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. 9382 _____ Сорочина М.В.

Преподаватель _____ Ефремов М.А.

Санкт-Петербург

2021

Цель работы.

Исследование возможности построения загрузочного модуля оверлейной структуры.

Порядок выполнения работы.

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 закончится аварийно.

Шаг 6. Занесите полученные результаты в виде скриншотов в отчет. Оформите отчет в соответствии с требованиями.

Ход работы.

На рис. 1 представлен вывод программы, запущенной в одной директории с оверлеями.

Рис. 1.

На рисунке видно, что оба оверлея были найдены и загружены с одного и того же адреса.

На рис. 2 представлен вывод программы, запущенной из другой директории.

Рис. 2.

На рисунке видно, что программа точно так же работает при запуске из другой директории.

На рис. 3 представлен вывод программы при отсутствии первого оверлея в директории.

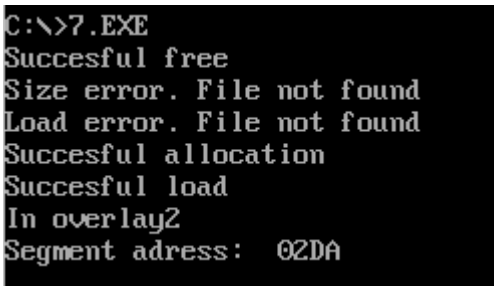

Рис. 3.

По рисунку видно, что программа не нашла один из модулей и вывела соответствующее сообщение об ошибке.

Ответы на контрольные вопросы.

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

   Поскольку в .COM модулях код располагается с адреса 100h, оверлейный сегмент необходимо вызывать по смещению 100h, иначе не будет PSP.

Выводы.

В ходе выполнения данной работы была изучена возможность построения загрузочного модуля оверлейной структуры.

ПРИЛОЖЕНИЕ А.

