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. 8383 _____ Колмыков В.Д.

Преподаватель _____ Ефремов М.А.

Санкт-Петербург
2020

Цель работы.

Исследование возможности построение загрузочного модуля оверлейной структуры.

Процедуры, используемые в работе.

Название процедуры	Описание
WRITE	Вывод строки из DX
FREE_MEMORY	Освобождение лишней памяти
EXECUTE_OVL	Загрузка, выполнение и удаление оверлейного модуля
GET_PATH	Получение пути до оверлея
GET_SIZE	Получение размера оверлея
LOAD_OVL	Загрузка и исполнение оверлея
WRITE_HEX_WORD	Печать слова в 16 с.с.
WRITE_HEX_BYTE	Печать байта в 16 с.с.

Ход работы.

Был написан и отлажен программный модуль типа EXE, который выполняет следующие функции:

1) Освобождает память для загрузки оверлеев.
2) Читает размер файла оверлея и запрашивает объем памяти, достаточный для загрузки.
3) Файл оверлейного сегмента загружается и выполняется
4) Освобождается память, отведенная под оверлей
5) Действия выполняются для второго оверлея

Оверлеи при выполнении выводят свой сегментный адрес. Результат выполнения программы предоставлен на рис. 1.

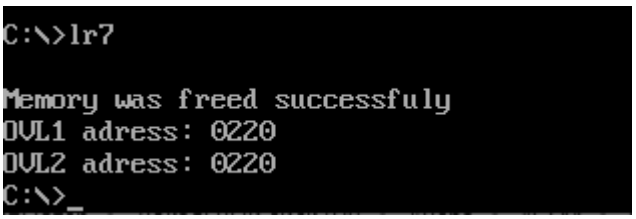

Рисунок 1 – Результат выполнения программы

Результат запуска программы из другого каталога представле на рис. 2.

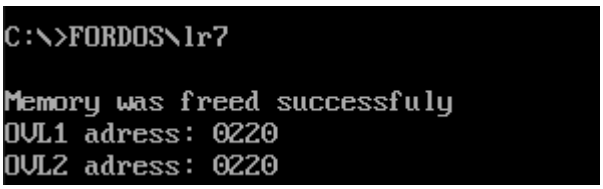

Рисунок 2 – Результат запуска программы из другой директории

Результат запуска программы, когда в каталоге нет второго оверлея представлен на рис. 3.

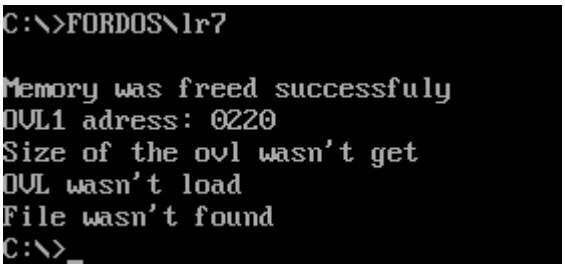

Рисунок 3 – Результат запуска программы, когда в каталоге нет второго оверлея

Ответы на контрольные вопросы.

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

Необходимо будет сохранять и восстанавливать регистры, а также учитывать, что обращение к модулю должно происходить со смещением 100h (PSP).

Выводы.

