

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

Студентка гр. 7381

Кушкеева А.О.

Преподаватель

Ефремов М.А.

Санкт-Петербург

2019

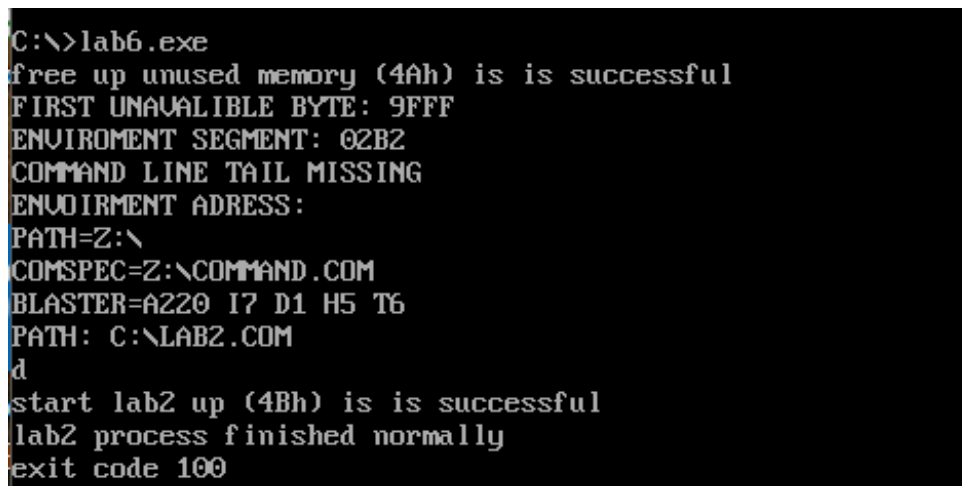
Цель работы

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.

Ход работы

1. Запустим отлаженную программу, когда текущим каталогом является каталог с разработанными модулями:



```
C:\>lab6.exe
free up unused memory (4Ah) is is successful
FIRST UNAVAILABLE BYTE: 9FFF
ENVIRONMENT SEGMENT: 02B2
COMMAND LINE TAIL MISSING
ENVIRONMENT ADDRESS:
PATH=Z:\
COMSPEC=Z:\COMMAND.COM
BLASTER=A220 I7 D1 H5 T6
PATH: C:\LAB2.COM
d
start lab2 up (4Bh) is is successful
lab2 process finished normally
exit code 100
```

Рис.1 Запуск программы lab6.exe

Программа вызывает другую программу (LAB2.COM), которая останавливается, ожидая символ с клавиатуры.

2. Запустим lab6.exe и введем комбинацию Ctrl-C. Программа и модуль в одной папке.

```
C:\>lab6.exe
free up unused memory (4Ah) is is successful
FIRST UNAVAILABLE BYTE: 9FFF
ENVIRONMENT SEGMENT: 02B2
COMMAND LINE TAIL MISSING
ENVIRONMENT ADDRESS:
PATH=Z:\
COMSPEC=Z:\COMMAND.COM
BLASTER=A220 I7 D1 H5 T6
PATH: C:\LAB2.COM
♥
start lab2 up (4Bh) is is successful
lab2 process finished normally
exit code 3
```

Рис.2 Запуск программы lab6.exe с Ctrl-C

Ожидалось, что при нажатии клавиш Ctrl-C программа остановится, но в Windows 2000 и более поздних версиях Ctrl-Break не поддерживается.

3. Запуск программы, когда текущим каталогом является какой-либо другой каталог, отличный от того в котором содержатся разработанные программные модули.

```
C:\>lab6.exe
free up unused memory (4Ah) is is successful
start lab2 up (4Bh) failed: file not found
```

Рис.3 Запуск программы из другого каталога.

Ответы на контрольные вопросы.

1. Как реализовано прерывание Ctrl-C?

Прерывание 23h вызывается, если была нажата комбинация клавиш Ctrl-C или Ctrl-Break. Адрес, по которому передается управление (0000:008c). Управление передаётся тогда, когда DOS распознает, что пользователь нажал Ctrl-Break или Ctrl-C. Адрес по вектору INT 23h

копируется в поле PSP Ctrl-Break Address функциями DOS 26h (создать PSP) и 4Ch (EXEC).

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?

Если код причины завершения 0, то вызываемая программа заканчивается в месте вызова функции 4Ch прерываний int 21h.

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

В месте, где программа ожидала ввода символа: в точке вызова функции 01h прерывания int 21h.

