

МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МОЭВМ

ОТЧЕТ
по лабораторной работе №6
по дисциплине «Операционные системы»
Тема: Построение модуля динамической структуры

Студент гр. 0382

Крючков А.М.

Преподаватель

Ефремов М.А.

Санкт-Петербург

2022

Постановка задачи

Цель работы.

Исследование возможности построения загрузочного модуля динамической структуры.

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

Шаг 1. Для выполнения лабораторной работы был написан и отлажен программный модуль типа .EXE, который выполняет требуемые по заданию функции.

Шаг 2. Пример работы программы, когда текущим каталогом является каталог с разработанными модулями. Программа вызывает другую программу, которая останавливается, ожидая символ с клавиатуры.



```
C:\>lab6.exe
1) Segment address of inaccessible memory taken from PSP in hexadecimal view: 9F
FFh
2) Environment segment address, safe program, in hexadecimal: 0225h
3) The tail of the command line in symbolic form:
4) The content of the environment area in symbolic form:
PATH=Z:\
COMSPEC=Z:\COMMAND.COM
BLASTER=A220 I7 D1 H5 T6
5) Path of the loaded module: C:\LAB2.COM f
Programm was finished: exit with code: f
```

Рисунок 1 — Пример работы программы, когда целевой модуль лежит в одном каталоге с lab6.exe.

Шаг 3. Была запущена и отлажена программа, когда текущим каталогом является каталог с разработанными модулями. Программа вызывает другую программу, которая останавливается, ожидая символ с клавиатуры. Была введена комбинация символов Ctrl-C. Однако прерывание по такой комбинации

клавиш в DOSBOX нет, поэтому результат работы программы повторяет шаг 2.

```
C:\>lab6.exe
1) Segment address of inaccessible memory taken from PSP in hexadecimal view: 9FFh
2) Environment segment address, safe program, in hexadecimal: 0225h
3) The tail of the command line in symbolic form:
4) The content of the environment area in symbolic form:
PATH=Z:\
COMSPEC=Z:\COMMAND.COM
BLASTER=A220 I7 D1 H5 T6
5) Path of the loaded module: C:\LAB2.COM ♥
Programm was finished: exit with code: ♥
```

Рисунок 2 — Введена комбинация клавиш ctrl+c

Шаг 4. Была запущена программа в другом каталоге.

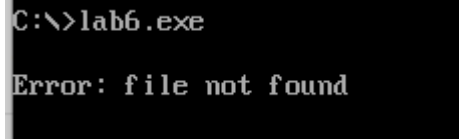
```
C:\ANOTHE~1>LAB6.EXE
1) Segment address of inaccessible memory taken from PSP in hexadecimal view: 9FFh
2) Environment segment address, safe program, in hexadecimal: 0225h
3) The tail of the command line in symbolic form:
4) The content of the environment area in symbolic form:
PATH=Z:\
COMSPEC=Z:\COMMAND.COM
BLASTER=A220 I7 D1 H5 T6
5) Path of the loaded module: C:\ANOTHE~1\LAB2.COM f
Programm was finished: exit with code: f
```

Рисунок 3 — Введён символ f.

```
C:\ANOTHE~1>LAB6.EXE
1) Segment address of inaccessible memory taken from PSP in hexadecimal view: 9FFh
2) Environment segment address, safe program, in hexadecimal: 0225h
3) The tail of the command line in symbolic form:
4) The content of the environment area in symbolic form:
PATH=Z:\
COMSPEC=Z:\COMMAND.COM
BLASTER=A220 I7 D1 H5 T6
5) Path of the loaded module: C:\ANOTHE~1\LAB2.COM ♥
Programm was finished: exit with code: ♥
```

Рисунок 4 — Иллюстрация работы программы (текущий каталог — «другой» каталог, вводимый символ — комбинация Ctrl-C)

Шаг 5. Была запущена и отлажена программа, когда модули находятся в разных каталогах.



```
C:\>lab6.exe  
Error: file not found
```

Рисунок 5 — Модули находятся в разных каталогах

Ответы на контрольные вопросы:

1) Как реализовано прерывание Ctrl-C?

Ответ. Вызывается прерывание `int 23h`. Исходное значение адреса обработчика `Ctrl-Break` восстанавливается из PSP при завершении программы. Таким образом, по завершении порожденного процесса будет восстановлен адрес обработчика `Ctrl-Break` из родительского процесса.

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

Ответ. В точке вызова функции `4ch` прерывания `int 21h`.

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

Ответ. В точке где программа считывает нажатый символ.

Вывод.

Было произведено исследование возможности построения загрузочного модуля динамической структуры.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

lab6.asm:

```
MYSTACK SEGMENT STACK
    DW 128 DUP(?)
MYSTACK ENDS
```

```
DATA SEGMENT
block_param dw 0
com_flag dw 0
com_seg dw 0
```

```
target_file db 'lab2.com', 0h
file_path db 128 DUP(0)
```