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

ПРИЛОЖЕНИЕ А
КОД ПРОГРАММЫ

```
DATA SEGMENT
	PSP_SEGMENT dw 0

	FLAG_IS_FREE_SUCCESS db 0
	STR_SUCCES_OF_FREED db 13, 10, "Memory was freed successfuly$"
	STR_ERROR_OF_FREED db 13, 10, "Memory wasn't freed$"
	STR_FREE_ERROR7 db 13, 10, "7: Memory block descriptor is
destroyed$"
	STR_FREE_ERROR8 db 13, 10, "8: Not enough memory for function$"
	STR_FREE_ERROR9 db 13, 10, "9: Invalid adress$"

	STR_GETSIZE_ERROR db 13, 10, "Size of the ovl wasn't get$"
	STR_GS_ERROR2 db 13, 10, "File wasn't found$"
	STR_GS_ERROR3 db 13, 10, "Path wasn't found$"

	STR_ERROR_LOAD db 13, 10, "OVL wasn't load$"
	STR_LOAD_ERROR1 db 13, 10, "Function doesn't exist$"
	STR_LOAD_ERROR2 db 13, 10, "File wasn't found$"
	STR_LOAD_ERROR3 db 13, 10, "Path wasn't found$"
	STR_LOAD_ERROR4 db 13, 10, "Too much file is open$"
	STR_LOAD_ERROR5 db 13, 10, "No access$"
	STR_LOAD_ERROR8 db 13, 10, "No memory$"
	STR_LOAD_ERROR10 db 13, 10, "Incorrect environment$"

	OVL1_NAME db "ovl1.ovl", 0
	OVL2_NAME db "ovl2.ovl", 0
	OFFSET_OVL_NAME dw 0

	STR_PATH db 100h dup(0)
	NAME_POS dw 0

	DTA_BUFF db 43 dup(0)
	OVL_PARAM_SEG dw 0
	OVL_ADRESS dd 0

	STUPID_MASM2 db 0
DATA ENDS
;_____________________________________________________
STACKK SEGMENT STACK
	dw 100h dup (0)
STACKK ENDS
;_____________________________________________________
CODE SEGMENT
	ASSUME CS:CODE, DS:DATA, SS:STACKK
;_____________________________________________________
	WRITE PROC
		push AX
		mov AH, 9h
		int 21h
		pop AX
		ret
	WRITE ENDP
;_____________________________________________________
	FREE_MEMORY PROC
		push AX
		push BX
		push CX
		push DX
```

```
            mov BX, offset STUPID_MASM    ;Попытка освобождения
            mov AX, offset STUPID_MASM2
            add BX, AX
            add BX, 40Fh
            mov CL, 4
            shr BX, CL
            mov AX, 4A00h
            int 21h

            jnc FREE_MEMORY_SUCCES  ;Проверка освобождения
            mov DX, offset STR_ERROR_OF_FREED
            call WRITE
            mov FLAG_IS_FREE_SUCCESS, 0
            cmp AX, 7
                  je FREE_MEMORY_ERROR_7
            cmp AX, 8
                  je FREE_MEMORY_ERROR_8
            cmp AX, 9
                  je FREE_MEMORY_ERROR_9
                  jmp FREE_MEMORY_RETURN
      FREE_MEMORY_ERROR_7:
            mov DX, offset STR_FREE_ERROR7
            call WRITE
            jmp FREE_MEMORY_RETURN
      FREE_MEMORY_ERROR_8:
            mov DX, offset STR_FREE_ERROR8
            call WRITE
            jmp FREE_MEMORY_RETURN
      FREE_MEMORY_ERROR_9:
            mov DX, offset STR_FREE_ERROR9
            call WRITE
            jmp FREE_MEMORY_RETURN
      FREE_MEMORY_SUCCES:
            mov DX, offset STR_SUCCES_OF_FREED
            call WRITE
            mov FLAG_IS_FREE_SUCCESS, 1

      FREE_MEMORY_RETURN:
            pop DX
            pop CX
            pop BX
            pop AX
            ret
      FREE_MEMORY ENDP
;_____________________________________________________
      GET_PATH PROC
            push AX
            push DI
            push SI
            push ES

            mov OFFSET_OVL_NAME, AX
            mov AX, PSP_SEGMENT
            mov ES, AX
            mov ES, ES:[2Ch]
            mov SI, 0
      GP_FIND0:
            mov AX, ES:[SI]
            inc SI
            cmp AX, 0
            jne GP_FIND0
            add SI, 3
```

```
            mov DI, 0
      GP_WRITE:
            mov AL, ES:[SI]
            cmp AL, 0
            je GP_WRITE_NAME
            cmp AL, '\'
            jne GP_ADD_SYMB
            mov NAME_POS, DI
      GP_ADD_SYMB:
            mov BYTE PTR [STR_PATH + DI], AL
            inc DI
            inc SI
            jmp GP_WRITE
      GP_WRITE_NAME:
            cld
            mov DI, NAME_POS
            inc DI
            add DI, offset STR_PATH
            mov SI, OFFSET_OVL_NAME
            mov AX, DS
            mov ES, AX
      GP_REWRITE_NAME_SYMB:
            lodsb
            stosb
            cmp AL, 0
            jne GP_REWRITE_NAME_SYMB

            pop ES
            pop SI
            pop DI
            pop AX
