МИНОБРНАУКИ РОССИИ

Санкт-Петербургский государственный

электротехнический университет

«ЛЭТИ» им. В.И. Ульянова (Ленина)

Кафедра математического обеспечения и применения ЭВМ

ОТЧЕТ

по лабораторной работе №6

по дисциплине «Операционные системы»

Тема: Построение модуля динамической структуры.

Студент гр. 8382 _____ Терехов А.Е.

Преподаватель _____ Ефремов М.А.

Санкт-Петербург

2020

Цель работы.

Исследование возможности построения загрузочного модуля динамической структуры. В отличие 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 4800h прерывания int 21h. Все загрузочные модули находятся в одном каталоге. Необходимо обеспечить возможность запуска модуля динамической структуры из любого каталога.

Ход работы.

1. Был написан и отлажен EXE-модуль. Программа подготавливает параметры для запуска загрузочного модуля из того же каталога, в котором он находится. Вызываемому модулю передается новая среда, созданная вызывающим модулем и новая командная строка. Вызываемый модуль успешно запускается. После запуска проверяется выполнение загрузчика, а затем результат выполнения вызываемой программы. Затем выявляется причина завершения и, в зависимости от значения, выводится сообщение. На рисунке 1 представлен результат выполнения программы при условии, что оба модуля находятся в одном каталоге.

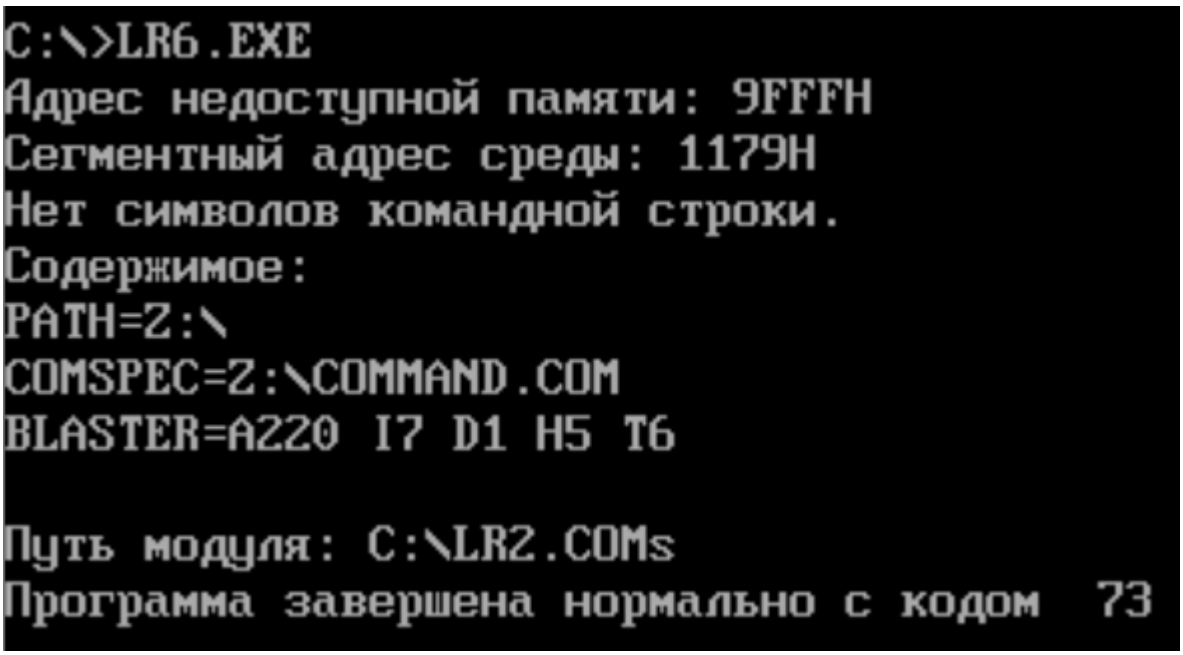

Рисунок 1. Результат выполнения программы.

Результат запуска программы при вводе комбинации Ctrl+C представлен на рисунке 2.

