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

ОТЧЕТ

по лабораторной работе №6
по дисциплине «Операционные системы»
Тема: Построение модуля динамической структуры

Студент гр.0382 _____ Злобин А.С.

Преподаватель _____ Ефремов М.А.

Санкт-Петербург
2022

Цель работы.

Исследование возможности построения загрузочного модуля динамической 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, как это делалось в работе 2. Понятно, что такое приложение должно запускаться в соответствии со стандартами ОС.

В работе исследуется интерфейс между вызывающим и вызываемым модулями по управлению и по данным. Для запуска вызываемого модуля используется функция 4B00h прерывания 21h. Все загрузочные модули находятся в одном каталоге. Необходимо обеспечить возможность запуска модуля динамической структуры из любого каталога.

Задание.

1. Для выполнения лабораторной работы необходимо написать и отладить программный модуль типа .EXE, который выполняет функции:

- Подготавливает параметры для запуска загрузочного модуля из того же каталога, в котором находится он сам. Вызываемому модулю передается новая среда, созданная вызывающим модулем и новая командная строка;

- Вызываемый модуль запускается с использованием загрузчика;

- После запуска проверяется выполнение загрузчика, а затем результат выполнения вызываемой программы. Необходимо проверять причину завершения и, в зависимости от значения, выводить соответствующее сообщение. Если причина завершения 0, то выводится код завершения.

В качестве вызываемой программы необходимо взять программу ЛР 2, которая распечатывает среду и командную строку. Эту программу следует немного модифицировать, вставив перед выходом из нее обращение к функции

ввода символа с клавиатуры. Введенное значение записывается в регистр AL и затем происходит обращение к функции выхода 4Ch прерывания int 21h.

2. Запустите отлаженную программу, когда текущим каталогом является каталог с разработанными модулями. Программа вызывает другую программу, которая останавливается, ожидая символ с клавиатуры.

Введите произвольный символ из числа A-Z. Посмотрите причину завершения и код. Занесите полученные данные в отчет.

3. Запустите отлаженную программу, когда текущим каталогом является каталог с разработанными модулями. Программа вызывает другую программу, которая останавливается, ожидая символ с клавиатуры.

Введите комбинацию символов Ctrl-C. Посмотрите причину завершения и код. Занесите полученные данные в отчет.

4. Запустите отлаженную программу, когда текущим каталогом является какой-либо другой каталог, отличный от того, в котором содержатся разработанные программные модули.

Повторите ввод комбинаций клавиш. Занесите полученные данные в отчет.

5. Запустите отлаженную программу, когда модули находятся в разных каталогах. Занесите полученные данные в отчет.

Выполнение работы.

1. Был написан программный модуль типа .EXE, состоящий из трех процедур:

- FREE_MEM – подготавливает место в памяти, необходимое для программы;

- PATH – подготавливает путь и имя вызываемого модуля.

- LOAD – загружает вызываемый модуль.

2. Запуск отлаженной программы, когда текущим каталогом является каталог с разработанными модулями. Был введен символ d. Результат представлен на рисунке 1:

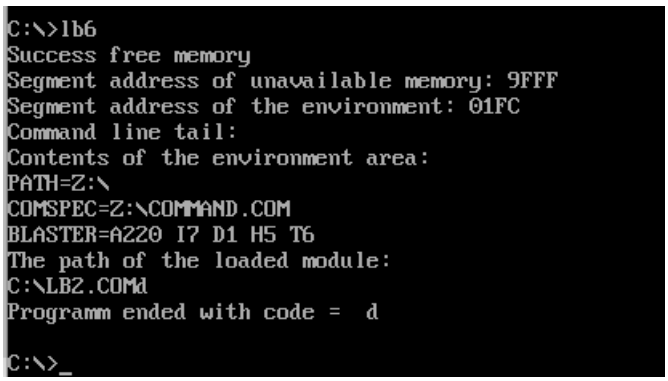

Рисунок 1 – Результат второго шага

3. Запуск отлаженной программы, когда текущим каталогом является каталог с разработанными модулями. Была нажата комбинация Ctrl-C. Результат представлен на рисунке 2. Как можно заметить, был напечатан символ сердечка, т.к. в эмуляторе DosBox не поддерживается прерывание Ctrl-C.

