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 Шангичев В. А.

Преподаватель Ефремов М.А.

Санкт-Петербург

2022

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 21h. Все загрузочные модули находятся в одном каталоге. Необходимо обеспечить возможность запуска модуля динамической структуры из любого каталога.

Задание.

1. Для выполнения лабораторной работы необходимо написать и отладить программный модуль типа .EXE, который выполняет функции:

- 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;

- Вызываемый модуль запускается с использованием загрузчика;

- 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.

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.

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.

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.

5. Запустите отлаженную программу, когда модули находятся в разных каталогах. Занесите полученные данные в отчет.

Выполнение работы.

1. Для выполнения лабораторной работы был написан файл `main.asm`. В нем были реализованы следующие процедуры:

`print` - выводит на экран сообщение по смещению, указанному в регистре `dx`.

`memory_alloc` - уменьшает количество памяти, занимаемое программой.

`free_mem` - вызывает предыдущую функцию и печатает сообщение об ошибке / успехе.

`handle_errors` - выводит сообщения об ошибках при запуске второй программы, если требуется.

`load` – вызывает вторую программу и печатает все сопутствующие сообщения.

`path` – сохраняет путь к файлу.

`main` – главная процедура.

2. Программа была запущена в каталоге компиляции. Был введен символ `q`. Результат представлен на рис. 1.

Рисунок 1 – запуск из каталога компиляции

3. При выполнении данного пункта были сохранены условия из предыдущего шага, но теперь выход из программы был реализован с помощью комбинации клавиш ctrl + c. В DosBox не налажена нормальная работа этой комбинации клавиш, однако на консоли можно увидеть символ “сердечко”, что было распознано второй программой, и привело к ее завершению (рис. 2)

Рисунок 2 – завершение работы программы после нажатия ctrl + C

4. Действия из двух предыдущих шагов были проделаны для случая, когда текущим каталогом является каталог, отличный от тех, в которых содержатся разработанные модули (рис. 3 - 4)

Рисунок 3 – ввод символа

Рисунок 4 – завершение по ctrl + C

5. Запуск программы для случая, когда вторая программа находится в другом каталоге.

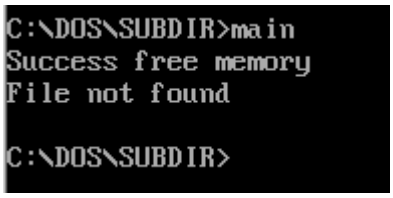

Рисунок 5 – программы в разных каталогах

Ответы на вопросы.

1. Как реализовано прерывание Ctrl-C?

Если было нажато сочетание клавиш ctrl+c и флаг break поставлен в значение on, то управление передаётся по адресу 0000:008Ch. Этот адрес копируется в PSP функциями 26h и 4Ch и восстанавливается из него при выходе из программы.

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

В месте вызова функции 4Ch прерывания 21h.

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

В месте вызова функции 01h прерывания 21h.

Выводы.

