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 _____ Охотникова Г.С.

Преподаватель _____ 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.

1. При выполнении данной лабораторной работы был написан модуль EXE, который выполняет указанные в задании функции.
2. Был запущен модуль lab5.exe, который заменяет символы “q”, “w”, “r” на “&”, “!”, “*”. Можно увидеть, что программа успешно работает.

Рисунок 1 — Результат работы модуля lab5.exe

3. Для проверки размещения прерывания в памяти был запущено модуль COM из лабораторной работы №3, который отображает карту памяти в виде списка блоков MCB.

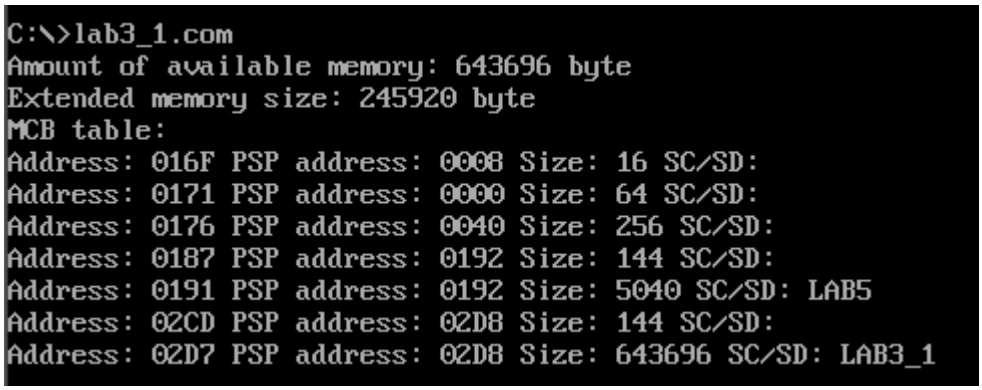

Рисунок 2 — Результат запуска модуля lab3_1.com

4. Был запущен модуль lab5.exe еще раз, чтобы продемонстрировать, что пользовательское прерывание можно загрузить только один раз. Затем был произведен запуск с ключом /un для выгрузки обработчика прерывания и освобождения памяти.

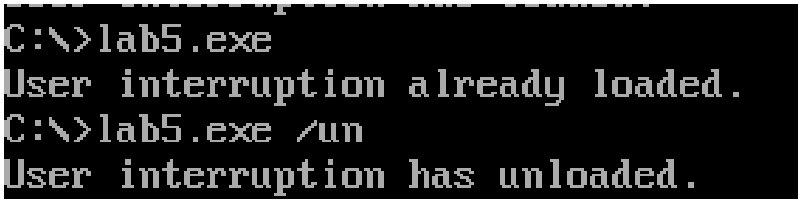

Рисунок 3 — Выгрузка пользовательского обработчика

5. Чтобы проверить, что пользовательское прерывание было действительно выгружено, был еще раз запущен модуль типа COM из третьей лабораторной работы.

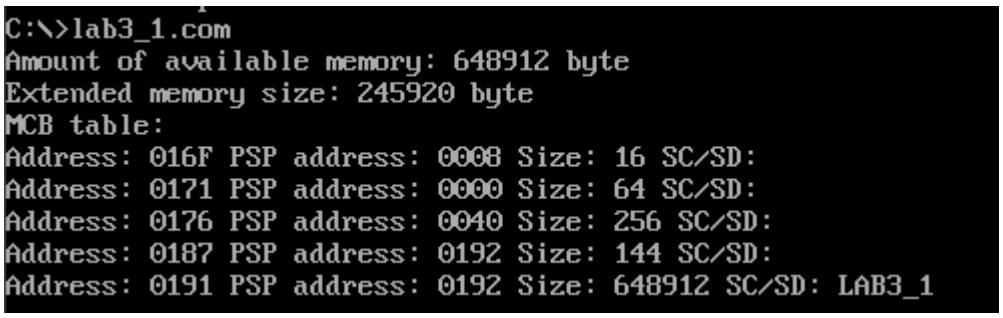

Рисунок 4 — Проверка выгрузки

Исходный программный код см. в приложении А.

Контрольные вопросы.

1. Какого типа прерывания использовались в работе?

Программные (21h) и аппаратные (09h, 16h).

2. Чем отличается скан код от ASCII?

Скан-код — код, который присвоен каждой клавише для того, чтобы определить, какая клавиша была нажата.

ASCII-код — это числовой код, который присвоен некоторому символу в таблице ASCII.

Выводы.

Были исследованы возможности встраивания пользовательского обработчика прерываний в стандартный обработчик от клавиатуры.

ПРИЛОЖЕНИЕ А.