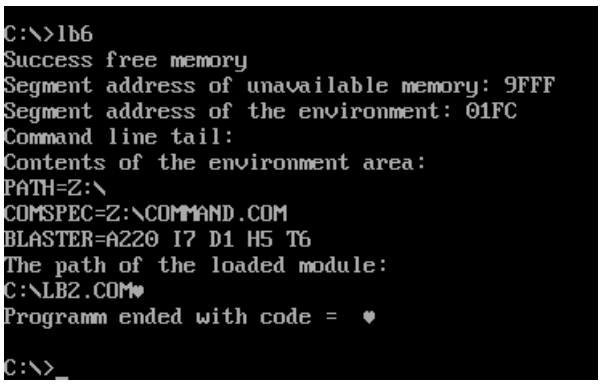

Рисунок 2 – Результаты третьего шага

4. Запуск отлаженной программы, когда текущим каталогом является какой-либо другой каталог, отличный от того, в котором содержатся разработанные программные модули. Был повторен ввод символа d (см. рисунок 3) и комбинации клавиш Ctrl-C (см. рисунок 4).

Рисунок 3 – Результаты четвертого шага (символ из A-Z)

Рисунок 4 – Результаты четвертого шага (символ из A-Z)

5. Запуск отлаженной программы, когда модули находятся в разных каталогах. Вывод представлен на рисунке 5.

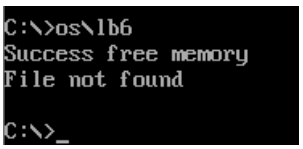

Рисунок 5 – Результаты пятого шага

Исходный код программы см. в приложении А.

Ответы на вопросы.

1. Как реализовано прерывание Ctrl-C?

После нажатия комбинации клавиш Ctrl-C срабатывает прерывание 23h, управление передается по адресу 0000:008C, адрес копируется в PSP с помощью функций 26h и 4Ch. Исходное значение адреса восстанавливается при выходе из программы.

2. В какой точке заканчивается вызываемая программа, если код причины завершения 0?

В точке вызова функции 4Ch прерывания int 21h.

3. В какой точке заканчивается программа по прерыванию Ctrl-C?

В точке ожидания ввода символа (на функции 01h прерывания 21h).

Выводы.

