполнение работы.

Функции, используемые в программах:

1. *_PRINT* — напечатать строку, адрес смещения до которой лежит в регистре DX.
2. *BYTE_TO_DEC*, *TETR_TO_HEX*, *BYTE_TO_HEX*, *WRD_TO_HEX* - вспомогательные функции перевода чисел в дес. и шест. системы счисления.
3. *FREE_UP_MEMORY* — высвобождает лишнюю память.
4. *GET_PATH* — получение пути к файлу оверлейного модуля, где BX - смещение к имени модуля.
5. *ALLOCATION* — выделение памяти для вызываемого модуля.
6. *LOAD_OVERLAY* — запуск оверлейного модуля.
7. *MAIN* — вызывающая функция.

Шаг 1.

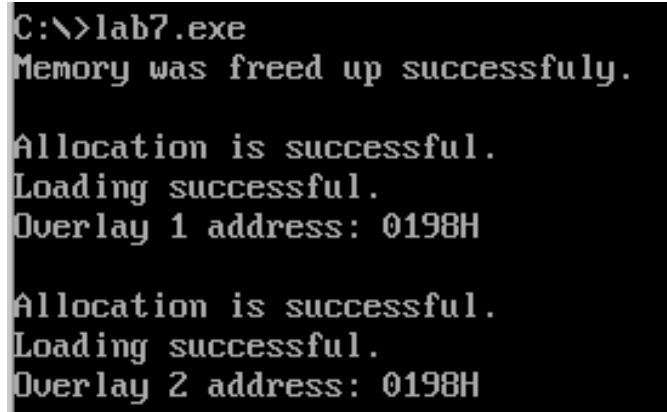
На первом шаге был написан и отлажен .EXE модуль, который освобождает память для загрузки оверлеев, читает размер файла оверлея и запрашивает объем памяти, достаточный для его загрузки, запускает оверлеи. Если встречаются ошибки, выводит соответствующие сообщения. В качестве запускаемых оверлейных модулей были написаны две программы, каждая из которых выводит адрес, начиная с которого она находится в памяти.

Шаг 2.

На втором шаге были написаны оверлейные модули, каждый из которых выводит адрес сегмента, куда он загружен.

Шаг 3.

На третьем шаге был запущен модуль .EXE. Результаты работы программ см. на рис. 2.



```
C:\>lab7.exe
Memory was freed up successfully.

Allocation is successful.
Loading successful.
Overlay 1 address: 0198H

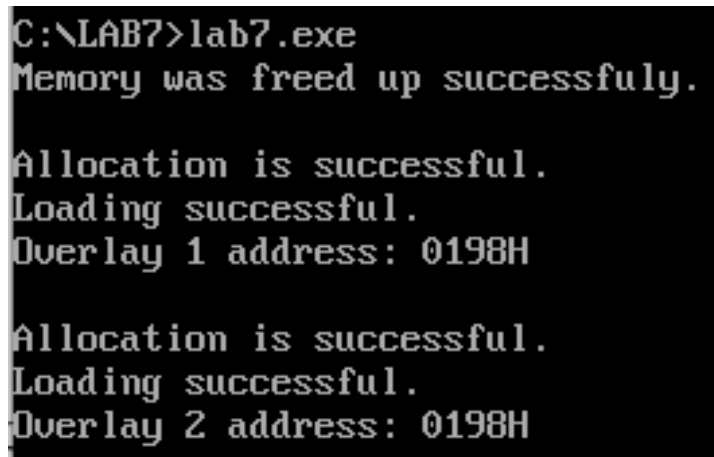
Allocation is successful.
Loading successful.
Overlay 2 address: 0198H
```

Рисунок 1: Результаты третьего шага

Как видно по рисунку, оба модуля действительно загружаются с одного адреса в памяти, перекрывая друг друга.

Шаг 4.

На четвертом шаге приложение было запущено из другого каталога. По рис. 2 видно, что оно запустилось и отработало успешно.



```
C:\LAB7>lab7.exe
Memory was freed up successfully.

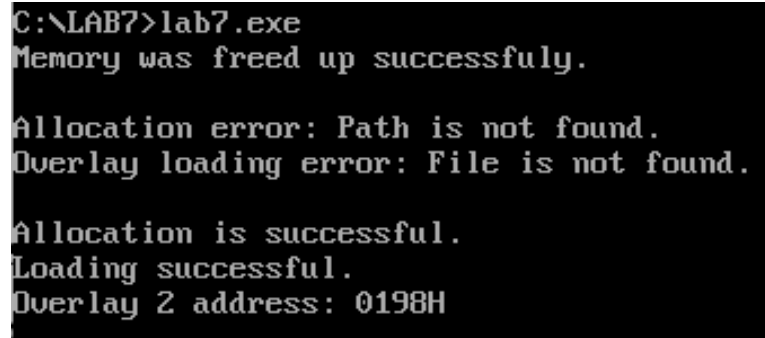
Allocation is successful.
Loading successful.
Overlay 1 address: 0198H

Allocation is successful.
Loading successful.
Overlay 2 address: 0198H
```

Рисунок 2: Результаты второго шага

Шаг 5.

