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. 8381 _____ Облизанов А.Д.

Преподаватель _____ Ефремов М.А.

Санкт-Петербург
2020

Цель работы.

Исследование возможности построение загрузочного модуля оверлейной структуры.

Основные теоретические положения.

Исследуется структура оверлейного сегмента и способ загрузки и выполнения оверлейных сегментов. Для запуска вызываемого оверлейного модуля используется функция 4B03h прерывания int 21h. Все загруженные 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.

Написание работы производилось на базе операционной системы Windows 10 в редакторе Visual Code. Сборка, отладка производились на базе эмулятора DOSBox 0.74-3.

Был написан текст исходного .EXE модуля с именем LR7.EXE. Описание процедур в программе представлено в табл. 1.

Таблица 1 – Описание процедур программы

Название	**Назначение**
PRINT_STRING	Вывод на экран строки, адрес которой содержится в DX
MEMORY_FREE	Процедура освобождения лишней зарезервированной программой памяти
EXECUTE_OVL	Загрузка и выполнение оверлейных сегментов

Основные этапы работы программы:

- Освобождается лишняя память, выделенная для моей программы
- Определяется размер оверлея, в случае его отсутствия выводится сообщение, и программа завершается
- Ищется файл оверлея, оверлей выполняется
- Осуществляется возврат в вызванную программу и очистка памяти под оверлей
- Операции повторяются для следующего оверлея

Результат запуска отлаженной программы, когда в текущей директории находятся оверлеи, представлен на рис. 1.

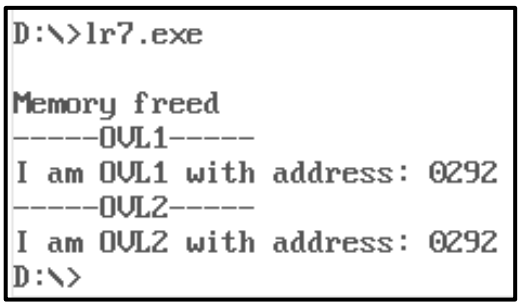

Рисунок 1 – Результат выполнения с оверлеями

Результат запуска отлаженной программы в другой директории, представлен на рис. 2.

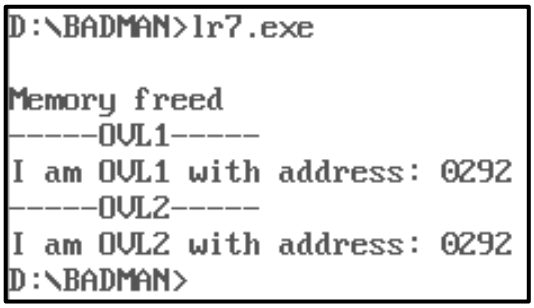

Рисунок 2 – Результат выполнения в другой директории

Результаты запуска отлаженной программы, когда один из оверлеев не находится в директории, представлен на рис. 3, на рис. 4, на рис. 5.

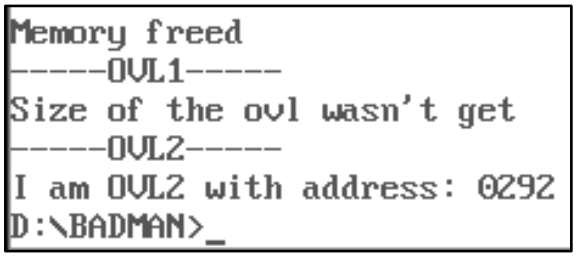

Рисунок 3 – Результат выполнения без одного оверлея

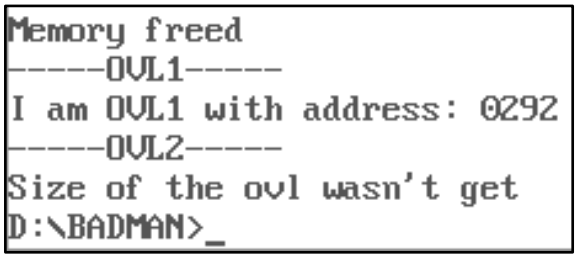

Рисунок 4 – Результат выполнения без другого оверлея

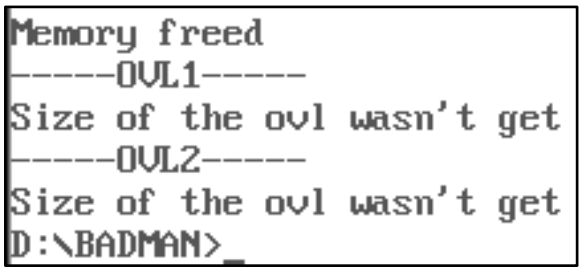

Рисунок 5 – Результат выполнения без обоих оверлеев

Контрольные вопросы.