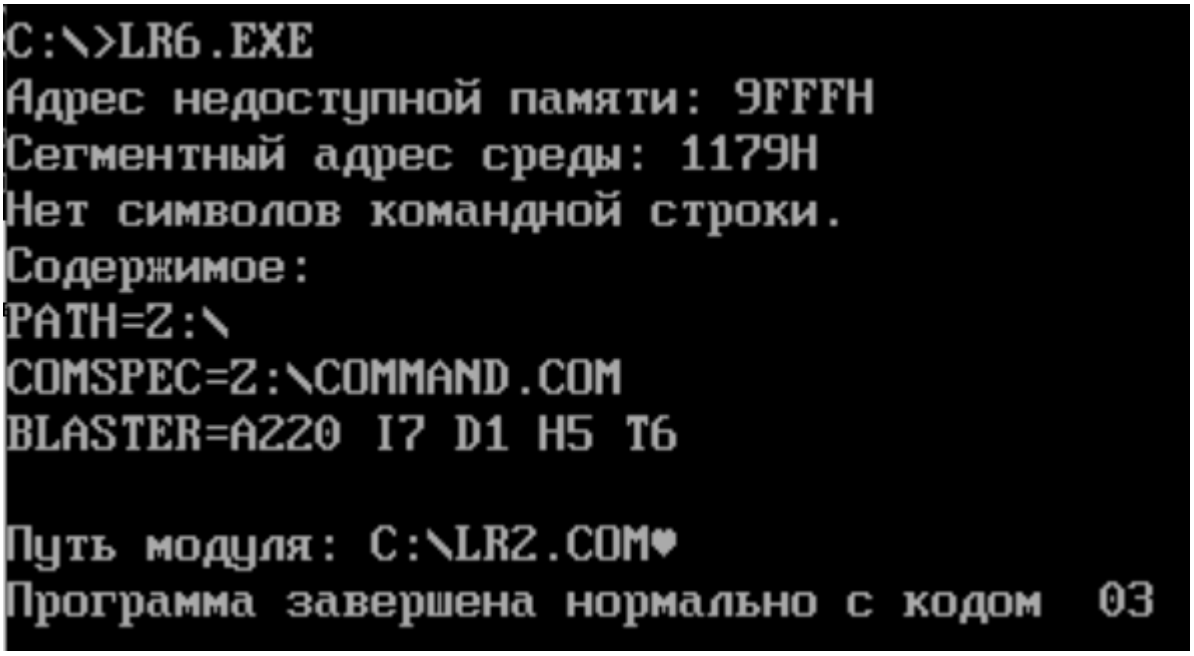

Рисунок 2. Результат выполнения программы при вводе Ctrl+C.

Результаты запуска файлов, находящихся в каталоге "6" представлены на рисунках 3 и 4.