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

```
mem_error db 0
error_7_memory_block_destroyed db 'Error. the control memory block
is destroyed', 0DH, 0AH, '$'
error_8_memory_for_execution db 'Error: not enough memory to
execute the function', 0DH, 0AH, '$'
error_9_invalid_ba db 'Error: invalid memory block address', 0DH,
0AH, '$'
```

```
error_1_number_func db 'Error: function number is incorrect', 0DH,
0AH, '$'
error_2_file_not_found db 'Error: file not found', 0DH, 0AH, '$'
error_5_disk_error db 'Disk error', 0DH, 0AH, '$'
error_8_insufficient_mem db 'Error: insufficient memory', 0DH, 0AH,
'$'
error_10_path db 'Error: wrong environment string', 0DH, 0AH, '$'
error_11_wrong_format db 'Error: wrong format', 0DH, 0AH, '$'
```

```
exit_zero db 'Programm was finished: exit with code: ', 0DH, 0AH,
'$'
exit_1 db 'termination by Ctrl-Break', 0DH, 0AH, '$'
exit_2 db 'device error termination', 0DH, 0AH, '$'
exit_3 db 'termination on function 31h leaving the program
resident.', 0DH, 0AH, '$'
```

```
newline    db 0dh,0ah,'$'  
next_com_line db 1h, 0dh
```

```
end_data_seg db 0  
data ends
```

```
code segment  
assume cs:code, ds:data, ss:mystack
```

```
free_mem proc far  
    push ax  
    push bx  
    push cx  
    push dx  
    push es  
  
    and dx, 0  
    mov mem_error, 0h  
  
    mov ax, offset end_data_seg  
    mov bx, offset end_code  
    add ax, bx  
    mov bx, 16  
    div bx  
    add ax, 50h  
    mov bx, ax  
    and ax, 0  
  
    mov ah, 4ah  
    int 21h  
  
    jnc end_free  
    mov mem_error, 1h  
    cmp ax, 7  
    jne check_8  
  
    mov dx, offset error_7_memory_block_destroyed  
    call print  
    jmp end_free  
  
check_8:  
    cmp ax, 8  
    jne check_9  
  
    mov dx, offset error_8_memory_for_execution  
    call print  
    jmp end_free  
  
check_9:  
    cmp ax, 9
```

```

jne end_free

mov dx, offset error_9_invalid_ba
call print
jmp end_free

end_free:
pop es
pop dx
pop cx
pop bx
pop ax
ret

free_mem endp

module_path proc near
    push ax
    push bx
    push bp
    push dx
    push es
    push di

    mov bx, offset file_path
    add di, 3

    find_loop:
    mov dl, es:[di]
    mov [bx], dl
    cmp dl, '.'
    je name_slash
    inc di
    inc bx
    jmp find_loop

    name_slash:
    mov dl, [bx]
    cmp dl, '\'
    je name_current_folder
    mov dl, 0h
    mov [bx], dl
    dec bx
    jmp name_slash

    name_current_folder:
    mov di, offset file_name
    inc bx

    file_name:
    mov dl, [di]
    cmp dl, 0h
    je module_path_end

```



```

    mov [bx], dl
    inc bx
    inc di
    jmp file_name

module_path_end:
    mov [bx], dl
    pop di
    pop es
    pop dx
    pop bp
    pop bx
    pop ax
    ret
module_path endp

get_path proc near
    push ax
    push dx
    push es
    push di

    xor di, di
    mov ax, es:[2ch]
    mov es, ax

    loop2:
    mov dl, es:[di]
    cmp dl, 0
    je end1
    inc di
    jmp loop2

    end1:
    inc di
    mov dl, es:[di]
    cmp dl, 0
    jne loop2

    call module_path

    pop di
    pop es
    pop dx
    pop ax
    ret
get_path endp

load proc far
    push ax
    push bx
    push cx

```

```

push dx
push ds
push es

mov keep_sp, sp
mov keep_ss, ss

call get_path

mov ax, data
mov es, ax
mov bx, offset block_param
mov dx, offset next_com_line
mov com_flag, dx
mov com_seg, ds
mov dx, offset target_file

mov ax, 4b00h
int 21h

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

push dx
mov dx, offset newline
call print
pop dx

jnc success_load

cmp ax, 1
jne check_error_2

mov dx, offset error_1_number_func
call print
jmp load_end

check_error_2:
cmp ax, 2
jne check_error_5
mov dx, offset error_2_file_not_found
call print
jmp load_end

check_error_5:
cmp ax, 5
jne check_error_8
mov dx, offset error_5_disk_error
call print
jmp load_end

```

```
check_error_8:
cmp ax, 8
jne check_error_10
mov dx, offset error_8_isufficient_mem
call print
jmp load_end
```

```
check_error_10:
cmp ax, 10
jne check_error_11
mov dx, offset error_10_path
call print
jmp load_end
```

```
check_error_11:
cmp ax, 11
jne load_end
mov dx, offset error_11_wrong_format
call print
jmp load_end
```

```
success_load:
mov ax, 4d00h
int 21h
```

```
cmp ah, 0
jne ctrl_exit
mov di, offset exit_zero
```

```
add di, 39
mov [di], al
mov dx, offset exit_zero
call print
jmp load_end
```

```
ctrl_exit:
cmp ah, 1
jne exit_error
mov dx, offset exit_1
call print
jmp load_end
```

```
exit_error:
cmp ah, 2
jne exit_int_31h
mov dx, offset exit_2
call print
jmp load_end
```

```
exit_int_31h:
cmp ah, 3
jne load_end
mov dx, offset exit_3
```

```

    call print
    jmp load_end

load_end:
    pop dx
    pop cx
    pop bx
    pop ax
    ret
load endp

print_byte proc near
    push ax

    mov ah, 02h
    int 21h

    pop ax
    ret
print_byte endp

print proc near

    push ax
    mov ah, 09h
    int 21h
    pop ax
    ret

print endp

main proc far
    mov ax, data
    mov ds, ax

    call free_mem

    cmp mem_error, 0h
    jne main_end

    call get_path
    call load

main_end:
    and al, 0

    mov ah, 4ch
    int 21h

main endp

```

```
    end_code:  
    code ends  
  
end main
```