Приложение А: Код программы lab6.asm

```
AStack SEGMENT STACK
    dw 64 dup (?)
AStack ENDS

;-----
DATA SEGMENT

    KEEP_SS          dw 0
    KEEP_SP          dw 0

    parameter_block  dw 0    ; segment address of environment
                        dd 0    ; segment and offset of command line
                        dd 0    ; segment and offset of the first FCB
                        dd 0    ; segment and offset of the second FCB

    path             db '0000000000000000', 0
    called_modul     db 'lab2.com', 0
    func_4Ah         db 'free up unused memory (4Ah) ', '$'
    func_4Bh         db 'start lab2 up (4Bh) ', '$'

    error_1          db 'failed: invalid function number', 0dh, 0ah, '$'
    error_2          db 'failed: file not found', 0dh, 0ah, '$'
    error_5          db 'failed: disc error', 0dh, 0ah, '$'
    error_7          db 'failed: MCB destroyed (error code 7)', 0dh, 0ah, '$'
    error_8          db 'failed: not enough memory (error code 8)', 0dh, 0ah, '$'
    error_9          db 'failed: invalid memory block address (error code 9)',
0dh, 0ah, '$'

    error_A          db 'failed: invalid environment string', 0dh, 0ah, '$'
    error_B          db 'failed: invalid format', 0dh, 0ah, '$'
    error_other      db 'failed: unknown error', 0dh, 0ah, '$'
    error_none       db 'is is successful', 0dh, 0ah, '$'
    exit_reason_0    db 'lab2 process finished normally', 0dh, 0ah, '$'
    exit_reason_1    db 'lab2 process finished with ctrl-break', 0dh, 0ah, '$'
    exit_reason_2    db 'lab2 process finished with device error', 0dh, 0ah, '$'
    exit_reason_3    db 'lab2 process finished with int 31h', 0dh, 0ah, '$'
    exit_code        db 'exit code ', '$'

DATA ENDS

;-----
CODE SEGMENT

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

;-----
    ; define 2 macroses for ease:
    push_registers macro
        push AX
        push BX
```

```
        push CX
        push DX

        push ds
        push es
        push di
        push si
    endm
```

```
pop_registers macro
    pop si
    pop di
    pop es
    pop ds

    pop DX
    pop CX
    pop BX
    pop AX
endm
```

```
;-----
```

```
BYTE_TO_DEC PROC near
    push CX
    push DX
    xor AH,AH
    xor DX,DX
    mov CX,10
loop_bd:
    div CX
    or DL,30h
    mov [SI],DL
    dec SI
    xor DX,DX
    cmp AX,10
    jae loop_bd
    cmp AL,00h
    je end_1
    or AL,30h
    mov [SI],AL
end_1:
    pop DX
    pop CX
    ret
BYTE_TO_DEC ENDP
```

```

OUTPUT_PROC PROC NEAR ;Вывод на экран сообщения
    push ax
    mov ah, 09h
    int 21h
    pop ax
    ret
OUTPUT_PROC ENDP

free_mem proc near
    push_registers

    mov ax, offset END_OF_PROGRAMM        ; get program size, bytes
    mov bx, ax
    and bx, 0Fh
    cmp bx, 0h
    je size_multiple_of_10h                ; if round up is not
needed
    add ax, 0fh                            ; for round up
size_multiple_of_10h:
    mov bl, 10h
    div bl                                ; get program
size, paragraphs
    mov bl, al
    mov bh, 0h                            ; bl = al = ax /
bl ; bh = 0
    mov ah, 4ah
    add bx, 100h                            ; for psp
    int 21h

    pop_registers
    ret
free_mem endp

error_handler proc near
    PUSHF
    push_registers
    jnc exit_error_handler                ; CF = 1
<=> 4ah failed

    check_01h:
        cmp ax, 01h
        jne check_02h
        lea dx, error_1
        jmp error_message
    check_02h:

```

```

        cmp ax, 02h
        jne check_05h
        lea dx, error_2
        jmp error_message
check_05h:
        cmp ax, 05h
        jne check_07h
        lea dx, error_5
        jmp error_message
check_07h:
        cmp ax, 07h
        jne check_08h
        lea dx, error_7
        jmp error_message
check_08h:
        cmp ax, 08h
        jne check_09h
        lea dx, error_8
        jmp error_message
check_09h:
        cmp ax, 09h
        jne check_Ah
        lea dx, error_9
        jmp error_message
check_Ah:
        cmp ax, 0Ah
        jne check_Bh
        lea dx, error_A
        jmp error_message
check_Bh:
        cmp ax, 0Bh
        jne other_error
        lea dx, error_A
        jmp error_message

other_error:
        lea dx, error_other
        jmp error_message

error_message:
        call OUTPUT_PROC
        mov ah, 4Ch
        int 21h

exit_error_handler:
        lea dx, error_none

