

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

Студентка гр. 0382

Здобнова К.Д.

Преподаватель

Ефремов М.А.

Санкт-Петербург

2022

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.

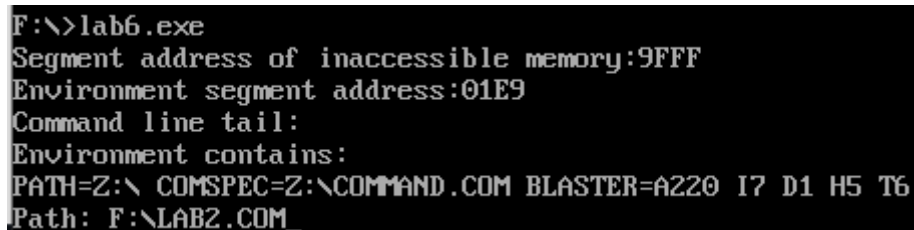
Шаг 4. Запустите отлаженную программу, когда текущим каталогом является какойлибо другой каталог, отличный от того, в котором содержатся разработанные программные модули. Повторите ввод комбинаций клавиш. Занесите полученные данные в отчет.

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

Выполнение работы.

Был написан и отлажен загрузочный модуль, подготавливающий параметры для запуска загрузочного модуля lab2, запускающий его и проверяющий его выполнение выводом символа из него.

Запустим его и увидим выполнение загрузочного модуля lab2.



```
F:\>lab6.exe
Segment address of inaccessible memory:9FFF
Environment segment address:01E9
Command line tail:
Environment contains:
PATH=Z:\ COMSPEC=Z:\COMMAND.COM BLASTER=A220 I7 D1 H5 T6
Path: F:\LAB2.COM_
```

Рисунок 1.

После чего введем символ, запрошенный в нем и увидим результат работы уже в выполнении загрузочного модуля lab6.

```
F:\>lab6.exe
Segment address of inaccessible memory:9FFF
Environment segment address:01E9
Command line tail:
Environment contains:
PATH=Z:\ COMSPEC=Z:\COMMAND.COM BLASTER=A220 I7 D1 H5 T6
Path: F:\LAB2.COMj
Normal
Returned: j
```

Рисунок 2.

Запустим снова программу и остановим ее, с помощью Ctrl+C, однако в связи с тем, что DOSBox не поддерживает данную комбинацию – будет символ – мы увидим нормальный результат работы программы. Однако, должен произойти выход из программы.

```
F:\>lab6.exe
Segment address of inaccessible memory:9FFF
Environment segment address:01E9
Command line tail:
Environment contains:
PATH=Z:\ COMSPEC=Z:\COMMAND.COM BLASTER=A220 I7 D1 H5 T6
Path: F:\LAB2.COM♥
Normal
Returned: ♥
```

Рисунок 3.

Изменим местоположение обоих загрузочных модулей в новый каталог и выполним снова программу. Результат, ожидаемо, нормальный.

```
F:\TMP>lab6.exe
Segment address of inaccessible memory:9FFF
Environment segment address:01E9
Command line tail:
Environment contains:
PATH=Z:\ COMSPEC=Z:\COMMAND.COM BLASTER=A220 I7 D1 H5 T6
Path: F:\TMP\LAB2.COMt
Normal
Returned: t
```

Рисунок 4.

Однако, при изменении местоположения в разные папки разных модулей увидим ошибку, что файл не был найден.

```
F:\TMP>lab6.exe
Error! No file found!
```

Рисунок 5.

Ответы на контрольные вопросы:

1) Как реализовано прерывание Ctrl+C?

При нажатии комбинации – возможны два варианта: программа будет реагировать на нее, как на ввод символа, либо произойдет переход по адресу, который записан в прерывании int 23h.

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

Данный код завершения подразумевает успешное завершение. Соответственно, программа закончится при выполнении 4ch прерывания int 21h

3) В какой точке закидывается вызываемая программа по прерыванию Ctrl+C?

Если клавиши нажаты, то программа завершается в том месте, в котором произошло это нажатие (в месте ожидания нажатия клавиши: 01h вектора прерывания 21h).

Выводы.

