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 Круглова В.Д.

Преподаватель Ефремов М.А.

Санкт-Петербург

2021

Цель работы.

Исследование возможности построения загрузочного модуля оверлейной структуры. Исследуется структура оверлейного сегмента и способ загрузки и выполнения оверлейных сегментов. Для запуска вызываемого 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.

Выполнение работы.

В процессе выполнения лабораторной работы был написан и отлажен программный модуль типа .EXE, выполняющий следующие функции:

- Освобождает память для загрузки оверлеев.
- Читает размер файла оверлея и запрашивает объем памяти, достаточный для его загрузки.
- Файл оверлейного сегмента загружается и выполняется.
- Освобождается память, отведенная для оверлейного сегмента.
- Затем предыдущие действия выполняются для следующего оверлейного сегмента.

Также были написаны и отлажены оверлейные сегменты, которые выводят адрес сегмента их загрузки.

Результат работы программы в одном каталоге показан на рисунке 1.

Результат работы программы при запуске из другого каталога показан на рисунке

2. На рисунках 3-4 показан вывод программы при отсутствии одного из оверлейных модулей.

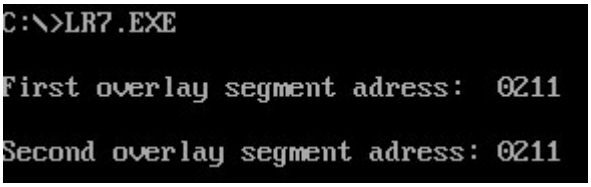

Рисунок 1. Запуск программы.

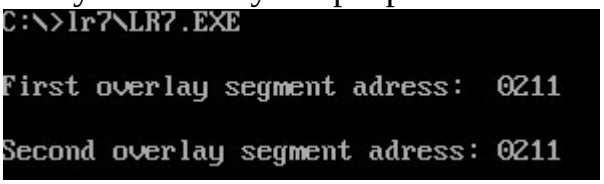

Рисунок 2. Запуск из другого каталога

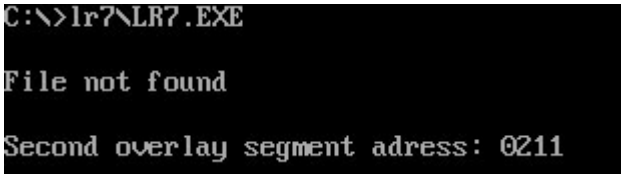

Рисунок 3. Запуск при отсутствии модуля 1

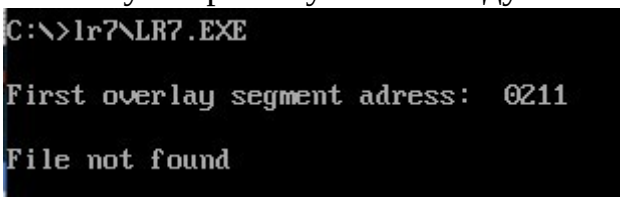

Рисунок 4. Запуск при отсутствии модуля 2

Контрольные вопросы.

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

Программа корректно работает, если в качестве оверлейного сегмента использовать .com модуль. Нужно лишь учитывать смещение psp.

Выводы.