В ходе работы были исследованы возможности построения загрузочного модуля динамической структуры. Была написана программа, вызывающая другую программу. Работа исполнительного файла была изучена при различном взаимном расположении программы и подпрограммы.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Файл main.asm

```
AStack SEGMENT STACK
 DW 32 DUP(?)
AStack ENDS

DATA SEGMENT
	PARAM dw 0
		dd 0
		dd 0
		dd 0

	file_name db 'lb2.COM', 0
	cmd db 1h, 0dh
	file_path db 128 DUP (?)

	memory_destroied_msg db 'The control memory block is destroyed', 0DH, 0AH,'$'
	not_enough_msg db 'Not enough memory to execute the function', 0DH, 0AH,'$'
	inv_address_msg db 'Invalid memory block address', 0DH, 0AH,'$'
	success_mem_msg db 'Success free memory', 0DH, 0AH,'$'
	free_mem_flag db 0

	invalid_function_msg db 'Invalid function number', 0DH, 0AH,'$'
	file_not_found_msg db 'File not found', 0DH, 0AH,'$'
	disk_error_msg db 'Disk error', 0DH, 0AH,'$'
	not_enough_load_msg db 'Not enough memory', 0DH, 0AH,'$'
	invalid_env_string_msg db 'Incorrect environment string', 0DH, 0AH,'$'
	incorrect_format_msg db 'Incorrect format', 0DH, 0AH,'$'

	success_end db 0DH, 0AH,'Programm ended with code =    ', 0DH, 0AH,'$'
```

```
      ctrl_c_end db 'Programm ended ctrl-break', 0DH, 0AH,'$'
      device_end db 'Programm ended device error', 0DH, 0AH,'$'
      int31_end db 'Programm ended int 31h', 0DH, 0AH,'$'

      keep_ss dw 0
      keep_sp dw 0
      keep_psp dw 0

      END_DATA db 0
DATA ENDS

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

; П Р О Ц Е Д У Р Ы
;----------------------
print PROC near
      push ax
      mov ah, 09h
      int 21h
      pop ax
      ret
print ENDP
;----------------------
memory_alloc proc near
      push ax
      push bx

      xor dx, dx
    mov ax, offset end_data
    mov bx, offset end_programm
    add ax, bx
    mov bx, 16
    div bx
    add ax, 50h
    mov bx, ax
```

```
    and ax, 0

    mov ah, 4ah
    int 21h

    pop bx
    pop ax
    ret
memory_alloc endp
;----------------------
FREE_MEM PROC near
     push dx
     push ax

     call memory_alloc

     jnc success_free

     cmp ax, 7
     je mem_destr

     cmp ax, 8
     je not_enough

     cmp ax, 9
     je inv_addr

     mem_destr:
          mov dx, offset memory_destroied_msg
          jmp finish_proc

     not_enough:
          mov dx, offset not_enough_msg
          jmp finish_proc

     inv_addr:
          mov dx, offset inv_address_msg
```

```
            jmp finish_proc

      success_free:
            mov dx, offset success_mem_msg
            mov free_mem_flag, 1

      finish_proc:
            call print
            pop dx
            pop ax
      ret
FREE_MEM ENDP
;-------------------------------------
handle_errors proc near
      ; bl:
      ; 1 - some errors were detected
      ; 0 - no errors
      ; dx - message
      push ax


      jnc finish_handle
      mov bl, 1

      cmp ax, 1
      je inv_func_msg

      cmp ax, 2
      je not_found

      cmp ax, 5
      je disk_error

      cmp ax, 8
      je not_enough_mem

      cmp ax, 10
```

```
      je env_error

      cmp ax, 11
      je not_correct_format


inv_func_msg:
      lea dx, invalid_function_msg
      jmp finish_handle

not_found:
      lea dx, file_not_found_msg
      jmp finish_handle

disk_error:
      lea dx, disk_error_msg
      jmp finish_handle

not_enough_mem:
      lea dx, not_enough_load_msg
      jmp finish_handle

env_error:
      lea dx, invalid_env_string_msg
      jmp finish_handle

not_correct_format:
      mov dx, offset incorrect_format_msg
      jmp finish_handle

finish_handle:
      pop ax
      ret
handle_errors endp
;---------------------------------
load proc near
      push ax
```

```
        push bx
        push cx
        push dx
        push ds
        push es

        mov keep_sp, sp
        mov ax, ss
        mov keep_ss, ax

        ; load program
        mov ax, DATA
        mov es, ax
        mov bx, offset PARAM
        mov dx, offset cmd
        mov [bx+2], dx
        mov [bx+4], ds
        mov dx, offset file_path
        xor bx, bx
        mov ax, 4B00h
        int 21h

        ; restore registers
        mov ss, keep_ss
        mov sp, keep_sp
        pop es
        pop ds

        call handle_errors
        cmp bl, 0
        jne load_print

success_load:
        mov ax, 4D00h
        int 21h

        cmp ah, 0
```

```
      jne ctrlc
      push di
      lea di, success_end
      mov [di+30], al
      pop si
      lea dx, success_end
      jmp load_print
ctrlc:
      cmp ah, 1
      jne device
      lea dx, ctrl_c_end
      jmp load_print
device:
      cmp ah, 2
      jne int_31h
      lea dx, device_end
      jmp load_print
int_31h:
      cmp ah, 3
      lea dx, int31_end

load_print:
      call print

end_load:
      pop dx
      pop cx
      pop bx
      pop ax
      ret
load ENDP
;-----------------------------------
path proc near
      push ax
      push bx
      push cx
      push dx
```

```
      push di
      push si
      push es

      mov ax, keep_psp
      mov es, ax
      mov es, es:[2Ch]
      mov bx, 0

find_zero:
      mov al, es:[bx]
      inc bx
      cmp byte ptr es:[bx], 0
      je second_zero
      jmp find_zero

      second_zero:
            cmp al, 0
            je skip_0_1
      jmp find_zero

skip_0_1:
      add BX, 3
      mov DI, 0

path_loop:
      mov dl, es:[bx]
      mov byte ptr [file_path+di], dl
      inc di
      inc bx
      cmp dl, 0
      je path_end_loop
      cmp dl, '\'
      jne path_loop
      mov cx, di
      jmp path_loop
path_end_loop:
```

```
      mov di, cx
      mov si, 0

_file_name:
      mov dl, byte ptr [file_name+si]
      mov byte ptr [file_path+di], dl
      inc di
      inc si
      cmp dl, 0
      jne _file_name

      pop es
      pop si
      pop di
      pop dx
      pop cx
      pop bx
      pop ax
      ret
path ENDP
;-----------------------------------
main proc far
      mov ax, data
      mov ds, ax
      mov keep_psp, es

      call free_mem

      cmp FREE_MEM_FLAG, 0
      je main_end
      call path
      call load
; Выход в DOS
main_end:
      xor AL, AL
      mov AH, 4Ch
      int 21h
```

```
MAIN ENDP
end_programm:
TESTPC ENDS
END MAIN
```

