МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра математического обеспечения и применения ЭВМ

ОТЧЕТ

по лабораторной работе № 7

по дисциплине «Операционные системы»

Тема: «Построение модуля оверлейной структуры»

Студент гр. 8304 _____ Сергеев А.Д.

Преподаватель _____ Губкин А.Ф.

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.

В этой работе также рассматривается приложение, состоящее из нескольких модулей, поэтому все модули помещаются в один каталог и вызываются с использованием полного пути.

Постановка задачи:

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 объём памяти, достаточный для его загрузки.
3. Файл оверлейного сегмента загружается и выполняется.
4. Освобождается память, отведённая 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 отчёт. Оформите отчёт в соответствии с требованиями.

Необходимые сведения для составления программы.

Для организации программы, имеющей оверлейную структуру, используется функция 4B03h прерывания int 21h. Эта функция позволяет в отведённую область памяти, начинающуюся с адреса сегмента, загрузить программу, находящуюся в файле на диске. Передача управления загруженной программе этой функцией не осуществляется и префикс сегмента программы (PSP) не создаётся.

Перед загрузкой оверлея вызывающая программа должна освободить память по функции 4Ah прерывания int 21h. Затем определить размер оверлея. Это можно сделать с помощью функции 4Eh прерывания 21h. Перед обращением к функции необходимо определить область памяти размером в 43 байта под буфер DTA, которую функция заполнит, если файл не будет найден.

Оверлейные сегмент не является загрузочным модулем типов .COM или .EXE. Он представляет собой кодовый сегмент, который оформляется в ассемблере как функция с точкой входа по адресу 0 и возврат осуществляется командой RETF. Это необходимо сделать, потому что возврат управления должен быть осуществлён в программу, выполняющую оверлейный сегмент. Если использовать функции выхода 4Ch прерывания int 21h, то программа закончит свою работу.

Описание программы.

В результате выполнения лабораторной работы была написана программа, описание функций которой представлено ниже.

- STRONG_OVERLOAD - загружает полный путь к файлу, адрес названия которого передан в AX в отдельную область памяти;

- OVER_EXEC - выделяет память, загружает туда модуль оверлейной структуры и выполняет его;
- MEMORY_FREE - освобождение незанятого программой места в памяти;
- PRINT_STRING - вывод сообщения на экран;

Оверлейные модули содержат следующие функции:

- PRINT - вывод сообщения на экран;
- TETR_TO_HEX - перевод символа в 16 системе счисления в ASCII-код этого символа;
- BYTE_TO_HEX - перевод байта в 16 системе счисления в ASCII-строку;
- WRD_TO_HEX - перевод регистра в 16 системе счисления в ASCII-строку и запись её по адресу DI;

Ход работы

Написание исходного кода производилось в редакторе Atom на базе операционной системы Windows 10, сборка и отладка производились в эмуляторе DOSBox.

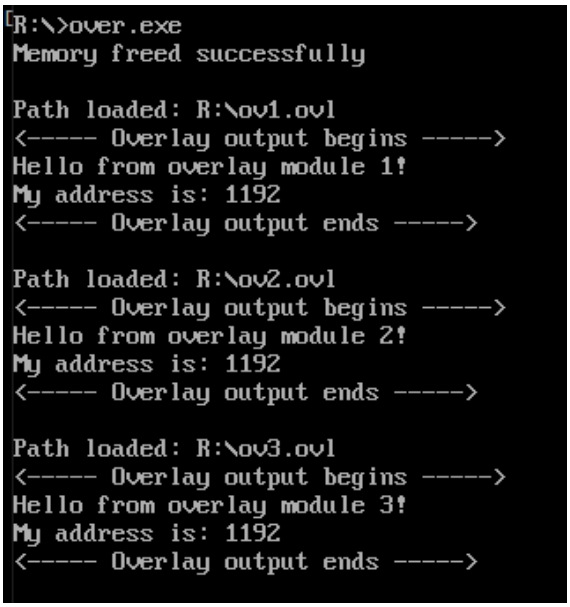

Рисунок 1 — Выполнение программы в одном каталоге с оверлейными модулями

Как видно из рисунка, программа завершилась в штатном режиме.

