

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

Студентка гр. 8382

Кузина А.М.

Преподаватель

Ефремов М.А.

Санкт-Петербург

2020

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.

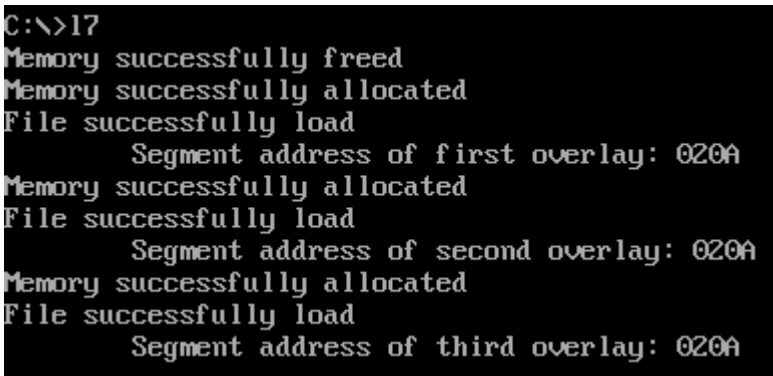
Ход выполнения работы.

Была написана программа, исходный код которой приведет в приложении А, выполняющая следующие функции:

- Освобождает память, не используемую программой, для загрузки оверлея.
- Читает размер файла оверлея и запрашивает память, достаточную для его загрузки.
- Загружает файл оверлея и он выполняется.
- Освобождает память, выделенную под оверлей.
- Повторяет предыдущие шаги для следующего оверлейного сегмента.

Также были написаны оверлейные модули, которые выводят свой номер и адрес сегмента, в который они загружены. На рисунке 1 представлена работа программы, когда все файлы находятся в одной текущей директории.

Рисунок 1 — результат работы программы.



```
C:\>I7
Memory successfully freed
Memory successfully allocated
File successfully load
    Segment address of first overlay: 020A
Memory successfully allocated
File successfully load
    Segment address of second overlay: 020A
Memory successfully allocated
File successfully load
    Segment address of third overlay: 020A
```

Затем приложение было запущено из другого каталога. Результат работы программы представлен на рисунке 2. На рисунках 3 и 4 соответственно

представлена работа программы при отсутствии 1го и 2го оверлейных модулей в текущем каталоге.

Рисунок 2 — результат работы программе при запуске из другого каталога.

```
C:\>test\l7
Memory successfully freed
Memory successfully allocated
File successfully load
    Segment address of first overlay: 020A
Memory successfully allocated
File successfully load
    Segment address of second overlay: 020A
Memory successfully allocated
File successfully load
    Segment address of third overlay: 020A
```

Рисунок 3 — результат работы программы при отсутствии 1го модуля в соответствующем каталоге.

```
C:\>test\l7
Memory successfully freed
File error - file is not found
```

Рисунок 4 — результат работы программы при отсутствии 2го модуля в соответствующем каталоге.

```
C:\>test\l7
Memory successfully freed
Memory successfully allocated
File successfully load
    Segment address of first overlay: 020A
File error - file is not found
```

Контрольные вопросы

- Как должна быть устроена программа, если в качестве оверлейного сегмента использовать .com модуль?

Необходимо сохранять значения регистров программы, а также учитывать смещение в 100h — под формирование PSP — для .com модулей. В остальном все будет работать корректно.

