

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

Гудов Н.Р.

Преподаватели

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.

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.

Программный код см. в Приложении А

Шаг 1.

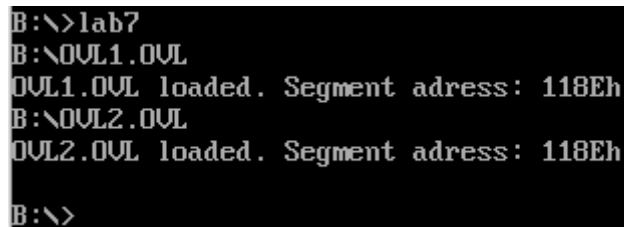
Написан программный модуль, освобождающий память для загрузки оверлея. Запрашивает память и загружает оверлей. Процедуры `prer_overlay` и `load` определяют размер и запускают оверлей используя функции `4Eh` и `4B03h` прерывания `21h`.

Шаг 2.

Написаны и отлажены оверлейные сегменты, выводящие адрес сегмента, в который они загружены.

Шаг 3.

Запуск программы из текущего каталога с модулем и оверлейными сегментами.



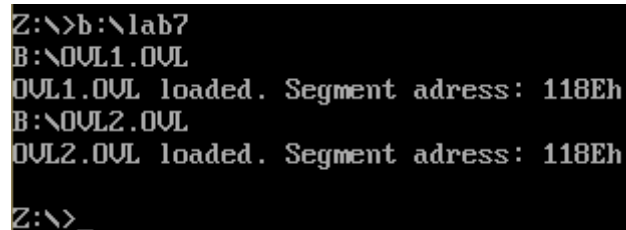
```
B:\>lab7
B:\>OVL1.OVL
OVL1.OVL loaded. Segment adress: 118Eh
B:\>OVL2.OVL
OVL2.OVL loaded. Segment adress: 118Eh
B:\>
```

Рисунок 1. Запуск того же каталога

Оверлеи запускаются с одного адреса, перекрывая друг друга.

Шаг 4.

Запуск программы из другого каталога



```
Z:\>b:\lab7
B:\>OVL1.OVL
OVL1.OVL loaded. Segment adress: 118Eh
B:\>OVL2.OVL
OVL2.OVL loaded. Segment adress: 118Eh
Z:\>_
```

Рисунок 2. Запуск из другого каталога

Результат запуска из другого каталога не отличается от предыдущего пункта.

Шаг 5.

Запуск программы при отсутствии одного из оверлейных модулей.



```
B:\>lab7
B:\OVL1.OVL
Overlay reading error
Overlay loading error
B:\OVL2.OVL
OVL2.OVL loaded. Segment adress: 118Eh
B:\>
```

Рисунок 3. Запуск с аварийным завершением

Запускается оставшийся оверлей. Программа завершается аварийно.

Вопросы.

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

В силу наличия PSP, необходимо обращаться к модулю со смещением в 100h.

Выводы.

В результате выполнения лабораторной работы была исследована возможность построения загрузочного модуля оверлейной структуры. Исследованы структура оверлейного сегмента, способ загрузки и выполнения оверлейных сегментов.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЕ КОДЫ ПРОГРАММ

Название файла: lab7.asm

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

DATA SEGMENT
    DTA db 43 dup(0)
    PARAMETERS      DW 0,0
    L_ADDRESS      DD 0
    PATH  DB 32 dup(0)
    OVL1_NAME      DB 'OVL1.OVL$'
    OVL2_NAME      DB 'OVL2.OVL$'
    MEM_ERROR      DB 'Error',13,10,'$'
    READ_ERROR     DB 13,10,'Reading error',13,10,'$'
    ALLOC_ERROR     DB 'Allocation error',13,10,'$'
    LOAD_ERROR     DB 'Loading error',13,10,'$'
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
;-----
MEM_FREE PROC NEAR
    push AX
    push BX
    push DX

    mov BX, offset end_address
    mov AX, ES
    sub BX, AX
    shr BX, 4
    inc BX
    mov AH, 4Ah
    int 21h

    jnc MEM_FREE_end
    mov DX, offset MEM_ERROR
    call PRINT
MEM_FREE_end:
    pop DX
    pop BX
    pop AX
    ret
MEM_FREE ENDP
```

