

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. 9382

Кузьмин Д. И.

Преподаватель

Ефремов М. А.

Санкт-Петербург

2021

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 4B00h прерывания int 21h. Все загрузочные модули находятся в одном каталоге. Необходимо обеспечить возможность запуска модуля динамической структуры из любого каталога.

Задание.

Шаг 1. Для выполнения лабораторной работы необходимо написать и отладить программный модуль типа .EXE, который выполняет функции: 1) 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. 2) Вызываемый модуль запускается с использованием загрузчика. 3) 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.

Шаг 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.

Шаг 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.

Шаг 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.

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

Выполнение работы.

- 1) Первым шагом было создание функции устанавливающей параметры для запуска вызываемой программы
- 2) Затем была создана функция, реализующая запуск вызываемой программы “LB2.COM”, при помощи функции 4b00h прерывания int 21h с сохранением и последующим восстановлением регистров ss, sp и ds
- 3) Далее была реализована обработка случаев удачного или неудачного запуска программ с выводом кода соответствующей ошибки.

Результаты работы программы см. в приложении А

Исходный код см. в приложении Б

Контрольные вопросы.

1) Как реализовано прерывание Ctrl-C?

При нажатии Ctrl-C вызывается прерывание INT 23H, завершающее работу текущей программы. Адрес в векторе INT 23H копируется в поле PSP Ctrl-Break Address функциями DOS 26H и 4cH. Исходное значение адреса обработчика Ctrl-Break восстанавливается из PSP при завершении программы.

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

Она заканчивается в точке вызова функции 4ch прерывания int 21h

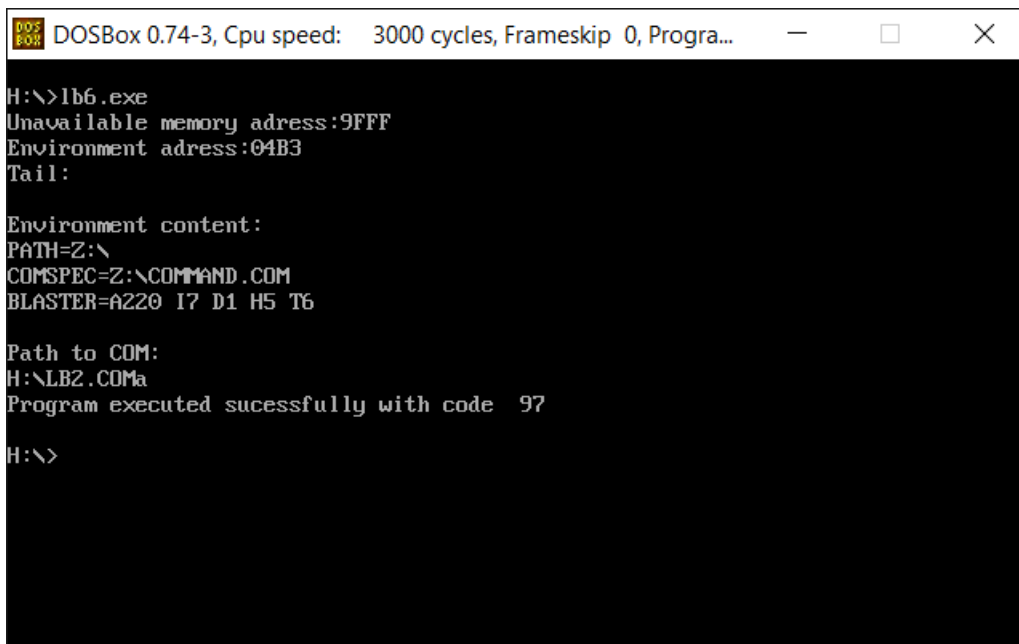
3) В какой точке заканчивается вызываемая программа по прерыванию Ctrl-C?

Вызываемая программа заканчивается в точке вызова функции 01h

Выводы.

Были изучены возможности построения загрузочного модуля динамической структуры.

