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

ОТЧЕТ

по лабораторной работе №5
по дисциплине «Операционные системы»
Тема: Сопряжение стандартного и пользовательского обработчиков прерываний

Студент гр. 9383 _____ Камзолов Н.А.

Преподаватель _____ Ефремов М.А.

Санкт-Петербург
2021

Постановка задачи.

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.

Результаты исследования проблем.

Шаг 1. Был написан и отлажен программный модуль типа .EXE, который выполняет, прописанные в задании функции.

Шаг 2. Была запущена программа, и можно было убедиться, что обработчик прерываний работает верно.

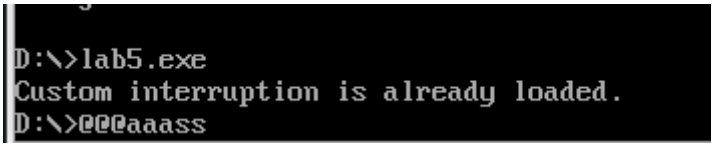

Рисунок 1 – Демонстрация корректной работы резидентного обработчика прерываний (сочетание клавиш ctrl+a меняется на @).

Шаг 3. Проверено размещение прерывания в памяти.

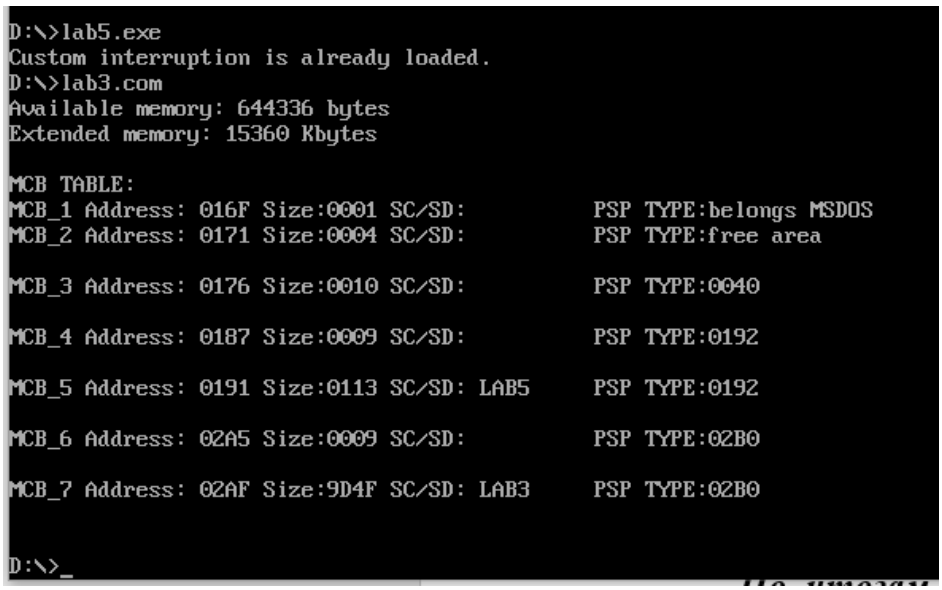

Рисунок 2 – Демонстрация корректного отображения обработчика прерываний в памяти(5-ая строка таблицы MCB).

Шаг 4. Программа была повторно запущена, чтобы удостовериться, что программа определяет установленный обработчик прерываний.

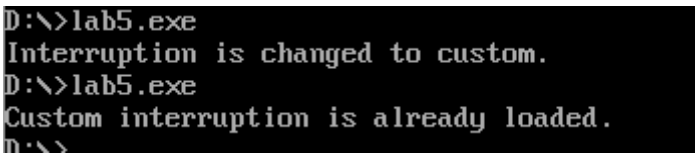

Рисунок 3 – Демонстрация корректного определения установленного обработчика прерывания при повторном запуске программы.

Шаг 5. Программа была запущена с ключом выгрузки, чтобы убедиться, что резидентный обработчик прерывания выгружен и память, занятая резидентом освобождена.