Название файла: lab5.asm

```
AStack SEGMENT  STACK
          DW 256 DUP(?)
AStack ENDS

DATA SEGMENT
      IS_LOAD DB 0
      IS_UNLOAD DB 0
      INTP_LOAD db "User interruption has loaded.$"
      INTP_LOADED db "User interruption already loaded.$"
      INTP_UNLOAD db "User interruption has unloaded.$"
      INTP_NOT_LOADED db "User interruption is not loaded.$"
DATA ENDS

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



PRINT PROC NEAR
    push ax
    mov ah, 09h
    int 21h
    pop ax
      ret
PRINT ENDP



INTERRUPT PROC FAR
jmp start

interrupt_data:
      keep_ip DW 0
      keep_cs DW 0
      keep_psp DW 0
      keep_ax DW 0
      keep_ss DW 0
      keep_sp DW 0
```

```
 intp_stack DW 256 DUP(0)
 sym DB 0
 sign DW 1234h

start:
     mov keep_ax, ax
     mov keep_sp, sp
     mov keep_ss, ss
     mov ax, seg intp_stack
     mov ss, ax
     mov ax, offset intp_stack
     add ax, 256
     mov sp, ax

    push ax
    push bx
     push cx
    push dx
    push si
    push es
    push ds

     mov ax, seg sym
     mov ds, ax

     in al, 60h   ;считывание номера клавиши
    cmp al, 10h    ;скан-символ
    je change_q
    cmp al, 11h
    je change_w
    cmp al, 13h
    je change_r

     pushf
     call dword ptr cs:keep_ip
     jmp end_p

change_q:
      mov sym, '&'
```

```
      jmp next
    change_w:
      mov sym, '!'
         jmp next
    change_r:
         mov sym, '*'

next:                    ;обработка аппартаного прерывания
      in al, 61h
      mov ah, al
      or al, 80h
      out 61h, al
      xchg al, al
      out 61h, al
      mov al, 20h
      out 20h, al

print_sym:
      mov ah, 05h ;запись символа в буфер клавитуры
      mov cl, sym
      mov ch, 00h
      int 16h
      or al, al
      jz end_p
      mov ax, 0040h
      mov es, ax
      mov ax, es:[1ah]
      mov es:[1ch], ax
      jmp print_sym

    end_p:
      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
INTERRUPT endp

END_I:
CHECK_LOAD PROC NEAR
      push ax
      push bx
      push si
      mov ah, 35h
      mov al, 09h
      int 21h

      mov si, offset sign
      sub si, offset INTERRUPT
      mov ax, es:[bx + si]
      cmp ax, sign
      jne load_end
      mov IS_LOAD, 1

load_end:
      pop  si
      pop  bx
      pop  ax
      ret
CHECK_LOAD ENDP


CHECK_UNLOAD PROC NEAR
    push ax
    push es
      mov ax, keep_psp
      mov es, ax
    cmp byte ptr es:[82h], '/'
```

```
    jne check_end
    cmp byte ptr es:[83h], 'u'
    jne check_end
    cmp byte ptr es:[84h], 'n'
    jne check_end
    mov IS_UNLOAD, 1

    check_end:
     pop es
         pop ax
         ret
CHECK_UNLOAD ENDP

INTERRUPT_LOAD PROC NEAR
     push ax
     push bx
     push cx
     push dx
     push ds
     push es

     mov ah, 35h
    mov al, 09h
    int 21h
     mov keep_cs, es
    mov keep_ip, bx
    mov ax, seg INTERRUPT
    mov dx, offset INTERRUPT
    mov ds, ax
    mov ah, 25h
    mov al, 09h

    int 21h

    pop ds
    mov dx, offset END_I
    mov cl, 4h
    shr dx, cl
    add dx, 10fh
```

```
    inc dx
    xor ax, ax
    mov ah, 31h
    int 21h

    pop es
    pop dx
    pop cx
    pop bx
    pop ax
     ret
INTERRUPT_LOAD ENDP


INTERRUPT_UNLOAD PROC NEAR
     cli
    push ax
    push bx
     push dx
    push ds
    push es
    push si

    mov ah, 35h
    mov al, 09h
    int 21h
    mov si, offset keep_ip
    sub si, offset INTERRUPT
    mov dx, es:[bx+si]
    mov ax, es:[bx+si+2]

    push ds
    mov ds, ax
    mov ah, 25h
    mov al, 09h
    int 21h
    pop ds

    mov ax, es:[bx+si+4]
```

```
    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 es
    pop ds
    pop dx
    pop bx
    pop ax
     ret
INTERRUPT_UNLOAD ENDP

MAIN PROC
    push ds
    xor ax, ax
     push ax

    mov ax, data
    mov ds, ax
    mov keep_psp, es

    call CHECK_LOAD
    call CHECK_UNLOAD
    cmp IS_UNLOAD, 1
    je unload
    mov al, IS_LOAD
    cmp al, 1
    jne load
    mov dx, offset INTP_LOADED
    call PRINT
    jmp end_main
```

```
    load:
        mov dx, offset INTP_LOAD
        call PRINT
        call INTERRUPT_LOAD
        jmp  end_main

    unload:
        cmp  IS_LOAD, 1
        jne  not_loaded
        mov dx, offset INTP_UNLOAD
        call PRINT
        call INTERRUPT_UNLOAD
        jmp  end_main

    not_loaded:
        mov  dx, offset INTP_NOT_LOADED
        call PRINT

    end_main:
        xor al, al
        mov ah, 4ch
        int 21h
MAIN ENDP
CODE ENDS
END MAIN
```