

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

Бочаров Г.С.

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 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, и написаны следующие функции:

- free_memory – освобождает место в памяти, необходимое для загрузки нашей программы;

- path – формирует полное имя вызываемого модуля.

- load – загружает вызываемый модуль из ЛБ_2.

- str_plus_str — дописывает вторую строку в начало первой

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

```
E:\LAB_6>a.exe

Memory was successfully free
Address of unavailable memory segment: 9FFF
Address of environment segment: 14CD
End of command line:
Contents of environment area:
COMSPEC=Z:\COMMAND.COM
BLASTER=A220 I7 D1 H5 T6
PATH=z:\;c:\bc\bin;c:\vc;c:\arj
Path of loaded module:E:\LAB_6\LB2.COMa

Program completed successfully || Code:  a

E:\LAB_6>S_
```

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. Был выведен символ сердечка, т.к. в DosBox проблемы с обработкой прерывания Ctrl-C.

```
E:\LAB_6>a.exe

Memory was successfully free
Address of unavailable memory segment: 9FFF
Address of environment segment: 14CD
End of command line:
Contents of environment area:
COMSPEC=Z:\COMMAND.COM
BLASTER=A220 I7 D1 H5 T6
PATH=z:\;c:\bc\bin;c:\vc;c:\arj
Path of loaded module:E:\LAB_6\LB2.COM♥

Program completed successfully || Code:  ♥

E:\LAB_6>S_
```

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.

```
E:\LAB_6\SCREENS> E:\LAB_6\la.exe

Memory was successfully free
Address of unavailable memory segment: 9FFF
Address of environment segment: 14CD
End of command line:
Contents of environment area:
COMSPEC=Z:\COMMAND.COM
BLASTER=A220 I7 D1 H5 T6
PATH=z:\;c:\bc\bin;c:\vc;c:\arj
Path of loaded module:E:\LAB_6\LB2.COMq

Program completed successfully !! Code:  q
```

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

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

```
E:\LAB_6>a.exe

memory was successfully free
file not found.

E:\LAB_6>S
1 2main 3View 4Edit 5C
```

Рисунок 4 – Результаты пятого шага

Исходный код программы см. в приложении А.

Ответы на вопросы.

1. Как реализовано прерывание Ctrl-C?

Прерывание 23h обрабатывает нажатие данной комбинации клавиш, управление передается обработчику с адресом 0000:008C, адрес заносится в PSP. Исходное значение адреса восстанавливается при выходе из программы.

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

После функции которая завершает работу программы, в нашей программе после вызова функции 4Ch прерывания int 21h.

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

В точке вызова функции ожидания ввода символа 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: a.asm

```
stack segment stack
        dw 128 dup(?)
stack ends
```

```
data segment
```

```
;???? ??????????
parametr_block dw 0
                dd 0
                dd 0
                dd 0
```

```
file_name db 'lb2.com', 0
```

```
path_ db 128 dup(0)
```

```
cmd_line db 1h, 0dh
```

```
keep_ss dw 0
keep_sp dw 0
keep_psp dw 0
```

```
problem_ind DB 0
```

```
Path_mes db
'Path :', 0dh, 0ah, '$'
```

```
mem_free_mes db 'memory was
successfully free ', 0dh, 0ah, '$'
```

```
mcb_crash_mes db 'mcb was
crashed.', 0dh, 0ah, '$'
```

```
no_memory_mes db 'not enough
memory.', 0dh, 0ah, '$'
```

```
ivalid_address_mes db 'invalid memory
addressess.', 0dh, 0ah, '$'
```

```
ivalid_fiunc_number_mes db 'invalid function
number.', 0dh, 0ah, '$'
```

```

        file_not_found_mes db                'file          not
found.', 0dh, 0ah, '$'

        disk_error_mes db                    'disk      error.',
0dh, 0ah, '$'

        not_enough_mem_mes db                'not          enough
memory to load programm', 0dh, 0ah, '$'

        enviroment_error_mes db              'error   environment.',
0dh, 0ah, '$'

        format_error_mes db                  'error   format.',
0dh, 0ah, '$'

        device_error_mess db                  0dh,      0ah,
'device error.' , 0dh, 0ah, '$'

        end_code db                          0dh,      0ah,
'program completed successfully || code:   ' , 0dh, 0ah, '$'

        end_ctrlc_mes db                     0dh,      0ah,
'program completed with ctrl-c' , 0dh, 0ah, '$'

        end_31h_mess db                      0dh,      0ah,
'program completed with int 31h' , 0dh, 0ah, '$'

        endl_s db                            0dh,0ah,'$'

        data_end db 0

data ends

code segment
    assume cs:code, ds:data, ss:stack

print proc near
    push ax
    mov ah, 09h
    int 21h
    pop ax
    ret
print endp

free_memory proc  near
    push ax
    push bx
    push cx
    push dx

