

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. 0382

Гудов Н.Р.

Преподаватели

Ефремов М.А.

Санкт-Петербург

2022

Цель работы.

Исследование возможности построения загрузочного модуля динамической структуры. В работе исследуется интерфейс между вызывающим и вызываемым модулями по управлению и по данным.

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.

Выполнение работы.

Программный код см. в Приложении А

Шаг 1.

Написан программный модуль, подготавливающий параметры для запуска загрузочного модуля. Состоит из процедуры освобождения неиспользуемой памяти FREE_MEM, процедуры поиска пути к вызываемой программе COMPOSE_PATH и процедуры запуска программы.

Шаг 2.

Запуск программы из того же каталога и введение латинского символа.



```
A:\>lab6
External memory segment: 9FC0h
Environment segment: 1F26h
Command-line tail:
Environment variables:
COMSPEC=C:\COMMAND.COM
PROMPT=$p$g
PATH=C:\DOS
TEMP=C:\DOS
Program path: A:\lab2.COM
N
Normal exit      Code:      78
A:\>
```

Рисунок 1. Запуск на втором шаге

В таком случае программа завершается нормально, выводя код ранее введенного символа.

Шаг 3.

Запуск программы из того же каталога и введение Ctrl+c.



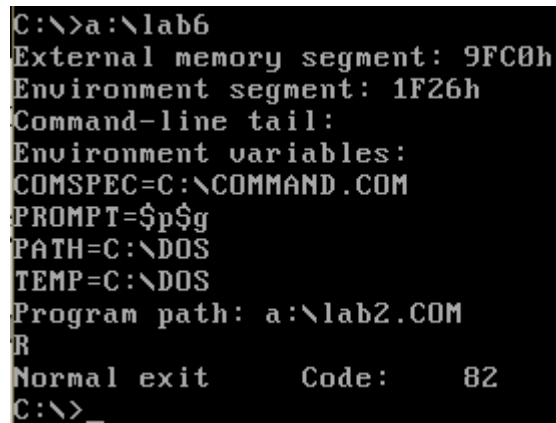
```
A:\>lab6
External memory segment: 9FC0h
Environment segment: 1F26h
Command-line tail:
Environment variables:
COMSPEC=C:\COMMAND.COM
PROMPT=$p$g
PATH=C:\DOS
TEMP=C:\DOS
Program path: A:\lab2.COM
^C
Ctrl-C exit
A:\>_
```

Рисунок 2. Запуск на третьем шаге

В таком случае программа завершается без ошибок, как и на предыдущем шаге.

Шаг 4.

Запуск программы из другого каталога.



```
C:\>a:\lab6
External memory segment: 9FC0h
Environment segment: 1F26h
Command-line tail:
Environment variables:
COMSPEC=C:\COMMAND.COM
PROMPT=$p$g
PATH=C:\DOS
TEMP=C:\DOS
Program path: a:\lab2.COM
R
Normal exit      Code:      82
C:\>_
```

Рисунок 3. Запуск на четвертом шаге при нажатии R

```

C:\>a:\lab6
External memory segment: 9FC0h
Environment segment: 1F26h
Command-line tail:
Environment variables:
COMSPEC=C:\COMMAND.COM
PROMPT=$p$g
PATH=C:\DOS
TEMP=C:\DOS
Program path: a:\lab2.COM
^C
Ctrl-C exit
C:\>_

```

Рисунок 4. Запуск на четвертом шаге при нажатии Ctrl+C

В таком случае программа завершается без ошибок, как и на предыдущем шаге. Вызов из другого каталога не влияет на работоспособность программы.

Шаг 5.

Запуск программы при нахождении модуля в другом каталоге.

```

A:\>lab6
Loading error
A:\>_

```

Рисунок 5. Запуск при нахождении модуля в другом каталоге

В таком случае программа завершается с ошибкой, так как не может найти модуль в своем каталоге.

Вопросы.

- 1) Как реализовано прерывание Ctrl-C?

Вызывается прерывание int 23h, которое завершает работу текущей программы. Адрес по вектору int 23h копируется в поле PSP Ctrl-Break Address. Исходное значение адреса обработчика Ctrl-Break восстанавливается из PSP при завершении программы.

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

После работы функции 4Ch прерывания int 21h, которая завершает работу программы.