В ходе работы были исследованы возможности построения загрузочного модуля динамической структуры, а также исследован интерфейс между вызывающим и вызываемым модулями по управлению и по данным.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: lb6.asm

```
AStack SEGMENT STACK
 DW 32 DUP(?)
AStack ENDS

DATA SEGMENT
     PARAM dw 0
               dd 0
               dd 0
               dd 0

     FILE_NAME db 'LB2.COM', 0
     CMD_L db 1h, 0dh
     FILE_PATH db 128 DUP (?)

     FREE_MEM_1 db 'The control memory block is destroyed',
0DH, 0AH,'$'
    FREE_MEM_2 db 'Not enough memory to execute the function',
0DH, 0AH,'$'
     FREE_MEM_3 db 'Invalid memory block address', 0DH,
0AH,'$'
     FREE_MEM_4 db 'Success free memory', 0DH, 0AH,'$'
     FREE_MEM_FLAG db 0

     LOAD_ERROR_1 db 'Invalid function number', 0DH, 0AH,'$'
     LOAD_ERROR_2 db 'File not found', 0DH, 0AH,'$'
     LOAD_ERROR_3 db 'Disk error', 0DH, 0AH,'$'
     LOAD_ERROR_4 db 'Not enough memory', 0DH, 0AH,'$'
     LOAD_ERROR_5 db 'Incorrect environment string', 0DH,
0AH,'$'
     LOAD_ERROR_6 db 'Incorrect format', 0DH, 0AH,'$'

     GOOD_END db 0DH, 0AH,'Programm ended with code =     ',
0DH, 0AH,'$'

     CTRLC_END db 'Programm ended ctrl-break', 0DH, 0AH,'$'
     DEVICE_END db 'Programm ended device error', 0DH, 0AH,'$'
     INT31_END db 'Programm ended int 31h', 0DH, 0AH,'$'

     KEEP_SS dw 0
     KEEP_SP dw 0
     KEEP_PSP dw 0

     END_DATA db 0
DATA ENDS

TESTPC SEGMENT
```

```
     ASSUME CS:TESTPC, DS:DATA, SS:AStack

; ПРОЦЕДУРЫ
;-----------------------
PRINT PROC near
     push AX
     mov AH, 09h
     int 21h
     pop AX
     ret
PRINT ENDP
;-----------------------
FREE_MEM PROC near
     push AX
     push BX
     push CX
     push DX

     lea BX, end_programm
     lea AX, END_DATA
     add BX, AX
     mov CL, 4
     shr BX, CL
     add BX, 2Bh
     mov AH, 4Ah
     int 21h

     jnc free_mem_suc

     mov FREE_MEM_FLAG, 0
     cmp AX, 7
     jne low_mem
     lea DX, FREE_MEM_1
     jmp free_mem_print
low_mem:
     cmp AX, 8
     jne inv_addr
     lea DX, FREE_MEM_2
     jmp free_mem_print
inv_addr:
     cmp AX, 9
     lea DX, FREE_MEM_3
     jmp free_mem_print

free_mem_suc:
     mov FREE_MEM_FLAG, 1
     lea DX, FREE_MEM_4

free_mem_print:
     call PRINT
```

```
end_free_mem:
     pop DX
     pop CX
     pop BX
     pop AX
     ret
FREE_MEM ENDP
;-----------------------------------
LOAD PROC near
     push AX
     push BX
     push CX
     push DX
     push DS
     push ES
     mov KEEP_SP, SP
     mov AX, SS
     mov KEEP_SS, AX

     mov AX, DATA
     mov ES, AX
     mov bx, offset PARAM
     mov dx, offset CMD_L
     mov [bx+2], dx
     mov [bx+4], ds
     mov dx, offset FILE_PATH

     mov ax, 4B00h
     int 21h

     mov ss, KEEP_SS
     mov sp, KEEP_SP
     pop es
     pop ds

     jnc success_load

     cmp AX, 1
     jne not_found
     lea DX, LOAD_ERROR_1
     jmp load_print
not_found:
     cmp AX, 2
     jne disk_error
     lea DX, LOAD_ERROR_2
     jmp load_print
disk_error:
     cmp AX, 5
     jne not_enough_mem
     lea DX, LOAD_ERROR_3
     jmp load_print
```

```
not_enough_mem:
     cmp AX, 8
     jne env_error
     lea DX, LOAD_ERROR_4
     jmp load_print

env_error:
     cmp AX, 10
     jne not_correct_format
     lea DX, LOAD_ERROR_5
     jmp load_print

not_correct_format:
     cmp AX, 11
     mov DX, offset LOAD_ERROR_6
     jmp load_print

success_load:
     mov AX, 4D00h
     int 21h

     cmp AH, 0
     jne ctrlc
     push DI
     lea DI, GOOD_END
     mov [DI+30], AL
     pop SI
     lea DX, GOOD_END
     jmp load_print
ctrlc:
     cmp AH, 1
     jne device
     lea DX, CTRLC_END
     jmp load_print
device:
     cmp AH, 2
     jne int_31h
     lea DX, DEVICE_END
     jmp load_print
int_31h:
     cmp AH, 3
     lea DX, INT31_END

load_print:
     call PRINT

end_load:
     pop DX
     pop CX
     pop BX
     pop AX
```

```
     ret
LOAD ENDP
;-----------------------------------
PATH PROC near
     push AX
     push BX
     push CX
     push DX
     push DI
     push SI
     push ES

     mov AX, KEEP_PSP
     mov ES, AX
     mov ES, ES:[2Ch]
     mov BX, 0

find_zero:
     inc BX
     cmp byte ptr ES:[BX-1], 0
     jne find_zero

     cmp byte ptr ES:[BX+1], 0
     jne find_zero

     add BX, 2
     mov DI, 0

path_loop:
     mov DL, ES:[BX]
     mov byte ptr [FILE_PATH+DI], DL
     inc DI
     inc BX
     cmp DL, 0
     je path_end_loop
     cmp DL, '\'
     jne path_loop
     mov CX, DI
     jmp path_loop
path_end_loop:
     mov DI, CX
     mov SI, 0

_file_name:
     mov DL, byte ptr [FILE_NAME+SI]
     mov byte ptr [FILE_PATH+DI], DL
     inc DI
     inc SI
     cmp DL, 0
     jne _file_name
```

```
        pop ES
        pop SI
        pop DI
        pop DX
        pop CX
        pop BX
        pop AX
        ret
    PATH ENDP
    ;-----------------------------------
    ; КОД
    MAIN PROC far
        mov ax, data
        mov ds, ax
        mov KEEP_PSP, ES

        call FREE_MEM
        cmp FREE_MEM_FLAG, 0
        je main_end
        call PATH
        call LOAD
    ; Выход в DOS
    main_end:
        xor AL, AL
        mov AH, 4Ch
        int 21h
    MAIN ENDP
    end_programm:
    TESTPC ENDS
    END MAIN
```