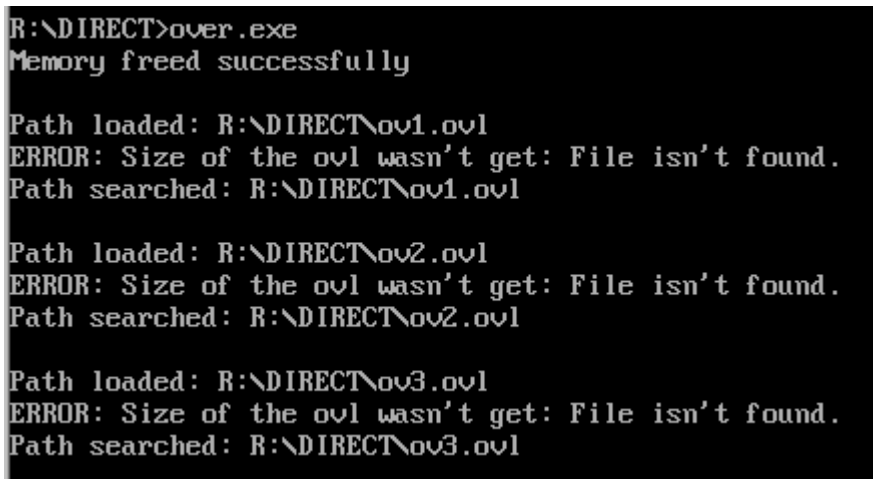

Рисунок 2 – Выполнение программы в другом каталоге

Как видно из рисунка, оверлейные модули в другом каталоге найдены не были.

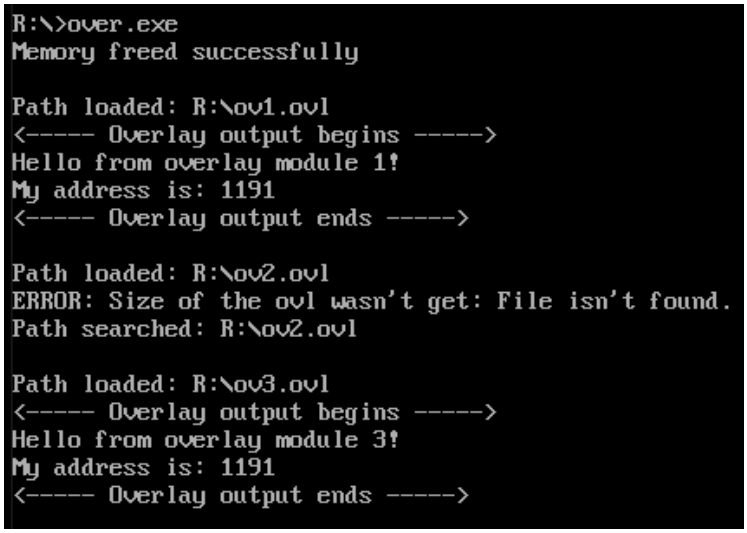

Рисунок 3 — Одного из модулей нет в каталоге

Как видно из рисунка, программа выполнила первый и третий модуль, выведя сообщение об ошибке во втором.

Вывод.

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

Контрольные вопросы.