```

;-----
SET_PATH PROC NEAR
    push dx
    push di
    push si
    push es

    mov ES, ES:[2Ch]
    mov SI, offset PATH
    xor DI, DI

    read_byte:
    mov DL, ES:[DI]
    check_byte:
    inc DI
    cmp DL, 0
    jne read_byte
    mov DL, ES:[DI]
    cmp DL, 0
    jne check_byte

    add DI, 3
    write_path:
    mov DL, ES:[DI]
    mov [SI], DL
    inc SI
    inc DI
    cmp DL, 0
    jne write_path

    backslash_loop:
    dec SI
    cmp PATH[si-1], '\'
    jne backslash_loop
    mov di, 0
    write_filename:
    mov dl, bx[di]
    mov PATH[si], dl
    inc si
    inc di
    cmp dl, '$'
    jne write_filename

    pop es
    pop si
    pop di
    pop dx
    ret
SET_PATH ENDP
;-----
PREP_OVERLAY PROC NEAR
    push AX
    push BX
    push CX
    push DX

    mov AH, 1Ah
    lea DX, DTA

```

```

int 21h

mov AH, 4Eh
lea DX, PATH
mov CX, 0
int 21h
jnc allocation
mov DX, offset READ_ERROR
call PRINT
jmp overlay_size_end

allocation:
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
mov DX, offset ALLOC_ERROR
call PRINT
jmp overlay_size_end
save_seg:
mov PARAMETERS, AX
mov PARAMETERS+2, AX

overlay_size_end:
pop DX
pop CX
pop BX
pop AX
ret
PREP_OVERLAY ENDP
;-----
LOAD PROC NEAR
push AX
push DX
push ES

call SET_PATH
mov DX, offset PATH
call PRINT
call PREP_OVERLAY

mov DX, offset PATH
push DS
pop ES
mov BX, offset PARAMETERS
mov AX, 4B03h
int 21h
jnc loaded

mov DX, offset LOAD_ERROR
call PRINT

```

```

    jmp overlay_end

loaded:
mov AX, PARAMETERS
mov word ptr L_ADDRESS + 2, AX
call L_ADDRESS
mov ES, AX
mov AH, 49h
int 21h

overlay_end:
pop ES
pop DX
pop AX
ret
LOAD ENDP
;-----
Main PROC FAR
    sub AX,AX
    push AX
    mov AX, DATA
    mov DS,AX

    call MEM_FREE
    mov BX, offset OVL1_NAME
    call LOAD
    mov BX, offset OVL2_NAME
    call LOAD
    xor AL,AL
    mov AH,4Ch
    int 21H
Main ENDP
    end_address:
CODE ENDS
END Main

```

Название файла: ovl1.asm

```

OVL1 SEGMENT
    ASSUME CS:OVL1, DS:NOTHING, SS:NOTHING, ES:NOTHING

Main PROC FAR
    push AX
    push DX
    push DI
    push DS

    mov AX,CS
    mov DS,AX
    mov DX, offset SEG_STR
    mov DI, DX
    add DI, 38
    call WRD_TO_HEX
    call PRINT

    pop DS
    pop DI

```



```

        pop DX
        pop AX
        retf
Main 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

PRINT PROC NEAR
    push AX
    mov AH, 09h
    int 21h
    pop AX
    ret
PRINT ENDP

SEG_STR DB 13,10,'OVL1.OVL loaded. Segment adress:      h',13,10,'$'
OVL1 ENDS
END Main

```

Название файла: ovl2.asm

```
OVL2 SEGMENT
    ASSUME CS:OVL2, DS:NOTHING, SS:NOTHING, ES:NOTHING

Main PROC FAR
    push AX
    push DX
    push DI
    push DS

    mov AX,CS
    mov DS,AX
    mov DX, offset SEG_STR
    mov DI, DX
    add DI, 38
    call WRD_TO_HEX
    call PRINT

    pop DS
    pop DI
    pop DX
    pop AX
    retf
Main 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

PRINT PROC NEAR
        push AX
        mov AH, 09h
        int 21h
        pop AX
        ret
PRINT ENDP

        SEG_STR DB 13,10,'OVL2.OVL loaded. Segment adress:      h',13,10,'$'
OVL2 ENDS
END Main

```