

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. 9383

Гладких А.А.

Преподаватель

Ефремов М.А.

Санкт-Петербург

2021

ПОСТАНОВКА ЗАДАЧИ

Цель работы.

Исследовать возможности построения загрузочного модуля динамической структуры. Исследовать интерфейс взаимодействия между вызывающим и вызываемым модулями по управлению и по данным. Написать и отладить приложение, состоящее из нескольких модулей.

Сведения о функциях и структурах данных управляющей программы.

Процедура	Описание
WRITEWRD	Функция печати строки на экран
WRITEBYTE	Функция печати символа на экран
ENDLINE	Функция печати символов переноса строки
FREE_UNUSED_MEMORY	Функция очистки выделенную под программу память
LOAD_MODULE	Функция загрузки вызываемого модуля
GET_PATH	Функция получения пути до каталога с вызывающим модулем
PARSE_PATH	Функция получения пути до вызываемого модуля
MAIN	Главная функция программы

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.

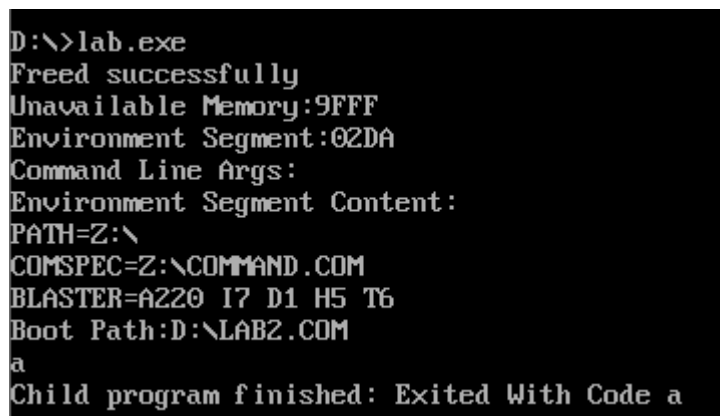
Исходный код.

Исходный код представлен в приложении А.

РЕЗУЛЬТАТЫ ИССЛЕДОВАНИЯ ПРОБЛЕМ

Шаг 1. Был написан и отлажен программный модуль типа .EXE, который выполняет поставленные в задании функции.

Шаг 2. Программа была запущена из каталога с разработанными модулями. Программа успешно вызывала .COM модуль. Вспомогательному модулю был введен символ «а». Причина завершения .EXE-модуля и код завершения отражены на рисунке 1.



```
D:\>lab.exe
Freed successfully
Unavailable Memory:9FFF
Environment Segment:02DA
Command Line Args:
Environment Segment Content:
PATH=Z:\
COMSPEC=Z:\COMMAND.COM
BLASTER=A220 I7 D1 H5 T6
Boot Path:D:\LAB2.COM
a
Child program finished: Exited With Code a
```

Рисунок 1 - Иллюстрация работы программы при вводе вызываемому модулю символа «а»

Шаг 3. Программа была запущена из каталога с разработанными модулями. После вызова .COM-модуля была введена комбинация Ctrl-C. Причина завершения .EXE-модуля отражена на рисунке 2. Так как в DOSBox не реализовано прерывание по Ctrl+C, программа считает комбинацию клавиш за символ сердца.



```
D:\>lab.exe
Freed successfully
Unavailable Memory:9FFF
Environment Segment:02DA
Command Line Args:
Environment Segment Content:
PATH=Z:\
COMSPEC=Z:\COMMAND.COM
BLASTER=A220 I7 D1 H5 T6
Boot Path:D:\LAB2.COM
♥
Child program finished: Exited With Code ♥
```

Рисунок 2 - Иллюстрация работы программы при вводе вызываемому модулю комбинации Ctrl+C

Шаг 4. Программа была запущена дважды из другого каталога. Сперва .EXE-модулю был введен символ «a», а затем комбинация клавиш Ctrl+C. Причины завершения .EXE-модуля отражены на рисунках 3-4.

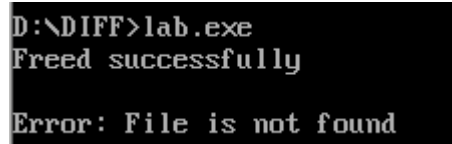
```
D:\DIFF>..\lab.exe
Freed successfully
Unavailable Memory:9FFF
Environment Segment:02DA
Command Line Args:
Environment Segment Content:
PATH=Z:\
COMSPEC=Z:\COMMAND.COM
BLASTER=A220 I7 D1 H5 T6
Boot Path:D:\LAB2.COM
a
Child program finished: Exited With Code a
```