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

Так как в .COM модулях код располагается с адреса 100h, то оверлейный сегмент необходимо вызывать по смещению 100h, иначе PSP не будет сформирован. Кроме того, необходимо сохранять и восстанавливать регистры.

Выводы.

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

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ. LR7.ASM

```
DATA SEGMENT
      PSP_SEGMENT dw 0

      OVL_PARAM_SEG                dw 0
      OVL_ADRESS                   dd 0

      STR_MEMORY_FREE        db 13, 10, "Memory freed$"
      STR_GETSIZE_ERROR            db 13, 10, "ERROR: Size of the ovl wasn't
get$"
      STR_NOFILE                   db 13, 10, "File wasn't found$"
      STR_NOPATH                   db 13, 10, "Path wasn't found$"
      STR_ERROR_LOAD               db 13, 10, "OVL wasn't load$"
      STR_OVL1_INFO                db 13, 10, "-----OVL1-----$"
      STR_OVL2_INFO                db 13, 10, "-----OVL2-----$"
      STR_OVL1                     db "ovl1.ovl", 0
      STR_OVL2                     db "ovl2.ovl", 0
      STR_PATH                     db 100h dup(0)
      OFFSET_OVL_NAME        dw 0
      NAME_POS                     dw 0
      MEMORY_ERROR                 dw 0

      DTA                          db 43 dup(0)
DATA ENDS

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

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

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

MEMORY_FREE       PROC
            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 STR_MEMORY_FREE
            call  PRINT_STRING
            jmp   MDEFAULT
      MCATCH:
            mov   MEMORY_ERROR, 1
      MDEFAULT:
      ret
MEMORY_FREE       ENDP


EXECUTE_OVL PROC
            push  AX
            push  BX
            push  CX
            push  DX
            push  SI

            mov   OFFSET_OVL_NAME, AX
            mov   AX, PSP_SEGMENT
            mov   ES, AX
            mov   ES, ES:[2Ch]
            mov   SI, 0
      FIND_ZERO:
            mov   AX, ES:[SI]
            inc   SI
            cmp   AX, 0
            jne   FIND_ZERO
            add   SI, 3
            mov   DI, 0
      WRITE_PATH:
            mov   AL, ES:[SI]
            cmp   AL, 0
            je          WRITE_PATH_NAME
            cmp   AL, '\'
            jne   NEW_SYMB
            mov   NAME_POS, DI
      NEW_SYMB:
            mov   BYTE PTR [STR_PATH + DI], AL
            inc   DI
            inc   SI
            jmp   WRITE_PATH
      WRITE_PATH_NAME:
```

```
      cld
      mov   DI, NAME_POS
      inc   DI
      add   DI, offset STR_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
      int   21h

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

      jnc   NOERROR
      mov   DX, offset STR_GETSIZE_ERROR
      call  PRINT_STRING
      cmp   AX, 2
      je          NOFILE
      cmp   AX, 3
      je          NOPATH
      jmp   PATH_ENDING
NOFILE:
      mov   DX, offset STR_NOFILE
      call  PRINT_STRING
      jmp   PATH_ENDING
NOPATH:
      mov   DX, offset STR_NOPATH
      call  PRINT_STRING
      jmp   PATH_ENDING
NOERROR:
      mov   SI, offset DTA
      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_SEG
      jmp   PATH_ENDING
SET_SEG:
      mov   OVL_PARAM_SEG, AX
      mov   DX, offset STR_PATH
      push  DS
      pop   ES
      mov   BX, offset OVL_PARAM_SEG
      mov   AX, 4B03h
      int   21h

      jnc   LO_SUCCESS
      mov   DX, offset STR_ERROR_LOAD
      call  PRINT_STRING
      jmp         PATH_ENDING

LO_SUCCESS:
      mov         AX, OVL_PARAM_SEG
      mov   ES, AX
      mov   WORD PTR OVL_ADRESS + 2, AX
      call  OVL_ADRESS
      mov   ES, AX
      mov   AH, 49h
      int   21h

PATH_ENDING:
      pop   SI
      pop   DX
      pop   CX
      pop   BX
      pop   AX
      ret
EXECUTE_OVL ENDP

BEGIN:
      mov   AX, DATA
      mov   DS, AX
      mov   PSP_SEGMENT, ES
      call  MEMORY_FREE
      cmp   MEMORY_ERROR, 1
      je          MAIN_END
      mov   DX, offset STR_OVL1_INFO
```

```
            call  PRINT_STRING
            mov   AX, offset STR_OVL1
            call  EXECUTE_OVL
            mov   DX, offset STR_OVL2_INFO
            call  PRINT_STRING
            mov   AX, offset STR_OVL2
            call  EXECUTE_OVL

      MAIN_END:
            mov AX, 4C00h
            int 21h
      PROGEND:
CODE ENDS
END BEGIN
```