```



```

        call OUTPUT_PROC
        pop_registers
        POPF                                ; recover flags state
        ret
error_handler endp

```

```

strcpy proc near ; copy string from ds:si to es:di
        ; not saving registers di, si
        push ax
strcpy_loop:
        mov al, ds:[si]
        mov es:[di], al
        inc si
        inc di
        cmp al, 00h                        ; cmp goes after copying so 0 is copied too
        je exit_strcpy
        jmp strcpy_loop
exit_strcpy:
        pop ax
        ret
strcpy endp

```

```

create_parameter_block proc near
        mov parameter_block + 0, 0          ; called program inherit
callers environment
        mov parameter_block + 2, es
        mov parameter_block + 4, 80h        ; command line
        mov parameter_block + 6, es
        mov parameter_block + 8, 5Ch        ; first FCB
        mov parameter_block + 10, es
        mov parameter_block + 12, 80h       ; second FCB
        ret
create_parameter_block endp

```

```

prepare_path proc near
        push_registers
        ; find first byte of path:
        mov es, es:[2Ch]
        xor si, si
path_loop:
        mov al, es:[si]
        inc si
        cmp al, 0
        jne path_loop                      ; one zero in a row
        mov al, es:[si]
        inc si

```

```

        cmp al, 0
        jne path_loop          ; one zero in a row
        add si, 2              ; skip two bytes after env. and before

; prerare segments for strcpy:
        push ds
        mov ax, es
        mov ds, ax            ; ds:si = env. var. seg. : path start offset
        pop es
        push es
        lea di, path          ; es:di = data seg. : string named path offset
        call strcpy            ; copy string from ds:si to es:di
        pop ds                ; recover ds = data segment

; find the last slash if the path:
slash_loop:
        dec di
        mov al, es:[di]
        cmp al, '\'
        jne slash_loop
        inc di

; prerare segments for strcpy:
        lea si, called_modul ; ds:si = called_modul (seg : offset)
        mov ax, ds
        mov es, ax
        call strcpy            ; copy string from ds:si to es:di
        mov es:[di], al
        pop_registers
        ret

prepare_path endp

start_me_up proc near
        mov dx, offset path
        mov ax, ds
        mov es, ax
        mov bx, offset parameter_block
        mov KEEP_SS, SS
        mov KEEP_SP, SP
        mov AX, 4B00h
        int 21h
        mov SS, KEEP_SS
        mov SP, KEEP_SP
        ret

start_me_up endp

check_exit_code proc near

```

```

        mov ah, 4Dh
        int 21h                ; get exit code of the last process
check_exit_reason_0:
        cmp ah, 00h
        jne check_exit_reason_1
        lea dx, exit_reason_0
        call OUTPUT_PROC
        lea si, exit_code
        add si, 0Ch
        call BYTE_TO_DEC
        mov dx, offset exit_code
        call OUTPUT_PROC
        ret
check_exit_reason_1:
        cmp ah, 01h
        jne check_exit_reason_2
        lea dx, exit_reason_1
        jmp exit_message
check_exit_reason_2:
        cmp ah, 02h
        jne check_exit_reason_3
        lea dx, exit_reason_2
        jmp exit_message
check_exit_reason_3:
        cmp ah, 02h
        jne print_exit_code
        lea dx, exit_reason_3
        jmp exit_message
exit_message:
        call OUTPUT_PROC

print_exit_code:
        ret
check_exit_code endp

MAIN PROC FAR
        mov AX, seg DATA
        mov DS, AX
        call free_mem
        lea dx, func_4Ah
        call OUTPUT_PROC
        call error_handler
        call create_parameter_block
        call prepare_path
        call start_me_up
        lea dx, func_4Bh

```

```
        call OUTPUT_PROC
        call error_handler
        jc exit_main
        call check_exit_code
exit_main:
        mov ah, 4Ch
        int 21h
        ret
MAIN ENDP

END_OF_PROGRAMM:
CODE ENDS

END MAIN
```