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

Студентка гр. 8382 _____ Рочева А.К.

Преподаватель _____ Ефремов М.А.

Санкт-Петербург

2020

Цель работы.

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

Ход выполнения.

Пользовательский обработчик прерывания получает управление по прерыванию (int 09h) при нажатии клавиши на клавиатуре. Он обрабатывает скан-код и, при условии, что нажатая клавиша — это клавиша Home (код 47h) выводит вместо нее символ треугольника (код 1Eh). Остальные символы не изменяются.

В процедуре CHECK_INT проверяется, установлено ли пользовательское прерывание. В процедуре LOAD_INT устанавливается резидентная функция для обработки прерывания, настраивается вектор прерывания. Выгрузка прерывания происходит в процедуре INT_UNLOAD по значению параметра командной строки /un (происходит восстановление стандартного вектора прерываний и освобождение памяти, занимаемой резидентном).

Код программы представлен в приложении А.

На рис. 1 представлен запуск модуля LAB5.EXE (загрузка прерывания) и его проверка.

Рис. 1 — запуск модуля LAB5.EXE и его проверка

На рис. 2 представлен запуск модуля LAB3.COM, выводящий блоки памяти. Как видно, прерывание разместилось в памяти (в блоках 4 и 5).

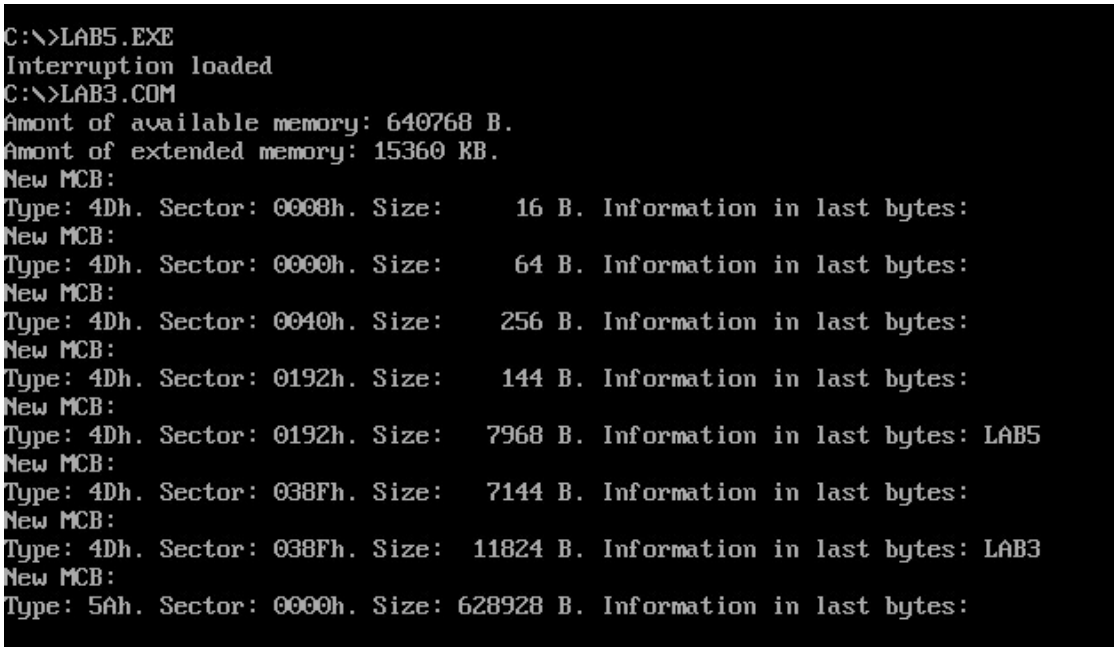

Рис. 2 - карта памяти после запуска LAB5.EXE

На рис. 3 представлен повторный запуск модуля LAB5.EXE.

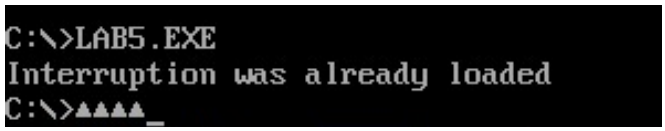

Рис. 3 — повторный запуск модуля LAB5.EXE

На рис. 4 представлен запуск модуля LAB5.EXE с ключом выгрузки /un и запуск модуля LAB3.EXE с выводом блоков памяти.