Выводы

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

ПРИЛОЖЕНИЕ А

Исходный код программы l7.asm

```
ASTACK SEGMENT      STACK
                    DB      100h  DUP(0)
ASTACK ENDS

DATA SEGMENT
MemN db 'Memory successfully freed', 13, 10, '$'
MemErr7 db 'Memory error - memory block was destroyed', 13, 10, '$'
MemErr8 db 'Memory error - not enough memory for function', 13, 10, '$'
MemErr9 db 'Memory error - incorrect memory block address', 13, 10, '$'
MemErrA      db      'Memory allocation error', 13, 10, '$'
MemA db 'Memory successfully allocated', 13, 10, '$'
FileErr2 db 'File error - file is not found', 13, 10, '$'
FileErr3 db 'File error - path is not found', 13, 10, '$'
LoadOk db 'File successfully load', 13, 10, '$'
Ovr1Err1 db 'Load error - incorrect function number', 13, 10, '$'
Ovr1Err2 db 'Load error - file is not found', 13, 10, '$'
Ovr1Err3 db 'Load error - path is not found', 13, 10, '$'
Ovr1Err4 db 'Load error - too many file already open', 13, 10, '$'
Ovr1Err5 db 'Load error - acces denied', 13, 10, '$'
Ovr1Err8 db 'Load error - not enough memory', 13, 10, '$'
Ovr1Err10 db 'Load error - incorrect enviroment', 13, 10, '$'
EndL db 13,10,'$'
Path db      128 DUP(0)
Ovr11 db '01.OVL', 0
Ovr12 db '02.OVL', 0
DTA      db 43 DUP(0)
Ovr1Seg      dw      0
Ovr1Addr dd 0
KeepSS dw 0
KeepSP dw 0
KeepDS dw 0
DataEnd dw 0
DATA ENDS

CODE SEGMENT
        ASSUME CS:CODE, DS:DATA, ES:DATA, SS:ASTACK

Fmem PROC
        push ax
        push bx
        ;определяем сколько памяти нужно оставить нашей программе
        push dx
        mov dx, offset DataEnd
        mov bx, offset ProgEnd
        add bx, dx
        push cx
        mov cl, 4
        shr bx, cl ;переводим в параграфы
        add bx, 28h
        pop cx
        mov ah, 4Ah ;всю остальную освобождаем
        int 21h
        jnc MRET
        cmp ax, 7
        je MERR7
        cmp ax, 8
```

```

        je MERR8
    cmp ax, 9
        je MERR9

MERR7:
    mov dx, offset MemErr7
    jmp MEnd
MERR8:
    mov dx, offset MemErr8
    jmp MEnd
MERR9:
    mov dx, offset MemErr9
MEnd:
    mov ah, 09h
    int 21h
    mov ah, 4Ch
    int 21h
MRET:
    mov dx, offset MEMN
    mov ah, 09h
    int 21h
    pop dx
    pop bx
    pop ax
ret
FMem ENDP

```

```

FPath PROC
;dx = имя нужного файла
    push ax
    push si
    push di
    push es

    mov es, es:[2Ch]
    mov si, 0

EnvLoop:
    mov ah, es:[si]
    cmp ah, 0
        je EnvEnd
    inc si
        jmp EnvLoop

EnvEnd:
    inc si
    mov ah, es:[si]
    cmp ah, 0
        jne EnvLoop
    add si, 3
    mov di, offset Path
Ploop:
    mov ah, es:[si]
    mov [di], ah
    cmp ah, 0
        je PEnd
    inc si
    inc di
        jmp Ploop

PEnd:

```

```

        sub    di, 6
        mov    si, dx

Floop:
        mov    ah, [si]
        mov    [di], ah
        cmp    ah, 0
        je     FEnd
        inc    si
        inc    di
        jmp    Floop

FEnd:
        pop    es
        pop    di
        pop    si
        pop    ax
        ret

;берем путь нашей программы, заменяем ее имя на имя нужного файла
FPath ENDP

```

```

AllMem PROC
        push ax
        push bx
        push cx
        push dx

        mov    dx, offset DTA ;устанавливаем DTA адрес
        mov    ah, 1Ah
        int     21h

        mov    dx, offset Path
        mov    cx, 0
        mov    ax, 4E00h
        int     21h
        jnc     AllMemOK
        cmp     ax, 2
        je      AllMemERR2
        cmp     ax, 3
        je      AllMemERR3

