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 _____ Афанасьев Н. С.

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.

На первом шаге была написана программа согласно заданию. При работе были использованы/созданы следующие процедуры:

- BYTE_TO_DEC – процедура, описанная в шаблоне, для печати десятичного числа в строку.
- PRINT – процедура для вывода строки, отступ на которую содержится в DX, на экран, используя функцию 09h прерывания 21h.
- FREE_UP_MEM – процедура для освобождения памяти, не используемой программой, с использованием функции 4Ah прерывания 21h. Если произошла ошибка, то выводится сообщение об ошибке.
- CREATE_PARAMETER_BLOCK – процедура для создания блока параметров. В качестве сегментного адреса указан 0, чтобы вызываемая программа наследовала среду вызывающей программы. В качестве сегмента и отступа командной строки указан PSP:80h, то есть командная строка вызывающей программы.
- COMPOSE_FILEPATH – процедура для подготовки строки с полным путём к файлу. Для начала записывается путь к вызываемой программе из переменных среды, затем ищется первый ‘\’ с конца, после этого с этой позиции записывается имя необходимого файла.

- START_MODULE – процедура для запуска программы через функцию 4B00h прерывания int 21h. Перед вызовом в регистры записываются отступы на сформированные блок параметров и строку с путём к файлу. Затем выводится информация о завершении программы. Если программа завершилась нормально, то считывается и выводится код нажатой клавиши.

Так как в DOSBox существуют проблемы с обработкой прерывания Ctrl+C, то тестирование проводилось на виртуальной машине с установленным DOS.

На втором шаге программа запущена, когда текущим каталогом является каталог с разработанными модулями. Был введён символ F. (см. рис. 1).

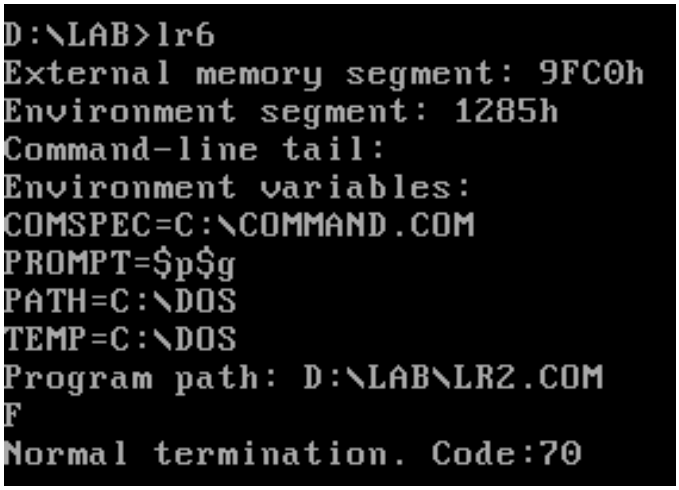

Рисунок 1 – Результат выполнения программы на первом шаге

Как видно из выведенного сообщения, программа завершилась нормально, код символа равен 70 (‘F’).

На третьем шаге программа вновь запущена, когда текущим каталогом является каталог с разработанными модулями. Был введена комбинация Ctrl+C.

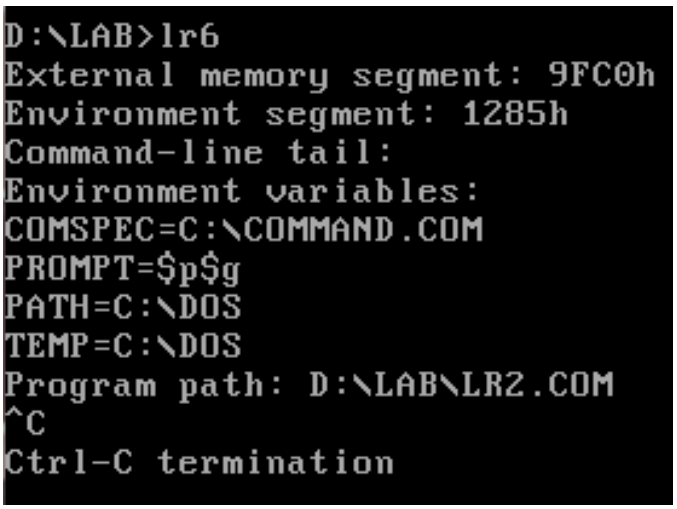

Рисунок 2 – Результат выполнения программы на втором шаге

Как видно из выведенного сообщения, программа завершилась через ввод Ctrl-C.

На четвёртом шаге были проведены те же процедуры из второго и третьего шага, но, когда текущий каталог отличается от каталога с файлами.

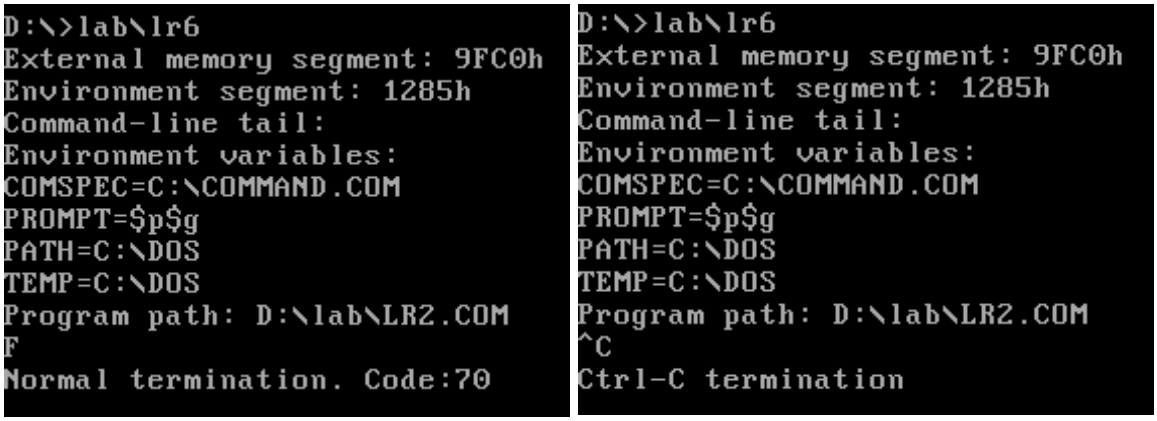

Рисунок 3 – Результаты выполнения программы на четвёртом шаге

Как видно из результатов, результаты совпадают с полученными ранее, то есть выбор текущего каталога не влияет на выполнения программы.