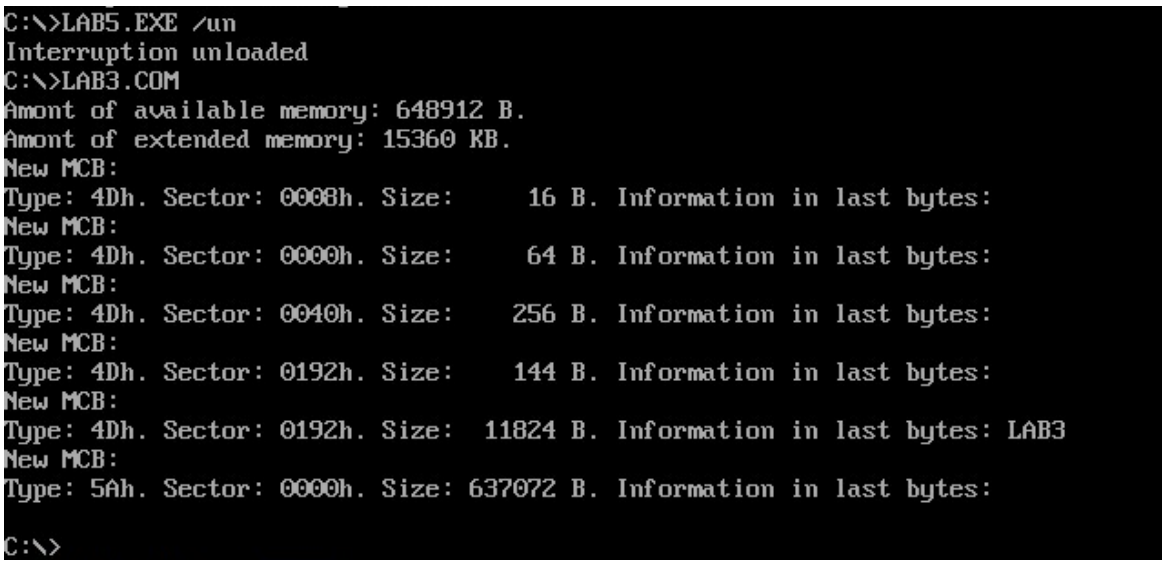

Рис. 4 - запуск модуля LAB5.EXE с ключом выгрузки и запуск LAB3.EXE

Как видно, произошла выгрузка прерывания, больше оно не занимает место в памяти.

Ответы на вопросы:

1. Какого типа прерывания использовались в работе?

   В работе использовались аппаратные прерывания (int 09h), прерывания BIOS (int 16h) и пользовательские прерывания (int 21h).

2. Чем отличается скан-код от кода ASCII?

   Скан-коды привязаны к каждой клавише на аппаратном уровне и не зависят ни от состояния индикаторов Caps Lock, Num Lock и Scroll Lock, ни от состояния управляющий клавиш (Shift, Alt, Ctrl). С помощью скан-кода драйвер распознает, какая клавиша была нажата. ASCII-коду же не может быть поставлена в соответствие определенная клавиша на клавиатуре, т. к. количество клавиш намного меньше. Т.е. ASCII-код представляет код символа в соответствии со стандартной таблицей кодировки.

