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. 8382 _____ Щемель Д.А.

Преподаватель _____ Ефремов М.А.

Санкт-Петербург

2020

Цель работы

исследование возможности построения загрузочного модуля динамической структуры. В отличии от предыдущих работ 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 моделями по управлению и по данным. Для запуска вызываемого модуля используется функции 4B00h прерывания int 21h. Все загрузочные модули находятся в одном каталоге. Необходимо обеспечить возможность запуска модуля динамической структуры из любого каталога.

Ход выполнения работы

Был написан исходный код для .EXE-модуля, выполняющий следующие действия:

1. Вычисляет путь до запускаемой программы и подготавливает необходимые для запуска параметры(среда, командная строка)
2. Вызывается модифицированная программа, разработанная в ходе лабораторной работы №2, которая считывает символ и устанавливает его в качестве кода выхода для функции 4ch.
3. В случае удачного вызова программы выводятся код завершения и причина завершения
4. В противном случае выводится код ошибки запуска программы

Результат работы программы при запуске программы, когда вызываемая программа находится в том же каталоге:

Результат работы программы при запуске программы, когда вызываемая программа находится в том же каталоге, но уже другом. Вызываемая программа завершается сочетанием клавиш crtl+c:

Результат работы программы, когда в каталоге с вызывающей программой нет вызываемой программы:

Контрольные вопросы

1. Как реализовано прерывание ctrl+c?