На пятом шаге приложение было запущено для случая, когда одного оверлейного модуля нет в каталоге. На рис. 3 видно, что программа работает корректно.



```
C:\LAB7>lab7.exe
Memory was freed up successfully.

Allocation error: Path is not found.
Overlay loading error: File is not found.

Allocation is successful.
Loading successful.
Overlay 2 address: 0198H
```

Рисунок 3: Результаты третьего шага

Исходный код программы см. в приложении А.

Ответы на контрольные вопросы.

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

Чтобы использовать .COM модули в качестве оверлейного сегмента нужно учитывать смещение 100h при обращении к блоку PSP, который необходимо организовать. Кроме того необходимо сохранять регистры, чтобы восстановить их в конце.

Выводы.

В ходе работы были исследованы возможности построения загрузочного модуля оверлейной структуры. Была исследована структура оверлейного сегмента, а также их способ загрузки в память и выполнения.

ПРИЛОЖЕНИЕ А

КОД МОДУЛЕЙ

Название файла: lab7.asm

```
AStack    SEGMENT    STACK
            DB 100 DUP('!')
AStack    ENDS

DATA      SEGMENT

            ErrorMessage db 'Memory free up error: $'
            Err7MSG db 'Memory control block (MCB) is destroyed.', 0DH, 0AH,
'$'
            Err8MSG db 'Not enough memory to execute the function.', 0DH,
0AH, '$'
            Err9MSG db 'Incorrect memory block address.', 0DH, 0AH, '$'
            SuccessFreeMSG db 'Memory was freed up successfully.', 0DH, 0AH, '$'
            flag db 0

            KEEP_SS dw 0
            KEEP_SP dw 0
            DTA db 43 dup(?)
            OverlayAdress dd 0
            LaunchParameters dw 0,0

            FileName1 db 'overlay1.ovl$'
            FileName2 db 'overlay2.ovl$'
            PathName db 50 dup (0)
            NewLine db 0DH, 0AH, '$'

            AllocErr db 'Allocation error: $'
            AllocErr2 db 'File is not found.', 0DH, 0AH, '$'
            AllocErr3 db 'Path is not found.', 0DH, 0AH, '$'
            AllocSuccessful db 'Allocation is successful.', 0DH, 0AH, '$'
            AllocForOvlErr db 'Alloc free up memory error.', 0DH, 0AH, '$'

            LoadErr db 'Overlay loading error: $'
            LoadErr1MSG db 'Non-existent function.', 0DH, 0AH, '$'
            LoadErr2MSG db 'File is not found.', 0DH, 0AH, '$'
            LoadErr3MSG db 'Path is not found.', 0DH, 0AH, '$'
            LoadErr4MSG db 'Too many opened files.', 0DH, 0AH, '$'
            LoadErr5MSG db 'No access.', 0DH, 0AH, '$'
            LoadErr8MSG db 'Not enough memory.', 0DH, 0AH, '$'
            LoadErr10MSG db 'Incorrect environment.', 0DH, 0AH, '$'
            LoadSuccessful db 'Loading successful.', 0DH, 0AH, '$'

DATA      ENDS

CODE      SEGMENT

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

_PRINT PROC NEAR

push AX

mov AH,09H
int 21H

pop AX
ret

_PRINT ENDP

FREE_UP_MEMORY PROC NEAR

push AX
push BX
push CX
push DX

lea BX,end_program
mov AX,ES
sub BX,AX
shr BX,4
inc BX
mov AH,4AH
int 21H
jnc success_free_up
mov flag,01H

mov DX,offset ErrorMessage
call _PRINT

cmp AX,07H
mov DX,offset Err7MSG
je end_free_up

cmp AX,08H
mov DX,offset Err8MSG
je end_free_up

cmp AX,09H
mov DX,offset Err9MSG
je end_free_up

success_free_up:
mov DX,offset SuccesFreeMSG

end_free_up:
call _PRINT

pop DX
pop CX

```
pop BX
pop AX
ret
```

```
FREE_UP_MEMORY ENDP
```

```
GET_PATH PROC NEAR
```

```
    ; BX - overlay file name
    push AX
    push BX
    push DX
    push SI
    push DI
    push ES
```

```
    xor DI,DI
    mov ES,ES:[2CH]
```

```
skip_content:
    mov DL,ES:[DI]
    cmp DL,0H
    je last_content
    inc DI
    jmp skip_content
```

```
last_content:
    inc DI
    mov DL,ES:[DI]
    cmp DL,0H
    jne skip_content
```

```
    add DI,3H
    mov SI,0H
```

```
write_path:
    mov DL,ES:[DI]
    cmp DL,0H
    je delete_file_name
    mov PathName[SI],DL
    inc DI
    inc SI
    jmp write_path
```

```
delete_file_name:
    dec SI
    cmp PathName[SI],'\ '
    je ready_add_file_name
    jmp delete_file_name
```

