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. 9383 _____ Лысова А.М.

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 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.

Задание.

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 закончиться аварийно.

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

Необходимые теоретические положения:

Для организации программы, имеющей оверлейную структуру, используется функция 4B03h прерывания int 21h. Эта функция позволяет в отведенную 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 создается. Обращение к функции 4B03h:

```
AX=4B03h - код функции;
DS:DX - указывает на строку ASCIIZ, содержащую путь к оверлею;
ES:BX - указатель на блок параметров, который представляет собой два слова памяти, содержащих сегментный адрес загрузки программы.
```

Если флаг переноса CF=1 после выполнения функции, то произошли ошибки и регистр AX содержит код ошибки. Значение регистра AX характеризует следующие ситуации:

- 1 - несуществующая функция;
- 2 - файл не найден;
- 3 - маршрут не найден;
- 4 - слишком много открытых файлов;
- 5 - нет доступа;
- 8 - мало памяти;
- 10 - неправильная среда.

Если флаг переноса CF=0, то оверлей загружен в память.

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 будет найден.

Функция использует следующие параметры:

CX - значение байта атрибутов, которое для файла имеет значение 0;

DS:DX - указатель на путь к файлу, который записывается в формате строки ASCIIZ.

Если флаг переноса CF=1 после выполнения функции, то произошли ошибки и регистр AX содержит код ошибки. Значение регистра AX характеризует следующие ситуации:

- 2 - файл не найден;

- 3 - маршрут не найден.

Если CF=0, то в области памяти буфера DTA со смещением 1Ah будет находится младшее слово размера файла, а в слове со смещением 1Ch - старшее слово размера памяти в байтах.

Полученный размер файла следует перевести в параграфы, причем следует взять большее целое числа параграфов. Затем необходимо отвести память с помощью функции 48h прерывания 21h. После этого необходимо сформировать параметры для функции 4B03h и выполнить ее.

После отработки оверлея необходимо освободить память с помощью функции 49h прерывания int 21h. Обращение к этой функции содержит следующие параметры:

AH=49h - код функции;

ES - сегментный адрес освобождаемой памяти.

Оверлейный 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 осуществлен в программу, выполняющую оверлейный сегмент. Если использовать функции выхода 4Ch прерывания int 21h, то программа закончит свою работу.

Выполнение работы:

Была написан и отлажен программный EXE модуль, который выполняет необходимые функции.

Были написаны и отлажены оверлейные сегменты, которые выводят адрес сегмента, в который они загружены.

Программа была запущена из одной директории:

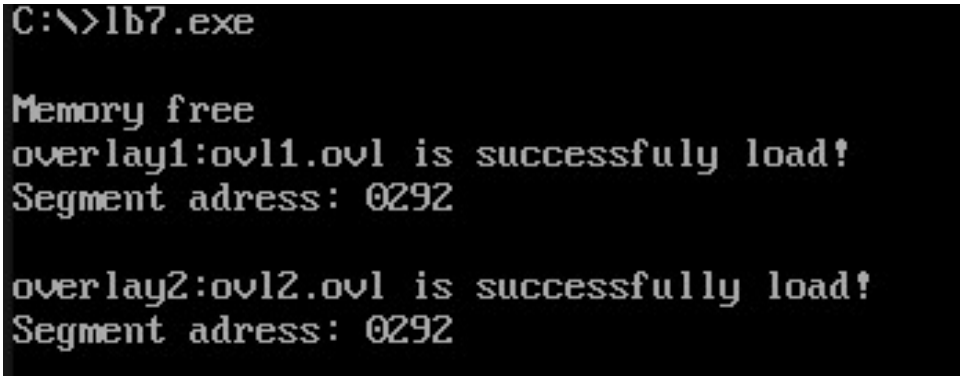

Рисунок 1: Запуск программы из текущей директории.

Также программа была запущена из другой директории:

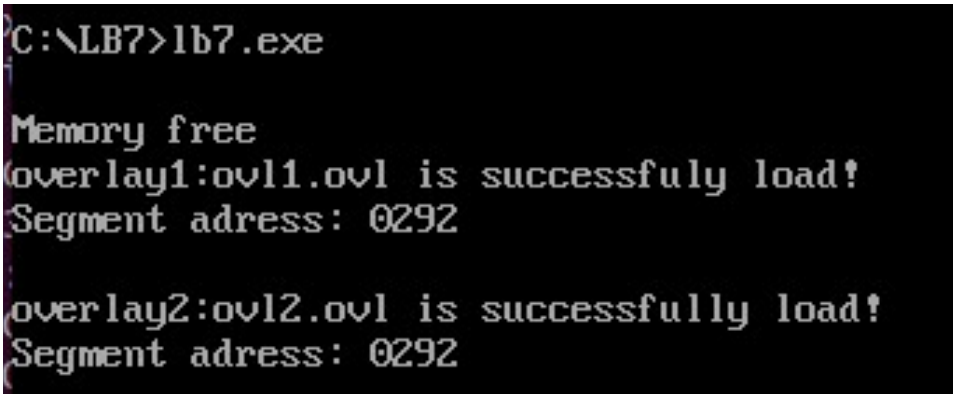

Рисунок 2: Запуск программы из другой директории.

