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 обработчика
прерываний.

Студент гр. 0382 Гудов Н.Р.

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.

Программный код см. в Приложении А

Шаг 1

Написан и отлажен программный модуль типа .EXE, выполняющий поставленные задачи:

1) Проверка установки пользовательского прерывания.
2) Установка резидентной функции для обработки прерывания.
3) Вывод сообщения, в случае, если прерывание уже установлено.
4) Выгрузка прерывания по значению параметра командной строки.

Разработаны следующие функции:

PRINT- процедура печати строки, с помощью функции 9h прерывания 21h.

INTER- обработчик прерывания. Проверяет была ли нажата соответствующая клавиша(LShift) путем считывания скан-кода клавиши из порта и циклически выводит символы алфавита на экран.

IS_LOADED- Прочитывает адрес из вектора прерывания. Сравнивает известное значение сигнатуры с реальным кодом в резиденте.

LOAD_INT- устанавливает обработчик прерывания, предварительно сохраняя первоначальный. Для выполнения этих действий используются функции 35h и 25 h прерывания 21h.

UNLOAD_INT- Возвращает первоначальный вариант прерывания. Для выполнения этих действий используются функции 35h и 25 h прерывания 21h.

CHECK- проверка установки прерывания.

Шаг 2

Зафиксируем информацию о состоянии блоков MCB перед началом работы программы.(рис1)

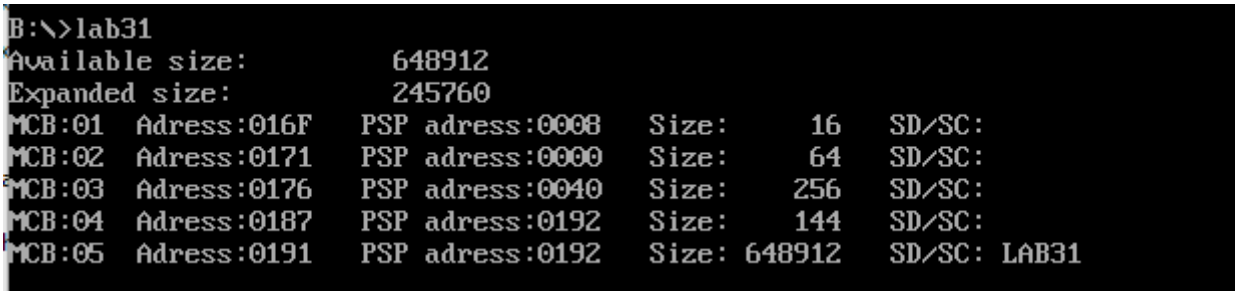

Рисунок 1

Далее запустим программу с пользовательским прерыванием. После нее снова выведем информацию о блоках MCB.(рис2)

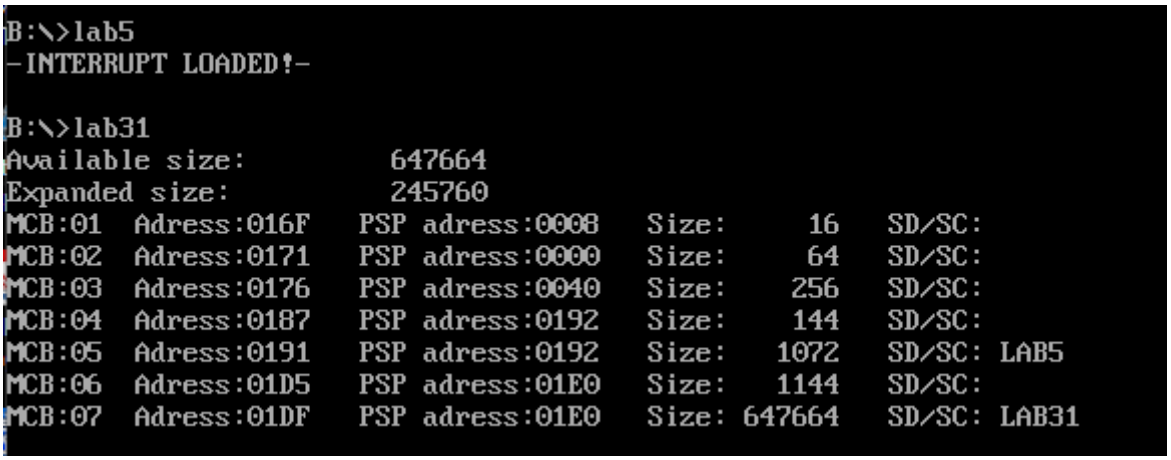

Рисунок 2

Шаг 3

Запустим программу заново и проверим функционал обработчика прерывания.(рис3)