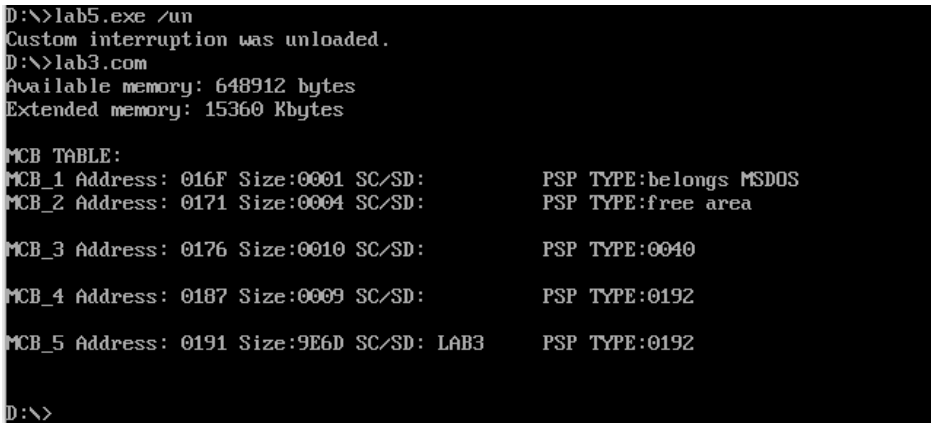

Рисунок 4 – Демонстрация корректной выгрузки резидентного обработчика прерываний.

По итогам выполнения работы можно ответить на контрольные вопросы:

1. Какого типа прерывания использовались в работе?

Ответ: 09h и 16h – аппаратное прерывание, 10h и 21h – программное прерывание.

2. Чем отличается скан-код от кода ASCII?

Ответ: Скан-код – уникальное число, однозначно определяющее нажатую клавишу, но не ASCII-код. ASCII код – это код символа ASCII таблицы.

Выводы.

Исследованы возможности встраивания пользовательского обработчика прерываний в стандартный обработчик от клавиатуры.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

lab5.asm:

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

DATA  SEGMENT
    ROUT_LOADED db "Custom interruption is already loaded.$"
    ROUT_IS_LOADING db "Interruption is changed to custom.$"
    ROUT_IS_NOT_LOADED db "Default interruption is set and can't be
unloaded.$"
    ROUT_IS_UNLOADED db "Custom interruption was unloaded.$"
DATA  ENDS

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

PRINT_BUF proc near
    push ax
    mov ah, 9h
    int 21h
    pop ax
    ret
PRINT_BUF endp


start_rout:
ROUT proc far
    jmp start_proc
    key_value db 0
    KEEP_PSP dw 0
    KEEP_IP dw 0
      KEEP_CS dw 0
    KEEP_SS DW 0
      KEEP_SP DW 0
      KEEP_AX DW 0
    ROUT_INDEX dw 1337h
    TIMER_COUNTER db 'Timer: 0000$'
    BStack DW 64 DUP(?)
start_proc:
    mov KEEP_SP, sp
    mov KEEP_AX, ax
    mov KEEP_SS, ss

    mov ax, seg BStack
    mov ss, ax
    mov ax, offset start_proc
    mov sp, ax

    mov ax, KEEP_AX


    push ax
    push bx
```

```
    push cx
    push dx
    push si
    push es
    push ds

     mov cx, 040h
     mov es, cx
     mov cx, es:[0017h]

    and cx, 0100b
    jz standart_interruption

    in al, 60h
    cmp al, 1Eh
    je do_req

standart_interruption:
    call dword ptr cs:[KEEP_IP]
    jmp restore_registers
do_req:
    in al, 61h
    mov ah, al
    or al, 80h
    out 61h, al
    xchg ah, al
    out 61h, al
    mov al, 20H
    out 20h, al

