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

Студентка гр. 7383 Маркова А.В.

Преподаватель Ефремов М.А.

Санкт-Петербург

2019

Постановка задачи.

Исследовать возможность построения загрузочного модуля оверлейной структуры. В отличии от предыдущих лабораторных работ в этой работе рассматривается приложение, состоящее из нескольких модулей, а не из одного модуля простой структуры. В этом случае разумно предположить, что все модули приложения находятся в одном каталоге и полный путь в этот каталог можно взять из среды. Понятно, что такое приложение должно запускаться в соответствии со стандартными ОС.

Также нужно исследовать способ загрузки и выполнения оверлейных сегментов. Для запуска вызываемого оверлейного модуля используется функция 4B03h прерывания int 21h.

Описание процедур, используемых в работе:

- LINE_OUTPUT – выводит сообщение на экран;
- DTA_SET – установка адреса области обмена с диском (DTA блока);
- TETR_TO_HEX – вспомогательная функция, которая переводит из двоичной в шестнадцатеричную систему счисления, используется для работы функции BYTE_TO_HEX;
- BYTE_TO_HEX – переводит байтовое число из регистра AL в шестнадцатеричную систему счисления;
- WRD_TO_HEX – переводит число из регистра AX в шестнадцатеричную систему счисления;
- FREE_MEMORY – освобождает место в памяти, используя функцию 4Ah прерывания int 21h;
- FIND_PATH – извлечение полного имени файла оверлея из среды;
- ALLOCATE_MEMORY_FOR_OVL – выделение памяти под оверлей;
- PROGRAM_CALL_OVL – вызов программы оверлея;
- PROCESSING – обработка оверлея: нахождение полного пути, выделение памяти под файл и вызов программы;
- BEGINNING – главная процедура.

Таблица 1 – Описание структуры данных:

Название	**Тип**	**Назначение**
Mem_7	db	Строка – сообщение об ошибке освобождения памяти: 7 – разрушен управляющий блок памяти; 8 – недостаточно памяти для выполнения функции; 9 – неверный адрес блока памяти
Mem_8	db	
Mem_9	db	
Load_1	db	Строка-сообщение об ошибке загрузки файла оверлея: 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
Load_2	db	
Load_3	db	
Load_4	db	
Load_5	db	
Load_8	db	
Load_10	db	
Err_alloc	db	Строка-сообщение об ошибке выделения памяти для загрузки оверлея
Path	db	Строка-сообщение для вывода вычисленного пути до оверлея
File_2	db	Строка-сообщение об ошибке поиска запускаемого файла оверлея: 2 – файл не найден; 3 – маршрут не найден
File_3	db	
OvlPath	db	Строка, содержащая полный путь до запускаемого оверлея
DTA	db	Переменная, содержащая сведения об области обмена с диском (DTA блок)
Keep_psp	dw	Переменная для сохранение содержимого регистра PSP
SegAdr	dw	Переменная, для сохранения сегментного адреса освобождённого для оверлея блока памяти

CallAdr	dd	Переменная, хранящая адрес, по которому вызывается оверлей
Ovl1	db	Название первого оверлея
Ovl2	db	Название второго оверлея

Ход работы.

1. Был написан и отлажен программный модуль .EXE, который выполняет следующие функции:
   - Освобождает память для загрузки оверлеев;
   - Читает размер файла оверлея и запрашивает объем памяти, достаточный для его загрузки;
   - Загружает и выполняет файл оверлейного сегмента;
   - Освобождает память, отведённую для оверлейного сегмента.
2. Смонтирован виртуальный диск k с каталогом tasm.
3. Было произведено транслирование программы с помощью строки tasm «имя файла», в последствии чего создался объектный файл.
4. Линковка загрузочного модуля с помощью команды tlink. Результат представлен на рис. 1.

```
Z:\>mount K D:/7383/LR7/TASM
Drive K is mounted as local directory D:/7383/LR7/TASM\

Z:\>K:

K:\>tasm lr7
Turbo Assembler  Version 3.1  Copyright (c) 1988, 1992 Borland International

Assembling file:   lr7.ASM
Error messages:    None
Warning messages:  None
Passes:            1
Remaining memory:  469k


K:\>tlink lr7
Turbo Link  Version 5.1 Copyright (c) 1992 Borland International
```

Рисунок 1 – Результат сборки программного модуля

5. Получен LR7.EXE модуль из исходного кода для .EXE модуля.
6. Были получены файлы с расширением .OVL путём конвертирования программных модулей, используя команду, показанную на рис. 2.

```
K:\>exe2bin.exe lr7_ovl1.exe lr7_ovl1.ovl
```

Рисунок 2 – Получение файлов с расширением .ovl

7. Запуск отлаженной программы, когда текущей каталог является каталогом с разработанными модулями. Оверлейный сегмент выводит адрес сегмента, в который он загружен. Выведенная информация представлена на рис. 3.