На пятом шаге программа была запущена, когда другой модуль находится в другом каталоге (см. рис. 4).

Рисунок 4 – Результаты выполнения программы на пятом шаге

Как видно из результата, возникла ошибка при запуске модуля из основной программы. Объясняется это тем, что модуль ищется в том же каталоге, что и исходный, но там его нет.

Программный код см. в Приложении А

Вопросы.

1) Как реализовано прерывание Ctrl-C?

   – При нажатии 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 функции 4Ch прерывания int 21h, которая завершает работу программы.

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

   – В данном случае после функции 01h прерывания int 21h, то есть при ожидании ввода символа.

Выводы.

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

ПРИЛОЖЕНИЕ А
ИСХОДНЫЕ КОДЫ ПРОГРАММ

Название файла: lr6.asm

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

DATA SEGMENT
     PARAMETERS DB 14 dup(0)
     FILEPATH   DB 64 dup(0)
     FILENAME   DB "LR2.COM", 0
     MEM_ERROR  DB 'Memory freeing error',13,10,'$'
     LOAD_ERROR DB 'Program loading error',13,10,'$'
     EXIT_0          DB 13,10,'Normal termination. Code:    ',13,10,'$'
     EXIT_1          DB 'Ctrl-C termination',13,10,'$'
     EXIT_2          DB 'Device error termination',13,10,'$'
     EXIT_3          DB 'Function 31h termination',13,10,'$'
     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_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
```

```
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_FILEPATH PROC NEAR
      push DX
      push DI
      push SI
      push ES

      mov ES, ES:[2Ch]
      mov SI, offset FILEPATH
      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 FILENAME
      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_FILEPATH ENDP

START_MODULE 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 FILEPATH
      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 EXIT_0
      cmp AH, 0
      je read_key
```

```
            mov DX, offset EXIT_1
            cmp AH, 1
            je print_exit_info
            mov DX, offset EXIT_2
            cmp AH, 2
            je print_exit_info
            mov DX, offset EXIT_3
            cmp AH, 3
            je print_exit_info

            read_key:
            mov SI, DX
            add SI, 29
            call BYTE_TO_DEC

            print_exit_info:
            call PRINT

            pop ES
            pop SI
            pop DX
            pop BX
            pop AX
            ret
      START_MODULE 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_FILEPATH
            call START_MODULE

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

end_address:
CODE ENDS
end Main
```

Название файла: lr2.asm

```
CODE SEGMENT
      ASSUME CS:CODE, DS:CODE, ES:NOTHING, SS:NOTHING
      ORG 100H
      START: jmp BEGIN

      MEM_SEG     DB "External memory segment:     h",0DH,0AH,'$'
      ENV_SEG  DB "Environment segment:     h",0DH,0AH,'$'
      CMD_TAIL DB "Command-line tail:",'$'
      ENV_VARS DB "Environment variables: ",0DH,0AH,'$'
      PR_PATH     DB "Program path: ",'$'
```

```
NEWLINE     DB 0DH,0AH,'$'

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

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

PRINT_INFO PROC NEAR
      push AX
      push CX
      push DX
      push DI
      push ES
;External memory segment
      mov DX, offset MEM_SEG
      mov DI, DX
      add DI, 28
```

```
      mov AX, CS:[2]
      call WRD_TO_HEX
      call PRINT
;Environment segment
      mov DX, offset ENV_SEG
      mov DI, DX
      add DI, 24
      mov AX, CS:[2Ch]
      call WRD_TO_HEX
      call PRINT
;Command-line tail
      mov DX, offset CMD_TAIL
      call PRINT
      xor CX,CX
      mov CL, CS:[80h]
      cmp CL, 0
      mov AH, 02h
      je lend
      mov DI, 81h
      lstart:
            mov DL, CS:[DI]
            int 21h
            inc DI
            loop lstart
      lend:
      mov DX, offset NEWLINE
      call PRINT
;Environment variables
      mov DX, offset ENV_VARS
      call PRINT
      mov DX, CS:[2Ch]
      mov ES, DX
      mov DI, 0
      _next:
            mov DL, ES:[DI]
      _print:
            int 21h
            inc DI
            cmp DL, 0
            jne _next
            mov DX, offset NEWLINE
            call PRINT
            mov DL, ES:[DI]
            cmp DL, 0
            jne _print
;Program path
      mov DX, offset PR_PATH
      call PRINT
      add DI, 3
      __next:
            mov DL, ES:[DI]
            int 21h
            inc DI
            cmp DL, 0
            jne __next
      mov DX, offset NEWLINE
      call PRINT
      pop ES
```

```
            pop DI
            pop DX
            pop CX
            pop AX
            ret
      PRINT_INFO ENDP

      BEGIN:
            call PRINT_INFO

            mov AH, 1
            int 21h

            mov AH,4Ch
            int 21H
CODE ENDS
END START
```