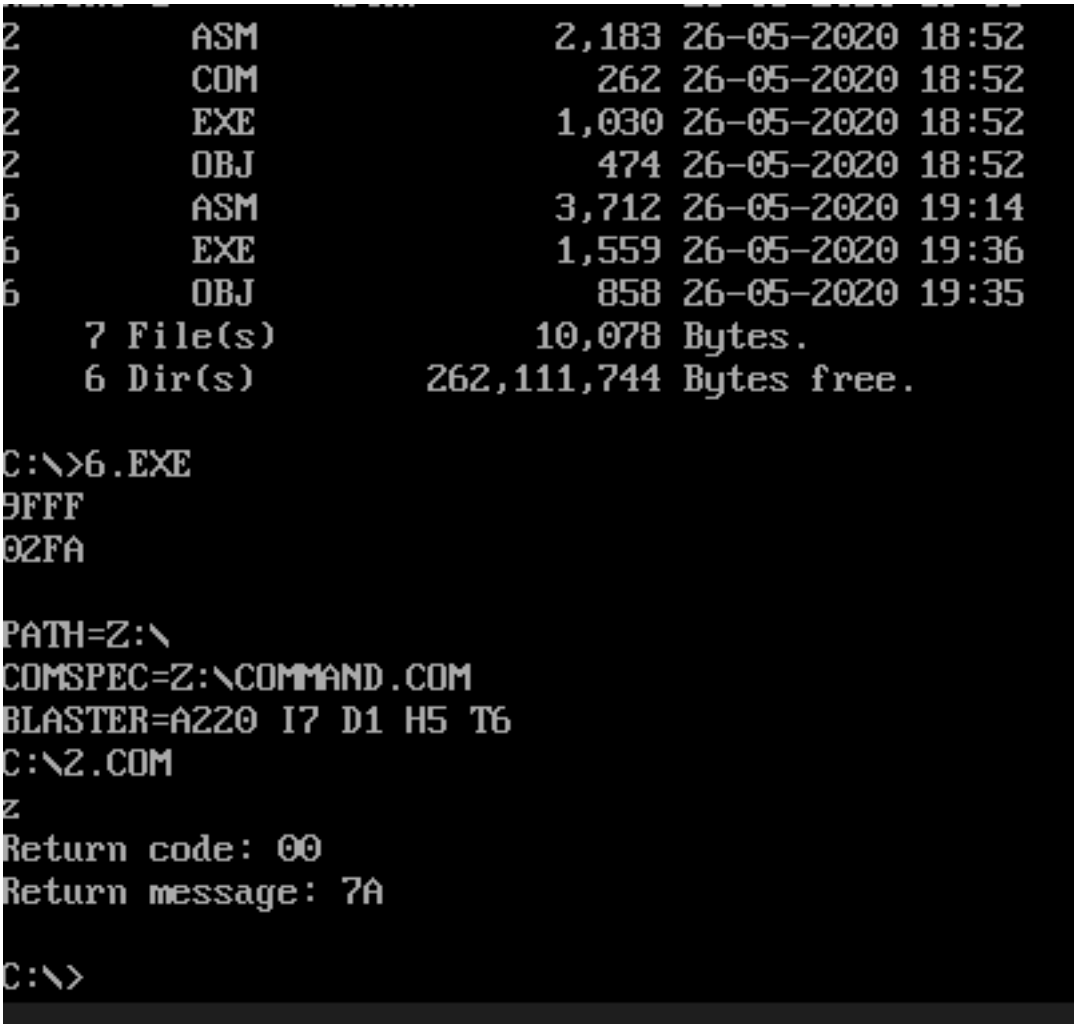

Рис. 1: Результат работы

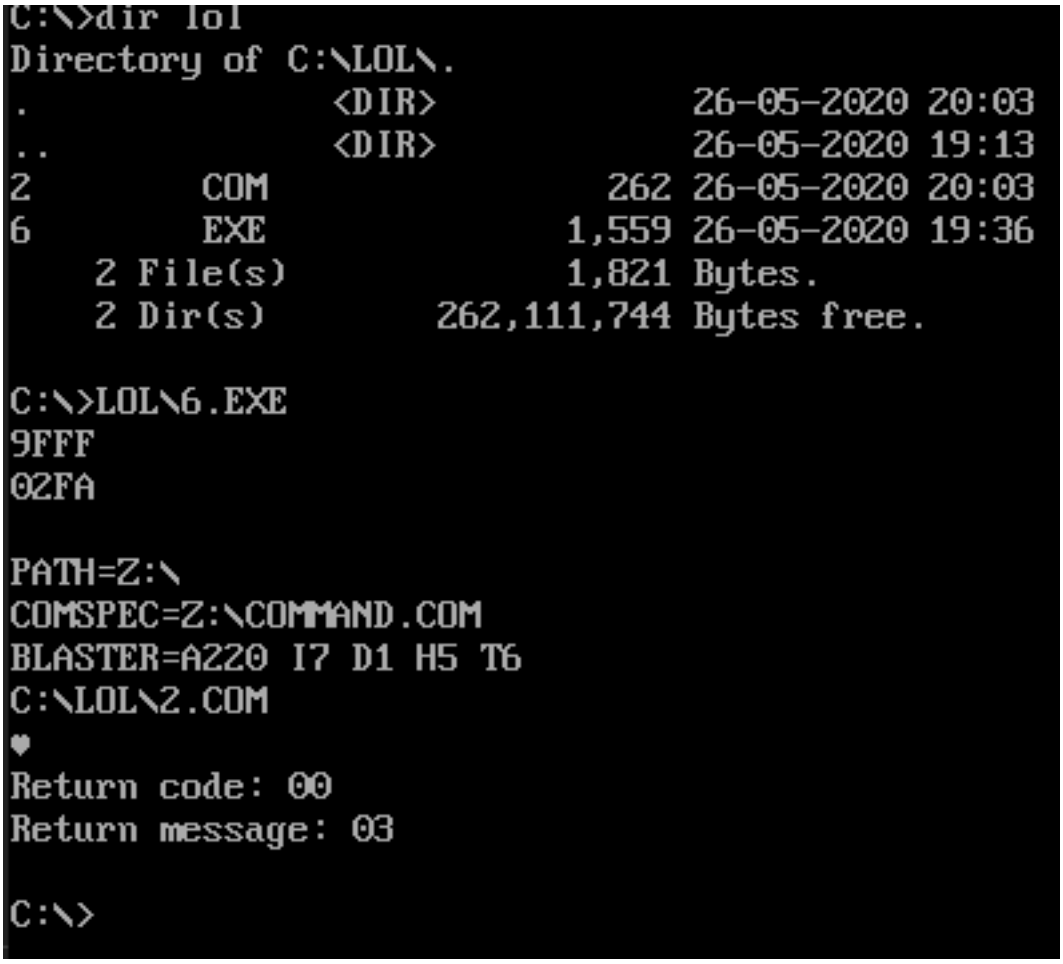

Рис. 2: Результат работы

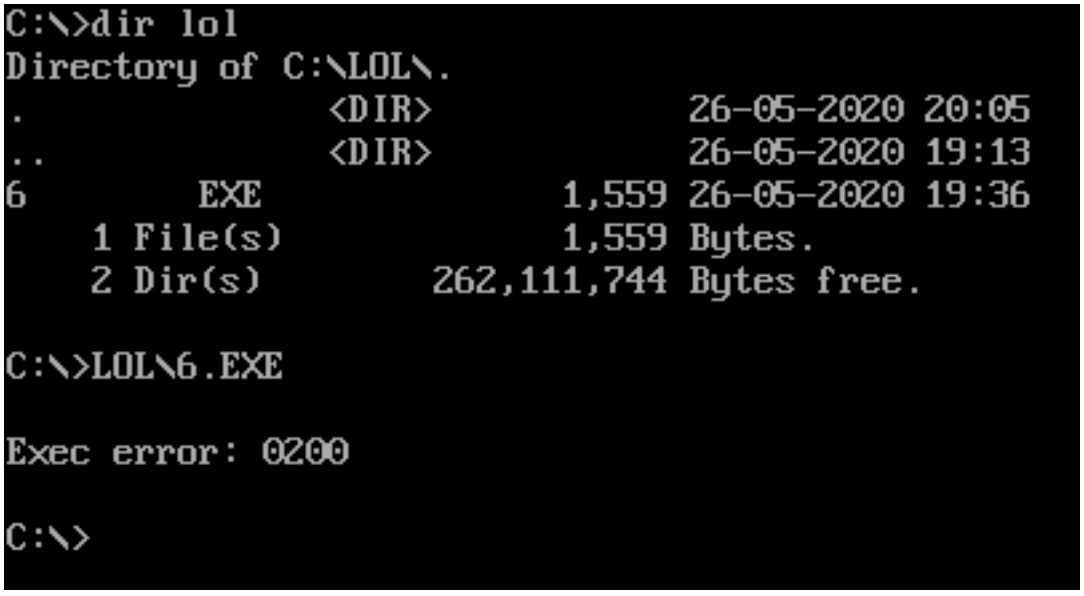

Рис. 3: Результат работы

При нажатии ctrl+c происходит вызов прерывания int 23h, которое передаёт управление вызывающей программе

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

   В точке вызова прерывания 4ch int 21h

3. В какой точке заканчивается вызываемая программа по прерыванию ctrl+c?

   В точке, где прерывание int 23h было вызвано. В нашем примере это точка считывания символа с клавиатуры (как видно на скриншотах выше - не работает на ВМ)

Вывод

В ходе выполнения лабораторной работы были получены практические навыки по построению загрузочного модуля динамической структуры.

ПРИЛОЖЕНИЕ А. ИСХОДНЫЙ КОД ПРОГРАММЫ 6.ASM

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

DATA SEGMENT
    PROGRAM_PATH db 80 dup(0)
    PARAMETER_BLOCK dw 0
    CMD_POINTER dd 0
    FIRST_FCB_STUB dd 0
    SECOND_FCB_STUB dd 0
    KEEP_SS dw 0
    KEEP_SP dw 0
    KEEP_DS dw 0

    EXEC_ERROR db 'Exec error: 0000', 13, 10, '$'
    RETURN_CODE db 'Return code: 00', 13, 10, '$'
    RETURN_MESSAGE db 'Return message: 00', 13, 10, '$'
    END_LINE db 13, 10, '$'
DATA ENDS