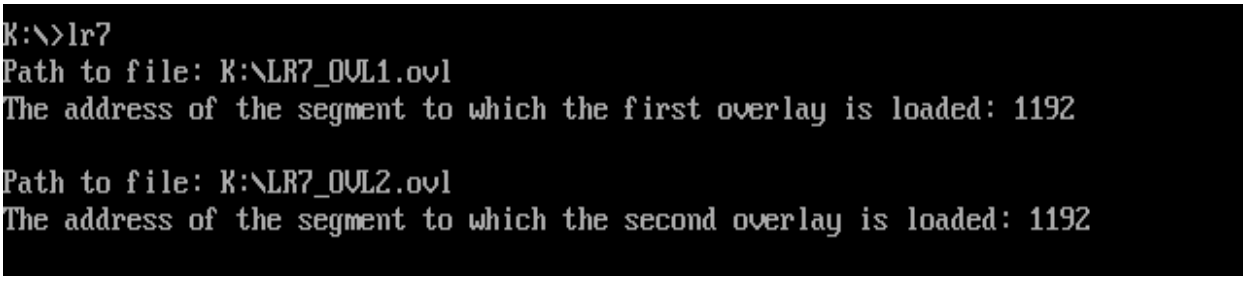

Рисунок 3 – Итог работы программы lr7.exe

Видно, что файлы были загружены с одного и того же адреса.

8. Запуск приложения из другого каталога. Результат показан на рис. 4.

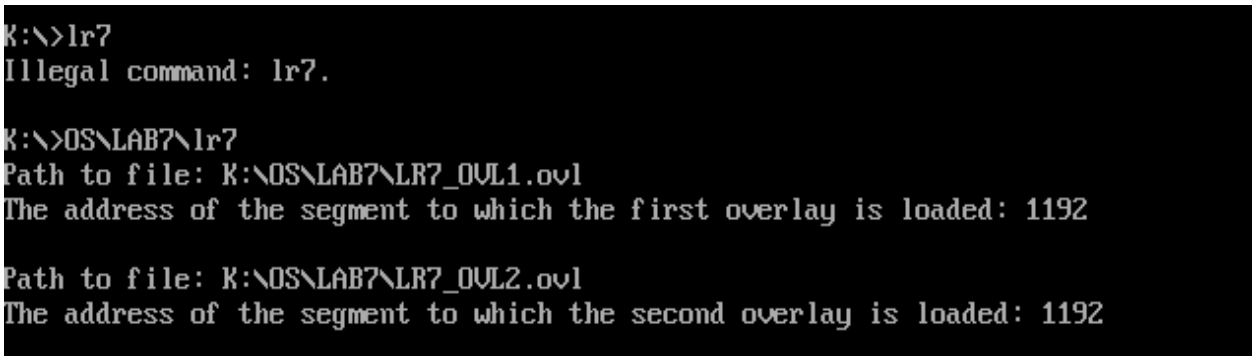

Рисунок 4 – Результат запуска программы lr7.exe

Программный модуль был запущен успешно.

9. Запуск приложения в случае, когда одного оверлея нет в каталоге. Выведенная информация представлена на рис. 5 – 6.

Рисунок 5 – Повторный запуск программы, когда отсутствует lr7_ovl1.ovl

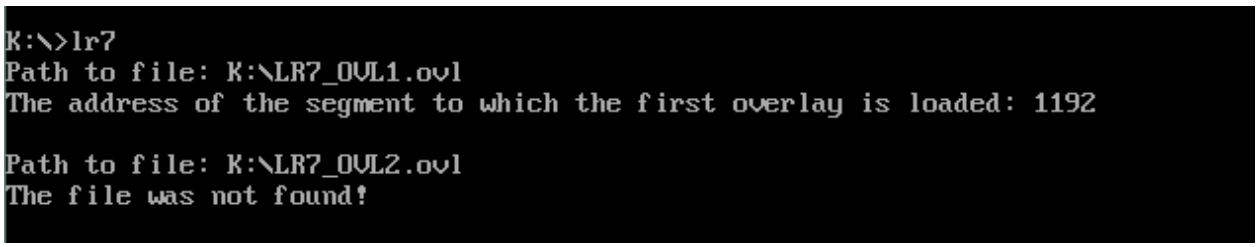

Рисунок 6 – Повторный запуск программы, когда отсутствует lr7_ovl2.ovl

В случае, когда одного из оверлеев нет в каталоге программа завершает свою работу аварийно.

