МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В. И. УЛЬЯНОВА (ЛЕНИНА)
КАФЕДРА МО ЭВМ

ОТЧЕТ

по лабораторной работе №5

по дисциплине «Операционные системы»

Тема: Сопряжение стандартного и пользовательского обработчиков прерываний.

Студент гр. 0382 _____ Афанасьев Н. С.

Преподаватели _____ Ефремов М.А.

Санкт-Петербург

2022

Цель работы.

Исследование возможности встраивания пользовательского обработчика прерываний в стандартный обработчик от клавиатуры. Пользовательский обработчик прерывания получает управление по прерыванию (int 09h) при нажатии клавиши на клавиатуре. Он обрабатывает скан-код и осуществляет определенные действия, если скан-код совпадает с определенными кодами, которые он должен обрабатывать. Если скан-код не совпадает с этими кодами, то управление передается стандартному прерыванию.

Задание.

Шаг 1. Для выполнения лабораторной работы необходимо написать и отладить программный модуль типа .EXE, который выполняет такие же функции, как в программе ЛР 4, а именно:

1) Проверяет, установлено ли пользовательское прерывание с вектором 09h.

2) Если прерывание не установлено то, устанавливает резидентную функцию для обработки прерывания и настраивает вектор прерываний. Адрес точки входа в стандартный обработчик прерывания находится в теле пользовательского обработчика. Осуществляется выход по функции 4Ch прерывания int 21h.

3) Если прерывание установлено, то выводится соответствующее сообщение и осуществляется выход по функции 4Ch прерывания int 21h.

Выгрузка прерывания по соответствующему значению параметра в командной строке /un. Выгрузка прерывания состоит в восстановлении стандартного вектора прерываний и освобождении памяти, занимаемой резидентом. Затем осуществляется выход по функции 4Ch прерывания int 21h.

Для того чтобы проверить установку прерывания, можно поступить следующим образом. Прочитать адрес, записанный в векторе прерывания. Предположим, что этот адрес указывает на точку входа в установленный резидент. На определенном, известном смещении в теле резидента

располагается сигнатура, некоторый код, который идентифицирует резидент. Сравнив известное значение сигнатуры с реальным кодом, находящимся в резиденте, можно определить, установлен ли резидент. Если значения совпадают, то резидент установлен. Длину кода сигнатуры должна быть достаточной, чтобы сделать случайное совпадение маловероятным.

Программа должна содержать код устанавливаемого прерывания в виде удаленной процедуры. Этот код будет работать после установки при возникновении прерывания. Он должен выполнять следующие функции:

1) Сохранить значения регистров в стеке при входе и восстановить их при выходе.

2) При выполнении тела процедуры анализируется скан-код.

3) Если этот код совпадает с одним из заданных, то требуемый код записывается в буфер клавиатуры.

4) Если этот код не совпадает ни с одним из заданных, то осуществляется передача управления стандартному обработчику прерывания.

Шаг 2. Запустите отлаженную программу и убедитесь, что резидентный обработчик прерывания 09h установлен. Работа прерывания проверяется введением различных символов, обрабатываемых установленным обработчиком и стандартным обработчиком.

Шаг 3. Также необходимо проверить размещение прерывания в памяти. Для этого запустите программу ЛР 3, которая отображает карту памяти в виде списка блоков МСВ. Полученные результаты поместите в отчет.

Шаг 4. Запустите отлаженную программу еще раз и убедитесь, что программа определяет установленный обработчик прерываний. Полученные результаты поместите в отчет.

Шаг 5. Запустите отлаженную программу с ключом выгрузки и убедитесь, что резидентный обработчик прерывания выгружен, то есть сообщения на экран не выводятся, а память, занятая резидентом освобождена. Для этого также следует запустить программу ЛР 3. Полученные результаты поместите в отчет.

Шаг 6. Ответьте на контрольные вопросы.

Выполнение работы.

Шаг 1.

