МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МОЭВМ

ЛАБОРАТОРНАЯ РАБОТА № 6

по дисциплине «Операционные системы»

ТЕМА: Построение модуля динамической структуры.

Студентка гр. 7381 _____ Алясова А.Н.

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) Написала и отладила 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.

2) Запустила отлаженную программу, когда текущим каталогом являлся каталог с разработанными модулями. Тут программа вызвала другую программу, которая остановилась, ожидая символ с клавиатуры.

Далее ввела произвольный символ из числа A-Z.

3) Запустила отлаженную программу, когда текущим каталогом являлся каталог с разработанными модулями. Тут программа вызвала другую программу, которая остановилась, ожидая символ с клавиатуры.

Ввела комбинацию символов Ctrl+C.

DOSBox не поддерживает комбинацию клавиш Ctrl + C, но программа должна вывести строку: reason 2 «Ctrl-Break termination. »

4) Запустила отлаженную программу, когда текущим каталогом являлся какой-либо другой каталог, отличный от того, в котором содержатся разработанные программные модули.

Повторила ввод комбинаций клавиш.

5) Запустила отлаженную программу, когда модули находятся в разных каталогах.

Функции программы, переменных и структур данных.

Название переменных	**Описание**
_ss (dw)	Для хранения значения регистра ss
_sp (dw)	Для хранения значения регистра sp
block (db)	Для хранения параграфов
path (db)	Для хранения пути
mem_error (db)	Хранит текст: Error: Memory can not be allocated!
lab2_name (db)	Хранит текст: lab2.com
mess (db)	Хранит текст: In lab6:
error1 (db)	Хранит текст: Error: invalid function number!
error2 (db)	Хранит текст: Error: file not found!
error3 (db)	Хранит текст: Error: disk error!
error4 (db)	Хранит текст: Error: not enought memory!

error5 (db)	Хранит текст: Error: wrong enviroment string!
error6 (db)	Хранит текст: Error: invalid format!
error7 (db)	Хранит текст: Error: unknown error!
reason (db)	Хранит текст: Termination reason:
reason1 (db)	Хранит текст: normal termination
reason2 (db)	Хранит текст: Ctrl-Break termination
reason3 (db)	Хранит текст: device error termination
reason4 (db)	Хранит текст: 31h function termination
reason5 (db)	Хранит текст: unknown termination reason
exit_code (db)	Хранит текст: Exit code:
endl (db)	Перенос строки
teststr (db)	Хранит текст: cmd tail to 2

Ответы на контрольные вопросы:

1. Как реализованно прерывание Ctrl+C?

В DOS существуют различные функции для работы с буфером клавиатуры. Одни из них реагируют на нажатие комбинации Ctrl+C как на ввод обычных символов (как например, функция 06h прерывания int 21h), в других случаях происходит переход по адресу, который записан на месте прерывания int 23h в векторе прерываний в ОП.

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

Код причины завершения 0 – это код успешного завершения программы. Программа завершается при выполнении функции 4ch прерывания int 21h.

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

Программа завершается в любой ее точке, сразу после обработки прерывания по Ctrl+C. Конкретно в данной программе, точкой завершения была инструкция, ожидающая ввода символа от пользователя (именно на этом моменте произошло завершение программы).