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

Программа завершится при ожидании ввода символа.

Выводы.

В результате выполнения лабораторной работы была исследована возможность построения загрузочного модуля динамической структуры. Был исследован интерфейс между вызывающим и вызываемым модулями по управлению и данными.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЕ КОДЫ ПРОГРАММ

Название файла: lab6.asm

```
AStack SEGMENT STACK
    DW 64 DUP(?)
AStack ENDS

DATA SEGMENT
    PARAMETERS DB 14 dup(0)
    PATH DB 64 dup(0)
    FILE DB "lab2.COM", 0
    MEM_ERROR DB 'Memory error',13,10,'$'
    LOAD_ERROR DB 'Loading error',13,10,'$'
    NORM_EXIT DB 13,10,'Normal exit Code:',13,10,'$'
    CTRL_EXIT DB 'Ctrl-C exit',13,10,'$'
    DEV_EXIT DB 'Device error exit',13,10,'$'
    FUNC_EXIT DB 'Function 31h exit',13,10,'$'
    KEEP_SS DW ?
    KEEP_SP DW ?
DATA ENDS

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

;-----
    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

;-----
    PRINT PROC NEAR
        push AX
        mov AH, 09h
        int 21h
        pop AX
        ret
    PRINT ENDP
```

```

;-----
FREE_UP_MEM PROC NEAR
    push AX
    push BX
    push DX
    push CX

    mov BX, offset end_address
    mov AX, ES
    sub BX, AX
    mov CL, 4
    shr BX, CL
    mov AH, 4Ah
    int 21h

    jnc free_mem_end
    mov DX, offset MEM_ERROR
    call PRINT

    free_mem_end:
    pop CX
    pop DX
    pop BX
    pop AX
    ret
FREE_UP_MEM ENDP

;-----
CREATE_PARAMETER_BLOCK PROC NEAR
    push AX
    push DI
    mov DI, offset PARAMETERS
    mov [DI+2], ES
    mov AX, 80h
    mov [DI+4], AX
    pop DI
    pop AX
    ret
CREATE_PARAMETER_BLOCK ENDP

;-----
COMPOSE_PATH PROC NEAR
    push DX
    push DI
    push SI
    push ES

    mov ES, ES:[2Ch]
    mov SI, offset PATH
    xor DI, DI

    read_byte:
    mov DL, ES:[DI]
    check_byte:
    inc DI
    cmp DL, 0
    jne read_byte
    mov DL, ES:[DI]
    cmp DL, 0
    jne check_byte

```



```

    add DI, 3
    write_path:
    mov DL, ES:[DI]
    mov [SI], DL
    inc SI
    inc DI
    cmp DL, 0
    jne write_path

    backslash_loop:
    mov DL, [SI-2]
    dec SI
    cmp DL, '\'
    jne backslash_loop

    mov DI, offset FILE
    write_filename:
    mov DL, [DI]
    mov [SI], DL
    inc SI
    inc DI
    cmp DL, 0
    jne write_filename

    pop ES
    pop SI
    pop DI
    pop DX
    ret
COMPOSE_PATH ENDP

```

```

;-----
BEGIN PROC NEAR
    push AX
    push BX
    push DX
    push SI
    push ES

    mov KEEP_SS, SS
    mov KEEP_SP, SP
    mov AX, DS
    mov ES, AX
    mov BX, offset PARAMETERS
    mov DX, offset PATH

    mov AX, 4B00h
    int 21h
    mov SS, KEEP_SS
    mov SP, KEEP_SP

    mov DX, offset LOAD_ERROR
    jc print_exit_info

    loaded:
    mov AH, 4Dh
    int 21h
    mov DX, offset NORM_EXIT
    cmp AH, 0
    je read_key

```

```

        mov DX, offset CTRL_EXIT
        cmp AH, 1
        je print_exit_info
        mov DX, offset DEV_EXIT
        cmp AH, 2
        je print_exit_info
        mov DX, offset FUNC_EXIT
        cmp AH, 3
        je print_exit_info

        read_key:
        mov SI, DX
        add SI, 28
        call BYTE_TO_DEC

        print_exit_info:
        call PRINT

        pop ES
        pop SI
        pop DX
        pop BX
        pop AX
        ret
BEGIN ENDP
;-----
Main PROC FAR
        sub AX, AX
        mov AX, DATA
        mov DS, AX

        call FREE_UP_MEM

        jc main_end
        call CREATE_PARAMETER_BLOCK
        call COMPOSE_PATH
        call BEGIN

        main_end:
        xor AL, AL
        mov AH, 4Ch
        int 21h
Main ENDP

end_address:
CODE ENDS
end Main