```

```

AllMemERR2:
        mov    dx, offset FileErr2
        mov    ah, 09h
        int     21h
        mov    dx, offset EndL
        mov    ah, 09h
        int     21h
        mov    ah, 4Ch
        int     21h

```

```

AllMemERR3:
        mov    dx, offset FileErr3
        mov    ah, 09h
        int     21h
        mov    dx, offset EndL
        mov    ah, 09h
        int     21h
        mov    ah, 4Ch
        int     21h

```

```

AllMemOK:
    mov bx, offset DTA ; получаем размер модуля
    mov ax, [bx+1Ch]
    mov bx, [bx+1Ah]

    mov cl, 12
    shl ax, cl
    mov cl, 4
    shr bx, cl
    add bx, ax
    inc bx

    mov ah, 48h ;выделяем память
    int 21h
    jnc AllMemEnd

```

```

AllMemErr:
    mov dx, offset MemErrA
    mov ah, 09h
    int 21h
    mov dx, offset EndL
    mov ah, 09h
    int 21h

    mov ah, 4Ch
    int 21h

```

```

AllMemEnd:
    mov Ovr1Seg, ax

    mov dx, offset MemA
    mov ah, 09h
    int 21h

    pop dx
    pop cx
    pop bx
    pop ax
    ret

```

```

AllMem ENDP

```

```

RunOvr1 PROC
    push ax
    push bx
    push dx
    push es

    push es
    mov KeepDS, ds
    mov KeepSS, ss
    mov KeepSP, sp

    mov bx, seg Ovr1Seg
    mov es, bx
    mov bx, offset Ovr1Seg

    mov dx, offset Path
    mov ax, 4B03h
    int 21h

```



```

mov     ds, KeepDS
mov     ss, KeepSS
mov     sp, KeepSP
pop     es

jnc     RunOvr1OK

cmp     ax, 1
je      ROvr1Err1
cmp     ax, 2
je      ROvr1Err2
cmp     ax, 3
je      ROvr1Err3
cmp     ax, 4
je      ROvr1Err4
cmp     ax, 5
je      ROvr1Err5
cmp     ax, 8
je      ROvr1Err8
cmp     ax, 10
je      ROvr1Err10

ROvr1Err1:
mov     dx, offset Ovr1Err1
jmp     ROvr1Err

ROvr1Err2:
mov     dx, offset Ovr1Err2
jmp     ROvr1Err

ROvr1Err3:
mov     dx, offset Ovr1Err3
jmp     ROvr1Err

ROvr1Err4:
mov     dx, offset Ovr1Err4
jmp     ROvr1Err

ROvr1Err5:
mov     dx, offset Ovr1Err5
jmp     ROvr1Err

ROvr1Err8:
mov     dx, offset Ovr1Err8
jmp     ROvr1Err

ROvr1Err10:
mov     dx, offset Ovr1Err10
jmp     ROvr1Err
ROvr1Err:
mov     ah, 09h
int     21h

mov     ax, 4C00h
int     21h

RunOvr1OK:

mov     dx, offset LoadOK
mov     ah, 09h

```

```

int 21h

mov ax, Ovr1Seg
mov word ptr Ovr1Addr+2, ax
call Ovr1Addr ; вызываем оверлей

mov ax, Ovr1Seg ;освобождаем память
mov es, ax
mov ah, 49h
int 21h
pop es
pop dx
pop bx
pop ax
ret

```

```
RunOvr1 ENDP
```

```
MAIN PROC
```

```

mov ax, DATA
mov ds, ax
call FMem
mov dx, offset Ovr11
call FPath
call AllMem
call RunOvr1
mov bx, 0
mov dx, offset Ovr12
call FPath
call AllMem
call RunOvr1

```

```

mov ax, 4C00h
int 21h

```

```
MAIN ENDP
```

```
ProgEND:
```

```
CODE ENDS
```

```
END MAIN
```