Рисунок 3 - Иллюстрация работы программы, запущенной из другого каталога, при вводе вызываемому модулю символа «a»

```
D:\DIFF>..\lab.exe
Freed successfully
Unavailable Memory:9FFF
Environment Segment:02DA
Command Line Args:
Environment Segment Content:
PATH=Z:\
COMSPEC=Z:\COMMAND.COM
BLASTER=A220 I7 D1 H5 T6
Boot Path:D:\LAB2.COM
♥
Child program finished: Exited With Code ♥
```

Рисунок 4 - Иллюстрация работы программы, запущенной из другого каталога, при вводе вызываемому модулю комбинации клавиш Ctrl+C

Шаг 5. Программа была запущена из каталога, в котором содержался лишь .EXE-модуль. Вывод программы представлен на рисунке 5.



```
D:\DIFF>lab.exe
Freed successfully

Error: File is not found
```

Рисунок 5 - Иллюстрация работы программы, запущенной из каталога, в котором лежит лишь .EXE-модуль

Шаг 6. Была произведена оценка результатов и были отвечены контрольные вопросы:

1. Как реализовано прерывание Ctrl-C?

Ответ: при нажатии комбинации клавиш Ctrl+C срабатывает прерывание int 23h – управление передается по адресу 0000:008C. Этот адрес копируется в поле PSP функциями DOS 26h, которая создает PSP, и 4Ch. Исходное значение адреса обработчика Ctrl+C восстанавливается из PSP при завершении программы.

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

Ответ: в таком случае программа завершится при обработке прерывания 4ch int21h.

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

Ответ: программа завершится в той точке, где была считана комбинация клавиш Ctrl+C.

Выводы.

Были исследованы возможности построения загрузочного модуля динамической структуры и интерфейс взаимодействия между вызывающим и вызываемым модулями по управлению и по данным. Было написано и отлажено приложение, состоящее из нескольких модулей. Программа была запущена в различных ситуациях.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: lab.asm

```
ASTACK SEGMENT STACK
    DW 256 DUP(?)
ASTACK ENDS

DATA SEGMENT
    exec_param_block dw 0
    cmd_off dw 0 ; сегмент командной строки
    cmd_seg dw 0 ; смещение командной строки
    fcb1 dd 0 ; сегмент и смещение первого FCB
    fcb2 dd 0 ; сегмент и смещение второго FCB

    child_cmd_line db 1h, 0dh
    second_module_name db 'lab2.com', 0h
    second_module_path db 128 DUP(0)

    keep_ss dw 0
    keep_sp dw 0

    error_mem_free db 0
    mcb_crash_string db 'Error: Memory Control Block has crashed', 0DH,
0AH, '$'
    not_enough_memory_string db 'Error: Not Enough Memory', 0DH, 0AH, '$'
    wrong_address_string db 'Error: Wrong Address', 0DH, 0AH, '$'
    free_without_error_string db 'Freed successfully', 0DH, 0AH, '$'

    child_error_function_number db 'Error: Function number is incorrect',
0DH, 0AH, '$'
    child_error_file_not_found db 'Error: File is not found', 0DH, 0AH,
'$'
    child_error_disk_error db 'Error: Something wrong with the disk', 0DH,
0AH, '$'
    child_error_not_enough_mem db 'Error: Not enough memory', 0DH, 0AH,
'$'
    child_error_path_string db 'Error: Something wrong with the path
param', 0DH, 0AH, '$'
```



```

child_error_wrong_format db 'Error: Wrong Format', 0DH, 0AH, '$'

child_std_exit db 'Child program finished: Exited With Code ', 0DH,
0AH, '$'
child_ctrl_exit db 'Child program finished: Ctrl+Break Exit', 0DH,
0AH, '$'
child_device_error_exit db 'Child program finished: Device Error
Exit', 0DH, 0AH, '$'
child_int31h_exit db 'Child program finished: became resident, int 31h
Exit', 0DH, 0AH, '$'

```

```

data_end db 0
DATA ENDS

```

```

CODE SEGMENT

```

```

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

```

```

WRITEWRD PROC NEAR

```

```

    push ax
    mov ah, 9
    int 21h
    pop ax
    ret

```

```

WRITEWRD ENDP

```

```

WRITEBYTE PROC NEAR

```

```

    push ax
    mov ah, 02h
    int 21h
    pop ax
    ret

```

```

WRITEBYTE ENDP

```

```

ENDLINE PROC NEAR

```

```

    push ax
    push dx

    mov dl, 0dh
    call WRITEBYTE

    mov dl, 0ah

```

```

        call WRITEBYTE

        pop dx
        pop ax
        ret
ENDLINE ENDP

FREE_UNUSED_MEMORY PROC FAR
        push ax
        push bx
        push cx
        push dx
        push es

        xor dx, dx

        mov error_mem_free, 0h

        mov ax, offset data_end
        mov bx, offset lafin
        add ax, bx

        mov bx, 10h
        div bx

        add ax, 100h

        mov bx, ax

        xor ax, ax

        mov ah, 4ah
        int 21h

        jnc free_without_error

        mov error_mem_free, 1h

;mcb crash
        cmp ax, 7
        jne not_enough_memory

