МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра математического обеспечения и применения ЭВМ

ОТЧЕТ

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

Студент гр. 8381 _____ Почаев Н.А.

Преподаватель _____ Ефремов М.А.

Санкт-Петербург
2020

Цель работы.

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

Основные теоретические положения.

Пользовательский обработчик прерываний 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 стандартном прерыванию.

Выполнение работы.

Выполнение работы производилось на базе операционной системы Windows XP (32 bit), запускаемой в системе виртуализации VMware Workstation, в редакторе Notepad++. Сборка и отладка модулей производились с помощью компилятора MASM и отладчика AFD. Также в работе был использован консольный файловый менеджер Far Manager и HEX-редактор HxD. Для дополнительного тестирования и проверки функциональности программы использовался DOSBox.

В ходе выполнения данной лаборыторной работы был написан и отлажен программный модуль типа EXE, который выполняет следующие функции:

1. Проверяет установлено ли пользовательское прерывание.
2. Устанавливает резидентную функцию для обработки прерывания, если она еще не установлена.
3. Если она уже остановлена выводится соответствующее сообщение.
4. По нажатию клавиши f1 прерывание будет выводить соответствующие сообщения.
5. Если скан-код не совпадает с данными, то осуществляется передача управления стандартному обработчику прерывания.

Функции, реализованный в программе, приведены в табл. 1.

Таблица 1 – Функции, реализованные в программе

Процедура	Описание
MY_CUSTOM_ITERRUPT	Резидентное прерывание, которое загружается в память и выполняет вывод символа от сообщения при нажатии на f1
WRITE_STRING	Вывод строки на экран
LOAD_FLAG	Проверка на наличия флага “/un”
IS_LOAD	Проверка на загрузку пользовательского прерывания в память
LOAD_ITERRAPT	Сохранение первоначального прерывания и загрузка пользовательского прерывания в память
UNLOAD_ITERRAPT	Выгрузка пользовательского прерывания из памяти, а также освобождение памяти и восстановление первоначальных прерываний
MAIN	Главная функция

1. Пример загрузки прерывания в память приведён на рис. ниже:

```
Z:\>SET BLASTER=A220 I7 D1 H5 T6

Z:\>mount D: D:\
Drive D is mounted as local directory D:\

Z:\>D:

D:\>main.exe
Iterrapt has been loaded
```

2. Пример вывода после нажати кнопки f1 несколько раз приведён на рис. ниже:

```
D:\>It works! 42. It works! 42. It works! 42. It works! 42._
```

3. Расположение в памяти прерывания представлено на рис. ниже:

```
D:\>LR3_1.COM
Availible memory: 648240 B
Extended memory :  15360 KB
 Address | MCB Type | PSP Address  |   Size   |    SD/SC
  016F        4D          0008           16
  0171        4D          0000           64
  0176        4D          0040          256
  0187        4D          0192          144
  0191        4D          0192          496         MAIN
  01B1        4D          01BC          144
  01BB        5A          01BC       648240         LR3_1
```

4. Выгрузка прерывания из памяти приведена на рис. ниже:

```
D:\>MAIN.EXE /un
Iterrapt is unloaded
```

5. Состояние памяти после данного действия приведено на рис. ниже:

```
D:\>LR3_1.COM
Availible memory: 648912 B
Extended memory :  15360 KB
 Address | MCB Type | PSP Address  |   Size   |    SD/SC
  016F        4D          0008           16
  0171        4D          0000           64
  0176        4D          0040          256
  0187        4D          0192          144
  0191        5A          0192       648912         LR3_1
```

Выводы.

В ходе выполнения данной лабораторной работы была исследованы возможности встраивания пользовательского обработчика прерываний в стандартный обработчик от клавиатуры. Программа загружает и выгружает резидент, а также производится проверка флагов и загрузки прерывание в память. С помощью rout при нажатии на клавишу f1 на экран посимвольно выводится строка, определённая в этом прерывании.

Ответы на контрольные вопросы.

1. Какого типа прерывания использовались в работе?

   Был реализован обработчик для аппаратного прерывания (от клавиатуры), в коде программы также использовались программные прерывания (например 21h).

2. Чем отличаетяс скан-код и ASCII код?

   Скан-код – код клавиши, позволящий опознавать нажатые клавиши драйверу клавиатуры. Контроллер пересылает скан-код в порт.

   Код ASCII – это уникальный код для каждого символа.

   Скан код характеризуете клавишу, а код ANCSII – символ.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ.

