

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. 7381

Адамов Я.В.

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 модуля 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.

Название функции	Описание
PrintMsg	Печать строки, адрес которой помещен в DX.
BYTE_TO_HEX	Байт в AL переводится в два символа шестн. числа в AX.
TETR_TO_HEX	Перевод 4-битного числа в код символа в 16 с/с. Вспомогательная функция для BYTE_TO_HEX.
DEAL	Запуск вызываемого модуля.
PAR_BL	Создаёт блока параметров.
MEMORY_	Освобождение неиспользуемой памяти.

Описание работы утилиты.

Программа подготавливает параметры для запуска загрузочного модуля из того же каталога, в котором находится он сам, после чего вызываемый модуль запускается с использованием загрузчика. После запуска проверяется выполнение загрузчика, а затем результат выполнения вызываемой программы. В качестве вызываемой программы выбрана утилита, созданная для лабораторной работы №2.

Демонстрация работы программы представлена на рис. 1-4.

```
Segment address of unavailable memory: 9FFFh
Segment address of environment: 025Ch
Tail:
Content of environment:
  PATH=Z:\
  COMSPEC=Z:\COMMAND.COM
  BLASTER=A220 I7 D1 H5 T6
Path of module:
  C:\Z.COM

g

All is good
Code of finish: 67
```

Рисунок 1 – запуск программы из текущего каталога.

```
Segment address of unavailable memory: 9FFFh
Segment address of environment: 025Ch
Tail:
Content of environment:
  PATH=Z:\
  COMSPEC=Z:\COMMAND.COM
  BLASTER=A220 I7 D1 H5 T6
Path of module:
  C:\Z.COM

♥

All is good
Code of finish: 03
```

Рисунок 2 – результат работы программы при нажатии комбинации клавиш Ctrl+C.

```

C:\>TASM\CODE.EXE

Segment address of unavailable memory: 9FFFh
Segment address of environment: 025Ch
Tail:
Content of environment:
  PATH=Z:\
  COMSPEC=Z:\COMMAND.COM
  BLASTER=A220 I7 D1 H5 T6
Path of module:
  C:\TASM\Z.COM

All is good
Code of finish: 03

```

Рисунок 3 – запуск программы не из текущего каталога.

```

C:\>CODE.EXE
Error of the file

```

Рисунок 4 – запуск программы, когда файлы находятся в разных каталогах.

Вывод.

В ходе выполнения лабораторной работы была исследована возможность построения загрузочного модуля динамической структуры.

Ответы на контрольные вопросы.

Как реализовано прерывание Ctrl+C?

При нажатии сочетания клавиш Ctrl+C, вызывается прерывание 23h, при этом управление передаётся по адресу 0000:008Ch. Данный адрес копируется в PSP функциями 26h и 4Ch, а затем восстанавливается при выходе из программы.

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

В месте вызова функции 4Ch прерывания 21h.

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

В месте вызова функции 01h прерывания 21h, то есть там, где ожидается ввод символа (но вместо прерывания вводится символ, получаемый комбинацией клавиш Ctrl+C: '♥').

Приложение А. 2.asm.

TESTPC SEGMENT

ASSUME CS:TESTPC, DS:TESTPC, ES:NOTHING, SS:NOTHING

ORG 100H

START: JMP BEGIN

; _____

; Данные

SegmentAddressOfUnavailableMemory db 0dh,0ah,'Segment address of
unavailable memory: h',0dh,0ah,'\$'

SegmentAddressOfEnvironment db 'Segment address of environment:
h',0dh,0ah,'\$'

Tail db 'Tail:', '\$'

ContentOfEnvironment db 'Content of environment:\$'

PathOfModule db 0dh,'Path of module:\$'

EndOfString db 0dh,0ah,'\$'

EndOfStringWithTab db 0dh,0ah,' \$'

; _____

; Процедуры

TETR_TO_HEX PROC near

and al,0fh

```
    cmp al,09
    jbe NEXT
    add al,07
NEXT: add al,30h
    ret
TETR_TO_HEX ENDP
```

```
BYTE_TO_HEX PROC near
    push cx
    mov ah,al
    call TETR_TO_HEX
    xchg al,ah
    mov cl,4
    shr al,cl
    call TETR_TO_HEX ;
    pop cx
    ret
BYTE_TO_HEX ENDP
```

```
WRD_TO_HEX PROC near
    push bx
    mov bh,ah
    call BYTE_TO_HEX
    mov [di],ah
    dec di
    mov [di],al
    dec di
    mov al,bh
```

```
    call BYTE_TO_HEX
    mov [di],ah
    dec di
    mov [di],al
    pop bx
    ret
WRD_TO_HEX ENDP
```

```
PrintMsg PROC near
    push ax
    mov ah,09h
    int 21h
    pop ax
    ret
PrintMsg ENDP
```

```
PrintSegmentAddressOfUnavailableMemory PROC near
    push ax
    push di
    push dx

    mov ax,es:[2]
    lea di,SegmentAddressOfUnavailableMemory
    mov dx,di
    add di,44
    call WRD_TO_HEX
    call PrintMsg
```



```
pop dx
pop di
pop ax
ret
```

