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 _____ Кондратов Ю.А.

Преподаватель _____ Ефремов М.А.

Санкт-Петербург

2022

Цель работы.

Исследование возможности построения загрузочного модуля динамической структуры. Исследование интерфейса 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 2 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.

Исходный программный код разработанного модуля представлен в приложении А.

Шаг 1. Был написан и отлажен программный модуль типа .EXE, выполняющий все необходимые по заданию действия.

Шаг 2. Программа была запущена из каталога с разработанными модулями. Была нажата клавиша “g”. Результат работы программы представлен на рисунке 1.

Рисунок 1 – Результат запуска модуля с нажатием клавиши “g”

Шаг 3. Программа была запущена из каталога с разработанными модулями. Была нажата комбинация клавиш Ctrl+C.

Рисунок 2 – Результат запуска модуля с нажатием комбинации “Ctrl+C”

Шаг 4. Разработанный модуль был запущен из другого каталога. На рисунках 3 и 4 соответственно представлены результаты при нажатии “g” и “Ctrl+C”.

Рисунок 3 – Результат запуска модуля из каталога OTHER с нажатием “g”

Рисунок 4 – Результат запуска модуля из каталога OTHER с нажатием “Ctrl+C”

Шаг 5. Модуль lab2.com был перемещён из каталога OTHER после чего вновь был запущен модуль lab6.exe.

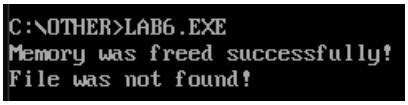

Рисунок 5 – Результат запуска модуля lab6.exe при отсутствии модуля lab2.com

Контрольные вопросы.

1. Как реализованы прерывание Ctrl+C?

Когда происходит нажатие сочетания клавиш Ctrl+C срабатывает прерывание - int 23h. Тогда управление передается по адресу - (0000:008C). С помощью функций 26h и 4ch этот адрес копируется в PSP. h и 4ch этот адрес копируется в PSP. При выходе из программы исходное значение адреса восстанавливается.

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

   Программа заканчивается в точке вызова функции 4ch прерывания int 21h.

3. В какой точке заканчивается вызываемая программа по прерыванию CtrlC?

   В точке, где была введена и считана комбинация клавиш Ctrl+C.

Выводы.