В ходе работы была исследована возможность построения загрузочного модуля оверлейной структуры.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ LR1COM.ASM

```
ASTACK SEGMENT  STACK
       DW 100h DUP(?)
ASTACK  ENDS

DATA SEGMENT
        KEEP_PSP dw 0
       OVL_PATH db 50 dup(0),'$'
        DTA db 43 dup(0)
        OVL_FLAG db 0
        OVL_ADR dw 0
        OVL_CALL dd 0

     FIRST_OVL_NAME db 'OVL1.OVL',0
     SECOND_OVL_NAME db 'OVL2.OVL',0

 ERR7_MEM_MSG db 13,10,'Memory control block is destroyed',13, 10,'$'
 ERR8_MEM_MSG db 13,10,'Not enough memory for function',13, 10,'$'
   ERR9_MEM_MSG db 13,10,'Invalid adress',13, 10,'$'

  ERR2_DTA_SIZE_MSG db 13,10,'File not found',13, 10,'$'
  ERR3_DTA_SIZE_MSG db 13,10,'Route not found',13, 10,'$'

 ERR_NO_MEM_TO_OVL_MSG db 13,10,'Failed to allocate memory',13,10,'$'

 ERR1_LOAD_MSG db 13,10,'Incorrect function number',13, 10,'$'
  ERR2_LOAD_MSG db 13,10,'File not found',13, 10,'$'
  ERR3_LOAD_MSG db 13,10,'Route not found',13, 10,'$'
 ERR4_LOAD_MSG db 13,10,'Too many opened files',13, 10,'$'
   ERR5_LOAD_MSG db 13,10,'Disk error',13, 10,'$'
  ERR8_LOAD_MSG db 13,10,'Not enough memory',13, 10,'$'
  ERRA_LOAD_MSG db 13,10,'Invalid environment',13, 10,'$'
        DATA_END db 0
DATA ENDS

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

;-------------------------------------------------------------------------------
WriteMsg PROC near
          push ax
    mov ah,09h
    int 21h
         pop ax
    ret
WriteMsg ENDP
;-------------------------------------------------------------------------------
FREE_MEM PROC near
          push ax
          push bx
          push cx
         push dx

          mov bx, offset _END
          mov      ax,      offset
```

```
          DATA_END   add   bx,
          ax    add    bx,    40Fh
          mov cl, 4
          shr bx, cl
         mov ah, 4Ah
         int 21h
         jnc      end_fm

        irpc var1, 789         ;аналог range-based c++
                 cmp ax, &var1&
                 je ERRM_&var1&
         endm

        irpc var2, 789
                 ERRM_&var2&:
  mov dx, offset ERR&var2&_MEM_MSG   call
WriteMsg
  mov ax, 4C00h   int 21h
         endm

end_fm:
          pop dx
          pop cx
          pop bx
          pop ax
         ret
FREE_MEM ENDP
;-------------------------------------------------------------------------------
GET_PATH proc near
          push ax
          push es
          push si
          push di
         push dx

        mov es, KEEP_PSP
   mov   es,   es:[2Ch]
mov   si,0      lea   di,
OVL_PATH env_skip:
        mov dl, es:[si]
        cmp dl, 00
 je env_end     inc
si    jmp  env_skip
env_end:
         inc si
        mov dl, es:[si]
        cmp dl, 00
          jne env_skip
          add si, 3
write_path:
           mov dl, es:[si]
         cmp dl, 00
           je write_name
           mov [di], dl
           inc si
           inc di
           jmp write_path
write_name:
          mov si, bp
file_name:
```

```
        mov dl, byte ptr [si]
        mov byte ptr [di-7], dl
          inc di
          inc si
         test dl, dl
        jne file_name

          pop dx
          pop di
          pop si
          pop es
          pop ax
          ret
GET_PATH ENDP
;--------------------------------------------------------------------------
OVL_SIZE PROC near
          push ax
          push bx
          push cx
          push dx
          push es
          push ds
          push si
          push di
          push ss
        push sp

          mov dx, seg DTA
          mov ds, dx
 mov dx, offset DTA  mov
ah, 1ah
         int 21h

       mov dx, seg OVL_PATH
         mov ds, dx
           mov dx, offset
OVL_PATH      mov ah, 4eh
          mov cx, 0
         int 21h

        jnc no_dta_size_err

 irpc var1, 23  cmp ax, &var1&
          je SIZE_ERR_&var1&
          endm

          irpc var2,23
                SIZE_ERR_&var2&:
                      mov dx, offset ERR&var2&_DTA_SIZE_MSG
                           call WriteMsg
                           mov
          OVL_FLAG,1              jmp
          end_ovl_size
          endm

no_dta_size_err:
        mov si, offset DTA

 mov bx, [si+1ch]  mov
cl, 12
```

```
        shr bx, cl

  mov   ax,  [si+1Ah]
mov cl, 4  shr ax, cl

          add bx, ax
          add bx, 2

        mov ax, 4800h
         int 21h
        jnc no_load_err

  lea  dx,  ERR_NO_MEM_TO_OVL_MSG
call WriteMsg  mov ax, 4ch
         int 21h

no_load_err:
       mov OVL_ADR, ax

end_ovl_siz
e:
          pop sp
          pop ss
          pop di
          pop si
          pop ds
          pop es
          pop dx
          pop cx
          pop bx
          pop ax
          ret
OVL_SIZE ENDP
;-------------------------------------------------------------------------------
OVL_LOAD PROC
near    push ax
          push bx
          push cx
          push dx
          push es
          push ds
          push si
          push di
          push ss
          push sp
          lea dx,
          OVL_PAT
          H push
          ds pop es
          lea bx,
          OVL_ADR
          mov ax, 4b03h
         int 21h
       jc load_not_success

          mov ax, OVL_ADR
          mov word ptr
OVL_CALL+2, ax
        call OVL_CALL
```

```
         ;free
memory         mov
es, ax         mov
ax, 4900h
        int 21h
       jmp end_ovl_load

load_not_success:
      irpc var3, 123458A
              cmp ax, 0&var3&h
                je LOAD_ERR&var3&
         endm

      irpc var4,123458A
              LOAD_ERR&var4&:
                      mov dx, offset
ERR&var4&_LOAD_MSG                      call
WriteMsg                        mov OVL_FLAG,1
                jmp end_ovl_load
        endm

end_ovl_load:
         pop sp
         pop ss
         pop di
         pop si
         pop ds
         pop es
         pop dx
         pop cx
         pop bx
         pop ax
        ret
OVL_LOAD ENDP
;-------------------------------------------------------------------------------
MAIN PROC far
         mov ax, DATA
         mov ds, ax
         mov KEEP_PSP,
         es call FREE_MEM

         ;first ovl
         mov         bp,         offset
         FIRST_OVL_NAME        call
         GET_PATH call OVL_SIZE
         cmp
         OVL_FLAG,   1
         mov
         OVL_FLAG,   0
         je
         load_sec_ovl
         call OVL_LOAD

load_sec_ovl:   ;second ovl  mov bp,
offset   SECOND_OVL_NAME       call
GET_PATH
       call OVL_SIZE

 cmp OVL_FLAG, 1     je
quit
```

```
        call OVL_LOAD

quit:
 xor al, al  mov
ah, 4ch
          int
21h      ret
MAIN    ENDP
_END:
CODE ENDS
END MAIN
```