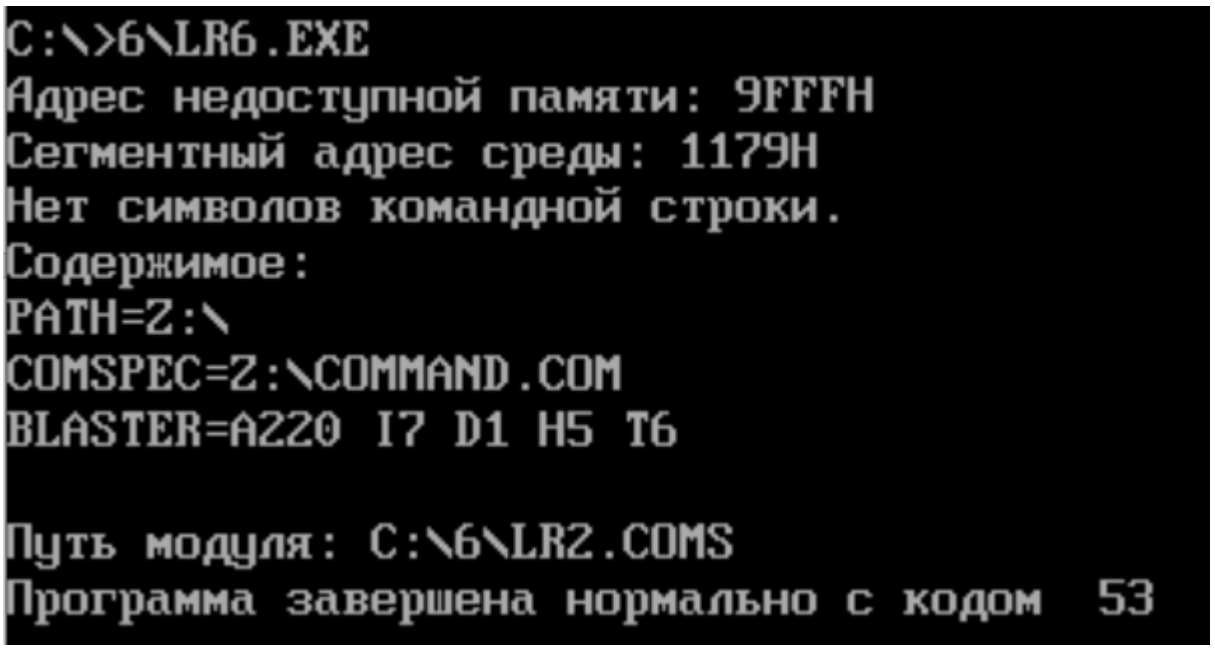

Рисунок 3. Результат выполнения программы, находящейся в каталоге "6".

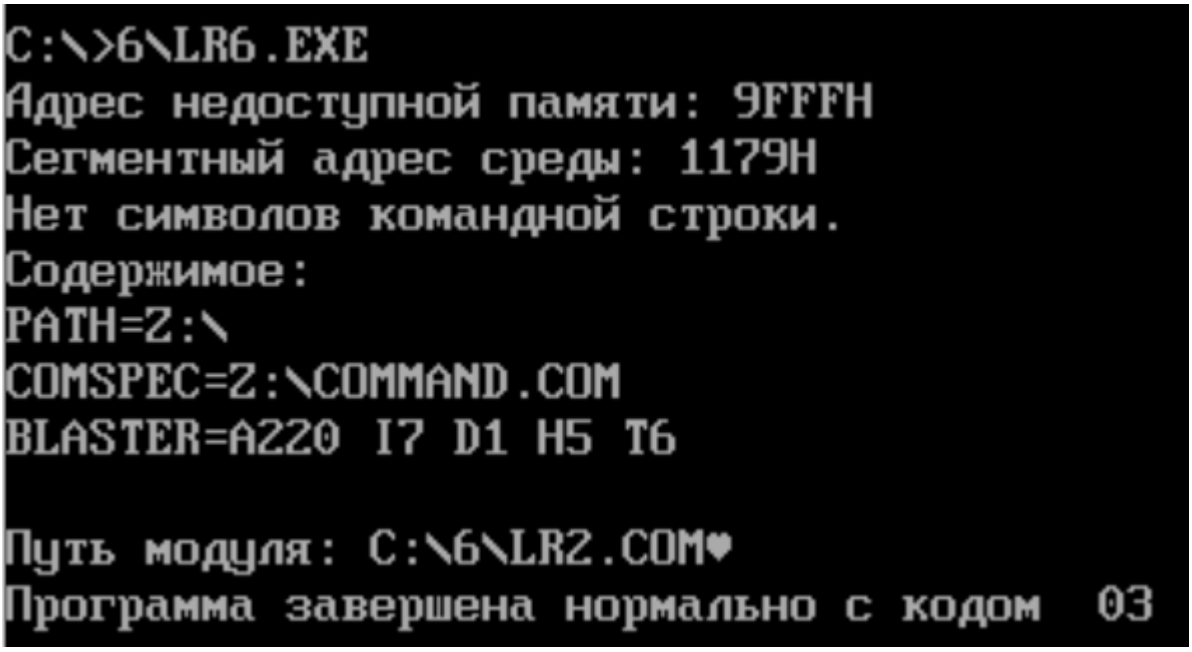

Рисунок 4. Результат выполнения программы, находящейся в каталоге "6" при вводе Ctrl+C.

В случае если модули находятся в разных каталогах, вызывающая программа не сможет найти вызываемую (рисунок 5).

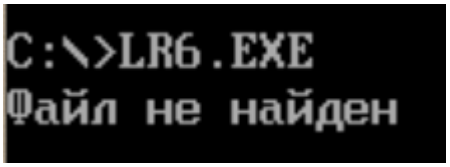

Рисунок 5. Результат выполнения программы, в случае, когда модули находятся в разных каталогах.