```

Название файла: lab2.asm

PC Segment

```

        Assume CS:PC, DS:PC, ES:NOTHING, SS:NOTHING
        ORG 100H

```

START: JMP BEGIN

;ÄÄííÛÄ

Unavailable_Memory_Msg db 'Unavailable memory address: ', 0ah, '\$'

```
Segment_Env_Addres_Msg db 'Environment address:      ', 0ah, '$'
Input_String db 'Input string:', '$'
```

```
; ĩĐİÖÄÄÓĐŮ
```

```
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 ; áàéò â AL ĩăđăăîăèòňŷ â äââ ñèîâîèâ øăňò. ÷èñèâ â
AX
```

```
    push CX
    mov AH,AL
    call TETR_TO_HEX
    xchg AL,AH
    mov CL,4
    shr AL,CL
    call TETR_TO_HEX ; â AL ñòàđøàŷ öèôđà
    pop CX ; â AH ìèääøàŷ
    ret
BYTE_TO_HEX ENDP
```

```
WRD_TO_HEX PROC near ; ĩăđăăîă â 16 ñ/ñ 16-òè đàçđŷăĭĭăĭ ÷èñèâ
; â AX - ÷èñèĭ, DI - àăđăň ĭĭñèăăĭăăĭ ñèîâîèâ
```

```
    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
```

```
BYTE_TO_DEC PROC near ; ĩăđăăîă â 10ň/ň, SI - àăđăň ĭĭëŷ ìèääøăé öèôđŮ
```

```
    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

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

Prnt_Unavailable proc near
        push ax
        push di
        push dx

        mov ax, es:[02h]
        mov di, offset Unavailable_Memory_Msg
        add di, 31
        call wrd_to_hex
        mov dx, offset Unavailable_Memory_Msg
        call print

        pop dx
        pop di
        pop ax
        ret
Prnt_Unavailable endp

Prnt_Env_Address proc near
        push ax
        push di
        push dx

        mov ax, es:[02Ch]
        mov di, offset Segment_Env_Addres_Msg
        add di, 24
        call wrd_to_hex
        mov dx, offset Segment_Env_Addres_Msg
        call print

        pop dx
        pop di
        pop ax
        ret
Prnt_Env_Address endp

Prnt_Input_String proc near
        push dx

```

```

push cx
push si
push ax

mov dx, offset Input_String
call print
mov cl, ds:[80h]
mov si, 081h
mov ah, 02h
cmp cl, 0
je end_

print_symbol:
    mov dl, [si]
    int 21h
    inc si
    loop print_symbol

end_:
    mov dl, 0ah
    int 21h

pop ax
pop si
pop cx
pop dx

ret

Prnt_Input_String endp

Prnt_String proc near
    push dx
    push ax
    mov ah, 02h

    print_sym:
        mov dl, ds:[si]
        inc si

        cmp dl, 0
        jz end_of_string

        int 21h
        jmp print_sym

    end_of_string:
        mov dl, 0ah
        int 21h

    pop ax
    pop dx
    ret
Prnt_String endp

Prnt_Environment_Content_And_Path proc near

```

```

push ds
push si
push ax
push cx
push dx

mov ds, es:[2ch]
mov si, 0

Prnt_Strings:
    call Prnt_String
    mov bl, ds:[si]
    cmp bl, 0
    jz print_path
    jmp Prnt_Strings

print_path:
    add si, 3
    call Prnt_String

the_end:
    pop dx
    pop cx
    pop ax
    pop si
    pop ds

ret

Prnt_Environment_Content_And_Path endp

BEGIN:

    call Prnt_Unavailable
    call Prnt_Env_Address
    call Prnt_Input_String
    call Prnt_Environment_Content_And_Path

    xor ax, ax
    mov ah, 01h
    int 21h

    mov ah, 4Ch
    int 21h

PC    ENDS
            END        START

```