Исходный код программы 7.asm.

```
astack              segment         stack
          dw        200   dup(?)
astack              ends

data                segment
          ovl1                      db        'ovl1.ovl',0
          ovl2                      db        'ovl2.ovl',0

          keep_psp        dw        0

          memErr7         db        'The control block destroyed',
     13,10, '$'
          memErr8         db        'Not enough memory to perform
     the function', 13,10, '$'
          memErr9         db        'Invalid address of the memory
     block', 13,10, '$'
          memSucces  db         'Succesful free', 13, 10, '$'

          sizeErr2            db        'Size error. File not
     found', 13,10,'$'
          sizeErr3            db        'Size error. Route not
     found', 13,10,'$'
          sizeSucces      db        'Succesful allocation',
     13,10,'$'

          loadErr1            db        'Load error. Non-existent
     function number', 13,10,'$'
          loadErr2            db        'Load error. File not
     found', 13,10,'$'
          loadErr3            db        'Load error. Route not
     found', 13,10,'$'
          loadErr4            db        'Load error. Too many open
     files', 13,10,'$'
          loadErr5            db        'Load error. No access',
     13,10,'$'
          loadErr8            db        'Load error. Not enough
     memory', 13,10,'$'
          loadErr10       db        'Load error. Wrong enviroment',
     13,10,'$'
          loadSuccess     db        'Succesful load', 13,10,'$'

          fullPath            db        128 dup(0)
          dtaMem          db        43   dup(?)
          ovlSegAdr       dd        0

          dataEnd             db        0
data                ends

code            segment
```

```
          assume cs:code, ds:data, es:nothing, ss:astack
;-------------------------------------------------
          PRINT           proc       near
                    push  ax
                    mov   ah, 09h
                    int         21h
                    pop         ax
                    ret
          PRINT           endp
;-------------------------------------------------
          freeExtraMem    proc       near
                    push  bx
                    push  dx
                    push  cx

                    mov   bx, offset progEnd
                    mov   ax, offset dataEnd
                    add         bx, ax

                    mov   cl, 4
                    shr         bx, cl
                    add         bx, 100h
                    mov   ah, 4ah
                    int         21h

                    jnc         freeMemSucces

                    cmp   ax, 7
                    je                mem_Err7
                    cmp   ax, 8
                    je                mem_Err8
                    cmp   ax, 9
                    je                mem_Err9

               mem_Err7:
                    mov   dx, offset memErr7
                    call        print
                    mov   ax, 0
                    jmp         freeMemEnd
               mem_Err8:
                    mov   dx, offset memErr8
                    call        print
                    mov   ax, 0
                    jmp         freeMemEnd
               mem_Err9:
                    mov   dx, offset memErr9
                    call        print
                    mov   ax, 0
                    jmp         freeMemEnd
               freeMemSucces:
                    mov   dx, offset memSucces
                    call        print
                    mov   ax, 1
```

```
                freeMemEnd:
                        pop         cx
                        pop         dx
                        pop         bx

                        ret
            freeExtraMem    endp
;-------------------------------------------------
            createPath proc
                        push  ax
                        push  cx
                        push  bx
                        push  di
                        push  si
                        push  es

                        mov   si, dx

                        mov   ax, keep_psp
                        mov   es, ax
                        mov   es, es:[2ch]

                        sub         bx, bx
                        printEnvVar:
                                    cmp   byte ptr es:[bx], 0
                                    je                varEnd
                                    inc         bx
                                    jmp         printEnvVar

                        varEnd:
                                    inc         bx
                                    cmp   byte ptr es:[bx+1], 0
                                    jne         printEnvVar

                        add         bx, 2
                        mov   di, 0
                        pathLoop:
                                    mov   dl, es:[bx]
                                    mov   byte ptr [fullPath+di], dl
                                    inc         bx
                                    inc         di
                                    cmp   dl, 0
                                    je                pathLoopEnd
                                    cmp   dl, '\'
                                    jne         pathLoop
                                    mov   cx, di
                                    jmp         pathLoop
                        pathLoopEnd:
                                    mov   di, cx

                        filenameLoop:
                                    mov   dl, byte ptr [si]
```

```
                                mov   byte ptr [fullPath+di], dl
                                inc         di
                                inc         si
                                cmp   dl, 0
                                jne         filenameLoop

                        pop         es
                        pop         si
                        pop         di
                        pop         bx
                        pop         cx
                        pop         ax

                        ret
                createPath endp
;-------------------------------------------------
                allocateMemForOvl     proc
                        push  bx
                        push  cx
                        push  dx

                        push  dx
                        mov   dx, offset dtaMem
                        mov   ah, 1ah
                        int         21h
                        pop         dx

                        sub         cx, cx
                        mov   ah, 4eh
                        int         21h

                        jnc         size_Succes

                        cmp   ax, 2
                        je                size_Err2
                        cmp   ax, 3
                        je                size_Err3

                        size_Err2:
                                    mov   dx, offset sizeErr2
                                    jmp         sizeErr
                        size_Err3:
                                    mov   dx, offset sizeErr3
                                    jmp         sizeErr
                        size_Succes:
                                    push  di
                                    mov   di, offset dtaMem
                                    mov   bx, [di+1ah]
                                    mov   ax, [di+1ch]
                                    pop         di

                                    push  cx
                                    mov   cl, 4
```

```
                        shr         bx, cl
                        mov   cl, 12
                        shr         ax, cl
                        pop         cx
                        add         bx, ax
                        inc         bx

                        mov   ah, 48h
                        int         21h

                        mov   word ptr ovlSegAdr, ax
                        mov   dx, offset sizeSucces
                        call        print

                        mov   ax, 1
                        jmp         sizeEnd

                sizeErr:
                        mov   ax, 0
                        call        print

                sizeEnd:
                pop     dx
                pop     cx
                pop     bx

                ret
        allocateMemForOvl       endp
;-------------------------------------------------
        loadOvl         proc
                push  ax
                push  bx
                push  cx
                push  dx
                push  ds
                push  es

                mov   ax, data
                mov   es, ax
                mov   dx, offset fullPath
                mov   bx, offset ovlSegAdr
                mov   ax, 4b03h
                int         21h

                jnc         load_Success

                cmp   ax, 1
                je                  load_Err1
                cmp   ax, 2
                je                  load_Err2
                cmp   ax, 3
                je                  load_Err3
                cmp   ax, 4
```

```
je              load_Err4
cmp   ax, 5
je              load_Err5
cmp   ax, 8
je              load_Err8
cmp   ax, 10
je              load_Err10

load_Err1:
          mov   dx, offset loadErr1
          jmp       print_and_end
load_Err2:
          mov   dx, offset loadErr2
          jmp       print_and_end
load_Err3:
          mov   dx, offset loadErr3
          jmp       print_and_end
load_Err4:
          mov   dx, offset loadErr4
          jmp       print_and_end
load_Err5:
          mov   dx, offset loadErr5
          jmp       print_and_end
load_Err8:
          mov   dx, offset loadErr8
          jmp       print_and_end
load_Err10:
          mov   dx, offset loadErr10
          jmp       print_and_end

print_and_end:
          call      print
          jmp       load_end

load_Success:
          mov   dx, offset loadSuccess
          call      print

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

          call      ovlSegAdr

          mov   es, ax
          mov   ah, 49h
          int       21h

load_end:
pop       es
pop       ds
pop       dx
```

```
                    pop         cx
                    pop         bx
                    pop         ax

                    ret
          loadOvl         endp
;-------------------------------------------------
          startOvl        proc
                    push  dx

                    call        createPath

                    mov   dx, offset fullPath
                    call        allocateMemForOvl
                    cmp   ax, 1
                    ;jne        ovl_end

                    call        loadOvl

                    ovl_end:
                    pop         dx

                    ret
          startOvl        endp
;-------------------------------------------------
          main            proc
                    push        DS        ;\  Сохранение адреса начала
     PSP в стеке
                    sub         AX,AX     ; > для последующего
     восстановления по
                    push        AX        ;/  команде ret, завершающей
     процедуру.
                    mov         AX,DATA             ; Загрузка
     сегментного
                    mov         DS,AX               ; регистра данных.
                    mov   keep_psp, es

                    call        freeExtraMem
                    cmp   ax, 0
                    je                end_main

                    lea         dx, ovl1
                    call        startOvl

                    lea         dx, ovl2
                    call        startOvl

               end_main:
                    ; Выход в DOS
                    xor AL,AL
                    mov AH,4Ch
                    int 21H
          main            endp
```

```
progEnd:

code  ends
end         main
```

