

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. 9383

Лысова А.М.

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 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.

Необходимые теоретические положения:

Для загрузки и выполнения одной программы из другой используется функция 4B00h прерывания int 21h (загрузчик ОС). Перед обращением к этой функции необходимо выполнить следующие действия:

1) Подготовить место в памяти. При начальном запуске программы ей отводится вся доступная в данный момент память OS, поэтому необходимо освободить место в памяти. Для этого можно использовать функцию 4Ah прерывания int 21h. Эта функция позволяет уменьшить отведенный программе блок памяти. Перед вызовом функции надо определить объем памяти, необходимый программе ЛР6 и задать в регистре ВХ число параграфов, которые будут выделяться программе. Если функция 4Ah не может быть

выполнена, то устанавливается флаг переноса CF=1 и в AX заносится код ошибки:

- 7 - разрушен управляющий блок памяти;
- 8 - недостаточно памяти для выполнения функции;
- 9 - неверный адрес блока памяти.

Поэтому после выполнения каждого прерывания int 21h следует проверять флаг переноса CF=1.

2) Создать блок параметров. Блок параметров - это 14-байтовый блок памяти, в который помещается следующая информация:

dw сегментный адрес среды
dd сегмент и смещение командной строки
dd сегмент и смещение первого FCB
dd сегмент и смещение второго FCB

Если сегментный адрес среды 0, то вызываемая программа наследует среду вызывающей программы. В противном случае вызывающая программа должна сформировать область памяти в качестве среды, начинающуюся с адреса кратного 16 и поместить этот адрес в блок параметров.

Командная строка записывается в следующем формате: первый байт - счетчик, содержащий число символов в командной строке, затем сама командная строка, содержащая не более 128 символов.

На блок параметров перед загрузкой вызываемой программы должны указывать ES:BX.

3) Подготовить строку, содержащую путь и имя вызываемой программы. В конце строки должен стоять код ASCII 0. На подготовленную строку должны указывать DS:DX.

4) Сохранить содержимое регистров SS и SP в переменных. При восстановлении SS и SP нужно учитывать, что DS необходимо также восстановить.

Когда вся подготовка выполнена, вызывается загрузчик OS следующей последовательностью команд:

mov AX,4B00h

int 21h

Если вызываемая программа не была загружена, то устанавливается флаг переноса CF=1 и в AX заносится код ошибки:

- 1 - если номер функции неверен;
- 2 - если файл не найден;
- 5 - при ошибке диска;
- 8 - при недостаточном объеме памяти;
- 10 - при неправильной строке среды;
- 11 - если не верен формат.

Если CF=0, то вызываемая программа выполнена и следует обработать ее завершение. Для этого необходимо воспользоваться функцией 4Dh прерывания int 21h. В качестве результата функция возвращает в регистре AH причину, а в регистре AL код завершения.

Причина завершения в регистре AH представляется следующими кодами:

- 0 - нормальное завершение;
- 1 - завершение по Ctrl-Break;
- 2 - завершение по ошибке устройства;
- 3 - завершение по функции 31h, оставляющей программу резидентной.

Код завершения формируется вызываемой программой в регистре AL перед выходом в OS с помощью функции 4Ch прерывания int 21h.

В качестве вызываемой программы целесообразно использовать программу, разработанную в Лабораторной работе №2, модифицировав ее следующим образом. Перед выходом из программы перед выполнением функции 4Ch прерывания int 21h следует запросить с клавиатуры символ и поместить введенный символ в регистр AL, в качестве кода завершения. Это можно сделать с помощью функции 01h прерывания int 21h.

