МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра математического обеспечения и применения ЭВМ

ОТЧЕТ

по практической работе № 7

по дисциплине «Операционные системы»

Тема: Построение модуля оверлейной структуры

Студентка гр.8382 _____ Наконечная А. Ю.

Преподаватель _____ Ефремов М. А.

Санкт-Петербург

2020

Цель работы.

Исследование возможности построения загрузочного модуля оверлейной структуры.

Постановка задачи.

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, отведённая 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 отчёт. Оформите отчёт в соответствии с требованиями.

Выполнение работы.

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 результате выполнения были получены следующие результаты (Рисунок 1 - 3):

Рисунок 1 – Запуск отлаженной программы из каталога с разработанными модулями.

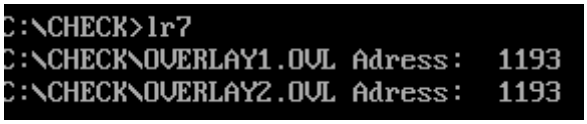

Рисунок 2 – Запуск отлаженной программы из каталога не с разработанными модулями.

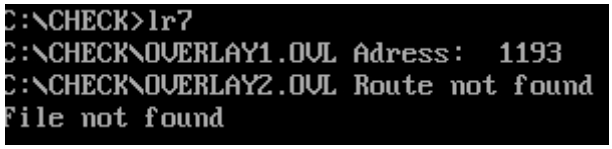

Рисунок 3 – Запуск отлаженной программы из каталога с разработанной программой, при отсутствие первого оверлея.

Ответы на контрольные вопросы.

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

При использовании оверлейного сегмента com модуля, нужно вызывать его по смещению 100h, т. к. в com файлах код располагается с этого адреса.

Выводы.

В ходе лабораторной работы был построен загрузочный модуль оверлейной структуры, а также оверлеи. Изучены дополнительные функции работы с памятью и способы загрузки и выполнения оверлейных сегментов.

ПРИЛОЖЕНИЕ А

Название исходного файла: lr7.asm

```
AStack    SEGMENT  STACK
    DW 100 dup(?)
AStack    ENDS

DATA  SEGMENT
                        STR_OVL1_NAME db 'OVERLAY1.OVL $'
                        STR_OVL2_NAME db 'OVERLAY2.OVL $'
                        MEM_ERROR7_STR db 'Memory control block destroyed
',13,10,'$'
                        MEM_ERROR8_STR db 'Not enough memory to execute
function ',13,10,'$'
                        MEM_ERROR9_STR db 'Invalid memory block address
',13,10,'$'
                        PATH db 128 dup (0)
                        PARAMETERS dw 0,0
                        ADDRESS dd 0
                        DTA db 43 dup(0)
                        STR_ERROR_FREE_MEMORY db 'Error free memory
',13,10,'$'
                        STR_ERROR_1 db 'Function does not exist ',
13,10,'$'
                        STR_ERROR_2 db 'File not found ',13,10,'$'
                        STR_ERROR_3 db 'Route not found ',13,10,'$'
                        STR_ERROR_4 db 'Too many files were opened
',13,10,'$'
                        STR_ERROR_5 db 'No access ',13,10,'$'
                        STR_MEMORY_8 db 'Low memory size for function
',13,10,'$'
DATA ENDS

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

```
GET_SIZE PROC NEAR
                        push AX
                        push BX
                        push CX
                        push DX
                        push BP
                        mov AH, 1Ah
                        lea DX, DTA
                        int 21h
                        mov AH, 4Eh
                        lea DX, PATH
                        mov CX, 0
                        int 21h
                        jnc MEM
                        lea DX, STR_ERROR_2
                        cmp AX, 2
                            je WRITE_ERR
                        lea DX, STR_ERROR_3
                        cmp AX, 3
                        je WRITE_ERR
WRITE_ERR:
                        push AX
                        mov AH,09h
                        int 21h
                        pop AX
                        jmp ENDING

MEM:
                        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
    lea DX, STR_ERROR_FREE_MEMORY
                        push AX
                        mov AH,09h
                        int 21h
                        pop AX
    jmp ENDING
SAVE:
    mov PARAMETERS, AX
    mov PARAMETERS + 2, AX
ENDING:
                        pop BP
                        pop DX
                        pop CX
                        pop BX
                        pop AX
                        ret
GET_SIZE ENDP

LOAD_OVL PROC NEAR
                        push AX
                        push DX
                        push ES
                        lea DX, PATH
                        push DS
```

```
                        pop ES
                        lea BX, PARAMETERS
                        mov AX, 4B03h
    int 21h
                        jnc LOAD
                        lea DX, STR_ERROR_1
                        cmp AX, 1
                            je L_ERR
                        lea DX, STR_ERROR_2
                        cmp AX, 2
                            je L_ERR
                        lea DX, STR_ERROR_3
                        cmp AX, 3
                            je L_ERR
                        lea DX, STR_ERROR_4
                        cmp AX, 4
                            je L_ERR
                        lea DX, STR_ERROR_5
                        cmp AX, 5
                            je L_ERR
                        lea DX, STR_MEMORY_8
                        cmp AX, 8
                            je L_ERR
L_ERR:
                        push AX
                        mov AH,09h
                        int 21h
                        pop AX
                        jmp GO_END
LOAD:
                        mov AX, PARAMETERS
    mov word ptr ADDRESS + 2, AX
```

```
    call ADDRESS
    mov ES, AX
                        mov AH, 49h
                        int 21h
GO_END:
                        pop ES
                        pop DX
                        pop AX
                        ret
LOAD_OVL ENDP