write_symbol:
    mov ah, 05h
    mov cl, '@'
    mov ch, 00h
    int 16h
    or al, al
    jnz skip
    jmp restore_registers
skip:
    mov al, es:[001Ah]
    mov es:[001Ch], al
    jmp write_symbol

restore_registers:
    pop ds
    pop es
    pop   si
    pop dx
    pop cx
    pop bx
    pop ax


    mov sp, KEEP_SP
    mov ax, KEEP_SS
    mov ss, ax
    mov ax, KEEP_AX
```

```
    mov al, 20H
    out 20H, al

    iret
end_rout:
ROUT endp


IF_NEED_UNLOAD proc near
     push ax
    push es

     mov al,es:[81h+1]
     cmp al,'/'
     jne end_if_need_unload

     mov al,es:[81h+2]
     cmp al,'u'
     jne end_if_need_unload

     mov al,es:[81h+3]
     cmp al,'n'
     jne end_if_need_unload

    mov cl,1h

end_if_need_unload:
    pop es
     pop ax
     ret
IF_NEED_UNLOAD endp


LOAD_ROUT PROC near
     push ax
     push dx

    mov KEEP_PSP, es

     mov ah,35h
     mov al,09h
     int 21h
    mov KEEP_IP, bx
    mov KEEP_CS, es

     push ds
     lea dx, ROUT
     mov ax, SEG ROUT
     mov ds,ax
     mov ah,25h
     mov al,09h
     int 21h
     pop ds

     lea dx, end_rout
     mov cl,4h
```

```
      shr dx,cl
      inc dx
      add dx,100h
     xor ax, ax
      mov ah,31h
      int 21h

      pop dx
      pop ax
      ret
LOAD_ROUT endp

UNLOAD_ROUT PROC near
      push ax
      push si

     cli
      push ds
      mov ah,35h
      mov al,09h
     int 21h

     mov si,offset KEEP_IP
     sub si,offset ROUT
     mov dx,es:[bx+si]
      mov ax,es:[bx+si+2]
     mov ds,ax
     mov ah,25h
     mov al,09h
     int 21h
     pop ds

     mov ax,es:[bx+si-2]
     mov es,ax
     push es

     mov ax,es:[2ch]
     mov es,ax
     mov ah,49h
     int 21h

     pop es
     mov ah,49h
     int 21h
     sti

     pop si
     pop ax
     ret
UNLOAD_ROUT endp

IF_LOADED proc near
      push ax
      push si

     push es
     push dx
```

```
      mov ah,35h
      mov al,09h
      int 21h

      mov si, offset ROUT_INDEX
      sub si, offset ROUT
      mov dx,es:[bx+si]
      cmp dx, ROUT_INDEX
      jne end_if_loaded
      mov ch,1h

end_if_loaded:
    pop dx
    pop es
      pop si
      pop ax
      ret
IF_LOADED ENDP

MAIN proc far
    push  DS
    push  AX
    mov   AX,DATA
    mov   DS,AX

    call IF_NEED_UNLOAD
    cmp cl, 1h
    je need_unload

    call IF_LOADED
    cmp ch, 1h
    je print_rout_is_already_set
    mov dx, offset ROUT_IS_LOADING
    call PRINT_BUF
    call LOAD_ROUT
    jmp exit

need_unload:
    call IF_LOADED
    cmp ch, 1h
    jne print_rout_cant_be_unloaded
    call UNLOAD_ROUT
    mov dx, offset ROUT_IS_UNLOADED
    call PRINT_BUF
    jmp exit

print_rout_cant_be_unloaded:
    mov dx, offset ROUT_IS_NOT_LOADED
    call PRINT_BUF
    jmp exit
print_rout_is_already_set:
    mov dx, offset ROUT_LOADED
    call PRINT_BUF
    jmp exit

exit:
```

```
    mov ah, 4ch
    int 21h
MAIN endp
CODE ends
END Main
```