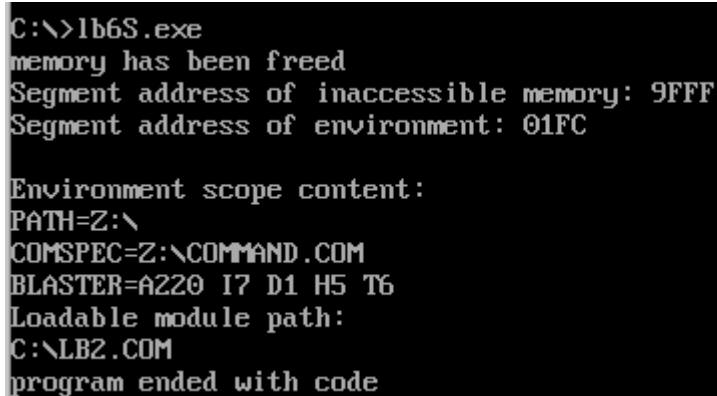
mov AH,01h

int 21h

Введенный символ остается в регистре AL и служит аргументом для функции 4Ch прерывания int 21h.

Выполнение работы:

Был написан и отлажен EXE модуль, выполняющий запуск программы из ЛР2.

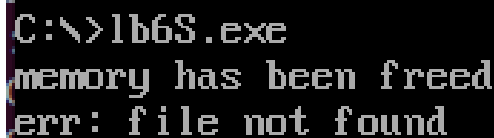


```
C:\>lb6S.exe
memory has been freed
Segment address of inaccessible memory: 9FFF
Segment address of environment: 01FC

Environment scope content:
PATH=Z:\
COMSPEC=Z:\COMMAND.COM
BLASTER=A220 I7 D1 H5 T6
Loadable module path:
C:\LB2.COM
program ended with code
```

Рисунок 1: Запуск программы из той же директории, что и ЛР2.

Но если запустить два модуля из разных директорий, ничего не выполнится.



```
C:\>lb6S.exe
memory has been freed
err: file not found
```

Рисунок 2: Запуск программы из отличной от ЛР2 директории.

Контрольные вопросы:

1) Как реализовано прерывание Ctrl-C?

При нажатии клавиш Ctrl-C управление передается по адресу 0000:008Ch. Этот адрес копируется в PSP функциями 26h и 4Ch, после чего восстанавливается из PSP при выходе из программы.

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

Если код завершения 0, то программа завершается при выполнении функции 4Ch прерывания int 21h.

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

Если во время выполнения программы было нажато Ctrl-C, то программа завершится непосредственно в том месте, в котором произошло нажатие клавиш.

Выводы.

Был построен загрузочный модуль динамической структуры, и получены навыки работы с памятью.