После чего был убран из директории один из оверлейных модулей:

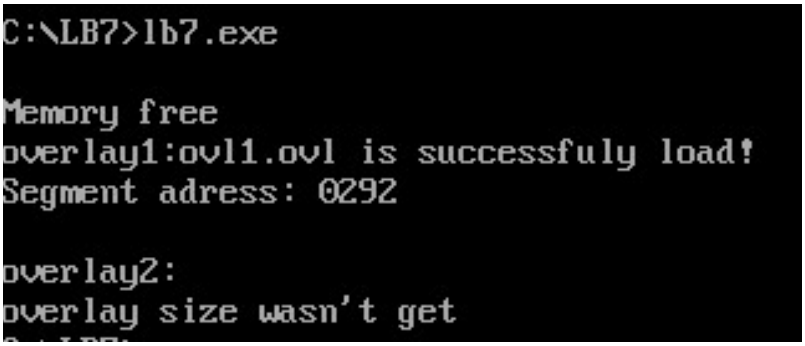

Рисунок 3: Запуск программы без одного оверлейного модуля.

Контрольные вопросы:

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

При использовании в качестве оверлейного сегмента COM модуля необходимо после того, как произойдет запись значений регистров в стек, поместить значение регистра CS в регистр DS, так как адрес сегмента данных совпадает с сегментом кода. А также вызвать данный модуль по смещению 100h, так как сегменты настроены на PSP.

Выводы.

Были исследование возможности построения загрузочного модуля оверлейной структуры и структура оверлейного сегмента, а также способ загрузки и выполнения оверлейных сегментов.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Файл: ovl1.asm

```
CODE SEGMENT
        ASSUME CS:CODE

MAIN PROC FAR
        push AX
        push DX
        push DS
        push DI

        mov AX, CS
        mov DS, AX
        lea DX, OVL_LOAD
        call WRITE_STRING

        lea DI, SEG_ADRESS
        add DI, 19
        mov AX, CS
        call WRD_TO_HEX

        lea DX, SEG_ADRESS
        call WRITE_STRING

        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
  call BYTE_TO_HEX
  mov [DI],AH
  dec DI
  mov [DI],AL
  pop BX
  ret
WRD_TO_HEX ENDP

WRITE_STRING PROC near
  push AX
  mov AH,09h
  int 21h
  pop AX
  ret
WRITE_STRING ENDP

OVL_LOAD db 'ovl1.ovl is successfuly load!',13,10,'$'
SEG_ADRESS db 'Segment adress:       ',13,10,'$'

CODE ENDS
END MAIN
```

Файл: ovl2.asm

```
CODE SEGMENT
        ASSUME CS:CODE

MAIN PROC FAR
        push AX
        push DX
        push DS
        push DI

        mov AX, CS
        mov DS, AX
        lea DX, OVL_LOAD
```

```
        call WRITE_STRING

        lea DI, SEG_ADRESS
        add DI, 19
        mov AX, cs
        call WRD_TO_HEX

        lea DX, SEG_ADRESS
        call WRITE_STRING

        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
  call BYTE_TO_HEX
  mov [DI],AH
  dec DI
```

```
  mov [DI],AL
  pop BX
  ret
WRD_TO_HEX ENDP

WRITE_STRING PROC near
  push AX
  mov AH,09h
  int 21h
  pop AX
  ret
WRITE_STRING ENDP

OVL_LOAD db 'ovl2.ovl is successfully load!',13,10,'$'
SEG_ADRESS db 'Segment adress:        ',13,10,'$'

CODE ENDS
END MAIN
```

