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. 8383 _____ Ларин А.

Преподаватель _____ Ефремов М.А,

Санкт-Петербург

2020

Цель работы.

Исследование возможности построения загрузочного модуля оверлейной структуры.

Выполнение

1. Написан код .EXE модуля, который выполняет 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.
- Предыдущие действия выполняются для следующего оверлейного
- сегмента.

2. Написан код оверлейных модулей, которые выводят адресс сегмента, в который загружены
3. Программа запущена, оба оверлейных модуля вывели один и тот же адрес. Результат приведен на рис. 1

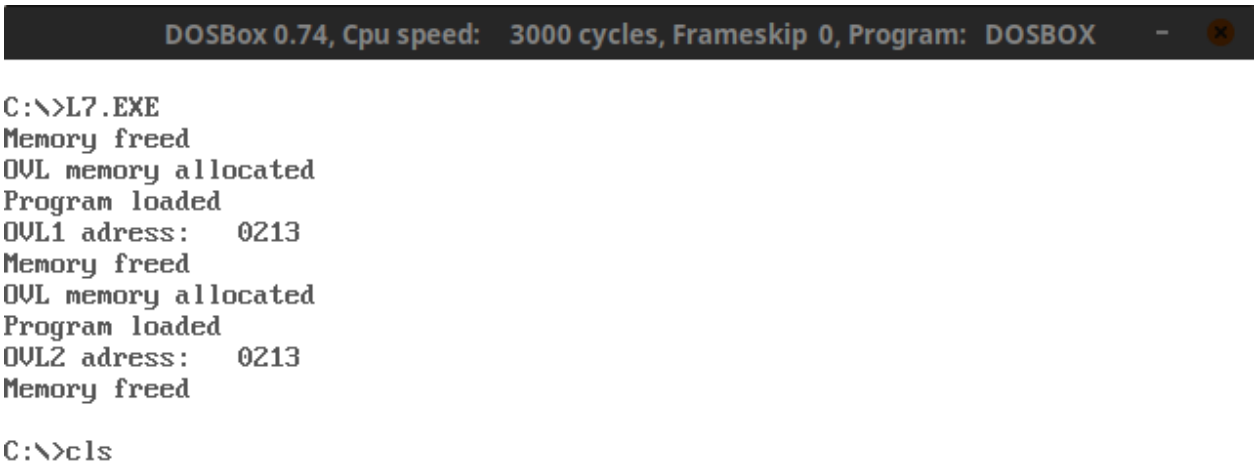

Рисунок 1 – Запуск программы

4. Программа запущена из другого каталога. Результат приведен на рис. 2

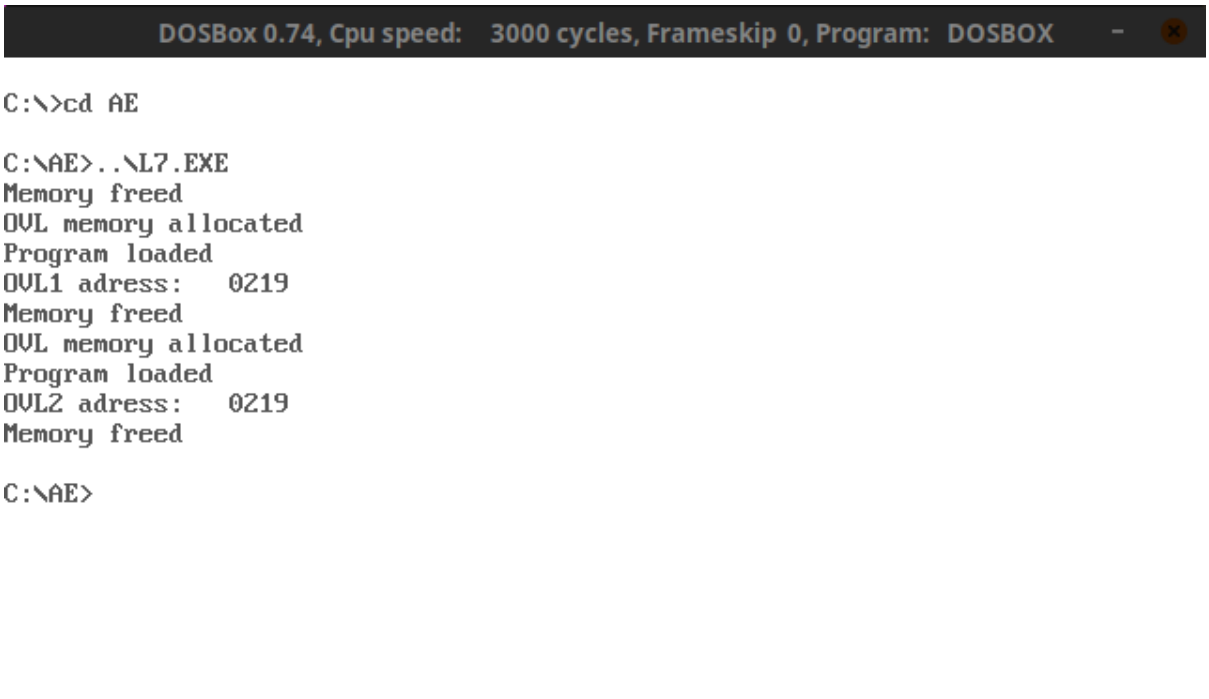

Рисунок 2 – Запуск программы из другого каталога

5. Программа запущена, когда второго оверлея нет в каталоге. Результат приведен на рис. 3

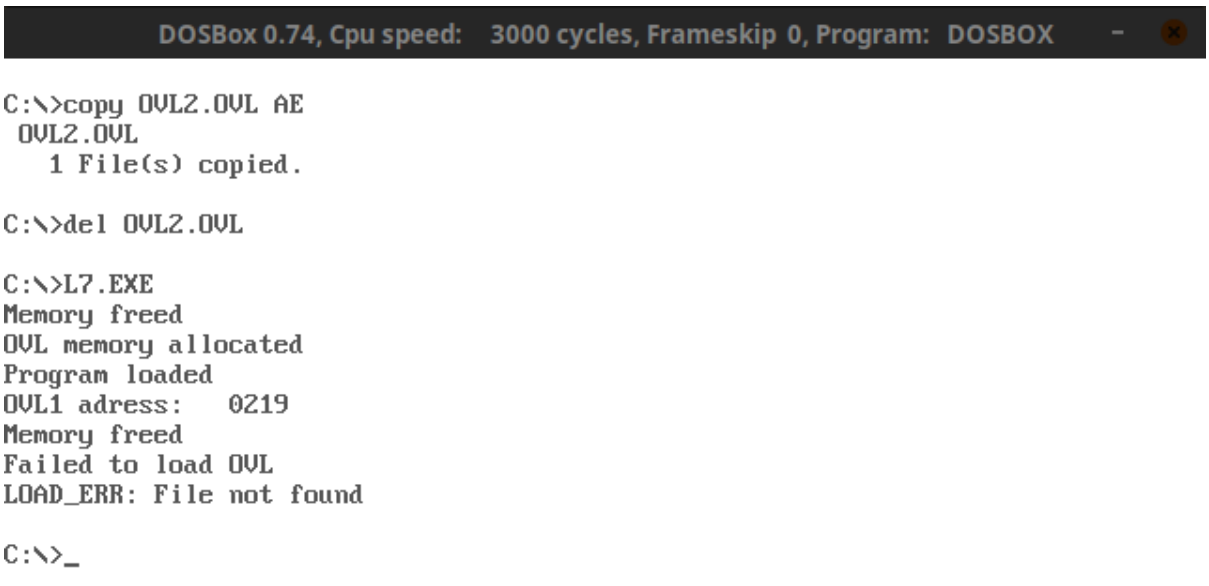

Рисунок 3 – Запуск программы без второго оверлея в директории

Контрольные вопросы

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

   Вызов модуля должен будет осуществляется со смещением 100h из за сегмента PSP, присутствующего в COM модуле

Выводы.

В результате работы были разобраны некоторые концепции языка ассемблера и работы операционной системы DOS. Были исследованы способы загрузки резидентной программы и установки своего обработчика прерываний. Написан модуль EXE с требуемым функционалом.