В ходе работы были исследованы возможности построения загрузочного модуля динамической структуры, а также интерфейс между вызывающим и вызываемыми модулями по управлению и по данным.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: lab6.asm

```
AStack segment stack
      DW 128 DUP(?)
AStack ENDS

DATA SEGMENT

      param_block DW 0
                  DD 0
                  DD 0
                  DD 0

      filename DB 'lab2.com', 0
      mem_flag DB 0
      cmd_line DB 1h, 0Dh
      cmd_line_pos DB 128 dup(0)
      old_ss DW 0
      old_sp DW 0
      psp DW 0

      mcb_err_msg DB 'MCB crashed!', 0dh, 0AH, '$'
      no_mem_err_msg DB 'Not enough memory to execute!', 0dh, 0AH, '$'
      mem_addr_err_msg DB 'Invalid memory address!', 0dh, 0AH, '$'
      free_mem_msg DB 'Memory was freed successfully!' , 0dh, 0AH, '$'

      func_num_err_msg DB 'Invalid function number!', 0dh, 0AH, '$'
      file_err_msg  DB 'File was not found!', 0Dh, 0Ah, '$'
      disk_err_msg DB 'Disk error!', 0dh, 0AH, '$'
      err_mem_msg DB 'Memory error!', 0dh, 0AH, '$'
      env_err_msg DB 'Enviroment string error!', 0dh, 0AH, '$'
      format_err_msg DB 'Format error!', 0dh, 0AH, '$'

      end_msg DB 0dh, 0AH, 'Program ended with code    ' , 0dh, 0AH, '$'
      end_break_msg DB 0dh, 0AH, 'Program ended because of Ctrl + C break' ,
0dh, 0AH, '$'
      end_device_msg DB 0dh, 0AH, 'Program ended because of device error' ,
0dh, 0AH, '$'
      end_int_msg DB 0dh, 0AH, 'Program ended because of int 31h' , 0dh, 0AH,
'$'

      END_DATA DB 0
DATA ENDS

CODE SEGMENT

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

print PROC
      push AX
      mov AH, 09h
      int 21h
      pop AX
      ret
```

```
print ENDP

free_mem PROC
      push AX
      push BX
      push CX
      push DX

      mov AX, offset END_DATA
      mov BX, offset global_end
      add BX, AX

      mov CL, 4
      shr BX, CL
      add BX, 2Bh
      mov AH, 4Ah
      int 21h

      jnc end_free
      mov mem_flag, 1

mcb_crash:
      cmp AX, 7
      jne no_mem
      mov DX, offset mcb_err_msg
      call print
      jmp free

no_mem:
      cmp AX, 8
      jne addr_err
      mov DX, offset no_mem_err_msg
      call print
      jmp free

addr_err:
      cmp AX, 9
      mov DX, offset mem_addr_err_msg
      call print
      jmp free

end_free:
      mov mem_flag, 1
      mov DX, offset free_mem_msg
      call print

free:
      pop DX
      pop CX
      pop BX
      pop AX
      ret
free_mem ENDP

load PROC
      push AX
      push BX
      push CX
```

```
      push DX
      push DS
      push ES
      mov old_sp, SP
      mov old_ss, SS
      mov AX, data
      mov ES, AX
      mov BX, offset param_block
      mov DX, offset cmd_line
      mov [BX+2], DX
      mov [BX+4], DS
      mov DX, offset cmd_line_pos

      mov AX, 4b00h
      int 21h

      mov SS, old_ss
      mov SP, old_sp
      pop ES
      pop DS

      jnc loads

      cmp AX, 1
      jne file_err
      mov DX, offset func_num_err_msg
      call print
      jmp load_end

file_err:
      cmp AX, 2
      jne disk_err
      mov DX, offset file_err_msg
      call print
      jmp load_end

disk_err:
      cmp AX, 5
      jne mem_err
      mov DX, offset disk_err_msg
      call print
      jmp load_end

mem_err:
      cmp AX, 8
      jne env_err
      mov DX, offset err_mem_msg
      call print
      jmp load_end

env_err:
      cmp AX, 10
      jne format_err
      mov DX, offset env_err_msg
      call print
      jmp load_end

format_err:
```

```
      cmp AX, 11
      mov DX, offset format_err_msg
      call print
      jmp load_end

loads:
      mov AH, 4Dh
      mov AL, 00h
      int 21h

      cmp AH, 0
      jne ctrl_break
      push DI
      mov DI, offset end_msg
      mov [DI+26], AL
      pop SI
      mov DX, offset end_msg
      call print
      jmp load_end

ctrl_break:
      cmp AH, 1
      jne device
      mov DX, offset end_break_msg
      call print
      jmp load_end

device:
      cmp AH, 2
      jne int_31h
      mov DX, offset end_device_msg
      call print
      jmp load_end

int_31h:
      cmp AH, 3
      mov DX, offset end_int_msg
      call print

load_end:
      pop DX
      pop CX
      pop BX
      pop AX
      ret
load ENDP

path PROC
      push AX
      push BX
      push CX
      push DX
      push DI
      push SI
      push ES

      mov AX, psp
      mov ES, AX
```

```
      mov ES, ES:[2ch]
      mov BX, 0

findz:
      inc BX
      cmp byte ptr ES:[BX-1], 0
      jne findz
      cmp byte ptr ES:[BX+1], 0
      jne findz

      add BX, 2
      mov DI, 0

_loop:
      mov dl, ES:[BX]
      mov byte ptr [cmd_line_pos + DI], dl
      inc DI
      inc BX
      cmp dl, 0
      je end_loop
      cmp dl, '\'
      jne _loop
      mov CX, DI
      jmp _loop
end_loop:
      mov DI, CX
      mov SI, 0

_fn:
      mov dl, byte ptr [filename + SI]
      mov byte ptr [cmd_line_pos + DI], dl
      inc DI
      inc SI
      cmp dl, 0
      jne _fn


      pop ES
      pop SI
      pop DI
      pop DX
      pop CX
      pop BX
      pop AX
      ret
path ENDP

main PROC FAR
      push DS
      xor AX, AX
      push AX
      mov AX, DATA
      mov DS, AX
      mov psp, ES
      call free_mem
      cmp mem_flag, 0
      je main_end
      call path
```

```
      call load
main_end:
      xor AL, AL
      mov AH, 4Ch
      int 21h

global_end:

main ENDP
CODE ENDS
END main
```