**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 |  | Кушкоева А.О. |
| Преподаватель |  | Ефремов М.А. |

Санкт-Петербург

2019

**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 4В00h 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:

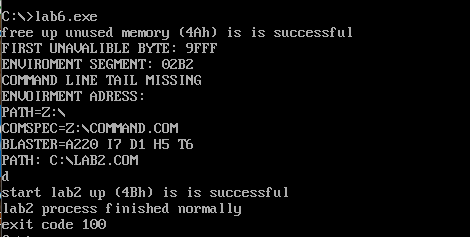


Рис.1 Запуск программы lab6.exe

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

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



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

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

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



Рис.3 Запуск программы из другого каталога.

**Ответы на контрольные вопросы.**

1. **Как реализовано прерывание Ctrl-C?**

Прерывание 23h вызывается, если была нажаты комбинации 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.

1. **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.

1. **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 A: 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\_l

or AL,30h

mov [SI],AL

end\_l:

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