ПРИЛОЖЕНИЕ А. ДЕМОНСТРАЦИЯ РАБОТЫ ПРОГРАММЫ

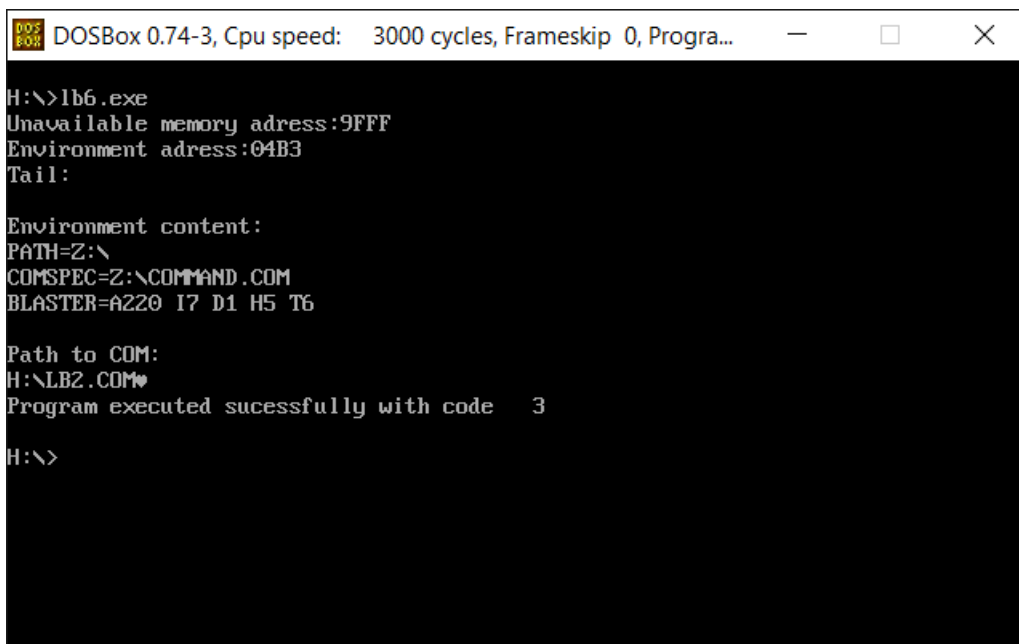


```
DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip 0, Progra...
H:\>lb6.exe
Unavailable memory address:9FFF
Environment address:04B3
Tail:

Environment content:
PATH=Z:\
COMSPEC=Z:\COMMAND.COM
BLASTER=A220 I7 D1 H5 T6

Path to COM:
H:\LB2.COMa
Program executed sucessfully with code 97
H:\>
```

Вызов программы LB2 и ее завершение после ввода символа ‘а’



```
DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip 0, Progra...
H:\>lb6.exe
Unavailable memory address:9FFF
Environment address:04B3
Tail:

Environment content:
PATH=Z:\
COMSPEC=Z:\COMMAND.COM
BLASTER=A220 I7 D1 H5 T6

Path to COM:
H:\LB2.COMa
Program executed sucessfully with code 3
H:\>
```

Вызов программы LB2 и ее завершение после ввода комбинации ‘Ctrl-C’. В DosBox данная комбинация не работает.

```
DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip 0, Progra...
H:\>cd dir2
H:\DIR2>lb6.exe
Unavailable memory adress:9FFF
Environment adress:04B3
Tail:
Environment content:
PATH=Z:\
COMSPEC=Z:\COMMAND.COM
BLASTER=A220 I7 D1 H5 T6
Path to COM:
H:\DIR2\LB2.COMa
Program executed sucessfully with code 97
H:\DIR2>
```

Вызов программы LB2.COM и ее завершение после введенного символа 'a', но в другом каталоге. Видно, что выполнение завершилось успешно, т.к. обе программы – вызывающая и вызываемая находятся в одном каталоге.

```
DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip 0, Progra...
H:\>cd dir3
H:\DIR3>lb6.exe
Program execution failed with code 2
H:\DIR3>_
```

Вызов программы LB2.COM из каталога, в которой этой программы нет. Видно, что выполнение завершилось с кодом ошибки 2 – файл не найден.