Выводы.

В ходе данной лабораторной работы была изучена возможность построения загрузочного модуля оверлейной структуры, а также исследован способ загрузки и выполнения оверлейных сегментов. Было создано приложение, состоящее из нескольких модулей, все модули которого помещаются в один каталог и вызываются с использованием полного пути. Код программы представлен в приложении А.

Ответы на контрольные вопросы.

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

При использовании в качестве оверлейного сегмента .COM модуль, необходимо вызывать его по смещению 100h, так как в .COM файлах код располагается с адреса 100h.

ПРИЛОЖЕНИЕ А

LR7.ASM

```
EOFLINE EQU '$'
ASTACK SEGMENT STACK
      DW 64 DUP (?)
ASTACK ENDS
;ДАННЫЕ
;---------------------------------------------------------------
DATA SEGMENT
      LOAD_1     DB 0DH, 0AH,'THE OVERLAY WAS NOT BEEN LOADED: NON-
EXISTENT FUNCTION!',0DH,0AH,EOFLINE
      LOAD_10   DB 0DH, 0AH,'THE OVERLAY WAS NOT BEEN LOADED: INCORRECT
ENVIRONMENT!',0DH,0AH,EOFLINE
      LOAD_4    DB 0DH, 0AH,'THE OVERLAY WAS NOT BEEN LOADED: TOO MANY
OPEN FILES!',0DH,0AH,EOFLINE
      LOAD_3     DB 0DH, 0AH,'THE OVERLAY WAS NOT BEEN LOADED: ROUTE
NOT FOUND!',0DH,0AH,EOFLINE
      LOAD_2    DB 0DH, 0AH,'THE OVERLAY WAS NOT BEEN LOADED: FILE NOT
FOUND!',0DH,0AH,EOFLINE
      LOAD_8     DB 0DH, 0AH,'THE OVERLAY WAS NOT BEEN LOADED: LOW
MEMORY!',0DH,0AH,EOFLINE
      LOAD_5     DB 0DH, 0AH,'THE OVERLAY WAS NOT BEEN LOADED: NO
ACCESS!',0DH,0AH,EOFLINE
      MEM_8        DB 0DH, 0AH,'NOT ENOUGH MEMORY TO PERFORM THE
FUNCTION!',0DH,0AH,EOFLINE
      ERR_ALLOC DB 0DH, 0AH,'FAILED TO ALLOCATE MEMORY TO LOAD
OVERLAY!',0DH,0AH,EOFLINE
      MEM_9          DB 0DH, 0AH,'WRONG ADDRESS OF THE MEMORY
BLOCK!',0DH,0AH,EOFLINE
      MEM_7               DB   0DH,   0AH,'MEMORY   CONTROL   UNIT
DESTROYED!',0DH,0AH,EOFLINE
      FILE_3    DB 0DH, 0AH,'THE ROUTE WAS NOT FOUND!',0DH,0AH,EOFLINE
      FILE_2    DB 0DH, 0AH,'THE FILE WAS NOT FOUND!',0DH,0AH,EOFLINE
      PATH      DB 'PATH TO FILE: ',EOFLINE
      OVL1    DB 'LR7_OVL1.OVL',0000H
      OVL2    DB 'LR7_OVL2.OVL',0000H
      OVLPATH   DB 64  DUP (?),EOFLINE
      DTA       DB 43 DUP (?)
      KEEP_PSP  DW 0000H
      SEGADR    DW 0000H
      CALLADR      DD 0000H
DATA ENDS
;---------------------------------------------------------------
CODE SEGMENT
```

```
    ASSUME CS:CODE, DS:DATA, ES:DATA, SS:ASTACK
START: JMP BEGINNING
;ПРОЦЕДУРЫ
;---------------------------------------------------------------
LINE_OUTPUT PROC NEAR
      MOV  AH, 0009H
      INT  21H
      RET
LINE_OUTPUT ENDP
;---------------------------------------------------------------
DTA_SET PROC NEAR
    PUSH DX
    LEA  DX, DTA
    MOV  AH, 1AH
    INT  21H
    POP  DX
DTA_SET ENDP

FREE_MEMORY PROC NEAR

      MOV  BX, OFFSET LAST_BYTE
      MOV  AX, ES
      SUB  BX, AX
      MOV  CL, 0004H
      SHR  BX, CL
      MOV  AH, 4AH
      INT  21H
      JNC  WITHOUT_ERROR
      JMP  WITH_ERROR

      WITHOUT_ERROR:
           RET

      WITH_ERROR:
    MEM_7_ERROR:
           CMP  AX, 0007H
           JNE  MEM_8_ERROR
           MOV  DX, OFFSET MEM_7
           JMP  END_ERROR
    MEM_8_ERROR:
            CMP  AX, 0008H
            JNE  MEM_9_ERROR
            MOV  DX, OFFSET MEM_8
            JMP  END_ERROR
      MEM_9_ERROR:
```

```
          MOV  DX, OFFSET MEM_9

    END_ERROR:
           CALL LINE_OUTPUT
           XOR  AL, AL
           MOV  AH, 4CH
           INT  21H
FREE_MEMORY ENDP

FIND_PATH PROC NEAR
      PUSH ES
      MOV  ES,  ES:[2CH]
      XOR  SI,  SI
      LEA  DI,  OVLPATH

    FIRST:
            INC  SI
            CMP  WORD PTR ES:[SI], 0000H
            JNE  FIRST
            ADD  SI, 0004H

    SECOND:
            CMP  BYTE PTR ES:[SI], 0000H