ПРИЛОЖЕНИЕ А ИСХОДНЫЙ КОД ПРОГРАММЫ

Файл: lb6.asm

```
AStack segment stack
    DW 128 dup(?)
AStack ENDS
```

```
DATA SEGMENT
    parameter_block DW 0
                    DD 0
                    DD 0
                    DD 0
    program DB 'lb2.com', 0
    mem_flag DB 0
    cmd_l DB 1h, 0Dh
    cl_pos DB 128 dup(0)
    keep_ss DW 0
    keep_sp DW 0
    keep_psp DW 0

    str_mcb_crash_err DB 'err: mcb crashed', 0dh, 0ah, '$'
    str_no_mem_err DB 'err: there is not enough memory to execute this function', 0dh, 0ah, '$'
    str_addr_err DB 'err: invalid memory address', 0dh, 0ah, '$'
    str_free_mem DB 'memory has been freed' , 0dh, 0ah, '$'

    str_fn_err DB 'err: invalid function number', 0dh, 0ah, '$'
    str_file_error DB 'err: file not found', 0dh, 0ah, '$'
```

```

str_disk_err DB 'err: disk error', 0dh, 0ah, '$'
str_memory_error DB 'err: insufficient memory', 0dh, 0ah, '$'
str_envs_err DB 'err: wrong string of environment ', 0dh, 0ah, '$'
str_format_err DB 'err: wrong format', 0dh, 0ah, '$'

str_norm_fin DB 0dh, 0ah, 'program ended with code  ', 0dh, 0ah, '$'
str_ctrl_end DB 0dh, 0ah, 'program ended by ctrl-break', 0dh, 0ah, '$'
str_device_err DB 0dh, 0ah, 'program ended by device error', 0dh, 0ah, '$'
str_int_end DB 0dh, 0ah, 'program ended by int 31h', 0dh, 0ah, '$'

end_data DB 0
DATA ENDS

CODE SEGMENT

ASSUME CS:CODE, DS:DATA, SS:AStack

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

free_memory proc
    push ax
    push bx
    push cx
    push dx

    mov ax, offset end_data
    mov bx, offset eeend
    add bx, ax

    mov cl, 4
    shr bx, cl
    add bx, 2bh
    mov ah, 4ah
    int 21h

    jnc _endf
    mov mem_flag, 1

mcb_crash:
    cmp ax, 7
    jne not_enought_memory
    mov dx, offset str_mcb_crash_err
    call print_str
    jmp freee
not_enought_memory:
    cmp ax, 8
    jne addr
    mov dx, offset str_no_mem_err
    call print_str
    jmp freee
addr:
    cmp ax, 9
    mov dx, offset str_addr_err
    call print_str
    jmp freee
_endf:
    mov mem_flag, 1
    mov dx, offset str_free_mem
    call print_str

freee:
    pop dx
    pop cx
    pop bx
    pop ax
    ret
free_memory endp

load proc
    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 parameter_block
    mov dx, offset cmd_l
    mov [bx+2], dx
    mov [bx+4], ds
    mov dx, offset cl_pos

    mov ax, 4b00h
    int 21h

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

    jnc loads

fn_err:
    cmp ax, 1
    jne file_err
    mov dx, offset str_fn_err
    call print_str
    jmp load_end

file_err:
    cmp ax, 2
    jne disk_err
    mov dx, offset str_file_error
    call print_str
    jmp load_end

disk_err:
    cmp ax, 5
    jne mem_err
    mov dx, offset str_disk_err
    call print_str
    jmp load_end

mem_err:
    cmp ax, 8
    jne envs_err
    mov dx, offset str_memory_error
    call print_str
    jmp load_end

envs_err:
    cmp ax, 10
    jne format_err
    mov dx, offset str_envs_err
    call print_str
    jmp load_end

format_err:
    cmp ax, 11
    mov dx, offset str_format_err
    call print_str
    jmp load_end

loads:
    mov ah, 4dh
    mov al, 00h
    int 21h

_nend:
    cmp ah, 0
    jne ctrlc
    push di
    mov di, offset str_norm_fin
    mov [di+26], al
    pop si
    mov dx, offset str_norm_fin
    call print_str
    jmp load_end

ctrlc:

```

```

        cmp ah, 1
        jne device
        mov dx, offset str_ctrl_end
        call print_str
        jmp load_end
device:
        cmp ah, 2
        jne int_31h
        mov dx, offset str_device_err
        call print_str
        jmp load_end
int_31h:
        cmp ah, 3
        mov dx, offset str_int_end
        call print_str

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

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

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

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

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

        add bx, 2
        mov di, 0

_loop:
        mov dl, es:[bx]
        mov byte ptr [cl_pos+di], dl
        inc di
        inc bx
        cmp dl, 0
        je _end_loop
        cmp dl, '\'
        jne _loop
        mov cx, di
        jmp _loop
_end_loop:
        mov di, cx
        mov si, 0

_fn:
        mov dl, byte ptr [program+si]
        mov byte ptr [cl_pos+di], dl
        inc di
        inc si
        cmp dl, 0
        jne _fn

        pop es
        pop si
        pop di
        pop dx

```

```

        pop cx
        pop bx
        pop ax
        ret
path 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 mem_flag, 0
    je _end
    call path
    call load
_end:
    xor al, al
    mov ah, 4ch
    int 21h

MAIN    ENDP

eeend:
CODE ENDS
END MAIN

```