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

ОТЧЕТ

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

Студент гр. 8381 Муковский Д.В.

Преподаватель Ефремов М.А.

Санкт-Петербург
2020

Цель работы.

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

Основные теоретические положения.

Клавиатура содержит микропроцессор, который воспринимает каждое нажатие на клавишу и посылает скан-код в порт микросхемы интерфейса с периферией. Когда скан-код поступает в порт, то вызывается аппаратное прерывание клавиатуры (INT 09H). Процедура обработки этого прерывания считывает номер клавиши из порта 60H, преобразует номер клавиши в соответствующий код, выполняет установку флагов в байтах состояния, загружает номер клавиши и полученный код в буфер клавиатуры.

В прерывании клавиатуры можно выделить три основных шага:

1. Прочитать скан-код и послать клавиатуре подтверждающий сигнал.

2. Преобразовать скан-код в номер кода или в установку регистра статуса клавиш-переключателей.

3. Поместить код клавиши в буфер клавиатуры.

Текущее содержимое буфера клавиатуры определяется указателями на начало и конец записи. Расположение в памяти необходимых данных представлено в таблице 1.

Таблица 1 – Буфер клавиатуры

Адрес в памяти	Размер в байтах	Содержимое
0040:001A	2	Адрес начала буфера клавиатуры
0040:001C	2	Адрес конца буфера клавиатуры
0040:001E	32	Буфер клавиатуры
0040:0017	2	Байты состояния

Флаги в байтах состояния устанавливаются в 1, если нажата соответствующая клавиша или установлен режим. Соответствие флагов и клавиш показано ниже на рис. 1.

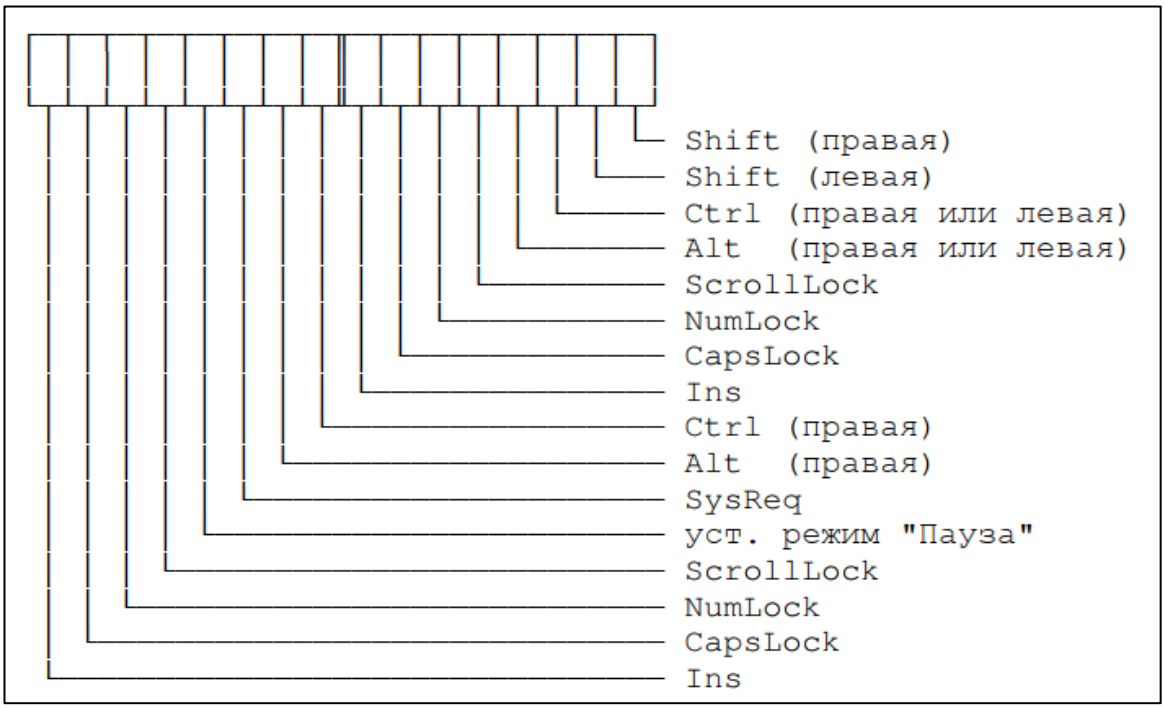

Рисунок 1 – Соответствие флагов и клавиш

Выполнение работы.

Написан текст исходного EXE модуля, который выполняет следующие функции: проверяет, установлено ли пользовательское прерывание с вектором 09h; устанавливает резидентную функцию для обработки прерывания и настраивает вектор прерываний, если прерывание не установлено, и осуществляется выход о функции 4Ch прерывания int 21h; 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.

Полученный исходный модуль был отлажен. Результаты выполнения программы представлены на рис. 2.