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

В начале выделенной для оверлейного сегмента памяти необходимо сформировать и поместить PSP, соответственно сместив точку входа на 100h байт. Также необходимо будет сохранить, а затем восстановить регистры.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ. OVER.ASM

```
ASSUME CS:CODE, DS:DATA, SS:STACKK


DATA SEGMENT

        MEM_FREE_SUCCESS        db      "Memory      freed
successfully", 13, 10, "$"

        MEM_FREE_ERROR          db  "Error  freeing  memory$",
13, 10, "$"


        DELIM_START                  db    "<-----    Overlay
output begins ----->", 13, 10, "$"

       DELIM_END                db  13,  10,  "<-----  Overlay
output ends ----->", 13, 10, "$"


        LOADED_PATH             db "Path loaded: $"

        END_L                        db 13, 10, "$"


        LOADING_ERROR           db  "ERROR:  Size  of  the  ovl
wasn't get: $"

        LOADING_NO_FILE         db  "File  isn't  found.",  13,
10, "$"

        LOADING_SEARCHED        db "Path searched: $"


        EXEC_MEMORY             db "Not enought memory!", 13,
10, "$"
```

```
        EXEC_LOADING            db  "Overlay  isn't  loaded!",
13, 10, "$"


        PARAMS                      dw 0
        ENTRY_ADRESS            dd 0


        OVERLAY_1                   db "ov1.ovl", 0
        OVERLAY_2                   db "ov2.ovl", 0
        OVERLAY_3                   db "ov3.ovl", 0
        MODULE_PATH             db 100 dup(0)
    DATA ENDS


    STACKK SEGMENT STACK
        dw 100h   dup (0)
    STACKK ENDS



    CODE SEGMENT
    PRINT_STRING PROC   near
            push      ax
            mov  ah, 09h
            int       21h
            pop  ax
```

```
        ret
PRINT_STRING ENDP

MEMORY_FREE PROC near
        push    ax
        push    bx
        push    cx
        push    dx

        mov  bx, offset PROGRAM_END
        mov  ax, es
        sub  bx, ax
        mov  cl, 4
        shr  bx, cl
        inc  bx

        xor  ax, ax
        mov  ah, 4Ah
        int  21h

        jc      M_ERR
        mov  dx, offset MEM_FREE_SUCCESS
        call PRINT_STRING
        jmp  M_DEF
```

```
M_ERR:
        mov  dx, offset MEM_FREE_ERROR
        call PRINT_STRING
        stc

M_DEF:
        pop  dx
        pop  cx
        pop  bx
        pop  ax

        ret
MEMORY_FREE ENDP

STRONG_OVERLOAD PROC
        push      ax
        push      es
        push      si
        push      di
        push      dx

        push      ax
        mov  es, es:[2Ch]
```

```
        mov  si, 0

SKIP:
        mov  ax, es:[si]
        inc  si
        cmp  ax, 0
        jne  SKIP
        add  si, 3
        mov  di, offset MODULE_PATH

EMPATH:
        mov  al, es:[si]
        mov  BYTE PTR [di], al
        cmp  al, '\'
        je        RECALL
        inc  di
        inc  si
        jmp  EMPATH

RECALL:
        pop  si

APPEND:
        inc  di
```

```
        mov  al, [si]
        cmp  al, 0
        je        PROCEED
        mov  BYTE PTR [di], al
        inc  si
        jmp  APPEND

PROCEED:
        mov  BYTE PTR [di], '$'

        mov  dx, offset LOADED_PATH
        call      PRINT_STRING
        mov  dx, offset MODULE_PATH
        call      PRINT_STRING
        mov  dx, offset END_L
        call      PRINT_STRING

        pop  dx
        pop  di
        pop  si
        pop  es
        pop  ax

        ret
```

```
STRONG_OVERLOAD ENDP


OVER_EXEC PROC
          push ax
          push bx
          push cx
          push dx
          push si
          push      es

          mov  ah, 4Eh
          mov  cx, 0
          mov  dx, offset MODULE_PATH
          int  21h

          jnc  NO_ERROR
          mov  dx, offset LOADING_ERROR
          call      PRINT_STRING
          cmp  ax, 2
          je        NO_FILE
          cmp  ax, 3
          je        NO_FILE
          cmp  ax, 12h
          je        NO_FILE
```

```
        jmp  EXEC_DEF


NO_FILE:
        mov  dx, offset LOADING_NO_FILE
        call      PRINT_STRING


        mov  dx, offset LOADING_SEARCHED
        call      PRINT_STRING
        mov  dx, offset MODULE_PATH
        call      PRINT_STRING
        mov  dx, offset END_L
        call      PRINT_STRING
        jmp  EXEC_DEF


NO_ERROR:
        mov  si, 0080h
        add  si, 1Ah
        mov  bx, [si]
        mov  ax, [si + 2]
        mov       cl, 4
        shr  bx, cl
        mov       cl, 12
        shl  ax, cl
        add  bx, ax
```

```
        add  bx, 2

        mov  ah, 48h
        int  21h

        jnc  PREPARE_DATA
        mov  dx, offset EXEC_MEMORY
        call      PRINT_STRING
        jmp  EXEC_DEF

PREPARE_DATA:
        mov  PARAMS, ax
        mov  WORD PTR ENTRY_ADRESS + 2, ax

        mov  dx, offset MODULE_PATH
        push      ds
        pop  es
        mov  bx, offset PARAMS
        mov  ax, 4B03h
        int  21h

        jnc  LOADED
        mov  dx, offset EXEC_LOADING
        call      PRINT_STRING
```

```
        jmp       EXEC_DEF

LOADED:
        mov       ax, PARAMS
        mov  es, ax

        mov  dx, offset DELIM_START
        call      PRINT_STRING

        call      ENTRY_ADRESS

        mov  dx, offset DELIM_END
        call      PRINT_STRING

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

EXEC_DEF:
        pop  es
        pop  si
        pop  dx
        pop  cx
        pop  bx
```

```
        pop  ax

        ret
OVER_EXEC ENDP

MAIN PROC
    BEGIN:
        mov  ax, DATA
        mov  ds, ax

        call      MEMORY_FREE
        jc        MAIN_END
        mov  dx, offset END_L
        call      PRINT_STRING

        mov  ax, offset OVERLAY_1
        call      STRONG_OVERLOAD
        call      OVER_EXEC
        mov  dx, offset END_L
        call      PRINT_STRING

        mov  ax, offset OVERLAY_2
        call      STRONG_OVERLOAD
        call      OVER_EXEC
```

```
            mov  dx, offset END_L
            call      PRINT_STRING

            mov  ax, offset OVERLAY_3
            call      STRONG_OVERLOAD
            call      OVER_EXEC

      MAIN_END:
            mov  ah, 4Ch
            int  21h
MAIN ENDP

      PROGRAM_END:
CODE ENDS

END BEGIN
```