            JE   THIRD
            MOV  DL, ES:[SI]
            MOV  [DI], DL
            INC  SI
            INC  DI
            JMP  SECOND

    THIRD:
            DEC  SI
            DEC  DI
            CMP  BYTE PTR ES:[SI], '\'
            JNE  THIRD
            INC  DI
            MOV  SI, BX
            PUSH DS
            POP  ES

    FOURTH:
            LODSB
            STOSB
            CMP  AL, 0000H
            JNE  FOURTH
```

```
            MOV  BYTE PTR [DI], EOFLINE
            MOV  DX, OFFSET PATH
            CALL LINE_OUTPUT
            LEA  DX, OVLPATH
            CALL LINE_OUTPUT
            POP  ES
            RET
FIND_PATH ENDP

ALLOCATE_MEMORY_FOR_OVL PROC NEAR
      PUSH DS
      PUSH DX
      PUSH CX
      XOR  CX, CX
      LEA  DX, OVLPATH
      MOV  AH, 4EH
      INT  21H
      JNC  FILE_IS_FOUND
      CMP  AX, 0003H
      JE   ERROR_3
      MOV  DX, OFFSET FILE_2
      JMP  EXIT_ERROR

    ERROR_3:
           MOV  DX, OFFSET FILE_3

    EXIT_ERROR:
           CALL LINE_OUTPUT
           POP  CX
           POP  DX
           POP  DS
           XOR  AL, AL
           MOV  AH, 4CH
           INT  21H

    FILE_IS_FOUND:
           PUSH ES
           PUSH BX
           MOV  BX, OFFSET DTA
           MOV  DX, [BX + 1CH]
           MOV  AX, [BX + 1AH]
           MOV  CL, 0004H
           SHR  AX, CL
           MOV  CL, 000CH
           SAL  DX, CL
```

```
            ADD  AX, DX
            INC  AX
            MOV  BX, AX
            MOV  AH, 48H
            INT  21H
            JNC  MEMORY_ALLOCATED
            MOV  DX, OFFSET ERR_ALLOC
            CALL LINE_OUTPUT
            XOR  AL, AL
            MOV  AH, 4CH
            INT  21H

    MEMORY_ALLOCATED:
            MOV  SEGADR, AX
            POP  BX
            POP  ES
            POP  CX
            POP  DX
            POP  DS
            RET
ALLOCATE_MEMORY_FOR_OVL ENDP

PROGRAM_CALL_OVL PROC NEAR
      PUSH DX
      PUSH BX
      PUSH AX
      MOV  BX, SEG SEGADR
      MOV  ES, BX
      LEA  BX, SEGADR
      LEA  DX, OVLPATH
      MOV  AX, 4B03H
      INT  21H
      JNC  IS_LOADED

      ERROR_CHECK:
            CMP  AX, 0001H
            LEA  DX, LOAD_1
            JE   PRINT_ERROR
            CMP  AX, 0002H
            LEA  DX, LOAD_2
            JE   PRINT_ERROR
            CMP  AX, 0003H
            LEA  DX, LOAD_3
            JE   PRINT_ERROR
            CMP  AX, 0004H
```

```
            LEA  DX, LOAD_4
            JE   PRINT_ERROR
            CMP  AX, 0005H
            LEA  DX, LOAD_5
            JE   PRINT_ERROR
            CMP  AX, 0008H
            LEA  DX, LOAD_8
            JE   PRINT_ERROR
            CMP  AX, 000AH
            LEA  DX, LOAD_10

      PRINT_ERROR:
             CALL LINE_OUTPUT
             JMP  FINISH

    IS_LOADED:
            MOV  AX, DATA
            MOV  DS, AX
            MOV  AX, SEGADR
            MOV  WORD PTR CALLADR + 0002H, AX
            CALL CALLADR
            MOV  AX, SEGADR
            MOV  ES, AX
            MOV  AX, 4900H
            INT  21H
            MOV  AX, DATA
            MOV  DS, AX

      FINISH:
             MOV  ES, KEEP_PSP
             POP  AX
             POP  BX
             POP  DX
             RET
PROGRAM_CALL_OVL ENDP

PROCESSING PROC NEAR
    CALL FIND_PATH
    CALL ALLOCATE_MEMORY_FOR_OVL
    CALL PROGRAM_CALL_OVL
    RET
PROCESSING ENDP
;-----------------------------------------------------------------
BEGINNING:
    MOV  AX, DATA
```

```
        MOV  DS, AX
        MOV  KEEP_PSP, ES
        CALL FREE_MEMORY
        CALL DTA_SET
        LEA  BX, OVL1
        CALL PROCESSING
        LEA  BX, OVL2
        CALL PROCESSING
        XOR  AL, AL
        MOV  AH, 4CH
        INT  21H
  LAST_BYTE:
        CODE ENDS
END  START
```