            ret
      GET_PATH ENDP
;_____________________________________________________
      GET_SIZE PROC
            push AX
            push BX
            push CX
            push DX
            push SI

            mov AX, 1A00h;установка DTA
            mov DX, offset DTA_BUFF
            int 21h

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

            jnc GS_SUCCESS
            mov DX, offset STR_GETSIZE_ERROR
            call WRITE
            cmp AX, 2
            je GS_ERROR2
            cmp AX, 3
            je GS_ERROR3
            jmp GS_END
      GS_ERROR2:
            mov DX, offset STR_GS_ERROR2
            call WRITE
            jmp GS_END
      GS_ERROR3:
```

```
            mov DX, offset STR_GS_ERROR3
            call WRITE
            jmp GS_END
      GS_SUCCESS:
            mov SI, offset DTA_BUFF
            add SI, 1Ah
            mov BX, [SI]
            mov AX, [SI + 2]
            shr BX, 4
            shl AX, 12;4 ->, 16 <-
            add BX, AX
            add BX, 2
            mov AX, 4800h
            int 21h

            jnc GS_SET_SEG
            mov DX, offset STR_ERROR_OF_FREED
            call WRITE
            jmp GS_END
      GS_SET_SEG:
            mov OVL_PARAM_SEG, AX

      GS_END:
            pop SI
            pop DX
            pop CX
            pop BX
            pop AX
            ret
      GET_SIZE ENDP
;_____________________________________________________
      LOAD_OVL PROC
            push AX
            push BX
            push DX
            push ES

            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 WRITE
            cmp AX, 1
            je LO_ERROR1
            cmp AX, 2
            je LO_ERROR2
            cmp AX, 3
            je LO_ERROR3
            cmp AX, 4
            je LO_ERROR4
            cmp AX, 5
            je LO_ERROR5
            cmp AX, 8
            je LO_ERROR8
            cmp AX, 10
            je LO_ERROR10
      LO_ERROR1:
            mov DX, offset STR_LOAD_ERROR1
```

```
            call WRITE
            jmp LO_END
      LO_ERROR2:
            mov DX, offset STR_LOAD_ERROR2
            call WRITE
            jmp LO_END
      LO_ERROR3:
            mov DX, offset STR_LOAD_ERROR3
            call WRITE
            jmp LO_END
      LO_ERROR4:
            mov DX, offset STR_LOAD_ERROR4
            call WRITE
            jmp LO_END
      LO_ERROR5:
            mov DX, offset STR_LOAD_ERROR5
            call WRITE
            jmp LO_END
      LO_ERROR8:
            mov DX, offset STR_LOAD_ERROR8
            call WRITE
            jmp LO_END
      LO_ERROR10:
            mov DX, offset STR_LOAD_ERROR10
            call WRITE
            jmp LO_END
      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
      LO_END:
            pop ES
            pop DX
            pop BX
            pop AX
            ret
      LOAD_OVL ENDP
;_____________________________________________________
      EXECUTE_OVL PROC
            call GET_PATH
            call GET_SIZE
            call LOAD_OVL
            ret
      EXECUTE_OVL ENDP
;_____________________________________________________
      BEGIN:
            mov AX, DATA
            mov DS, AX
            mov PSP_SEGMENT, ES
            call FREE_MEMORY
            cmp FLAG_IS_FREE_SUCCESS, 0
            je TO_DOS
            mov AX, offset OVL1_NAME
            call EXECUTE_OVL
            mov AX, offset OVL2_NAME
            call EXECUTE_OVL

      TO_DOS:
            mov AX, 4C00h
```

```
            int 21h
      STUPID_MASM:
CODE ENDS
END BEGIN
```

```
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_RESULT
            call WRITE
            call WRITE_HEX_WORD

            pop DS
            pop DX
            pop AX
            retf
      MAIN ENDP
;_____________________________________________________
      STR_RESULT db 13, 10, "OVL1 adress: $"
;_____________________________________________________
      WRITE PROC
            push AX
            mov AH, 9h
            int 21h
            pop AX
            ret
      WRITE 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
```