Ответы на вопросы.

1. Как реализовано прерывание Ctrl-C?

При вводе данной комбинации вызывается прерывание 23h и управление передается вызвавшему процессу.

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

Программа завершается на строчке int 21h при условии, что ah = 4ch.

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

Программа завершается на строчке, где ожидается ввод символа, то есть строки:

xor al,al

mov ah, 01h

int 21h.

Вывод.

В ходе работы был построен модуль динамической структуры, то есть программа в ходе работы вызывает другую программу и анализирует ее завершение.

ПРИЛОЖЕНИЕ А

```
AStack SEGMENT STACK
    DW 100h DUP(0)
AStack ENDS

DATA SEGMENT
    M_DESTROYED_MCB db'Разрушен управляющий блок памяти',0Dh,0Ah,'$'
    M_NO_MEMORY     db'Недостаточно памяти для выполнения функции',
0Dh,0Ah,'$'
    M_WRONG_ADRESS  db'Неверный адрес блока памяти',0Dh,0Ah,'$'
    PARAMBLK    dw 0
                dd 0
                dd 0
                dd 0
    PATH db 128 dup(0)
    KEEP_SP dw 0
    KEEP_SS dw 0
    M_INVALID_FUNC      db'Не верный номер функции',0Dh,0Ah,'$'
    M_FILE_NOT_EXIST    db'Файл не найден',0Dh,0Ah,'$'
    M_DISK_ERROR        db'Ошибка диска',0Dh,0Ah,'$'
    M_NOT_ENOUGH_MEM    db'Недостаточно памяти',0Dh,0Ah,'$'
    M_INVALID_STRING    db'Неправильная строка среды',0Dh,0Ah,'$'
    M_WRONG_FORMAT      db'Не верный формат',0Dh,0Ah,'$'
    M_T_NORMAL          db 0Dh,0Ah,'Программа завершена нормально $'
    M_T_CTRL_BREAK     db 0Dh,0Ah,'Программа завершена по Ctrl-Break
$'
    M_T_DEVICE_ERROR     db 0Dh,0Ah,'Программа завершена по ошибке
устройства $'
    M_T_31H              db 0Dh,0Ah,'Программа завершена по функции
31h $'
    M_T_CODE            db'с кодом      ',0Dh,0Ah,'$'
DATA ENDS

CODE SEGMENT

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

TETR_TO_HEX PROC near
        and al,0Fh
        cmp al,09
        jbe NEXT
        add al,07
NEXT:   add al,30h; код нуля
        ret
TETR_TO_HEX ENDP

BYTE_TO_HEX PROC near
; байт в al переводится в два символа шестн. 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

WRITE proc near
    push ax
    mov ah,09h
    int 21h
    pop ax
    ret
WRITE ENDP

FREE_MEM proc near
    push ax
    push bx
    push cx
    push dx
    mov bx, offset LAST
    mov ax, es
    sub bx, ax
    mov cl, 4
    shr bx, cl
    mov ah, 4ah
    int 21h
    jc ERROR_FREE
    jmp FREE_OK
ERROR_FREE:
    cmp ax, 7
    je DESTR_MCB
    cmp ax, 8
    je NO_MEM
    cmp ax, 9
    je WRONG_ADRESS
DESTR_MCB:
    lea dx, M_DESTROYED_MCB
    jmp PRINT_ERROR
NO_MEM:
    lea dx, M_NO_MEMORY
    jmp PRINT_ERROR
WRONG_ADRESS:
    lea dx, M_WRONG_ADRESS
    jmp PRINT_ERROR
PRINT_ERROR:
    call WRITE
    xor ax, ax
    mov ah, 4ch
    int 21h
FREE_OK:
    pop dx
    pop cx
    pop bx
    pop ax
    ret
FREE_MEM ENDP

CREATE_PARAMBLK proc near
    push ax
    push bx
