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. 0382 _____ Афанасьев Н. С.

Преподаватели _____ Ефремов М.А.

Санкт-Петербург

2022

Цель работы.

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

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

Выполнение работы.

При работе были использованы/созданы следующие процедуры:

- TETR_TO_HEX, BYTE_TO_HEX, WRD_TO_HEX – процедуры, описанные в шаблоне, для печати шестнадцатеричного числа.
- PRINT – процедура для вывода строки, отступ на которую содержится в DX, на экран, используя функцию 09h прерывания 21h.
- FREE_UP_MEM – процедура для освобождения памяти, не используемой программой, с использованием функции 4Ah прерывания 21h. Если произошла ошибка, то выводится сообщение об ошибке.
- COMPOSE_FILEPATH – процедура для подготовки строки с полным путём к файлу. Для начала записывается путь к вызываемой программе из переменных среды, затем ищется первый ‘\’ с конца, после этого с этой позиции записывается имя необходимого файла.
- PREP_OVERLAY – процедура для определения размера оверлейного сегмента с помощью функции 4Eh прерывания 21h, которая (в случае успешного нахождения файла) заполняет буфер DTA, область памяти в 43 байта под который выделяется заранее. Полученный и взятый из DTA размер файла переводится в целое число параграфов, для которых затем выделяется память с помощью функции 48h прерывания 21h. После этого подготавливаются параметры для следующего шага.
- LOAD_OVERLAY – процедура для непосредственного запуска оверлея через функцию 4B03h прерывания int 21h, которой передаются подготовленные ранее параметры. После отработки оверлея память из-под него освобождается с помощью функции 49h прерывания 21h

На первом и втором шаге из описанных процедур были составлены .EXE модуль и два оверлейных сегмента.

На третьем шаге программа запущена, когда текущим каталогом является каталог с разработанным модулем и оверлейными сегментами. (см. рис. 1).

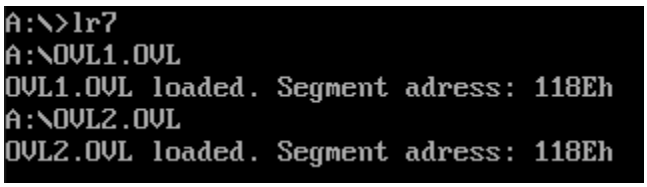

Рисунок 1 – Результат выполнения программы на третьем шаге

Как видно из выведенного сообщения, программа поочерёдно запустила два оверлейных сегмента, при чём запускались они с одного и того же адреса, перекрывая друг друга.

На третьем шаге программа была запущена из другого каталога (см. рис. 2).

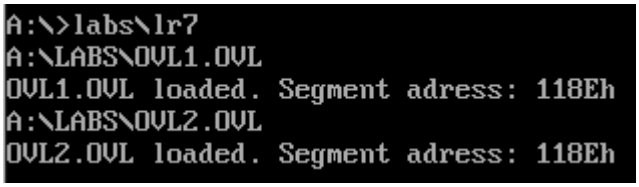

Рисунок 2 – Результат выполнения программы на четвёртом шаге

Как видно из результата, результаты совпадают с полученными ранее, то есть выбор текущего каталога не влияет на выполнения программы.

На четвёртом шаге программа была запущена, когда первого оверлейного сегмента не было в каталоге с основным модулем (см. рис. 3).

Рисунок 3 – Результат выполнения программы на пятом шаге

Как видно из результата, программа запустила второй оверлей, но закончилась аварийно при вызове первого оверлея, которого нет в каталоге.

Программный код см. в Приложении А

Вопросы.

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

– Программа должна учитывать, что в .COM модулях присутствует PSP, поэтому необходимо обращаться к модулю со смещением в 100h.

Выводы.

В результате выполнения лабораторной работы была исследована возможность построения загрузочного модуля оверлейной структуры. Также были исследованы структура оверлейного сегмента и способ загрузки и выполнения оверлейных сегментов.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЕ КОДЫ ПРОГРАММ