При работе были использованы/созданы следующие процедуры:

- TETR_TO_HEX, BYTE_TO_HEX, WRD_TO_HEX – процедуры, описанные в шаблоне, для перевода двоичных кодов в символы шестнадцатеричных чисел.
- PRINT – процедура для вывода в консоль строки, отступ на которую содержится в DX, на экран, используя функцию 09h прерывания 21h.
- IS_LOADED – процедура для проверки того, что пользовательский обработчик прерывания уже был установлен. Для этого сначала загружается вектор прерывания 09h через функцию 35h прерывания 21h, а затем проверяется сигнатура, установленная в созданном обработчике прерывания.
- LOAD_INT – процедура для установки обработчика прерывания. Сначала сохраняется оригинальный вектор прерывания, полученный через функцию 35h прерывания 21h. Затем устанавливается пользовательский обработчик прерывания через функцию 25h прерывания 21h. Затем используется функция 31h прерывания 21h, чтобы оставить процедуру прерывания резидентной в памяти.
- UNLOAD_INT – процедура для выгрузки пользовательского прерывания и возвращение оригинального, который запоминался на этапе установки, через функции 35h и 25h прерывания 21h. Также происходит освобождение памяти через функцию 49h прерывания 21h.
- CHECK_CMD_KEY – процедура для проверки ввода ключа “/un” при вызове программы через проверку байтов в блоке PSP, содержащего хвост командной строки.

INTERRUPT – процедура, являющаяся обработчиком прерывания. При каждом вызове сначала проверяет, нажат ли CapsLock (значение 9-го бита в байтах состояния клавиатуры – в 0040:0017h). Затем считывается скан-код

клавиши из порта 60h, если это не тильда, то управление передаётся стандартному обработчику прерывания. Если нажата тильда, то обработчик сначала посылает сигнал подтверждения микропроцессору клавиатуры через порт 61h, а затем посимвольно выводит строку “hello world”, повторяя её заново, если был достигнут конец строки, регистр символов определяется состоянием CapsLock’a. Символы записываются в буфер клавиатуры с помощью функции 05h прерывания 16h, если был достигнут конец буфера, то запись происходит в начало.

В процедуре MAIN вызываются вышеописанные процедуры при выполнении соответствующих условий (необходимость загрузки или выгрузки прерывания) и выводятся соответствующие информативные сообщения.

Шаг 2.

Программа была запущена, результаты проверены (см. рис. 1). Сначала вводились обычные символы, потом была зажата тильда, затем был включён и выключен CapsLock.

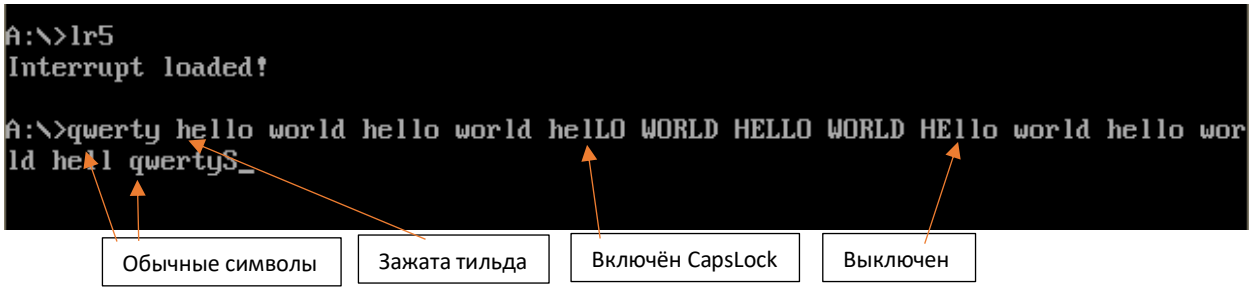

Рис.1 – Выполнение программы

Шаг 3.

Для проверки размещения прерывания в памяти была использована программа из предыдущей лабораторной работы (см. рис. 2).

