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. 9381 _____ Любимов В.А.

Преподаватель _____ Ефремов М.А.

Санкт-Петербург

2021

Цель работы

Исследование возможности построения загрузочного модуля динамической структуры. В отличии 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.

Описание функций и структуры данных

1. PRINT_MES – при помощи функции 9h из прерывания 21h выводит строку на экран.
2. FREE_MEM_FOR_MODULE – освобождает память для запуска вызываемого модуля. Обрабатывает возможные ошибки.
3. LOAD_MODULE – загружает выполняемый модуль. Обрабатывает возможные ошибки.
4. PATH_MAKING – создаёт путь для выполняемого модуля.

Ход выполнения работы

1. Запускаем программу из каталога с разработанными модулями. При нажатии клавиши в диапазоне A-Z программа выводит нажатую клавишу, как код корректного завершения.

```
F:\>lb6
Memory for calling module has been cleared!
Segment address of restricted memory: 9FFF
Segment address of the environment: 01FF
Tail contenet:
Content of the environment:
        PATH=Z:\
        COMSPEC=Z:\COMMAND.COM
        BLASTER=A220 I7 D1 H5 T6
Path to the launched module: F:\LB2.COM
h
Program has been finished successfully with code: h
```

2. Запускаем программу из каталога с разработанными модулями. При нажатии комбинации Ctrl+C программа корректно завершила работу.

```
F:\>lb6
Memory for calling module has been cleared!
Segment address of restricted memory: 9FFF
Segment address of the environment: 01FF
Tail contenet:
Content of the environment:
        PATH=Z:\
        COMSPEC=Z:\COMMAND.COM
        BLASTER=A220 I7 D1 H5 T6
Path to the launched module: F:\LB2.COM
♥
Program has been finished successfully with code: ♥
```

3. Запускаем программу из каталога отличного от каталога с разработанными модулями. При нажатии клавиши в диапазоне A-Z программа выводит нажатую клавишу, как код корректного завершения.

```
F:\TEST>lb6
Memory for calling module has been cleared!
Segment address of restricted memory: 9FFF
Segment address of the environment: 01FF
Tail contenet:
Content of the environment:
        PATH=Z:\
        COMSPEC=Z:\COMMAND.COM
        BLASTER=A220 I7 D1 H5 T6
Path to the launched module: F:\TEST\LB2.COM
j
Program has been finished successfully with code: j
```

4. Запускаем программу из каталога отличного от каталога с разработанными модулями. При нажатии комбинации Ctrl+C программа корректно завершила работу.

```
F:\TEST>lb6
Memory for calling module has been cleared!
Segment address of restricted memory: 9FFF
Segment address of the environment: 01FF
Tail contenet:
Content of the environment:
        PATH=Z:\
        COMSPEC=Z:\COMMAND.COM
        BLASTER=A220 I7 D1 H5 T6
Path to the launched module: F:\TEST\LB2.COM
♥
Program has been finished successfully with code: ♥
```

5. Запускаем программу, когда разработанные модули находятся в различных каталогах.

```
F:\TEST>lb6
Memory for calling module has been cleared!
File has not been found!
```

Ответы на контрольные вопросы

1. Как реализовано прерывание Ctrl-C?

Когда нажимается соответствующая комбинация клавиш, управление передаётся по адресу 0000:008ch. Затем при помощи функций DOS 26h и 4Ch данный адрес помещается в PSP. При завершении программы этот адрес восстанавливается из PSP.

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

В момент выполнения функции 4ch прерывания 21h.

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

В месте, где было вызвано данное прерывания. В этой программе при вызове функции 01h прерывания 21h.

Вывод

В ходе выполнения работы была изучена возможность построения модуля динамической структуры.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

lab6.asm

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

DATA SEGMENT
       mem_free_mes db 'Memory for calling module has been cleared!' , 0dh, 0ah, '$'
    mcb_error_mes db 'MCB has been crashed!', 0dh, 0ah, '$'
       no_memory_mes db 'Not enough memory!', 0dh, 0ah, '$'
       mem_adress_err_mes db 'Invalid memory address!', 0dh, 0ah, '$'
       err_func_num_mes db 'Invalid function number!', 0dh, 0ah, '$'
       no_file_mes db 'File has not been found!', 0dh, 0ah, '$'
       disk_err_mes db 'Disk error!', 0dh, 0ah, '$'
       memory_err_mes db 'Inadequate amount of memory!', 0dh, 0ah, '$'
       envir_err_mes db 'Error with environment!', 0dh, 0ah, '$'
       format_err_mes db 'Wrong format!', 0dh, 0ah, '$'
       device_err_mes db 0dh, 0ah, 'Error with computer!' , 0dh, 0ah, '$'