Название файла: lr7.asm

```
AStack    SEGMENT  STACK
    DW 64 DUP(?)
AStack    ENDS

DATA  SEGMENT
      DTA db 43 dup(0)
      PARAMETERS      DW 0,0
      L_ADDRESS  DD 0
      FILEPATH   DB 32 dup(0)
      OVL1_NAME  DB 'OVL1.OVL$'
      OVL2_NAME  DB 'OVL2.OVL$'
      MEM_ERROR  DB 'Memory freeing error',13,10,'$'
      READ_ERROR DB 13,10,'Overlay reading error',13,10,'$'
      ALLOC_ERROR     DB 'Memory allocation error',13,10,'$'
      LOAD_ERROR DB 'Overlay loading error',13,10,'$'
DATA ENDS

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

      FREE_UP_MEM PROC NEAR
            push AX
            push BX
            push DX

            mov BX, offset end_address
            mov AX, ES
            sub BX, AX
            shr BX, 4
            inc BX
            mov AH, 4Ah
            int 21h

            jnc free_mem_end
            mov DX, offset MEM_ERROR
            call PRINT

            free_mem_end:
            pop DX
            pop BX
            pop AX
            ret
      FREE_UP_MEM ENDP

      COMPOSE_FILEPATH PROC NEAR
            push dx
            push di
            push si
            push es
```

```
        mov ES, ES:[2Ch]
        mov SI, offset FILEPATH
        xor DI, DI

        read_byte:
        mov DL, ES:[DI]
        check_byte:
        inc DI
        cmp DL, 0
        jne read_byte
        mov DL, ES:[DI]
        cmp DL, 0
        jne check_byte

        add DI, 3
        write_path:
        mov DL, ES:[DI]
        mov [SI], DL
        inc SI
        inc DI
        cmp DL, 0
        jne write_path

        backslash_loop:
        dec SI
        cmp FILEPATH[si-1],'\'
        jne backslash_loop

        mov di,0
        write_filename:
        mov dl,bx[di]
        mov FILEPATH[si],dl
        inc si
        inc di
        cmp dl,'$'
        jne write_filename

        pop es
        pop si
        pop di
        pop dx
        ret
COMPOSE_FILEPATH ENDP


PREP_OVERLAY PROC NEAR
        push AX
        push BX
        push CX
        push DX

        mov AH, 1Ah
        lea DX, DTA
        int 21h

        mov AH, 4Eh
        lea DX, FILEPATH
        mov CX, 0
```

```
      int 21h
      jnc allocation
      mov DX, offset READ_ERROR
      call PRINT
      jmp overlay_size_end

      allocation:
      mov SI, offset DTA
      add SI, 1Ah
      mov BX, [SI]
      shr BX, 4
      mov AX, [SI+2]
      shl AX, 12
      add BX, AX
      add BX, 2
      mov AH, 48h
      int 21h
      jnc save_seg
      mov DX, offset ALLOC_ERROR
      call PRINT
      jmp overlay_size_end

      save_seg:
      mov PARAMETERS, AX
      mov PARAMETERS+2, AX

      overlay_size_end:
      pop DX
      pop CX
      pop BX
      pop AX
      ret
PREP_OVERLAY ENDP

LOAD_OVERLAY PROC NEAR
      push AX
      push DX
      push ES

      call COMPOSE_FILEPATH
      mov DX, offset FILEPATH
      call PRINT
      call PREP_OVERLAY

      mov DX, offset FILEPATH
      push DS
      pop ES
      mov BX, offset PARAMETERS
      mov AX, 4B03h
      int 21h
      jnc loaded

      mov DX, offset LOAD_ERROR
      call PRINT
      jmp overlay_end

      loaded:
      mov AX, PARAMETERS
```

```
            mov word ptr L_ADDRESS + 2, AX
            call L_ADDRESS
            mov ES, AX
            mov AH, 49h
            int 21h

            overlay_end:
            pop ES
            pop DX
            pop AX
            ret
      LOAD_OVERLAY ENDP

      PRINT PROC NEAR
            push AX
            mov AH, 09h
            int 21h
            pop AX
            ret
      PRINT ENDP

      Main PROC FAR
            sub AX,AX
            push AX
            mov AX, DATA
            mov DS,AX

            call FREE_UP_MEM
            mov BX, offset OVL1_NAME
            call LOAD_OVERLAY
            mov BX, offset OVL2_NAME
            call LOAD_OVERLAY

            xor AL,AL
            mov AH,4Ch
            int 21H
      Main ENDP
      end_address:
CODE ENDS
END Main
```

Название файла: ovl1.asm

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

      Main PROC FAR
            push AX
            push DX
            push DI
            push DS

            mov AX,CS
            mov DS,AX
            mov DX, offset SEG_STR
            mov DI, DX
            add DI, 38
            call WRD_TO_HEX
```

```
        call PRINT

        pop DS
        pop DI
        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

    PRINT PROC NEAR
        push AX
        mov AH, 09h
        int 21h
        pop AX
        ret
    PRINT ENDP

    SEG_STR DB 13,10,'OVL1.OVL loaded. Segment adress:     h',13,10,'$'
OVL1 ENDS
END Main
```

Название файла: ovl2.asm

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

     Main PROC FAR
          push AX
          push DX
          push DI
          push DS

          mov AX,CS
          mov DS,AX
          mov DX, offset SEG_STR
          mov DI, DX
          add DI, 38
          call WRD_TO_HEX
          call PRINT

          pop DS
          pop DI
          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

      PRINT PROC NEAR
            push AX
            mov AH, 09h
            int 21h
            pop AX
            ret
      PRINT ENDP

      SEG_STR DB 13,10,'OVL2.OVL loaded. Segment adress:    h',13,10,'$'
OVL2 ENDS
END Main
```