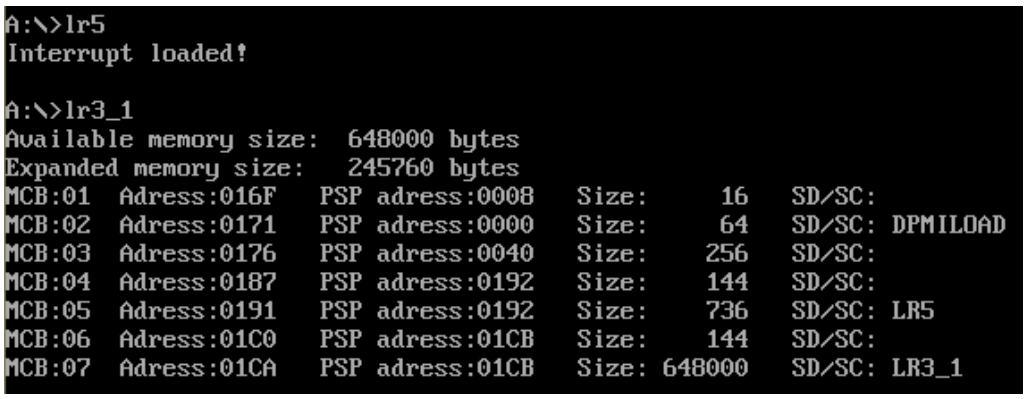

Рис.2 – Проверка размещения в памяти

Как видно из результатов размещение прерывания в памяти подтверждено (прерывание занимает 2 блока общим размером в 880 байт).

Шаг 4.

Программа была запущена несколько раз: 2 раза без ключа и один раз с ключом (см. рис. 2).

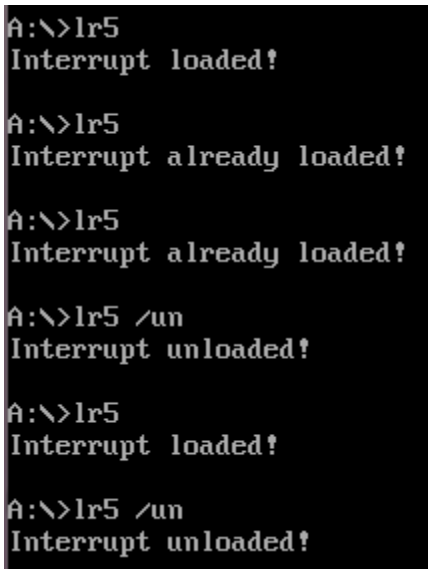

Рис.3 – Повторное выполнение программы: с ключом и без

Как можно заметить программа видит установленное прерывание и не устанавливает его повторно. При запуске с ключом программа восстанавливает оригинальный вектор прерывания.

Шаг 5.

После выгрузки прерывания была вновь запущена программа из прошлой лабораторной работы (см. рис. 3)