ПРИЛОЖЕНИЕ Б

ИСХОДНЫЙ КОД ПРОГРАММЫ. OV1.COM

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


OVL1 SEGMENT

MAIN1 PROC FAR

          push      ds

          push      dx

          push      di

          push      ax

          push      bx


          mov  ax, cs

          mov  ds, ax

          mov  bx, offset MESSAGE

          add  bx, 48

          mov  di, bx

          mov  ax, cs

          call      WRD_TO_HEX

          mov  dx, offset MESSAGE

          call      PRINT


          pop  bx

          pop  ax
```

```
          pop  di
          pop  dx
          pop  ds

          retf
MAIN1 ENDP

PRINT PROC NEAR
     push      ax
     mov  ah, 09h
     int  21h
     pop  ax
     ret
PRINT 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

    MESSAGE   DB 'Hello from overlay module 1!', 13, 10, 'My
address is:     $'

    OVL1 ENDS
    END
```

ПРИЛОЖЕНИЕ В

ИСХОДНЫЙ КОД ПРОГРАММЫ. OV2.COM

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


OVL2 SEGMENT

MAIN1 PROC FAR

          push      ds

          push      dx

          push      di

          push      ax

          push      bx


          mov  ax, cs

          mov  ds, ax

          mov  bx, offset MESSAGE

          add  bx, 48

          mov  di, bx

          mov  ax, cs

          call      WRD_TO_HEX

          mov  dx, offset MESSAGE

          call      PRINT


          pop  bx

          pop  ax
```

```
          pop  di
          pop  dx
          pop  ds

          retf
MAIN1 ENDP

PRINT PROC NEAR
     push      ax
     mov  ah, 09h
     int  21h
     pop  ax
     ret
PRINT 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

    MESSAGE   DB 'Hello from overlay module 2!', 13, 10, 'My
address is:     $'

    OVL2 ENDS
    END
```

ПРИЛОЖЕНИЕ Г

ИСХОДНЫЙ КОД ПРОГРАММЫ. OV3.COM

```
ASSUME CS:OVL3, DS:OVL3, SS:NOTHING, ES:NOTHING


OVL3 SEGMENT

MAIN1 PROC FAR

          push      ds

          push      dx

          push      di

          push      ax

          push      bx


          mov  ax, cs

          mov  ds, ax

          mov  bx, offset MESSAGE

          add  bx, 48

          mov  di, bx

          mov  ax, cs

          call      WRD_TO_HEX

          mov  dx, offset MESSAGE

          call      PRINT


          pop  bx

          pop  ax
```

```
          pop  di
          pop  dx
          pop  ds

          retf
MAIN1 ENDP

PRINT PROC NEAR
     push      ax
     mov  ah, 09h
     int  21h
     pop  ax
     ret
PRINT 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

    MESSAGE   DB 'Hello from overlay module 3!', 13, 10, 'My
address is:     $'

    OVL3 ENDS
    END
```