ПРИЛОЖЕНИЕ Б

ИСХОДНЫЙ КОД ПРОГРАММЫ. OVL1.ASM

```
CODE SEGMENT
      ASSUME CS:CODE, DS:NOTHING, SS:NOTHING

MAIN PROC FAR
            push  AX
            push  DX
            push  DS

            mov   AX, CS
            mov   DS, AX
            mov   DX, offset STR_OUTPUT
            call  PRINT_STRING
            call  WRITE_HEX_WORD

            pop   DS
            pop   DX
            pop   AX
      retf
MAIN ENDP

STR_OUTPUT        db 13, 10, "I am OVL1 with address: $"

PRINT_STRING PROC
            push  AX
            mov   AH, 9h
            int   21h
            pop   AX
      ret
PRINT_STRING ENDP

      WRITE_HEX_BYTE PROC
            push  AX
            push  BX
            push  DX

            mov   AH, 0
            mov   BL, 16
            div   BL
            mov   DX, AX
            mov   AH, 02h
            cmp   DL, 0Ah
            jl          PRINT
            add   DL, 7
      PRINT:
```

```
            add   DL, '0'
            int   21h;

            mov   DL, DH
            cmp   DL, 0Ah
            jl          PRINT2
            add   DL, 7
      PRINT2:
            add   DL, '0'
            int   21h;

            pop   DX
            pop   BX
            pop   AX
            ret
WRITE_HEX_BYTE ENDP

WRITE_HEX_WORD PROC
            push  AX
            push  AX
            mov   AL, AH
            call  WRITE_HEX_BYTE
            pop   AX
            call  WRITE_HEX_BYTE
            pop   AX
      ret
WRITE_HEX_WORD ENDP

CODE ENDS
END MAIN
```

ПРИЛОЖЕНИЕ В

ИСХОДНЫЙ КОД ПРОГРАММЫ. OVL2.ASM

```
CODE SEGMENT
      ASSUME CS:CODE, DS:NOTHING, SS:NOTHING

MAIN PROC FAR
            push  AX
            push  DX
            push  DS

            mov   AX, CS
            mov   DS, AX
            mov   DX, offset STR_OUTPUT
            call  PRINT_STRING
            call  WRITE_HEX_WORD

            pop   DS
            pop   DX
            pop   AX
      retf
MAIN ENDP

STR_OUTPUT        db 13, 10, "I am OVL2 with address: $"

PRINT_STRING PROC
            push  AX
            mov   AH, 9h
            int   21h
            pop   AX
      ret
PRINT_STRING ENDP

WRITE_HEX_BYTE PROC
            push  AX
            push  BX
            push  DX

            mov   AH, 0
            mov   BL, 16
            div   BL
            mov   DX, AX
            mov   AH, 02h
            cmp   DL, 0Ah
            jl          PRINT
            add   DL, 7
      PRINT:
```

```
            add   DL, '0'
            int   21h;

            mov   DL, DH
            cmp   DL, 0Ah
            jl          PRINT2
            add   DL, 7
      PRINT2:
            add   DL, '0'
            int   21h;

            pop   DX
            pop   BX
            pop   AX
            ret
WRITE_HEX_BYTE ENDP

WRITE_HEX_WORD PROC
            push  AX
            push  AX
            mov   AL, AH
            call  WRITE_HEX_BYTE
            pop   AX
            call  WRITE_HEX_BYTE
            pop   AX
      ret
WRITE_HEX_WORD ENDP

CODE ENDS
END MAIN
```