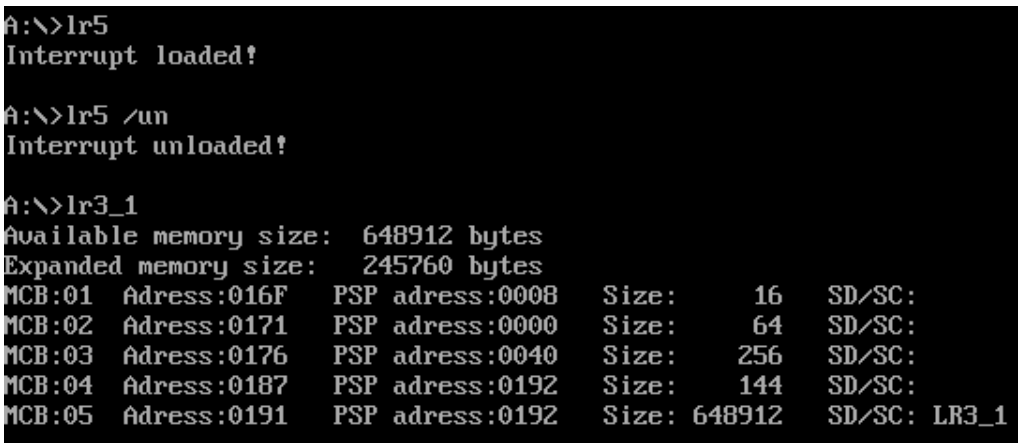

Рис.4 – Выгрузка прерывания и удаление из памяти

Как можно видеть, в памяти не осталось блоков, связанных с пользовательским прерыванием.

Программный код см. в Приложении А

Вопросы.

1) Какого типа прерывания использовались в работе?

   – int 16h (сервис клавиатуры – функция BIOS), int 21h (функции DOS), в том числе пользовательское прерывание, сопряжённое со стандартным по вектору 09h.

2) Чем отличается скан код от кода ASCII?

   – Скан-код – это код клавиши на клавиатуре, а код ASCII – это код символа из таблицы ASCII.

Выводы.

В ходе выполнения лабораторной работы был построен обработчик прерывания при нажатии клавиши на клавиатуре. Построенный пользовательский обработчик может передавать управление стандартному обработчику прерывания, если код клавиши не соответствуют необходимому скан-коду.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЕ КОДЫ ПРОГРАММ