Файл: lb7.asm

```
AStack SEGMENT STACK
        DW 128h DUP (0)
AStack ENDS

DATA SEGMENT
        PSP DW 0

        OVL_SEGMENT DW 0
        ADDRESS_OVL       DD 0

        NO_PATH  DB 13, 10, "Path wasn't found$"
        LOAD_ERROR  DB 13, 10, "Overlay wasn't load$"
        OVERLAY1  DB 13, 10, "Overlay1: $"
        OVERLAY2  DB 13, 10, "Overlay2: $"
        MEMORY_FREE       DB 13, 10, "Memory successfully free$"
        SIZE_ERROR  DB 13, 10, "Overlay size wasn't get$"
        NO_FILE  DB 13, 10, "File wasn't found$"
        OVL1  DB "ovl1.ovl", 0
        OVL2  DB "ovl2.ovl", 0

        PATH  DB 128h DUP(0)

        OFFSET_OVL_NAME  DW 0
        NAME_POS  DW 0
        MEMORY_ERROR  DW 0

        DTA_BUFFER  DB 43 DUP(0)
DATA ENDS

CODE SEGMENT
```

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

PRINT_STRING  PROC      near
        push    AX
        mov     AH, 09h
        int             21h
        pop     AX
        ret
PRINT_STRING      ENDP

MEM_FREE  PROC near
        lea     BX, PROGEND
        mov     AX, ES
        sub     BX, AX
        mov     CL, 8
        shr     BX, CL
        sub     AX, AX
        mov     AH, 4Ah
        int     21h
        jc              MCATCH
        mov     DX, offset MEMORY_FREE
        call    PRINT_STRING
        jmp     MDEFAULT
MCATCH:
        mov     MEMORY_ERROR, 1
MDEFAULT:
   ret
MEM_FREE  ENDP


OVERLAY PROC near
        push    AX
        push    BX
        push    CX
        push    DX
        push    SI

        mov     OFFSET_OVL_NAME, AX
        mov     AX, PSP
        mov     ES, AX
        mov     ES, ES:[2Ch]
        mov     SI, 0
zero_find:
        mov     AX, ES:[SI]
        inc     SI
        cmp     AX, 0
        jne     zero_find
        add     SI, 3
        mov     DI, 0
write_path:
        mov     AL, ES:[SI]
        cmp     AL, 0
```

```
        je              write_name_of_path
        cmp     AL, '\'
        jne     new_symbol
        mov     NAME_POS, DI
new_symbol:
        mov     BYTE PTR [PATH + DI], AL
        inc     DI
        inc     SI
        jmp     write_path
write_name_of_path:
        cld
        mov     DI, NAME_POS
        inc     DI
        add     DI, offset PATH
        mov     SI, OFFSET_OVL_NAME
        mov     AX, DS
        mov     ES, AX
UPDATE:
        lodsb
        stosb
        cmp     AL, 0
        jne     UPDATE
        mov     AX, 1A00h
        mov     DX, offset DTA_BUFFER
        int     21h

        mov     AH, 4Eh
        mov     CX, 0
        mov     DX, offset PATH
        int     21h

        jnc     no_error
        mov     DX, offset SIZE_ERROR
        call    PRINT_STRING
        cmp     AX, 2
        je              jmp_no_file
        cmp     AX, 3
        je              jmp_no_path
        jmp     path_end
jmp_no_file:
        mov     DX, offset NO_FILE
        call    PRINT_STRING
        jmp     path_end
jmp_no_path:
        mov     DX, offset NO_PATH
        call    PRINT_STRING
        jmp     path_end
no_error:
        mov     SI, offset DTA_BUFFER
        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     AX, 4800h
        int     21h

        jnc     set_segment
        jmp     path_end
set_segment:
        mov     OVL_SEGMENT, AX
        mov     DX, offset PATH
        push    DS
        pop     ES
        mov     BX, offset OVL_SEGMENT
    mov         AX, 4B03h
        int     21h

        jnc     load_success
        mov     DX, offset LOAD_ERROR
        call    PRINT_STRING
        jmp     path_end

load_success:
        mov     AX, OVL_SEGMENT
        mov     ES, AX
        mov     WORD PTR ADDRESS_OVL + 2, AX
        call    ADDRESS_OVL
        mov     ES, AX
        mov     AH, 49h
        int     21h

path_end:
        pop     SI
        pop     DX
        pop     CX
        pop     BX
        pop     AX
        ret
OVERLAY ENDP

MAIN PROC
        mov     AX, DATA
        mov     DS, AX
        mov     PSP, ES
        call    MEM_FREE
        cmp     MEMORY_ERROR, 1
        je      ending
        mov     DX, offset OVERLAY1
        call    PRINT_STRING
```

```
        mov     AX, offset OVL1
        call    OVERLAY
        mov     DX, offset OVERLAY2
        call    PRINT_STRING
        mov     AX, offset OVL2
        call    OVERLAY

ending:
        mov AX, 4C00h
        int 21h
PROGEND:

MAIN ENDP
CODE ENDS
END     MAIN
```