Выводы

В ходе выполнения работы была исследована возможность встраивания пользовательского обработчика прерываний в стандартный обработчик от клавиатуры.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД LAB5.ASM

```
AStack SEGMENT STACK
    dw 100h dup(?)
AStack ENDS

DATA SEGMENT
    INT_LOADED db 0
      MESSAGE_INT_LOADED db 'Interruption loaded$'
      MESSAGE_INT_ALREADY_LOADED db 'Interruption was already loaded$'
      MESSAGE_INT_NOT_LOADED db 'Interruption unloaded$'
DATA ENDS

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


ROUT PROC FAR
      jmp Start
      INT_COUNT db 'Interruption: 0000$'
      INT_ID dw 4040h
      KEEP_AX dw 0
      KEEP_SS dw 0
      KEEP_SP dw 0
      KEEP_IP dw 0
      KEEP_CS dw 0
      KEEP_PSP DW 0
      INT_STACK dw 100h dup(0)
      REQ_KEY db 47h

Start:
      mov KEEP_SS, ss
      mov KEEP_SP, sp
      mov KEEP_AX, ax
      mov ax, seg INT_STACK
      mov ss, ax
      mov ax, offset INT_STACK
      add ax, 100h
      mov sp, ax

      push bx
      push cx
      push dx
      push si
      push ds
      push bp
      push es

      in al, 60h
      cmp al, REQ_KEY
      je doReq
      pushf
      call dword ptr cs:KEEP_IP
      jmp ROUT_END

doReq:
      push ax
      in al, 61h
      mov ah, al
      or al, 80h
      out 61h, al
      xchg ah, al
      out 61h, al
      mov al, 20h
      out 20h, al
      pop ax

WriteToBuff:
      mov ah, 05h
```

```
      mov cl, 1Eh
      mov ch, 00h
      int 16h
      or al, al
      jz ROUT_END
      mov ax, 0040h
      mov es, ax
      mov ax, es:[1Ah]
      mov es:[1Ch], ax
      jmp WriteToBuff

ROUT_END:
      pop es
      pop bp
      pop ds
      pop si
      pop dx
      pop cx
      pop bx
      mov sp, KEEP_SP
      mov ax, KEEP_SS
      mov ss, ax
      mov ax, KEEP_AX
      mov al, 20h
      out 20h, al
      iret
ROUT ENDP
LAST_BYTE:


LOAD_INT PROC near
      push ax
      push bx
      push cx
      push dx
      push ds
      push es


      mov ah, 35h ; функция получения вектора
      mov al, 09h ; номер вектора
      int 21h
      mov KEEP_IP, bx ; запоминание смещения
      mov KEEP_CS, es ; и сегмента


      push ds
      mov dx, offset ROUT
      mov ax, seg ROUT
      mov ds, ax
      mov ah, 25h
      mov al, 09h
      int 21h ; восстанавливаем вектор
      pop ds

      mov dx, offset LAST_BYTE
      mov cl, 4h ; перевод в параграфы
      shr dx, cl
      add dx, CODE
      inc dx
      xor ax, ax
      mov ah, 31h
      int 21h

      pop es
      pop ds
      pop dx
      pop cx
      pop bx
      pop ax
      ret
LOAD_INT ENDP
```

```
INT_UNLOAD PROC near
      push ax
      push bx
      push dx
      push ds
      push es
      push si

      cli
      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]
      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
INT_UNLOAD ENDP


CHECK_INT PROC near
      push ax
      push bx
      push si

      mov ah, 35h
      mov al, 09h
      int 21h
      mov si, offset INT_ID
      sub si, offset ROUT
      mov ax, es:[bx+si]
      cmp ax, 4040h
      jne EndCheck
      mov INT_LOADED, 1

EndCheck:
      pop si
      pop bx
      pop ax
      ret
CHECK_INT ENDP


WRITE PROC near
    push ax
```

```
    mov ah, 09h
    int 21h
    pop ax
    ret
WRITE ENDP


MAIN PROC FAR
      mov ax, DATA
      mov ds, ax
      mov KEEP_PSP, es

      call CHECK_INT

      mov ax, KEEP_PSP
      mov es, ax
      cmp byte ptr es:[81h+1], '/'
      jne WithoutUN
      cmp byte ptr es:[81h+2], 'u'
      jne WithoutUN
      cmp byte ptr es:[81h+3], 'n'
      jne WithoutUN
      mov dx, offset MESSAGE_INT_NOT_LOADED
      call WRITE
      call INT_UNLOAD
      jmp EndInt

WithoutUN:
      mov al, INT_LOADED
      cmp al, 1
      je IntAlready
      mov dx, offset MESSAGE_INT_LOADED
      call WRITE
      call LOAD_INT
      jmp EndInt

IntAlready:
      mov dx, offset MESSAGE_INT_ALREADY_LOADED
      call WRITE

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