ПРИЛОЖЕНИЕ А
КОД ИСХОДНОЙ ПРОГРАММЫ

```
code segment
     assume cs: code, ds: code, ss:Tstack

     block           db 14 dup(0)
     path       db 50 dup(0)
     _ss        dw 0
     _sp        dw 0
     mem_error  db 'Error: Memory can not be allocated!$'
     lab2_name  db 'lab2.com', 0
     mess       db 'In lab6:$'
     error1          db 'Error: invalid function number!$'
     error2          db 'Error: file not found!$'
     error3          db 'Error: disk error!$'
     error4          db 'Error: not enought memory!$'
     error5          db 'Error: wrong enviroment string!$'
     error6          db 'Error: invalid format!$'
     error7          db 'Error: unknown error!$'
     reason          db 'Termination reason: $'
     reason1    db 'normal termination$'
     reason2    db 'Ctrl-Break termination$'
     reason3    db 'device error termination$'
     reason4    db '31h function termination$'
     reason5    db 'unknown termination reason$'
     exit_code  db 'Exit code: $'
     endl       db  13, 10, '$'
    teststr          db  4,' cmd tail to lab2 $',0
print proc near
    push   ax
    push   dx
    mov    ah, 09h
    int    21h
    pop    dx
    pop    ax
    ret
   print endp

print_symb proc near
     push  ax
     push  dx
     mov        ah, 02h
```

```
      int        21h
      pop        dx
      pop        ax
      ret
   print_symb endp

main proc near
      mov   ax, seg code
      mov   ds, ax

;free memory
      mov        bx, seg code
      add        bx, offset CodeSegEnd
      add        bx, 256 ;Stack
      mov   cl, 4h
      shr   bx, cl
      mov        ah, 4Ah
      int   21h
      jnc        mem_ok
      mov        dx, offset mem_error
      call  print
      mov        dx,   offset endl
      call  print
      jmp        main_exit

mem_ok:
      ;path
      mov        es, es:[002Ch]
      xor        bx, bx

;мотаем до пути
next:
      mov   dl, byte PTR es:[bx]
      cmp   dl, 0h
      je         first_0
      inc   bx
      jmp   next
first_0:
      inc   bx
      mov   dl, byte PTR es:[bx]
      cmp   dl, 0h
      je         second_0
      jmp   next
```

```
second_0:
     add        bx,3

;переписываем путь
     push  si
     mov        si, offset path
next1:
     mov   dl, byte PTR es:[bx]
     mov        [si], dl
     inc        si
     inc        bx
     cmp        dl, 0
     jne        next1

;удаляем имя файла
next2:
     mov        al, [si]
     cmp        al, '\'
     je         next3
     dec        si
     jmp        next2

next3:
     inc        si

;дописываем имя файла lab2.com
     push  di
     mov        di, offset lab2_name
next4:
     mov        ah, [di]
     mov        [si], ah
     inc        si
     inc        di
     cmp        ah, 0
     jne        next4

     pop        di
     pop        si

;подготовка к вызову lab2
     mov        _sp, sp
     mov        _ss, ss
     mov        ax, ds
     mov        es, ax
```

```
      push  ax
      mov         ax, seg teststr
      mov         [block+3], ah
      mov         [block+2], al
      pop         ax
      mov         [block+4], offset teststr
      mov         bx, offset block
      mov         dx, offset path
      mov         ax, 4B00h
      int   21h

      mov         dx, offset endl
      call  print
      call  print
      mov         dx, offset mess
      call  print
      mov         dx, offset endl
      call  print

      jc          errors
      jmp         run_ok

errors:
      cmp   ax, 1
      jne   err2
      mov   dx, offset error1
      call  print
      mov         dx, offset endl
      call  print
      jmp         main_exit

err2:
      cmp   ax, 2
      jne   err3
      mov   dx, offset error2
      call  print
      mov         dx, offset endl
      call  print
      jmp         main_exit

err3:
      cmp   ax, 5
      jne   err4
      mov   dx, offset error3
```

```
      call  print
      mov         dx, offset endl
      call  print
      jmp         main_exit

err4:
      cmp   ax, 8
      jne   err5
      mov   dx, offset error4
      call  print
      mov         dx, offset endl
      call  print
      jmp         main_exit

err5:
      cmp   ax, 10
      jne   err6
      mov   dx, offset error5
      call  print
      mov         dx, offset endl
      call  print
      jmp         main_exit

err6:
      cmp   ax, 11
      jne   err7
      mov   dx, offset error6
      call  print
      mov         dx, offset endl
      call  print
      jmp         main_exit

err7:
      mov   dx, offset error7
      call  print
      mov         dx, offset endl
      call  print
      jmp         main_exit

run_ok:
      mov   ax, seg code
      mov   ds, ax
      mov   ss, _ss
      mov   sp, _sp
```

```
      mov   ah, 4Dh
      int   21h

      push  ax
      mov   dx, offset reason
      call  print

      cmp   ah, 0
      jne   reason_tag2
      mov         dx, offset reason1
      call  print
      mov         dx, offset endl
      call  print
      jmp   print_exit_code

reason_tag2:
      cmp   ah, 1
      jne   reason_tag3
      mov         dx, offset reason2
      call  print
      mov         dx, offset endl
      call  print
      jmp   print_exit_code

reason_tag3:
      cmp   ah, 2
      jne   reason_tag4
      mov         dx, offset reason3
      call  print
      mov         dx, offset endl
      call  print
      jmp   print_exit_code

reason_tag4:
      cmp   ah, 3
      jne   reason_tag5
      mov         dx, offset reason4
      call  print
      mov         dx, offset endl
      call  print
      jmp   print_exit_code

reason_tag5:
```

```
      mov        dx, offset reason5
      call  print
      mov        dx, offset endl
      call  print

print_exit_code:
      mov        dx, offset exit_code
      call  print
      pop        ax
      mov   dl, al
      call  print_symb
      mov        dx, offset endl
      call  print

main_exit:
      xor   al, al
      mov   ah, 4Ch
      int   21h
      ret
      main endp

CodeSegEnd:
code ends

Tstack segment stack
      dw 128 dup (?)
Tstack ends

end main
```