В ходе выполнения лабораторной работы был написан загрузочный модуль динамической структуры, а так же был изучен принцип работы с памятью.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: lab6.asm

```
MY_STACK SEGMENT STACK
    dw 128 dup (?)
MY_STACK ENDS

CODE SEGMENT
    ASSUME CS: CODE, DS: CODE, SS:MY_STACK

    KEEP_PGPH db 14 dup(0)
    KEEP_PATH db 50 dup(0)
    KEEP_SS    dw 0
    KEEP_SP    dw 0
    lab2 db 'lab2.com', 0
    retCode    db 'Returned: $'
    nextLine   db 13, 10, '$'

    term_1      db 'Normal$'
    term_2      db '^C$'
    term_3      db 'Error with device$'
    term_4      db 'int 31h$'
    term_5      db 'Unknown error$'

    err_0       db 'Error! Memory can not be allocated!$'
    err_1       db 'Error! Wrong number!$'
    err_2       db 'Error! No file found!$'
    err_3       db 'Error! Disk error!$'
    err_4       db 'Error! Need more memory!$'
    err_5       db 'Error! Wrong enviroment!$'
    err_6       db 'Error! Wrong format!$'
    err_7       db 'Error! Unknown error!$'

print PROC NEAR
    push ax
    mov ah, 09h
    int 21h
    pop ax
    ret
print ENDP

main PROC NEAR
    mov ax, seg code
    mov ds, ax
    mov bx, seg code
    add bx, OFFSET EndProg
    add bx, 256
    mov cl, 4h
    shr bx, cl
    mov ah, 4Ah
    int 21h
    jnc memCheck
    mov dx, OFFSET err_0
    call print
    mov dx, OFFSET nextLine
    call print
    jmp return
```

```

memCheck:
    mov     es, es:[002Ch]
    xor     bx, bx

check_0:
    mov     dl, byte PTR es:[bx]
    cmp     dl, 0h
    je      ch1
    inc     bx
    jmp     check_0

ch1:
    inc     bx
    mov     dl, byte PTR es:[bx]
    cmp     dl, 0h
    je      ch2
    jmp     check_0

ch2:
    add     bx, 3
    push    si
    mov     si, OFFSET KEEP_PATH

check_1:
    mov     dl, byte PTR es:[bx]
    mov     [si], dl
    inc     si
    inc     bx
    cmp     dl, 0
    jne     check_1

check_2:
    mov     al, [si]
    cmp     al, '\'
    je      check_3
    dec     si
    jmp     check_2

check_3:
    inc     si
    push    di
    mov     di, OFFSET lab2

check_4:
    mov     ah, [di]
    mov     [si], ah
    inc     si
    inc     di
    cmp     ah, 0
    jne     check_4

    pop     di
    pop     si

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

```

```

        mov     bx, OFFSET KEEP_PGPH
        mov     dx, OFFSET KEEP_PATH
        mov     ax, 4B00h
        int     21h

        mov     dx, OFFSET nextLine
        call    print
        jc      err1Check
        jmp     NoErr

err1Check:
        cmp     ax, 1
        jne     err2Check
        mov     dx, OFFSET err_1
        jmp     printErr

err2Check:
        cmp     ax, 2
        jne     err3Check
        mov     dx, OFFSET err_2
        jmp     printErr

err3Check:
        cmp     ax, 5
        jne     err4Check
        mov     dx, OFFSET err_3
        jmp     printErr

err4Check:
        cmp     ax, 8
        jne     err5Check
        mov     dx, OFFSET err_4
        jmp     printErr

err5Check:
        cmp     ax, 10
        jne     err6Check
        mov     dx, OFFSET err_5
        jmp     printErr

err6Check:
        cmp     ax, 11
        jne     err7Check
        mov     dx, OFFSET err_6
        jmp     printErr

err7Check:
        mov     dx, OFFSET err_7
        jmp     printErr

printErr:
        call    print
        mov     dx, OFFSET nextLine
        call    print
        jmp     return

NoErr:
        mov     ax, seg code
        mov     ds, ax
        mov     ss, KEEP_SS