Название файла: lb2.asm

```
    TESTPC SEGMENT
        ASSUME CS:TESTPC, DS:TESTPC, ES:NOTHING, SS:NOTHING
        ORG  100H
    START:    jmp BEGIN

    ; ДАННЫЕ
    SAUM db 'Segment address of unavailable memory:      ', 0DH,
0AH, '$'
    SAE db 'Segment address of the environment:     ', 0DH, 0AH,
'$'
    CLT db 'Command line tail:  ', '$'
    ECLT db 'Command line tail is empty', 0DH, 0AH, '$'
    CEA db 'Contents of the environment area:  ',0DH, 0AH,'$'
    PLM db 'The path of the loaded module:  ', 0DH, 0AH, '$'

    ; ПРОЦЕДУРЫ
    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 16-го 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, в 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
;--------------------------------------
PRINT PROC near
     push AX
     mov AH, 09h
     int 21h
     pop AX
     ret
PRINT ENDP
;----------------------
PRINT_SYM PROC near
     push AX
     mov AH, 02h
     int 21h
```

```
     pop AX
     ret
PRINT_SYM ENDP
;----------------------
PSAUM PROC near
     mov AX, DS:[2h]
     mov DI, offset SAUM + 42
     call WRD_TO_HEX
     mov DX, offset SAUM
     call PRINT
     ret
PSAUM ENDP
;---------------------------------
PSAE PROC near
     mov AX, DS:[2Ch]
     mov DI, offset SAE + 39
     call WRD_TO_HEX
     mov DX, offset SAE
     call PRINT
     ret
PSAE ENDP
;---------------------------------
PCEA PROC near
     mov DX, offset CEA
     call PRINT
     mov ES, DS:[2Ch]
     xor DI, DI
line:
     mov DL, ES:[DI]
     cmp DL, 0h
     je end_line
     call PRINT_SYM
     inc DI
     jmp line
end_line:
     mov DL, 0Dh
     call PRINT_SYM
     mov DL, 0Ah
     call PRINT_SYM
     inc DI
     mov DL, ES:[DI]
     cmp DL, 0h
     jne line

     mov DX, offset PLM
     call PRINT
     add DI, 3
path_line:
     mov DL, ES:[DI]
     cmp DL, 0h
     je end_path
```

```
     call PRINT_SYM
     inc DI
     jmp path_line
end_path:
     ret
PCEA ENDP
;-----------------------------------
PCLT PROC near
     xor CX, CX
     mov CL, DS:[80h]
     cmp CL, 0h
     je empty
     mov DX, offset CLT
     call PRINT
     mov SI, 81h
loop_clt:
     mov DL, DS:[SI]
     call PRINT_SYM
     inc SI
     loop loop_clt
     mov DL, 0Dh
     call PRINT_SYM
     mov DL, 0Ah
     call PRINT_SYM
     ret
empty:
     mov DX, offset ECLT
     call PRINT
     ret
PCLT ENDP
;-----------------------------------
; КОД
BEGIN:
     call PSAUM
     call PSAE
     call PCLT
     call PCEA
     xor AL, AL
     mov AH, 01h
     int 21h
     mov AH, 4Ch
     int 21h
TESTPC ENDS
     END START
```