CODE SEGMENT
ASSUME CS:CODE, DS:DATA, SS:_STACK

EXIT PROC near
    xor AL, AL
    mov AH, 4ch
    int 21h
    ret
EXIT ENDP

PRINT PROC near
    push ax
```

```
    mov ah, 09h
    int 21h
    pop ax
    ret
PRINT ENDP

ADJUST_SIZE PROC near
    push ax
    push bx

    mov ah, 4ah
    mov bx, offset END_LABEL
    int 21h

    pop bx
    pop ax
    ret
ADJUST_SIZE ENDP

PREPARE_PARAMETER_BLOCK PROC near
    push si
    push ax

    mov si, offset CMD_POINTER
    mov [si], es
    mov al, 80h
    mov [si + 2], al

    mov si, offset FIRST_FCB_STUB
    mov [si], es
    mov al, 5ch
    mov [si + 2], al

    mov si, offset SECOND_FCB_STUB
```

```
    mov [si], es
    mov ah, 6ch
    mov [si + 2], ah

    pop ax
    pop si
    ret
PREPARE_PARAMETER_BLOCK ENDP

FIND_START_PATH PROC near
; OUT: si - offset to end of path var
    push es
    push ax
    push dx

    mov es, es:[2ch]
    mov si, 0
    CMP_WITH_0000_AND_INC:
        mov al, es:[si]
        mov ah, es:[si+1]
        cmp ax, 0000h
        je FINISH_FIND_END_PATH
    inc si
    jmp CMP_WITH_0000_AND_INC

    FINISH_FIND_END_PATH:
    add si, 4
    pop dx
    pop ax
    pop es
    ret
FIND_START_PATH ENDP

COPY_PATH PROC near
```

```
; IN si - start path IN PSP
; OUT di - end of path IN DATA
    push ax
    push es
    push si

    mov es, es:[2ch]
    mov di, offset PROGRAM_PATH
    CMP_WITH_COPY_AND_INC:
    mov al, es:[si]
    mov [di], al
    cmp al, 0
    je FINISH_COPY_PATH
    inc si
    inc di
    jmp CMP_WITH_COPY_AND_INC

    FINISH_COPY_PATH:
    pop si
    pop es
    pop ax
    ret
COPY_PATH ENDP

PREPARE_DATA PROC near
; OUT: ds:dx - path
    push bx
    push di

    call PREPARE_PARAMETER_BLOCK
    call FIND_START_PATH
    call COPY_PATH

    mov dx, offset PROGRAM_PATH
```

```
    sub di, 5
    mov [di], byte ptr '2'
    mov [di+1], byte ptr '.'
    mov [di+2], byte ptr 'C'
    mov [di+3], byte ptr 'O'
    mov [di+4], byte ptr 'M'

    pop di
    pop bx
    ret
PREPARE_DATA ENDP

SAVE_DATA PROC near
    mov KEEP_SS, ss
    mov KEEP_SP, sp
    mov KEEP_DS, ds
    ret
SAVE_DATA ENDP

RESTORE_DATA PROC near
    mov ss, KEEP_SS
    mov sp, KEEP_SP
    mov ds, KEEP_DS
    ret
RESTORE_DATA 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
; IN: al
; OUT: ax
    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

PROCEED_RESULT PROC near
    push ax
    push bx
    push si
    push dx

    mov ax, 4d00h
    int 21h

    mov bl, al

    mov al, ah
    call BYTE_TO_HEX
    mov si, offset RETURN_CODE
    add si, 13
    mov [si], ax

    mov dx, offset RETURN_CODE
```

```
    call PRINT

    mov al, bl
    call BYTE_TO_HEX
    mov si, offset RETURN_MESSAGE
    add si, 16
    mov [si], ax

    mov dx, offset RETURN_MESSAGE
    call PRINT

    pop dx
    pop si
    pop bx
    pop ax
    ret
PROCEED_RESULT ENDP

RUN_PROGRAM PROC near
    push ax
    push es
    push bx
    push dx
    push si

    call PREPARE_DATA
    call SAVE_DATA

    mov ax, seg PROGRAM_PATH
    mov es, ax
    mov bx, offset PARAMETER_BLOCK

    mov ax, 4b00h
    int 21h
```

```
    call RESTORE_DATA

    mov dx, offset END_LINE
    call PRINT

    jnc EXEC_OK

    mov bh, ah
    call BYTE_TO_HEX
    mov si, offset EXEC_ERROR
    add si, 12
    mov [si], ax
    mov al, bh
    call BYTE_TO_HEX
    add si, 2
    mov [si], ax
    mov dx, offset EXEC_ERROR
    call PRINT
    jmp FINISH_RUN_PROGRAM