```



```

    mov bx, offset end_address
    mov ax, es
    sub bx, ax
    mov cl, 4
    shr bx, cl          ; byte to par
    mov ah, 4ah
    int 21h

    jnc end_proc

    mov problem_ind, 1

    cmp ax, 7
    je error_crash

    cmp ax, 8
    je error_no_memory

    cmp ax, 9
    je error_address

error_crash:
    mov dx, offset mcb_crash_mes
    call print
    jmp ret_p

error_no_memory:
    mov dx, offset no_memory_mes
    call print
    jmp ret_p

error_address:
    mov dx, offset ivalid_address_mes
    call print
    jmp ret_p

end_proc:
    mov dx, offset endl_s
    call print
    mov dx, offset mem_free_mes
    call print

ret_p:
    pop dx
    pop cx
    pop bx
    pop ax

```

```

    ret
free_memory endp

```

```

str_plus_str proc near
push dx

```

```

add_loop:  ;si - source    di - dest

```

```

    mov dl, byte ptr [si]
    mov byte ptr [di], dl
    inc di
    inc si
    cmp dl, 0

```

```

    jne add_loop

```

```

pop dx
    ret
str_plus_str endp

```

```

path proc near

```

```

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

```

```

    mov ax, keep_psp                ; parse psp
    mov es, ax
    mov es, es:[2ch]                ; segment adress of env
    mov bx, 0

```

```

skip_env:                                ; skip_env

```

```

    cmp byte ptr es:[bx], 0
    je separator_word

```

```

    inc bx

```

```

    jmp skip_env

```

```

separator_word:

```

```

    inc bx

```

```

    cmp byte ptr es:[bx], 0

```

```

        je read_p

        inc bx
        jmp skip_env

read_p:
        add bx, 3
        mov di, 0
read_loop:

        mov dl, es:[bx]
        mov byte ptr [path_ + di], dl
        inc di
        inc bx
        cmp dl, 0
        je create_full_name
        cmp dl, '\\'
        jne read_loop
        mov cx, di
        jmp read_loop

create_full_name:
        mov si, offset file_name
        mov di, offset path_
        add di, cx
        call str_plus_str

        mov dx, offset Path_mes
        call print

        mov si, offset path_
print_path_deb:
        mov dl, byte ptr [si]
        mov ah, 02h
        int 21h
        inc si
        cmp dl, 0
        jne print_path_deb

        mov dx, offset endl_s
        call print

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

```

```

        ret
path endp

load proc near
    push ax
    push bx
    push cx
    push dx
    push ds
    push es

    mov keep_sp, sp
    mov keep_ss, ss

    mov ax, data
    mov es, ax

    mov bx, offset parametr_block
    mov dx, offset cmd_line

    mov [bx+2], dx            ; cmd
    mov [bx+4], ds            ; 1-st fcb

    mov dx, offset path_

    mov ax, 4b00h              ;????? ?????????? ??
    int 21h

    mov ss, keep_ss
    mov sp, keep_sp
    pop es
    pop ds

    jnc load_okey

    cmp ax, 1
    je error_no_func

    cmp ax, 2
    je error_no_file

    cmp ax, 5
    je error_disk

    cmp ax, 8

```

```

    je error_memory

    cmp ax, 10
    je error_enviroment

    cmp ax, 11
    je error_format

error_no_func:
    mov dx, offset ivalid_fiunc_number_mes
    call print

    jmp end_load

error_no_file:
    mov dx, offset file_not_found_mes
    call print
    jmp end_load

error_disk:

    mov dx, offset disk_error_mes
    call print
    jmp end_load

error_memory:

    mov dx, offset not_enough_mem_mes
    call print
    jmp end_load

error_enviroment:

    mov dx, offset enviroment_error_mes
    call print
    jmp end_load

error_format:
    mov dx, offset format_error_mes
    call print
    jmp end_load

load_okey:
    mov ah, 4dh      ; program completion key handling
    mov al, 00h
    int 21h

    cmp ah, 0
    je normal_0

```

```

        cmp ah, 1
        je contrl

        cmp ah, 2
        je error_device

        cmp ah, 3
        je error_device

normal_0:
        push di

        mov di, offset end_code

        mov [di+44], al ;      program completion code

        pop si

        mov dx, offset endl_s

        call print

        mov dx, offset end_code

        call print

        jmp end_load

contrl:
        mov dx, offset end_ctrlc_mes
        call print
        jmp end_load

error_device:
        mov dx, offset device_error_mess
        call print
        jmp end_load

end_interrupt:
        mov dx, offset end_31h_mess
        call print

end_load:
        pop dx
        pop cx
        pop bx
        pop ax
        ret

```

```

load endp

main proc far

    push ds
    xor ax, ax
    push ax
    mov ax, data
    mov ds, ax
    mov keep_psp, es
    call free_memory

    cmp problem_ind, 0
    jne end_main

    call path
    call load

end_main:

    xor al, al
    mov ah, 4ch
    int 21h

main endp

end_address:

code ends

end main

```