```

```

        mov dx, offset mcb_crash_string
        call WRITEWRD
        jmp free_unused_memory_end

not_enough_memory:
        cmp ax, 8
        jne wrong_address

        mov dx, offset not_enough_memory_string
        call WRITEWRD
        jmp free_unused_memory_end

wrong_address:
        cmp ax, 9
        jne free_unused_memory_end

        mov dx, offset wrong_address_string
        call WRITEWRD
        jmp free_unused_memory_end

free_without_error:
        mov dx, offset free_without_error_string
        call WRITEWRD

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

FREE_UNUSED_MEMORY ENDP

LOAD_MODULE 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 exec_param_block

    mov dx, offset child_cmd_line
    mov cmd_off, dx
    mov cmd_seg, ds

    mov dx, offset second_module_path

    mov ax, 4b00h
    int 21h

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

    call ENDLINE

    jnc loaded_successfully

;function number error
    cmp ax, 1
    jne load_file_not_found
    mov dx, offset child_error_function_number
    call WRITEWRD
    jmp load_module_end

load_file_not_found:
    cmp ax, 2
    jne load_disk_error
    mov dx, offset child_error_file_not_found
    call WRITEWRD
    jmp load_module_end

```

```

load_disk_error:
    cmp ax, 5
jne load_not_enough_memory
mov dx, offset child_error_disk_error
call WRITEWRD
jmp load_module_end

load_not_enough_memory:
    cmp ax, 8
jne load_path_error
mov dx, offset child_error_disk_error
call WRITEWRD
jmp load_module_end

load_path_error:
    cmp ax, 10
jne load_wrong_format
mov dx, offset child_error_path_string
call WRITEWRD
jmp load_module_end
load_wrong_format:
    cmp ax, 11
jne load_module_end
mov dx, offset child_error_wrong_format
call WRITEWRD
jmp load_module_end

loaded_successfully:
    mov ax, 4d00h
int 21h

;std_exit
    cmp ah, 0
jne ctrl_exit
mov di, offset child_std_exit

    add di, 41
    mov [di], al
    mov dx, offset child_std_exit
call WRITEWRD
jmp load_module_end

```

```

ctrl_exit:
    cmp ah, 1
jne device_error_exit
mov dx, offset child_ctrl_exit
call WRITEWRD
jmp load_module_end

device_error_exit:
    cmp ah, 2
jne int31h_exit
mov dx, offset child_device_error_exit
call WRITEWRD
jmp load_module_end

int31h_exit:
    cmp ah, 3
jne load_module_end
mov dx, offset child_int31h_exit
call WRITEWRD
jmp load_module_end

load_module_end:

    pop dx
    pop cx
    pop bx
    pop ax
    ret
LOAD_MODULE ENDP

GET_PATH PROC NEAR

    push ax
    push dx
    push es
    push di

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

```

```

content_loop:
    mov dl, es:[di]
    cmp dl, 0
    je end_string2

    inc di
    jmp content_loop

end_string2:

    inc di

    mov dl, es:[di]
    cmp dl, 0
    jne content_loop

    call PARSE_PATH

    pop di
    pop es
    pop dx
    pop ax
    ret

GET_PATH ENDP

PARSE_PATH PROC NEAR

    push ax
    push bx
    push bp
    push dx
    push es
    push di

    mov bx, offset second_module_path

    add di, 3

boot_loop:
    mov dl, es:[di]
    mov [bx], dl

```

```

    cmp dl, '.'
    je parse_to_slash

    inc di
    inc bx

    jmp boot_loop

parse_to_slash:
    mov dl, [bx]
    cmp dl, '\'
    je get_second_module_name
    mov dl, 0h
    mov [bx], dl

    dec bx
    jmp parse_to_slash

get_second_module_name:
    mov di, offset second_module_name
    inc bx

add_second_module_name:
    mov dl, [di]
    cmp dl, 0h
    je parse_path_end

    mov [bx], dl

    inc bx
    inc di

    jmp add_second_module_name

parse_path_end:
    mov [bx], dl

    pop di
    pop es
    pop dx
    pop bp
    pop bx

```



```

        pop ax
        ret

PARSE_PATH ENDP

MAIN PROC FAR
    mov ax, data
    mov ds, ax

    call FREE_UNUSED_MEMORY

    cmp error_mem_free, 0h
    jne main_end

    call GET_PATH
    call LOAD_MODULE

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

MAIN ENDP

labin:
CODE ENDS

END MAIN

```