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

MY_CUSTOM_ITERRUPT PROC far
    jmp custom_interrupt

    PSP dw ?
    KEEP_IP dw 0
    KEEP_CS dw 0
    ITERRUPT_ID dw 8f17h

    STR_ITERRUPT db 'It works! 42. $'

    KEEP_SS dw ?
    KEEP_SP dw ?
    KEEP_AX dw ?
    REQ_KEY db 3Bh
    STR_INDEX db 0
    ITERRUPT_STACK dw 32 dup (?)
    END_IT_STACK dw ?

custom_interrupt:
   mov KEEP_SS,ss
   mov KEEP_SP,sp
   mov KEEP_AX,ax

   mov ax,cs
   mov ss,ax
   mov sp,offset END_IT_STACK

   push bx
   push cx
   push dx

   in al,60h
   cmp al,REQ_KEY
   je m_do_req
   call dword ptr cs:KEEP_IP
   jmp m_iter_end

m_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

    xor bx,bx
    mov bl,STR_INDEX

m_write_s:
   mov ah,05h
   mov cl,STR_ITERRUPT[bx]
   cmp cl,'$'
   je m_str_end
   mov ch,00h
   int 16h
   or al,al
   jnz m_skip
   inc bl
   mov STR_INDEX,bl
   jmp m_iter_end

m_skip:
   mov ax,0C00h
   int 21h
   jmp m_write_s

m_str_end:
   mov STR_INDEX,0

m_iter_end:

      pop dx
      pop cx
      pop bx

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

   iret
m_iterrapt_end:
MY_CUSTOM_ITERRUPT ENDP

WRITE_STRING PROC near
   push AX
```

```
   mov AH,09h
   int 21h
   pop AX
   ret
WRITE_STRING ENDP

LOAD_FLAG PROC near
   push ax

   mov PSP,es
   mov al,es:[81h+1]
   cmp al,'/'
   jne m_load_flag_end
   mov al,es:[81h+2]
   cmp al, 'u'
   jne m_load_flag_end
   mov al,es:[81h+3]
   cmp al, 'n'
   jne m_load_flag_end
   mov flag,1h

m_load_flag_end:
   pop ax
   ret
LOAD_FLAG ENDP

IS_LOAD PROC near
   push ax
   push si

   mov ah,35h
   mov al,1Ch
   int 21h
   mov si,offset ITERRUPT_ID
   sub si,offset MY_CUSTOM_ITERRUPT
   mov dx,es:[bx+si]
   cmp dx, 8f17h
   jne m_is_load_end
   mov flag_load,1h
m_is_load_end:
   pop si
   pop ax
   ret
IS_LOAD ENDP

LOAD_ITERRAPT PROC near
   push ax
```

```
   push dx

   call IS_LOAD
   cmp flag_load,1h
   je m_already_load
   jmp m_start_load

m_already_load:
   lea dx,STR_ALR_LOAD
   call WRITE_STRING
   jmp m_end_load

m_start_load:
   mov AH,35h
      mov AL,1Ch
      int 21h
      mov KEEP_CS, ES
      mov KEEP_IP, BX

    push ds
    lea dx, MY_CUSTOM_ITERRUPT
    mov ax, seg MY_CUSTOM_ITERRUPT
    mov ds,ax
    mov ah,25h
    mov al, 1Ch
    int 21h
    pop ds
    lea dx, STR_SUC_LOAD
    call WRITE_STRING

    lea dx, m_iterrapt_end
    mov CL, 4h
    shr DX,CL
    inc DX
    mov ax,cs
    sub ax,PSP
    add dx,ax
    xor ax,ax
    mov AH,31h
    int 21h

m_end_load:
   pop dx
   pop ax
   ret
LOAD_ITERRAPT ENDP
```

```
UNLOAD_ITERRAPT PROC near
   push ax
   push si

   call IS_LOAD
   cmp flag_load,1h
   jne m_cant_unload
   jmp m_start_unload

m_cant_unload:
   lea dx,STR_IST_LOAD
   call WRITE_STRING
   jmp m_unload_end

m_start_unload:
   CLI
   PUSH DS
   mov ah,35h
      mov al,1Ch
      int 21h
   mov si,offset KEEP_IP
      sub si,offset MY_CUSTOM_ITERRUPT
      mov dx,es:[bx+si]
      mov ax,es:[bx+si+2]
   MOV DS,AX
   MOV AH,25H
   MOV AL, 1CH
   INT 21H
   POP DS
   STI

   lea dx,STR_IS_UNLOAD
   call WRITE_STRING

   mov ax,es:[bx+si-2]
   mov es,ax
   mov ax,es:[2ch]
   push es
   mov es,ax
   mov ah,49h
   int 21h
   pop es
   int 21h

m_unload_end:
   pop si
   pop ax
```

```
   ret
UNLOAD_ITERRAPT ENDP

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

   call LOAD_FLAG
   cmp flag, 1h
   je m_unload_iterrapt
   call LOAD_ITERRAPT
   jmp m_end

m_unload_iterrapt:
   call UNLOAD_ITERRAPT

m_end:
   mov ah,4ch
   int 21h
Main      ENDP
CODE      ENDS

ASTACK    SEGMENT  STACK
   DW 64 DUP(?)
ASTACK    ENDS

DATA      SEGMENT
   flag db 0
   flag_load db 0

   STR_IST_LOAD  DB  'Iterrapt is not load',       0AH, 0DH,'$'
   STR_ALR_LOAD  DB  'Iterrapt is already loaded', 0AH, 0DH,'$'
   STR_SUC_LOAD  DB  'Iterrapt has been loaded',   0AH, 0DH,'$'
   STR_IS_UNLOAD  DB 'Iterrapt is unloaded',       0AH, 0DH,'$'
DATA      ENDS
          END Main
```