       exit_code_mes db 0dh, 0ah, 'Program has been finished successfully with code:    '
, 0dh, 0ah, '$'
       ctrl_exit_mes db 0dh, 0ah, 'Program has been finished with ctrl-break
combination!' , 0dh, 0ah, '$'
       inter_exit_mes db 0dh, 0ah, 'Program has been finished because of function 31h!' ,
0dh, 0ah, '$'

       calling_module_name db 'lb2.com', 0
       flag db 0
       cmd db 1h, 0dh
       pos db 128 DUP(0)
       keep_ss DW 0
       keep_sp DW 0
       psp_state DW 0
       block_of_parametres DW 0
               dd 0
               dd 0
               dd 0

       data_end db 0
DATA ENDS

CODE SEGMENT

assume cs:CODE, ds:DATA, ss:STACK

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

FREE_MEM_FOR_MODULE PROC NEAR
       push ax
       push bx
       push cx
       push dx

       mov ax, offset data_end
```

```
        mov bx, offset ENDProgram
        add bx, ax

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

        jnc end_free
        mov flag, 0

        cmp ax, 7
        je mcb_err

        cmp ax, 8
        je mem_limit

        cmp ax, 9
        je adress_err

mcb_err:
        mov dx, offset mcb_error_mes
        call PRINT_MES
        jmp end_free_func

mem_limit:
        mov dx, offset no_memory_mes
        call PRINT_MES
        jmp end_free_func

adress_err:
        mov dx, offset mem_adress_err_mes
        call PRINT_MES
        jmp end_free_func

end_free:
        mov flag, 1
        mov dx, offset mem_free_mes
        call PRINT_MES

end_free_func:
        pop dx
        pop cx
        pop bx
        pop ax
        ret
FREE_MEM_FOR_MODULE ENDP

PATH_MAKING PROC NEAR
        push ax
        push bx
        push cx
        push dx
        push di
        push si
        push es

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

path_finding:
        inc bx
```

```
        cmp byte ptr es:[bx - 1], 0
        jne path_finding

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

        add bx, 2
        mov di, 0

loop_finding:
        mov dl, es:[bx]
        mov byte ptr [pos + di], dl
        inc di
        inc bx
        cmp dl, 0
        je end_loop_finding
        cmp dl, '\'
        jne loop_finding
        mov cx, di
        jmp loop_finding

end_loop_finding:
        mov di, cx
        mov si, 0

end_path_making:
        mov dl, byte ptr [calling_module_name + si]
        mov byte ptr [pos + di], dl
        inc di
        inc si
        cmp dl, 0
        jne end_path_making

        pop es
        pop si
        pop di
        pop dx
        pop cx
        pop bx
        pop ax
        ret
PATH_MAKING ENDP

LOAD_MODULE 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 block_of_parametres
        mov dx, offset cmd
        mov [bx + 2], dx
        mov [bx + 4], ds
        mov dx, offset pos

        mov ax, 4b00h
        int 21h
```

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

        jnc start_load

        cmp ax, 1
        je func_num_err

        cmp ax, 2
        je file_err

        cmp ax, 5
        je disk_err

        cmp ax, 8
        je mem_err

        cmp ax, 10
        je envir_err

        cmp ax, 11
        je format_err

func_num_err:
        mov dx, offset err_func_num_mes
        call PRINT_MES
        jmp end_load

file_err:
        mov dx, offset no_file_mes
        call PRINT_MES
        jmp end_load

disk_err:
        mov dx, offset disk_err_mes
        call PRINT_MES
        jmp end_load

mem_err:
        mov dx, offset memory_err_mes
        call PRINT_MES
        jmp end_load

envir_err:
        mov dx, offset envir_err_mes
        call PRINT_MES
        jmp end_load

format_err:
        mov dx, offset format_err_mes
        call PRINT_MES
        jmp end_load

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

        cmp ah, 0
        je usual_exit

        cmp ah, 1
        je ctrl_exit
```

```
        cmp ah, 2
        je device_err

        cmp ah, 3
        je inter_exit

usual_exit:
        push di
        mov di, offset exit_code_mes
        mov [di + 52], al
        pop si
        mov dx, offset exit_code_mes
        call PRINT_MES
        jmp end_load

ctrl_exit:
        mov dx, offset ctrl_exit_mes
        call PRINT_MES
        jmp end_load

device_err:
        mov dx, offset device_err_mes
        call PRINT_MES
        jmp end_load

inter_exit:
        mov dx, offset inter_exit_mes
        call PRINT_MES

end_load:
        pop dx
        pop cx
        pop bx
        pop ax
        ret
LOAD_MODULE ENDP

MAIN_PROC PROC FAR
        push ds
        xor ax, ax
        push ax

        mov ax, DATA
        mov ds, ax
        mov psp_state, es

        call FREE_MEM_FOR_MODULE
        cmp flag, 0
        je end_main
        call PATH_MAKING
        call LOAD_MODULE

end_main:
        xor al, al
        mov ah, 4ch
        int 21h
MAIN_PROC ENDP

ENDProgram:
CODE ENDS
END MAIN_PROC
```