ПРИЛОЖЕНИЕ А

L7.ASM

```
DATA SEGMENT
  IS_MEM_FREED db 0
  KEEP_PSP dw 0
  CLN dw 0
  PATH db 80h dup(0)
  PROG_NAME db "L2.COM", 0
  KEEP_SP dw 0
  KEEP_SS dw 0

  STR_RET_CODE db 13, 10, "Ret.cod:        $",0
  STR_NEW_LINE db 0DH,0AH,'$'
  STR_MEM_ERROR db 'Failed to free memory$'
  STR_MEM_SUCCESS db 'Memory freed$'
  STR_MEM_ERROR7 db 'MEM_ERR: MCB corrupted$'
  STR_MEM_ERROR8 db 'MEM_ERR: Not genough memory to
execute$'
  STR_MEM_ERROR9 db 'MEM_ERR: Invalid memory block adress$'

  STR_LOAD_ERROR db 'Failed to load program$'
  STR_LOAD_SUCCESS db 'Program loaded$'
  STR_LOAD_ERROR1 db 'LOAD_ERR: Invalid function nunmber$'
  STR_LOAD_ERROR2 db 'LOAD_ERR: File not found$'
  STR_LOAD_ERROR5 db 'LOAD_ERR: disk error$'
  STR_LOAD_ERROR8 db 'LOAD_ERR: Not enough memory$'
  STR_LOAD_ERROR10 db 'LOAD_ERR: Invalid environment line$'
  STR_LOAD_ERROR11 db 'LOAD_ERR: Wrong format$'

  STR_END_CAUSE0 db 'Program finished normally$'
  STR_END_CAUSE1 db 'Program finished by Ctrl+Break$'
  STR_END_CAUSE2 db 'Program finished by divice error$'
  STR_END_CAUSE3 db 'Program went resident$'

  PARAMS     dw ? ;сегментный адрес среды
             dd ? ;сегмент и смещение командной строки
             dd ? ;сегмент и смещение первого FCB
             dd ? ;сегмент и смещение второго FCB

  DATA_END db 0
DATA ENDS

PSTACK SEGMENT STACK
  dw 128 dup(0)
PSTACK ENDS
```

```
CODE SEGMENT
 assume cs:CODE, ds:DATA, ss:PSTACK, es:NOTHING
;===============================================
FREE_MEM proc near
 push ax
 push bx
 push cx
 push dx
 push es

 mov ax,offset DATA_END
 mov bx,offset MEM_END
 add bx,ax
 add bx,30Fh
 mov cx,4
 shr bx,cl
 xor al,al
 mov ah,4Ah
 int 21h

 jnc MEM_FREED
 mov dx,offset STR_MEM_ERROR
 call PRINT
 call LN
 mov IS_MEM_FREED,0

 cmp ax, 7
   je MEM_ERROR7
 cmp ax, 8
   je MEM_ERROR8
 cmp ax, 9
   je MEM_ERROR9

 MEM_ERROR7:
   mov dx, offset STR_MEM_ERROR7
   jmp FREE_MEM_END
 MEM_ERROR8:
   mov dx, offset STR_MEM_ERROR8
   jmp FREE_MEM_END
 MEM_ERROR9:
   mov dx, offset STR_MEM_ERROR9
   jmp FREE_MEM_END

 MEM_FREED:
 mov IS_MEM_FREED,1
 mov dx, offset STR_MEM_SUCCESS
 FREE_MEM_END:
 call PRINT
```

```
  call LN

  pop es
  pop dx
  pop cx
  pop bx
  pop ax
  ret
FREE_MEM endp

;=========================================================
PROCESS proc near

  push ax
  push cx
  push dx
  push si
  push di
  push es

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


SEEK_START:
  mov ax, es:[si]
  inc si
  cmp ax, 0
  jne SEEK_START
  add si, 3
  mov di, 0
SEEK_CLN:
  mov al, es:[si]
  cmp al, 0
  je APPEND_NAME
  cmp al, '\'
  jne ADD_PATH_SYM
  mov CLN, di
ADD_PATH_SYM:
  mov BYTE PTR [PATH + di], al
  inc si
  inc di
  jmp SEEK_CLN
APPEND_NAME:
  mov di, CLN
  inc di
```

```