```



```

        mov     sp, KEEP_SP

        mov     ah, 4Dh
        int     21h

        push    ax

        cmp     ah, 0
        jne     term2Check
        mov     dx, OFFSET term_1
        jmp     printTerm

term2Check:
        cmp     ah, 1
        jne     term3Check
        mov     dx, OFFSET term_2
        jmp     printTerm

term3Check:
        cmp     ah, 2
        jne     term4Check
        mov     dx, OFFSET term_3
        jmp     printTerm

term4Check:
        cmp     ah, 3
        jne     term5Check
        mov     dx, OFFSET term_4
        jmp     printTerm

term5Check:
        mov     dx, OFFSET term_5

printTerm:
        call    print
        mov     dx, OFFSET nextLine
        call    print
        jmp     ExitCode

ExitCode:
        mov     dx, OFFSET retCode
        call    print
        pop     ax
        mov     dl, al
        mov     ah, 02h
        int     21h
        mov     dx, OFFSET nextLine
        call    print

return:
        xor     al, al
        mov     ah, 4Ch
        int     21h
        ret

main ENDP
EndProg:
CODE ENDS
END main

```

Название файла: lab2.asm

```
TESTPC SEGMENT
    ASSUME CS:TESTPC, DS:TESTPC, ES:NOTHING, SS:NOTHING
    ORG 100H
START: JMP BEGIN
; ДАННЫЕ
Seg_address_un    db    'Segment address of inaccessible memory: ', 0dh,
0ah, '$'
Seg_address_env   db    'Environment segment address: ', 0dh, 0ah, '$'
Tail_cmd          db    'Command line tail: ', 0dh, 0ah, '$'
Env_cont          db    'Environment contains: ', 0dh, 0ah, '$'
path              db    'Path: $'
nextLine          db    0dh, 0ah, '$'
;ПРОЦЕДУРЫ
;-----
PRINT PROC near
    mov ah, 09h
    int 21h
    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
; 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
;-----
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
;-----
BYTE_TO_DEC PROC near
; перевод в 10с/с, SI - адрес поля 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_l
    or AL,30h
    mov [SI],AL
end_l: pop DX
    pop CX
    ret
BYTE_TO_DEC ENDP
;-----
; КОД
BEGIN:
;Type
    mov ax, es:[0002h]
    mov di, OFFSET Seg_adress_un+42; сдвиг по строке
    call WRD_TO_HEX
    mov dx, OFFSET Seg_adress_un
    call PRINT
    mov dx, OFFSET nextLine
    call PRINT

    mov ax, es:[002Ch]
    mov di, OFFSET Seg_adress_env+31
    call WRD_TO_HEX
    mov dx, OFFSET Seg_adress_env
    call PRINT
    mov dx, OFFSET nextLine
    call PRINT

    mov dx, OFFSET Tail_cmd
    call PRINT
    xor cx, cx
    xor bx, bx
    mov cl, byte PTR es:[80h]
    mov bx, 81h
first:
    cmp cx, 0h
    je after_first
    mov dl, byte PTR es:[bx]
    mov ah, 02h
    int 21h
    inc bx
    dec cx
    jmp first; Повторяем
after_first:

```

```

        push es
        mov dx, OFFSET Env_cont
        call PRINT
        mov bx, es:[002Ch]
        mov es, bx
        xor bx, bx

after_second:
        mov dl, byte PTR es:[bx]
        cmp dl, 0h
        je second
        mov ah, 02h
        int 21h
        inc bx
        jmp after_second
second:
        int 21h
        inc bx
        mov dl, byte PTR es:[bx]
        cmp dl, 0h
        je skip
        jmp after_second

skip:
        mov dx, OFFSET nextLine
        call PRINT

        add bx, 3
        mov dx, OFFSET path
        call PRINT

third:
        mov dl, byte PTR es:[bx]
        cmp dl, 0h
        je skip_last
        mov ah, 02h
        int 21h
        inc bx
        jmp third

skip_last:

; Выход в DOS
        xor AL,AL
        mov ah,01h
        int 21h
        mov AH,4Ch
        int 21H
        ret
TESTPC ENDS
        END START ;конец модуля, START - точка входа

```