LR7_OVL1.ASM

```
  LR7_OVL1 SEGMENT
  ASSUME CS:LR7_OVL1, DS:LR7_OVL1, ES:NOTHING, SS:NOTHING
  ;-------------------------------------------------------------------
  BEGINNING PROC FAR
        PUSH DS
        PUSH DX
        PUSH DI
        PUSH AX
        MOV  AX, CS
        MOV  DS, AX
        LEA  BX, STRFORPRINT
        ADD  BX, 46H
        MOV  DI, BX
        MOV  AX, CS
        CALL WRD_TO_HEX
        LEA  DX, STRFORPRINT
        CALL LINE_OUTPUT
        POP  AX
        POP  DI
        POP  DX
        POP  DS
        RETF
  BEGINNING ENDP
  ;-------------------------------------------------------------------
  LINE_OUTPUT PROC NEAR
        MOV  AH, 0009H
        INT  21H
        RET
  LINE_OUTPUT 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
      ;-------------------------------------------------------------------
      STRFORPRINT DB 0DH,0AH,'THE ADDRESS OF THE SEGMENT TO WHICH THE FIRST
OVERLAY IS LOADED:                  ',0DH,0AH,'$'
      ;-------------------------------------------------------------------
      LR7_OVL1 ENDS
END
```

LR7_OVL2.ASM

```
LR7_OVL2 SEGMENT
ASSUME CS:LR7_OVL2, DS:LR7_OVL2, ES:NOTHING, SS:NOTHING
;-----------------------------------------------------------------
BEGINNING PROC FAR
      PUSH DS
      PUSH DX
      PUSH DI
      PUSH AX
      MOV  AX, CS
      MOV  DS, AX
      LEA  BX, STRFORPRINT
      ADD  BX, 47H
      MOV  DI, BX
      MOV  AX, CS
      CALL WRD_TO_HEX
      LEA  DX, STRFORPRINT
      CALL LINE_OUTPUT
      POP  AX
      POP  DI
      POP  DX
      POP  DS
      RETF
BEGINNING ENDP
;-----------------------------------------------------------------
LINE_OUTPUT PROC NEAR
      MOV  AH, 0009H
      INT  21H
      RET
LINE_OUTPUT 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
      ;-------------------------------------------------------------------
      STRFORPRINT DB 0DH,0AH,'THE ADDRESS OF THE SEGMENT TO WHICH THE SECOND
OVERLAY IS LOADED:                  ','$'
      ;-------------------------------------------------------------------
      LR7_OVL2 ENDS
END
```