PrintSegmentAddressOfUnavailableMemory ENDP

PrintSegmentAddressOfEnvironment PROC near

```
push ax
push di
push dx
```

```
mov ax,es:[2Ch]
lea di,SegmentAddressOfEnvironment
mov dx,di
add di,35
call WRD_TO_HEX
call PrintMsg
```

```
pop dx
pop di
pop ax
ret
```

PrintSegmentAddressOfEnvironment ENDP

PrintCommandLineTail PROC near

```

push ax
push cx
push dx
push si

lea dx,Tail
call PrintMsg
xor ax,ax
mov al,es:[80h]
add al,81h
mov si,ax
push es:[si]
mov byte ptr es:[si+1],'$'
push ds
mov cx,es
mov ds,cx
mov dx,81h
call PrintMsg
lea dx,EndOfString
call PrintMsg

pop ds
pop es:[si]
pop si
pop dx
pop cx
pop ax
ret

```

```
PrintCommandLineTail ENDP
```

```
PrintEnvironmentContentAndModulePath PROC near
```

```
    push si
```

```
    push es
```

```
    push ax
```

```
    push bx
```

```
    push cx
```

```
    push dx
```

```
    lea dx,ContentOfEnvironment
```

```
    call PrintMsg
```

```
    mov bx,1
```

```
    mov es,es:[2ch]
```

```
    mov si,0
```

```
p_1:
```

```
    lea dx,EndOfStringWithTab
```

```
    call PrintMsg
```

```
    mov ax,si
```

```
p_2:
```

```
    cmp byte ptr es:[si],0
```

```
    je p_3
```

```
    inc si
```

```
    jmp p_2
```

```
p_3:
```

```
    push es:[si]
```

```
    mov byte ptr es:[si], '$'
```

```

    push ds
    mov cx,es
    mov ds,cx
    mov dx,ax
    call PrintMsg
    pop ds
    pop es:[si]
    cmp bx,0
    jz p_4
    inc si
    cmp byte ptr es:[si],01h
    jne p_1
    lea dx,PathOfModule
    call PrintMsg
    mov bx,0
    add si,2
    jmp p_1
p_4:
    pop dx
    pop cx
    pop bx
    pop ax
    pop es
    pop si
    ret
PrintEnvironmentContentAndModulePath ENDP

```

; _____
; Код

BEGIN:

```
    call PrintSegmentAddressOfUnavailableMemory
    call PrintSegmentAddressOfEnvironment
    call PrintCommandLineTail
    call PrintEnvironmentContentAndModulePath
    lea dx,EndOfString
    call PrintMsg
    call PrintMsg
    mov ah,01h
    int 21h
    mov ah,04Ch
    int 21h
    ret
```

TESTPC ENDS

END START

Приложение Б. code.asm.

```
AStack SEGMENT STACK
```

```
    DW 100h DUP(?)
```

```
AStack ENDS
```

```
;
```

```
DATA SEGMENT
```

```
ERRMEM db 'Error of clear memory: $'
```

```
MCB_ db 'MCB is destroyed$'
```

```
NOTMEM db 'Not enough memory$'
```

```
ERRADR db 'Error of the address$'
```

```
ERRFUNCT db 'Error of function number$'
```

```
ERRFILE db 'Error of the file$'
```

```
ERRDISK db 'Error of the disk$'
```

```
NOTMEM_ db 'Not enough memory$'
```

```
ERRENV db 'Error of env$'
```

```
ERRFORM db 'Error of format$'
```

```
GOOD db 'All is good$'
```

```
END_CTRL db 'End ctrl$'
```

```
ERRDEVICE db 'Error of device$'
```

```
ERRRES db 'End 31h$'
```

```
CODEELEM db 'Code of finish: $'
```

```
STRING db 0DH,0AH,'$'
```

```
PARAM_S dw 0
```

```
    dd ?
```

```
    dd 0
```

```
        dd 0
_PATH db 50h dup ('$')
KEEP_SS dw 0
KEEP_SP dw 0
QQQ db 'qqqqqqqqqqqqqqq$'
```

```
DATA ENDS
```

```
;_____
```

```
CODE SEGMENT
```

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

```
PrintMsg PROC near
```

```
        push ax
        mov ah,09h
        int 21h
        pop ax
        ret
```

```
PrintMsg ENDP
```

```
TETR_TO_HEX PROC near
```

```
        and al,0fh
        cmp al,09
        jbe NEXT
        add al,07
```