ПРИЛОЖЕНИЕ Б.

Исходный код программы ovl1.asm.

```
MY SEGMENT
          ASSUME    CS:MY, DS:nothing, ES:nothing, SS:nothing
          main           proc      far
                    jmp start
                    inOverlay  db   'In overlay1', 13,10,'$'
                    adress              db   'Segment adress:  $'
                    endl                db   13, 10, '$'
               start:
                    push  ax
                    push  dx
                    push  ds

                    mov   ax, cs
                    mov   ds, ax

                    mov   dx, offset inOverlay
                    call      print

                    mov   dx, offset adress
                    call      print

                    mov   ax, cs
                    call      printWord

                    mov   dx, offset endl
                    call      print

                    pop       ds
                    pop   dx
                    pop       ax
                    retf
          main           endp
;-------------------------------------------------
          PRINT          proc      near
```

```
                push  ax
                mov   ah, 09h
                int       21h
                pop       ax
                ret
        PRINT         endp
;-------------------------------------------------
        printWord  proc
                xchg  ah, al
                call      printByte
                xchg  ah, al
                call      printByte
                ret
        printWord  endp
;-------------------------------------------------
        printByte      proc
                push  ax
                push  bx
                push  dx

                call      byte_to_hex
                mov   bh, ah

                mov   dl, al
                mov   ah, 02h
                int 21h

                mov   dl, bh
                mov   ah, 02h
                int 21h

                pop       dx
                pop       bx
                pop       ax
                ret
        printByte      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
;байт в AL переводится в два символа шест. числа в AX
           push CX
           mov AH,AL
           call TETR_TO_HEX
           xchg AL,AH
           mov CL,4
           shr AL,CL
           call TETR_TO_HEX ;в AL старшая цифра
           pop CX                ;в AH младшая
           ret
BYTE_TO_HEX ENDP
MY ENDS
           end main
```