```
ready_add_file_name:
    mov DI,-1
```

```
add_file_name:
```



```

        inc SI
        inc DI
        mov DL,BX[DI]
        cmp DL,'$'
        je path_end
        mov PathName[SI],DL
        jmp add_file_name

path_end:
        mov DL,BX[7]
        pop ES
        pop DI
        pop SI
        pop DX
        pop BX
        pop AX
        ret

GET_PATH ENDP


ALLOCATION PROC NEAR

        push AX
        push BX
        push CX
        push DX
        PUSH BP

        mov DX,offset DTA
        mov AH,1AH
        int 21H

        mov DX,offset PathName
        mov CX,0
        mov AH,4EH
        int 21H

        jnc success_alloc

        mov DX,offset AllocErr
        call _PRINT

        cmp AX,02H
        mov DX,offset AllocErr2
        je end_alloc

        mov DX,offset AllocErr3
        jmp end_alloc

success_alloc:

        mov SI,offset DTA
        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 save_seg
lea DX,AllocForOvlErr
jmp end_alloc

```

```

save_seg:
mov DX,offset AllocSuccessful
mov LaunchParameters,AX
mov LaunchParameters+2,AX

```

```

end_alloc:
call _PRINT
pop BP
pop DX
pop CX
pop BX
pop AX
ret

```

ALLOCATION ENDP

LOAD_OVERLAY PROC NEAR

```

push AX
push DX
push ES

lea DX,PathName
push DS
pop ES
lea BX, LaunchParameters
mov AX,4B03h
int 21H

jnc loading_successful

mov DX,offset LoadErr
call _PRINT

cmp AX,01H
mov DX,offset LoadErr1MSG
je print_err

cmp AX,02H
mov DX,offset LoadErr2MSG
je print_err

cmp AX,03H
mov DX,offset LoadErr3MSG

```

```

    je print_err

    cmp AX,04H
    mov DX,offset LoadErr4MSG
    je print_err

    cmp AX,05H
    mov DX,offset LoadErr5MSG
    je print_err

    cmp AX,08H
    mov DX,offset LoadErr8MSG
    je print_err

    cmp AX,0AH
    mov DX,offset LoadErr10MSG
    je print_err

print_err:
    call _PRINT
    jmp end_load

loading_successful:
    mov DX,offset LoadSuccessful
    call _PRINT
    mov AX,LaunchParameters
    mov word ptr OverlayAddress+2,AX

    call OverlayAddress

    mov ES,AX
    mov AH, 49H
    int 21H

end_load:
    pop ES
    pop DX
    pop AX
    ret
LOAD_OVERLAY ENDP

MAIN PROC FAR
    sub AX,AX
    push AX
    mov AX,DATA
    mov DS,AX

    call FREE_UP_MEMORY
    cmp flag,0
    jne final
    mov DX,offset NewLine
    call _PRINT

    lea bx,FileName1

```

```

    call GET_PATH
    call ALLOCATION
    call LOAD_OVERLAY

    mov DX,offset NewLine
    call _PRINT

    lea bx,FileName2
    call GET_PATH
    call ALLOCATION
    call LOAD_OVERLAY

final:
    xor AL,AL
    mov AH,4Ch
    int 21H
MAIN ENDP
end_program:
CODE ENDS
    END MAIN

```

Название файла: overlay1.asm

```

OVERLAY1 SEGMENT
ASSUME CS:OVERLAY1, DS:NOTHING, SS:NOTHING

MAIN PROC FAR

    push AX
    push DX
    push DI
    push DS

    mov AX,CS
    mov DS,AX
    mov DI,offset ovl_addr + 22
    call WRD_TO_HEX
    mov DX,offset ovl_addr
    call _PRINT

    pop ds
    pop di
    pop dx
    pop ax
    retf

MAIN ENDP

ovl_addr db "Overlay 1 address:      H", 0DH, 0AH, '$'

_PRINT PROC NEAR

```

```

    push ax

    mov ah, 09h
    int 21h

    pop ax
    ret

_PRINT 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

```

```
OVERLAY1 ENDS  
END MAIN
```

Название файла: overlay2.asm

```
OVERLAY2 SEGMENT  
ASSUME CS:OVERLAY2, DS:NOTHING, SS:NOTHING
```

```
MAIN PROC FAR
```

```
    push AX  
    push DX  
    push DI  
    push DS
```

```
    mov AX,CS  
    mov DS,AX  
    mov DI,offset ovl_addr + 22  
    call WRD_TO_HEX  
    mov DX,offset ovl_addr  
    call _PRINT
```

```
    pop ds  
    pop di  
    pop dx  
    pop ax  
    retf
```

```
MAIN ENDP
```

```
ovl_addr db "Overlay 2 address:      H", 0DH, 0AH, '$'
```

```
_PRINT PROC NEAR
```

```
    push ax
```

```
    mov ah, 09h  
    int 21h
```

```
    pop ax  
    ret
```

```
_PRINT 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

OVERLAY2 ENDS
END MAIN

```