Рисунок 3

Шаг 4

Вернем исходное прерывание и посмотрим на состояние блоков MCB.(рис4)

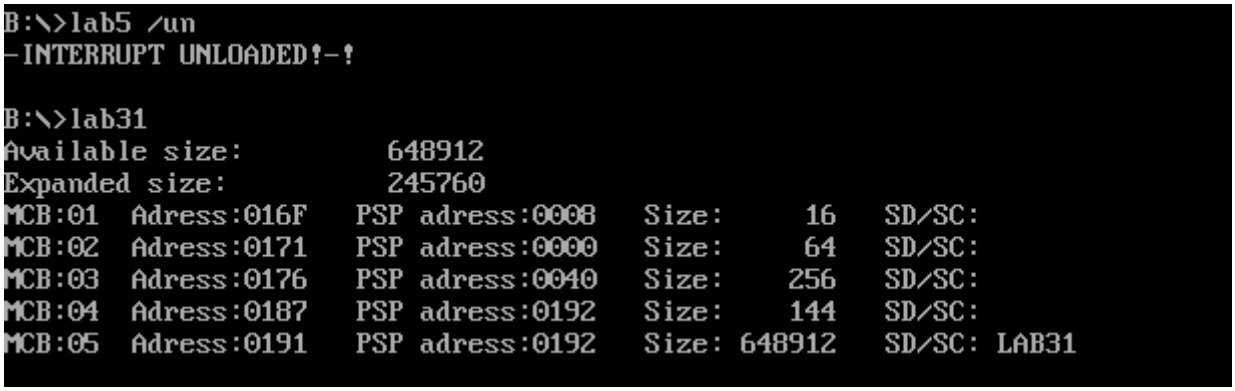

Рисунок 4

Заметим, что блоки, хранящие пользовательское прерывания удалены – прерывание выгружено.

Вопросы.

1) Какого типа прерывания использовались в работе?

   В работе использовались аппаратные прерывания int 16h, int 09h и программное int 21h

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 прерываний сигналов таймера, который выводит информацию о количестве вызовов на экран. Изучен механизм прерываний в DOS.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЕ КОДЫ ПРОГРАММ

Название файла: lab5.asm

```
Astack SEGMENT STACK
   DW 128 DUP(?)
Astack ENDS

DATA SEGMENT
   flag       DB 0
   msg_loaded         DB '-INTERRUPT LOADED!-',0DH,0AH,'$'
   msg_unloaded       DB '-INTERRUPT UNLOADED!-!',0DH,0AH,'$'
   msg_already DB '-INTERRUPT ALREADY LOADED!-',0DH,0AH,'$'
DATA ENDS

CODE SEGMENT
   ASSUME CS:CODE, DS:DATA, SS:Astack

    ;ПРОЦЕДУРЫ
    ;-----------------------------------------------------
    PRINT PROC NEAR
         push AX
         mov AH, 09h
         int 21h
         pop AX
         ret
   PRINT ENDP
   ;-----------------------------------------------------
   INTER 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 'a b c d e f $'
         STR_INDEX  DB 0
         CAPS_LOCK  DB 0
         REQ_KEY          DB 2Ah
         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 ;взять значение порта управления клавиатурой
mov AH, AL ;сохранить его
or AL, 80h ;установить бит разрешения для клавиатуры
out 61h, AL ;и вывести его в управляющий порт
xchg AH, AL ;извлечь исходное значение порта
out 61h, AL ;и записать его обратно
mov AL, 20h ;послать сигнал "конец прерывания"
out 20h, AL ;контроллеру прерываний 8259

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:
INTER 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 INTER
     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 INTER
      mov AX, seg INTER
      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 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 ENDP
;-------------------------------------------------------
main PROC FAR
      push DS
      xor AX, AX
      mov AX, DATA
     mov DS, AX

     mov PSP, ES
     call CHECK
     cmp flag, 1
     je int_unload
     call IS_LOADED
     cmp flag, 0
     je int_load
     mov DX, OFFSET msg_already
     call PRINT
     jmp final

      int_load:
      mov DX, OFFSET msg_loaded
     call PRINT
     call LOAD_INT
     jmp final

      int_unload:
     call IS_LOADED
     cmp flag, 0
     je unloaded
     call UNLOAD_INT
      unloaded:
     mov DX, OFFSET msg_unloaded
     call PRINT

      final:
      pop DS
     mov AH, 4Ch
     int 21h
main ENDP
```

```
CODE ENDS

END main
```