```

```
    push cx
    lea bx, PARAMBLK
    mov ax, es
    mov cx, 0
    mov [bx], cx
    mov [bx+2], ax
    mov cx, 80h
    mov [bx+4], cx
    mov [bx+6], ax
    mov cx, 5ch
    mov [bx+8], cx
    mov [bx+10], ax
    mov cx, 6ch
    mov [bx+12], cx
    pop cx
    pop bx
    pop ax
    ret
CREATE_PARAMBLK ENDP

CREATE_CMDSTR proc near
    push es
    push dx

    mov es, es:[2ch]
    mov si, 0
CYCLE_ENV:
    mov dl, es:[si]
    cmp dl, 00h
    je END_CYCLE_ENV
    inc si
    jmp CYCLE_ENV
END_CYCLE_ENV:
    inc si
    mov dl, es:[si]
    cmp dl, 00h
    jne CYCLE_ENV
    add si, 3
    push di
    lea di, PATH

CYCLE_PATH:
    mov dl, es:[si]
    cmp dl, 00h
     je END_CYCLE_PATH
     mov [di], dl
    inc si
    inc di
     jmp CYCLE_PATH
END_CYCLE_PATH:
    sub di, 7
    mov [di], byte ptr 'L'
    mov [di+1], byte ptr 'R'
    mov [di+2], byte ptr '2'
    mov [di+3], byte ptr '.'
    mov [di+4], byte ptr 'C'
    mov [di+5], byte ptr 'O'
```

```
    mov [di+6], byte ptr 'M'
    mov [di+7], byte ptr 0
    pop dx
    pop di
    pop es
    ret
CREATE_CMDSTR ENDP

CHILD_RUN proc near
    push ds
    mov KEEP_SP, sp
    mov KEEP_SS, ss
    pop es
    lea bx, PARAMBLK
    lea dx, PATH
    xor al, al
    mov ah, 4bh
    int 21h
    mov sp, KEEP_SP
    mov ss, KEEP_SS
    jnc CORRECT_LOAD
    cmp ax, 1
    je INV_FUN_NUMBER
    cmp ax, 2
    je FILE_NOT_EXIST
    cmp ax, 5
    je DISK_ERROR
    cmp ax, 8
    je NOT_ENOUGH_MEMORY
    cmp ax, 10
    je INV_STR_ENV
    cmp ax, 11
    je FORMAT_NOT_MATCH
INV_FUN_NUMBER:
    lea dx, M_INVALID_FUNC
    jmp END_WITH_ERR
FILE_NOT_EXIST:
    lea dx, M_FILE_NOT_EXIST
    jmp END_WITH_ERR
DISK_ERROR:
    lea dx, M_DISK_ERROR
    jmp END_WITH_ERR
NOT_ENOUGH_MEMORY:
    lea dx, M_NOT_ENOUGH_MEM
    jmp END_WITH_ERR
INV_STR_ENV:
    lea dx, M_INVALID_STRING
    jmp END_WITH_ERR
FORMAT_NOT_MATCH:
    lea dx, M_WRONG_FORMAT
    jmp END_WITH_ERR
END_WITH_ERR:
    call WRITE
    xor al, al
    mov ah, 4ch
    int 21h
CORRECT_LOAD:
```

```
    xor al, al
    mov ah, 4dh
    int 21h
    cmp ah, 0
    je NORMAL
    cmp ah, 1
    je CTRL_BREAK
    cmp ah, 2
    je DEVICE_ERROR
    cmp ah, 3
    je STAY_RESIDENT
NORMAL:
    lea dx, M_T_NORMAL
    jmp PRINT_TERM
CTRL_BREAK:
    lea dx, M_T_CTRL_BREAK
    jmp PRINT_TERM
DEVICE_ERROR:
    lea dx, M_T_DEVICE_ERROR
    jmp PRINT_TERM
STAY_RESIDENT:
    lea dx, M_T_31H
    jmp PRINT_TERM
PRINT_TERM:
    call WRITE
    lea di, M_T_CODE
    call BYTE_TO_HEX
    add di, 9
    mov [di], al
    inc di
    mov [di], ah
    lea dx, M_T_CODE
    call WRITE
        xor ax, ax
    mov ah, 4ch
    int 21h
    ret
CHILD_RUN ENDP

MAIN proc far
    mov ax, DATA
    mov ds, ax

    call FREE_MEM
    call CREATE_PARAMBLK
    call CREATE_CMDSTR
    call CHILD_RUN

    xor ax, ax
    mov ah, 4ch
    int 21h
    ret
MAIN ENDP
CODE ENDS
LAST SEGMENT
LAST ENDS
END MAIN
```