    EXEC_OK:
        call PROCEED_RESULT

    FINISH_RUN_PROGRAM:
    pop si
    pop dx
    pop ax
    pop es
    pop bx
    ret
RUN_PROGRAM ENDP

MAIN:
    mov ax, DATA
```

```
    mov ds, ax

    call ADJUST_SIZE
    call RUN_PROGRAM

    call EXIT
    END_LABEL:
CODE ENDS
END MAIN
```

ПРИЛОЖЕНИЕ В. ИСХОДНЫЙ КОД ПРОГРАММЫ 2.ASM

```
PSPRESEARCH SEGMENT
       ASSUME CS:PSPRESEARCH, DS:PSPRESEARCH, ES:NOTHING, SS:NOTHING
       ORG 100H
START: JMP MAIN

START_FORBIDEN db '0000', 10, 13, '$'
ENV_ADDRESS db '0000', 10, 13, '$'
CRLF db 10, 13, '$'

EXIT PROC near
    mov AH, 4ch
    int 21h
    ret
EXIT ENDP

PRINT PROC near
    push ax
    mov ah, 09h
    int 21h
    pop ax
    ret
PRINT 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

BYTE_TO_DEC PROC near
    push cx
    push dx
    xor ah, ah
    xor dx, dx
    mov cx, 10
    loop_bd:
        div cx
        or dl, 30h
        mov [si], dl
        dec si
```

```
        xor dx, dx
        cmp ax, 10
        jae loop_bd
        cmp al, 00h
        je end_l
        or al, 30h
        mov [si], al
    end_l:
        pop dx
        pop cx
    ret
BYTE_TO_DEC ENDP

MAIN:
    mov dx, es:[2h]
    mov al, dh
    call BYTE_TO_HEX
    mov si, offset START_FORBIDEN
    mov [si], ax
    mov si, offset START_FORBIDEN
    mov al, dl
    call BYTE_TO_HEX
    mov [si+2], ax
    mov dx, offset START_FORBIDEN
    call PRINT

    mov dx, es:[2ch]
    mov al, dh
    call BYTE_TO_HEX
    mov si, offset ENV_ADDRESS
    mov [si], ax
    mov si, offset ENV_ADDRESS
    mov al, dl
    call BYTE_TO_HEX
```

```
mov [si+2], ax
mov dx, offset ENV_ADDRESS
call PRINT

mov cl, es:[80h]
mov si, 81h
mov ah, 2h
cmp cl, 0
je FINISH_PRINT_NEXT_CHAR_FROM_CL
PRINT_NEXT_CHAR_FROM_CL:
    mov dl, [si]
    int 21h
    inc si
    loop PRINT_NEXT_CHAR_FROM_CL
FINISH_PRINT_NEXT_CHAR_FROM_CL:
    mov dx, offset CRLF
    call PRINT

mov es, es:[2ch]
mov dl, es:[0]
mov si, 0
PRINT_NEXT_CHAR_FROM_ENV:
    int 21h
    inc si
    mov dl, es:[si]
    cmp dl, 0
    je PRINT_NEXT_LINE
    jmp PRINT_NEXT_CHAR_FROM_ENV
PRINT_NEXT_LINE:
    mov dx, offset CRLF
    call PRINT
    inc si
    mov dl, es:[si]
    cmp dl, 0
```

```
        je FINISH_PRINT_ENV
        jmp PRINT_NEXT_CHAR_FROM_ENV
    FINISH_PRINT_ENV:
    add si, 3
    mov dl, es:[si]
    PRINT_NEXT_CHAR_FROM_PATH:
        int 21h
        inc si
        mov dl, es:[si]
        cmp dl, 0
        je FINISH_PRINT_NEXT_CHAR_FROM_PATH
        jmp PRINT_NEXT_CHAR_FROM_PATH
    FINISH_PRINT_NEXT_CHAR_FROM_PATH:
        mov dx, offset CRLF
        call PRINT

    mov ah, 1h
    int 21h

    call EXIT
PSPRESEARCH ENDS
END START
```