Название файла: lr5.asm

```
CODE SEGMENT
     ASSUME CS:CODE, DS:DATA, SS:ASTACK

     INTERRUPT PROC FAR
          jmp int_start
          PSP              DW ?
          KEEP_IP    DW ?
          KEEP_CS    DW ?
          KEEP_SS    DW ?
          KEEP_SP    DW ?
          INT_ID     DW 0ABCDh
          STRING           DB 'hello world $'
          STR_INDEX  DB 0
          CAPS_LOCK  DB 0
          REQ_KEY          DB 41
          INT_STACK  DW 128 dup(?)
          STACK_TOP  DW ?

          int_start:
          mov KEEP_SS, SS
        mov KEEP_SP, SP
        mov SP, CS
        mov SS, SP
        mov SP, OFFSET STACK_TOP
          push AX
        push BX
        push CX
          push ES

          mov CAPS_LOCK, 0
        mov AX, 40h
        mov ES, AX
        mov AX, ES:[17h]
        and AX, 1000000b
        cmp AX, 0h
        je read_scan_code
        mov CAPS_LOCK, 1

          read_scan_code:
          in AL, 60h
          cmp AL, REQ_KEY
          je signal_to_keyboard
          call dword ptr CS:KEEP_IP
          jmp int_end

          signal_to_keyboard:
          in AL, 61h
          mov AH, AL
          or AL, 80h
          out 61h, AL
```

```
      xchg AH, AL
      out 61h, AL
      mov AL, 20h
      out 20h, AL

      print_letter:
      xor BX, BX
      mov BL, STR_INDEX
      mov AH, 05h
      mov CL, STRING[BX]
      cmp CL, '$'
      jne check_caps
      mov BL, 0
      mov Cl, STRING[0]

      check_caps:
      cmp CAPS_LOCK, 0b
      je to_buffer
      cmp CL, ' '
      je to_buffer
      add CL, -32

      to_buffer:
      mov CH, 00h
      int 16h
      or AL, AL
      jnz reset_buffer
      inc BL
      mov STR_INDEX, BL
      jmp int_end

      reset_buffer:
      mov AX, 40h
      mov ES, AX
      mov AX, ES:[1Ah]
      mov ES:[1Ch], AX
      jmp print_letter

      int_end:
      pop ES
   pop CX
   pop BX
      pop AX
   mov SP, KEEP_SS
   mov SS, SP
   mov SP, KEEP_SP
   mov AL, 20h
      out 20h, AL
   iret
      int_last_byte:
INTERRUPT ENDP

IS_LOADED PROC NEAR
      push AX
      push BX
   push DX
   push SI
```

```
   mov FLAG, 1
   mov AH, 35h
   mov AL, 09h
   int 21h
   mov SI, OFFSET INT_ID
   sub SI, OFFSET INTERRUPT
   mov DX, ES:[BX+SI]
   cmp DX, 0ABCDh
   je loaded
   mov FLAG, 0

      loaded:
   pop SI
   pop DX
      pop BX
   pop AX
      ret
IS_LOADED ENDP

LOAD_INT PROC NEAR
      push DS
      push ES
      push AX
      push BX
      push CX
      push DX
      MOV AH, 35h
      MOV AL, 09h
      INT 21h
      MOV KEEP_IP, BX
      MOV KEEP_CS, ES
      mov DX, offset INTERRUPT
      mov AX, seg INTERRUPT
      mov DS, AX
      mov AH, 25h
      mov AL, 09h
      int 21h
      mov DX, offset int_last_byte
      mov CL,4
      shr DX,CL
      inc DX
      mov AX, CS
   sub AX, PSP
   add DX, AX
   xor AX, AX
      mov AH,31h
      int 21h

      pop DX
      pop CX
      pop BX
      pop AX
      pop ES
      pop DS
      ret
LOAD_INT ENDP

UNLOAD_INT PROC NEAR
```

```
      push DS
      push ES
      push AX
      push BX
      push DX

      cli
      mov AH, 35h
      mov AL, 09h
      int 21h
      mov DX, ES:[offset KEEP_IP]
      mov AX, ES:[offset KEEP_CS]
      mov DS, AX
      mov AH, 25h
      mov AL, 09h
      int 21h

      mov AX, ES:[offset PSP]
      mov ES, AX
      mov DX, ES:[2ch]
      mov AH, 49h
      int 21h
      mov ES, DX
      mov AH, 49h
      int 21h
      sti

      pop DX
      pop BX
      pop AX
      pop ES
      pop DS
      ret
UNLOAD_INT ENDP

CHECK_CMD_KEY PROC NEAR
      push AX

      mov FLAG, 0
   mov AL, ES:[82h]
   cmp AL, '/'
   jne no_key
   mov AL, ES:[83h]
   cmp AL, 'u'
   jne no_key
   mov AL, ES:[84h]
   cmp AL, 'n'
   jne no_key
   mov FLAG, 1

      no_key:
   pop     AX
   ret
CHECK_CMD_KEY ENDP

PRINT PROC NEAR
      push AX
      mov AH, 09h
```

```
            int 21h
            pop AX
            ret
      PRINT ENDP

      main PROC FAR
            push DS
            xor AX, AX
            mov AX, DATA
         mov DS, AX

         mov PSP, ES
         call CHECK_CMD_KEY
         cmp FLAG, 1
         je int_unload

         call IS_LOADED
         cmp FLAG, 0
         je int_load
         mov DX, OFFSET ALREADY_LOADED_MSG
         call PRINT
         jmp exit

            int_load:
            mov DX, OFFSET LOADED_MSG
         call PRINT
         call LOAD_INT
         jmp exit

            int_unload:
         call IS_LOADED
         cmp FLAG, 0
         je unloaded
         call UNLOAD_INT
            unloaded:
         mov DX, OFFSET UNLOADED_MSG
         call PRINT

            exit:
            pop DS
         mov AH, 4Ch
         int 21h
      main ENDP
CODE ENDS

ASTACK SEGMENT STACK
      DW 128 DUP(?)
ASTACK ENDS

DATA SEGMENT
      FLAG               DB 0
      LOADED_MSG         DB 'Interrupt loaded!',0DH,0AH,'$'
      UNLOADED_MSG       DB 'Interrupt unloaded!',0DH,0AH,'$'
      ALREADY_LOADED_MSG DB 'Interrupt already loaded!',0DH,0AH,'$'
DATA ENDS
END main
```