Файл lb2.asm

```
LAB2  Segment
        Assume CS:LAB2, DS:LAB2, ES:NOTHING, SS:NOTHING
        ORG 100H
START: JMP BEGIN


; Данные
unavailable_memory_msg db 'Unavailable memory address:      ', 0ah,
'$'
segment_env_addres_msg db 'Environment address:     ', 0ah, '$'
input_string db 'Input string:', '$'

;-----------------


; 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
```

```
;байт в AL переводится в два символа шест. чи
сла в AX
   push CX
   mov AH,AL
   call TETR_TO_HEX
   xchg AL,AH
   mov CL,4
   shr AL,CL
   call TETR_TO_HEX ;в AL старшая цифра
   pop CX ;в AH младшая
   ret
BYTE_TO_HEX ENDP
;--------------------------------
WRD_TO_HEX PROC near
;перевод в 16 с/с 16-ти разрядного числа
; в AX - число, DI - адрес последнего символа
   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
; перевод в 10с/с, SI - адрес поля младшей цифр
ы
   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
  ; процедура вывода
  ; dx - смещение сообщения
  push ax
  mov ah, 09h
  int 21h
  pop ax
  ret
print ENDP
;--------------------------------------

print_unavailable proc near
  ; процедура вывода сегментного адреса
  ; первого байта недоступной памяти
```

```
  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
print_unavailable endp
;-----------------------------------------

print_env_address proc near
  ; процедура вывода сегментного адреса
  ; среды
  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
print_env_address endp
```

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

print_input_string proc near
  ; печатает хвост командной строки
  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
```

```
print_input_string endp
;-----------------------------------------

print_string proc near
  ; п е ч а т а е т  н а б о р  с и м в о л о в  д о  н у л я . si - а д р е
с
  ; п е р в о г о  с и м в о л а  в  с т р о к е
  ; si  п о с л е  в ы п о л н е н и я  п р о ц е д у р ы  у к а з ы в а е т
  ; н а  п е р в ы й  с и м в о л  п о с л е  н у л я
  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
print_string endp

;-----------------------------------------
print_environment_content_and_path proc near
  ; п е ч а т а е т  с о д е р ж и м о е  о б л а с т и  с р е д ы  и  п у т ь
  ; з а г р у ж а е м о г о  м о д у л я
```

```
  push ds
  push si
  push ax
  push cx
  push dx

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

  ; п р е д ы д у щ и й  с и м в о л

  print_strings:
    call print_string
    mov bl, ds:[si]
    cmp bl, 0
    jz print_path
    jmp print_strings

  print_path:
    add si, 3 ; п р о п у с к  д в у х  б а й т о в  п е р е д  п у т е м
    call print_string

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


  ret

print_environment_content_and_path endp
;-----------------------------------------


BEGIN:
```

```
  call print_unavailable
  call print_env_address
  call print_input_string
  call print_environment_content_and_path

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

  ; Выход в DOS
  mov ah, 4Ch
  int 21H

LAB2  ENDS
        END     START
```