NEXT:

add al,30h

ret

TETR_TO_HEX ENDP

BYTE_TO_HEX PROC near

push cx

mov ah,al

call TETR_TO_HEX

xchg al,ah

mov cl,4

shr al,cl

call TETR_TO_HEX

pop cx

ret

BYTE_TO_HEX ENDP

FUNC_ PROC

mov ax,AStack

sub ax,CODE

add ax,100h

mov bx,ax

mov ah,4ah

int 21h

jnc stepF

call DEAL

stepF:

```
    call PAR_BL
    push es
    push bx
    push si
    push ax
    mov es,es:[2ch]
    mov bx,-1
```

stepS:

```
    add bx,1
    cmp word ptr es:[bx],0000h
    jne stepS
    add bx,4
    mov si,-1
```

stepT:

```
    add si,1
    mov al,es:[bx+si]
    mov _PATH[si],al
    cmp byte ptr es:[bx+si],00h
    jne stepT
    add si,1
```

stepT2:

```
    mov _PATH[si],0
    sub si,1
    cmp byte ptr es:[bx+si],'\ '
    jne stepT2
    add si,1
    mov _PATH[si],'2'
```

```
    add si,1
    mov _PATH[si], '.'
    add si,1
    mov _PATH[si], 'C'
    add si,1
    mov _PATH[si], 'O'
    add si,1
    mov _PATH[si], 'M'
    pop ax
    pop si
    pop bx
    pop es
    ret
FUNC_ ENDP
```

```
MEMORY_ PROC
    mov ax,AStack
    mov bx,es
    sub ax,bx
    add ax,10h
    mov bx,ax
    mov ah,4Ah
    int 21h
    jnc FIN

    lea dx,ERRMEM
    call PrintMsg
```

```
    cmp ax,7
    lea dx,MCB_
    je MEM_PRINT
    cmp ax,8
    lea dx,NOTMEM
    je MEM_PRINT
    cmp ax,9
    lea dx,ERRADR
```

MEM_PRINT:

```
    call PrintMsg
    lea dx,STRING
    call PrintMsg
```

```
    xor AL,AL
    mov AH,4Ch
    int 21H
```

FIN:

```
    ret
```

MEMORY_ ENDP

PAR_BL PROC

```
    mov ax,es:[2Ch]
    mov PARAM_S,ax
    mov PARAM_S+2,es
    mov PARAM_S+4,80h
    ret
```

PAR_BL ENDP

DEAL PROC

```
    lea dx,_PATH
    xor ch,ch
    mov cl,es:[80h]
    cmp cx,0
    je UNTAIL
    mov si,cx
    push si
```

METOCKKA:

```
    mov al,es:[81h+si]
    mov [offset _PATH+si-1],al
    dec si
    loop METOCKKA
    pop si
    mov [_PATH+si-1],0
    mov dx,offset _PATH
```

UNTAIL:

```
    push ds
    pop es
    mov bx,offset PARAM_S
    mov KEEP_SP,sp
    mov KEEP_SS,ss
    mov ax,4b00h
    int 21h
    jnc FIN_
```

```

push ax
mov ax,DATA
mov ds,ax
pop ax
mov SS,KEEP_SS
mov SP,KEEP_SP
cmp ax,1
lea dx,ERRFUNCT
je DEAL_PRINT
cmp ax,2
lea dx,ERRFILE
je DEAL_PRINT
cmp ax,5
lea dx,ERRDISK
je DEAL_PRINT
cmp ax,8
lea dx,NOTMEM_
je DEAL_PRINT
cmp ax,10
lea dx,ERRENV
je DEAL_PRINT
cmp ax,11
lea dx,ERRFORM
DEAL_PRINT:
call PrintMsg
lea dx,STRING
call PrintMsg
xor al,al

```

```

    mov ah,4Ch
    int 21H
FIN_:
    lea dx,STRING
    call PrintMsg
    call PrintMsg
    mov ax,4d00h
    int 21h
    cmp ah,0
    lea dx,GOOD
    je REASONS
    cmp ah,1
    lea dx,END_CTRL
    je REASONS
    cmp ah,2
    lea dx,ERRDEVICE
    je REASONS
    cmp ah,3
    lea dx,ERRRES
REASONS:
    call PrintMsg
    lea dx,STRING
    call PrintMsg
    lea dx,CODEELEM
    call PrintMsg
    call BYTE_TO_HEX
    push ax
    mov ah,02h

```

```
    mov dl,al
    int 21h
    pop ax
    xchg ah,al
    mov ah,02h
    mov dl,al
    int 21h
    lea dx,STRING
    call PrintMsg
    ret
DEAL ENDP
```

BEGIN:

```
    mov ax,DATA
    mov ds,ax
    call MEMORY_
    call FUNC_
    call DEAL
    xor AL,AL
    mov AH,4Ch
    int 21H
CODE ENDS
    END BEGIN
```