ПРИЛОЖЕНИЕ Б. ИСХОДНЫЙ КОД

Файл lb6.asm

```
ASTACK      SEGMENT      STACK
            DW 300 DUP (?)
ASTACK      ENDS

DATA SEGMENT
block_destroyed db 'Block destroyed error' , 13, 10, '$'
memory_not_enough db 'Memory not enough error', 13, 10, '$'
invalid_address_of_block db 'Invalid address of block error', 13, 10, '$'
ctrlbreak_finish db 13, 10, 'Program exitde with ctrlbreak', 13, 10, '$'
ok_finish db 13, 10, 'Program executed sucessfully with code      ', 13, 10, '$'
bad_finish db 'Program execution failed with code      ', 13, 10, '$'
prog_name db 'LB2.COM', 0
file_name_addr dw 0
psp dw 0
epb dw 0 ;сегментный адрес среды
            dd 0 ;сегмент и смещение командной строки
            dd 0 ;сегмент и смещение FCB
            dd 0 ;сегмент и смещение FCB
path db 50 dup (0)
DATA ENDS

CODE SEGMENT
        .386
        ASSUME CS:CODE, DS:DATA, SS:ASTACK
; Процедуры

WRITE_MSG PROC near
            push ax
            mov AH, 09h
            int 21h
            pop ax
            ret
WRITE_MSG ENDP
;-----
BYTE_TO_DEC PROC near
;Перевод в 10чную с/с, SI - адрес младшей цифры
            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_1
                                or AL,30h
                                mov [SI],AL
end_1:                          pop DX
                                pop CX
                                ret
BYTE_TO_DEC ENDP
;-----

SET_EPB PROC NEAR
                                mov ax,es:[2ch]
                                mov epb,ax
                                mov epb+2,es
                                mov epb+4,80h
                                ret
SET_EPB ENDP

PROCESS_EXECUTION PROC NEAR
                                push ax
                                mov ah, 4dh
                                int 21h
                                cmp ah, 0
                                je codezero
                                cmp ah, 1
                                je break

                                codezero:
                                mov SI, OFFSET ok_finish
                                add SI, 43
                                call BYTE_TO_DEC
                                mov DX, OFFSET ok_finish
                                call WRITE_MSG
                                jmp _end

                                break:
                                lea dx, ctrlbreak_finish
                                call WRITE_MSG
                                _end:
                                pop ax
                                ret
PROCESS_EXECUTION ENDP
;-----

PROCESS_FAILURE PROC NEAR
                                push ax

                                mov SI, OFFSET bad_finish
                                add SI, 37
                                call BYTE_TO_DEC
                                mov DX, OFFSET bad_finish
                                call WRITE_MSG
                                pop ax
                                ret
PROCESS_FAILURE ENDP
;-----

START_PROGRAM PROC NEAR

```



```

        jmp start
        keep_ss dw 0
        keep_sp dw 0
        keep_ds dw 0

start:
        call GET_PATH_TO_FILE
        ;выделение памяти под программу и переход к обработке
ВОЗМОЖНЫХ ОШИБОК
        push bx
        mov bx, offset end_point
        mov ax, cs
        add bx, ax
        shr bx, 4h
        add bx, 40h
        mov ah, 4ah
        int 21h
        jc carry

        pop bx

        ;сохранение регистров
        mov keep_ss, ss
        mov keep_sp, sp
        mov keep_ds, ds

        ;загружаем путь к файлу в DS:DX
        mov dx, offset path

        ;устанавливаем параметры
        call SET_EPB
        mov bx, offset epb

        ;вызов программы
        mov ax, 4B00h
        int 21h
        jnc executed

        call PROCESS_FAILURE
        jmp restore

executed:
        call PROCESS_EXECUTION
        ;восстановление регистров
restore:
        mov ss, keep_ss
        mov sp, keep_sp
        mov ds, keep_ds

        jmp exitfromhere

        ;обработка ошибок
carry:
        cmp ax, 7
        je destroyed
        cmp ax, 8
        je not_enough

```

```

        cmp ax, 9
        je invalid_adress
        jmp exitfromhere
    destroyed:
        mov dx, offset block_destryoed
        jmp exitfromhere
    not_enough:
        mov dx, offset memory_not_enough
        jmp exitfromhere
    invalid_adress:
        mov dx, offset invalid_adress_of_block

    exitfromhere:
        ret
START_PROGRAM ENDP
GET_PATH_TO_FILE PROC NEAR

        push di
        push si
        push ax
        push bx
        push cx
        push dx
        push es

        mov ax, psp
        mov es, ax
        mov es, es:[2ch]
        mov bx, 0

    continue:
        inc bx
        cmp byte ptr es:[bx-1], 0
        jne continue

        cmp byte ptr es:[bx+1], 0
        jne continue

        add bx, 2
        mov di, 0

    check:
        mov dl, es:[bx]
        mov byte ptr [path+di], dl
        add di, 1
        add bx, 1
        cmp dl, 0
        je end_check
        cmp dl, '\'
        jne check
        mov cx, di
        jmp check
    end_check:
        mov di, cx
        mov si, offset prog_name

```

```

        write_end:
            mov dl, byte ptr [si]
            mov byte ptr [path+di], dl
            inc di
            inc si
            cmp dl, 0
            jne write_end

            pop es
            pop dx
            pop cx
            pop bx
            pop ax
            pop si
            pop di
            ret

GET_PATH_TO_FILE ENDP

MAIN      PROC FAR

            mov ax, DATA
            mov ds, ax
            mov psp, es
            call START_PROGRAM
            mov ah, 4ch
            int 21h
            end_point:

MAIN ENDP
CODE ENDS
        END MAIN

```