FILE_NAME PROC NEAR
                        push DX
                        push DI
                        push SI
                        push ES
                        xor DI, DI
                        mov ES, ES:[2ch]
SKIP:
                        mov DL, ES:[DI]
                        cmp DL, 0h
                             je LAST
                        inc DI
                        jmp SKIP
LAST:
                        inc DI
                        mov DL, ES:[DI]
                        cmp DL, 0h
                        jne SKIP
                        add DI, 3h
                        mov SI, 0
WRITE:
```

```
                        mov DL, ES:[DI]
                        cmp DL, 0h
                            je DELETE
                        mov PATH[SI], DL
                        inc DI
                        inc SI
                        jmp WRITE
DELETe:
                        dec SI
                        cmp PATH[si], '\'
                            je READY
                        jmp DELETE
READY:
                        mov DI,-1
ADD_NAME:
                        inc SI
                        inc DI
                        mov DL, BX[DI]
                        cmp DL, '$'
                            je PATH_END
                        mov PATH[SI], DL
                        jmp ADD_NAME
PATH_END:
                        mov PATH[SI],'$'
                        pop ES
                        pop SI
                        pop DI
                        pop DX
                        ret
FILE_NAME ENDP

FREE_MEMORY PROC
```

```
                        push AX
                        push BX
                        mov BX, 4096
                        mov AH, 4Ah
                        int 21h
                        jnc MEM_END
                        mov DX, offset MEM_ERROR7_STR
                        cmp AX,7
                            je MEM_WRITE
                        mov DX, offset MEM_ERROR8_STR
                        cmp AX,8
                            je MEM_WRITE
                        mov DX, offset MEM_ERROR9_STR
                        cmp AX,9
                            je MEM_WRITE
MEM_END:
                        pop BX
                        pop AX
                        ret
MEM_WRITE:
                        push AX
                        mov AH, 09h
                        int 21h
                        pop AX
                        ret
FREE_MEMORY ENDP

Main PROC FAR
                        mov AX, DATA
                        mov DS, AX
                        call FREE_MEMORY
                        push DX
```

```
                        push BX
                        lea BX, STR_OVL1_NAME
                        call FILE_NAME
                        pop BX
                        lea DX, PATH
                        push AX
                        mov AH, 09h
                        int 21h
                        pop AX
                        call GET_SIZE
                        call LOAD_OVL
                        pop DX
                        push DX
                        push BX
                        lea BX, STR_OVL2_NAME
                        call FILE_NAME
                        pop BX
                        lea DX, PATH
                        push AX
                        mov AH, 09h
                        int 21h
                        pop AX
                        call GET_SIZE
                        call LOAD_OVL
                        pop DX

    ;Выход в DOS
                        xor AL,AL
                        mov AH,4Ch
                        int 21h
Main ENDP
CODE ENDS
```

```
END Main
```

ПРИЛОЖЕНИЕ В

Название исходного файла: OVERLAY1.asm

```
OVERLAY1 SEGMENT
                        ASSUME    CS:OVERLAY1,    DS:NOTHING,    SS:NOTHING,
ES:NOTHING


MAIN PROC FAR
                        push AX
                        push BX
                        push DX
                        push DS
                        push DI
                        mov AX, CS
                        mov DS, AX
                        mov BX, offset ADDRESS_STR
                        add BX, 12
                        mov DI, BX
                        mov AX, CS
                        call WRD_TO_HEX
                        mov DX, offset ADDRESS_STR
                        mov AH,09h
                        int 21h
                        pop DI
                        pop DS
                        pop DX
                        pop BX
                        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
;байт в AL переводится в два символа шест. числа в AX
   push CX
   mov AH,AL
   call TETR_TO_HEX
   xchg AL,AH
   mov CL,4
   shr AL,CL
   call TETR_TO_HEX ;в AL старшая цифра
   pop CX ;в AH младшая
   ret
BYTE_TO_HEX ENDP

WRD_TO_HEX PROC near
;перевод в 16 с/с 16-ти разрядного числа
; в AX - число, DI - адрес последнего символа
   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

ADDRESS_STR db 'Adress:        ',0DH,0AH, '$'

OVERLAY1 ENDS
END MAIN
```

ПРИЛОЖЕНИЕ С

Название исходного файла: OVERLAY2.asm

```
OVERLAY2 SEGMENT
                        ASSUME    CS:OVERLAY2,    DS:NOTHING,    SS:NOTHING,
ES:NOTHING


MAIN PROC FAR
                        push AX
                        push BX
                        push DX
                        push DS
                        push DI
                        mov AX, CS
                        mov DS, AX
                        mov BX, offset ADDRESS_STR
                        add BX, 12
                        mov DI, BX
                        mov AX, CS
                        call WRD_TO_HEX
                        mov DX, offset ADDRESS_STR
                        mov AH,09h
                        int 21h
                        pop DI
                        pop DS
                        pop DX
                        pop BX
                        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
;байт в AL переводится в два символа шест. числа в AX
   push CX
   mov AH,AL
   call TETR_TO_HEX
   xchg AL,AH
   mov CL,4
   shr AL,CL
   call TETR_TO_HEX ;в AL старшая цифра
   pop CX ;в AH младшая
   ret
BYTE_TO_HEX ENDP

WRD_TO_HEX PROC near
;перевод в 16 с/с 16-ти разрядного числа
; в AX - число, DI - адрес последнего символа
   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

ADDRESS_STR db 'Adress:        ',0DH,0AH, '$'

OVERLAY2 ENDS
END MAIN
```