  add di, offset PATH
  mov si, offset PROG_NAME
  mov ax, ds
  mov es, ax
APPEND_SYMB:
  mov al, ds:[si]
  mov ds:[di], al
  inc di
  inc si
  cmp al, 0
  jne APPEND_SYMB


  pop dx
  pop si
  pop di
  pop es
  pop cx
  pop ax
  ret
PROCESS endp
;=========================================================
LOAD proc near
  push ax
  push bx
  push dx

  push ds
  push es
  mov KEEP_SP, SP
  mov KEEP_SS, ss

  mov ax, DATA
  mov es, ax
  mov bx, offset PARAMS
  mov dx, offset CLN
  mov [bx + 2], dx
  mov [bx + 4], ds
  mov dx, offset PATH
  mov ax, 4B00h
  int 21h

  mov ss, ds:KEEP_SS
  mov SP, ds:KEEP_SP
  pop es
  pop ds

  jnc LOAD_SUCCESS
```

```
  mov dx, offset STR_LOAD_ERROR
  call PRINT
  call LN
  cmp ax, 1
  je LOAD_ERROR1
  cmp ax, 2
  je LOAD_ERROR2
  cmp ax, 5
  je LOAD_ERROR5
  cmp ax, 8
  je LOAD_ERROR8
  cmp ax, 10
  je LOAD_ERROR10
  cmp ax, 11
  je LOAD_ERROR11
LOAD_ERROR1:
  mov dx, offset STR_LOAD_ERROR1
  call PRINT
  call LN
  jmp LOAD_END
LOAD_ERROR2:
  mov dx, offset STR_LOAD_ERROR2
  call PRINT
  call LN
  jmp LOAD_END
LOAD_ERROR5:
  mov dx, offset STR_LOAD_ERROR5
  call PRINT
  call LN
  jmp LOAD_END
LOAD_ERROR8:
  mov dx, offset STR_LOAD_ERROR8
  call PRINT
  call LN
  jmp LOAD_END
LOAD_ERROR10:
  mov dx, offset STR_LOAD_ERROR10
  call PRINT
  call LN
  jmp LOAD_END
LOAD_ERROR11:
  mov dx, offset STR_LOAD_ERROR11
  call PRINT
  call LN
  jmp LOAD_END
LOAD_SUCCESS:
  mov ax, 4D00h
  int 21h
```

```
 mov di, offset STR_RET_CODE
 mov [di + 10], al
 mov dx, offset STR_RET_CODE
 call PRINT
 call LN
 cmp ah, 0
 je LOAD_END0
 cmp ah, 1
 je LOAD_END1
 cmp ah, 2
 je LOAD_END2
 cmp ah, 3
 je LOAD_END3
LOAD_END0:
 mov dx, offset STR_END_CAUSE0
 call PRINT
 jmp LOAD_END
LOAD_END1:
 mov dx, offset STR_END_CAUSE1
 call PRINT
 jmp LOAD_END
LOAD_END2:
 mov dx, offset STR_END_CAUSE2
 call PRINT
 jmp LOAD_END
LOAD_END3:
 mov dx, offset STR_END_CAUSE3
 call PRINT


LOAD_END:
 pop dx
 pop bx
 pop ax
 ret
LOAD endp
;=====================================================
;UTILS
;=====================================================
PRINT PROC NEAR
 PUSH ax
 MOV AH, 09H
 INT 21H
 POP ax
 RET
PRINT   ENDP
;=====================================================
LN PROC
```

```
 push ax
 push dx
 mov dx, offset STR_NEW_LINE
 mov ah, 9h
 int 21h
 pop dx
 pop ax
 ret
LN ENDP
;=======================================================
PRINTLN PROC NEAR
 call PRINT
 call LN
 RET
PRINTLN ENDP



;==========MAIN=====MAIN====MAIN====MAIN===============
MAIN PROC
;MAIN init
 mov  ax, DATA                        ;ds setup
 mov  ds, ax
 mov KEEP_PSP, es


 call FREE_MEM

 cmp IS_MEM_FREED,1
 jne EXIT
 call PROCESS
 call LOAD

EXIT:
 xor al,al
 mov ah,4Ch
 int 21h

MAIN ENDP

MEM_END:
CODE ENDS




END MAIN
```

ПРИЛОЖЕНИЕ Б

OVL1.ASM

```
CODE SEGMENT
 ASSUME CS:CODE, DS:NOTHING, SS:NOTHING
 MAIN PROC FAR
   push AX
   push DX
   push DS
   push DI

   mov AX, CS
   mov DS, AX
   mov DI, offset STR
   add DI, 18
   call WRD_TO_HEX
   MOV DX, offset STR
   call PRINT

   pop DI
   pop DS
   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
   xor   ah,ah
   call BYTE_TO_HEX
   mov   [di],ah
   dec   di
   mov   [di],al
   pop   bx
   ret
 WRD_TO_HEX   ENDP

 PRINT PROC NEAR
   push DX
   push AX

   mov AH, 09h
   int 21h

   pop AX
   pop DX
   ret
 PRINT ENDP


STR db "OVL1 adress:          ", 13, 10, '$'

CODE ENDS
END MAIN
```

OVL2.ASM

```
CODE SEGMENT
 ASSUME CS:CODE, DS:NOTHING, SS:NOTHING
 MAIN PROC FAR
   push AX
   push DX
   push DS
   push DI

   mov AX, CS
   mov DS, AX
   mov DI, offset STR
   add DI, 18
   call WRD_TO_HEX
   MOV DX, offset STR
   call PRINT

   pop DI
   pop DS
   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
    xor   ah,ah
    call BYTE_TO_HEX
    mov   [di],ah
    dec   di
    mov   [di],al
    pop   bx
    ret
  WRD_TO_HEX   ENDP

  PRINT PROC NEAR
    push DX
    push AX

    mov AH, 09h
    int 21h

    pop AX
    pop DX
    ret
  PRINT ENDP


STR db "OVL2 adress:          ", 13, 10, '$'

CODE ENDS
END MAIN
```