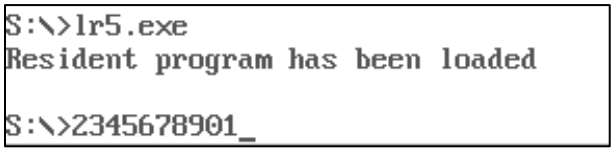

Рисунок 2 – Результат выполнения LR5.EXE

Прерывание работает следующим образом: вместо нажатой цифры выводится цифра на единицу больше, а вместо «9» выводится «0».

Необходимо было проверить размещение прерывания в памяти. Для этого была запущена программа LR3_1.COM, которая отображает карту памяти в виде списка блоков MCB. Результат выполнения программы представлен на рис. 3.

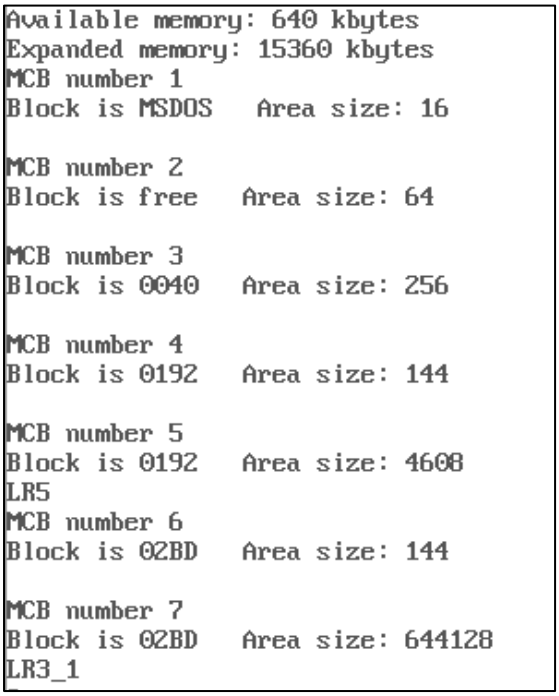

Рисунок 3 – Состояние памяти после загрузки прерывания

После повторного запуска программа определила установленный обработчик прерываний. Результат выполнения программы представлен на рис. 4.

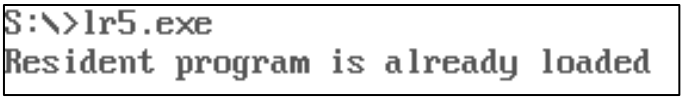

Рисунок 4 – Повторный запуск программы

Далее программа была запущена с ключом выгрузки «/un» для выгрузки резидентного обработчика прерываний. Далее, чтобы убедиться, что резидентный обработчик прерывания выгружен и память, занятая резидентом освобождена была запущена программа LR3_1.COM. Результаты выполнения программы представлен на рис. 5.

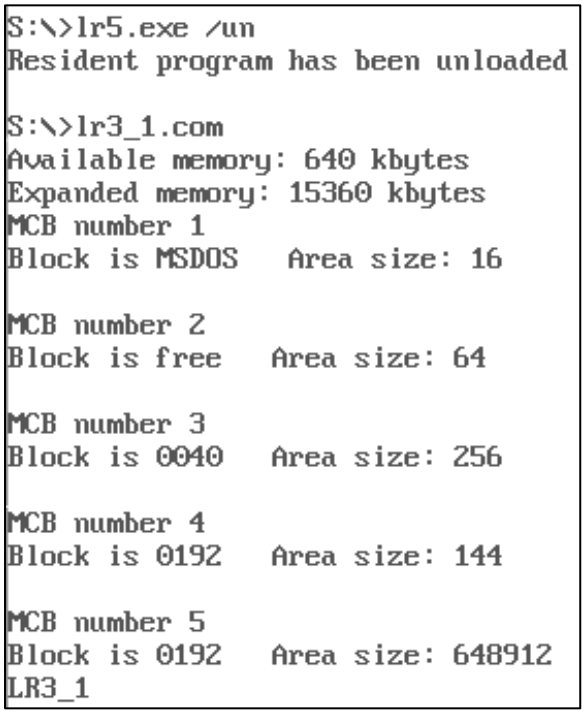

Рисунок 5 – Состояние памяти после выгрузки прерывания

Из скриншота видно, что память резидентного обработчика была высвобождена.

Контрольные вопросы

- **Какого типа прерывания использовались в работе?**
  - Аппаратные прерывания (INT 09H)
  - Прерывания функций BIOS для обслуживания аппаратуры компьютера (INT 16H)
  - Прерывания функций DOS (INT 21H)

- **Чем отличается скан-код от кода ASCII?**

Скан-код - это код клавиши на клавиатуре, которую нажимает пользователь, а ASCII код - это код конкретного символа необходимый для однозначного определения этого самого символа.

Вывод.

В ходе выполнения лабораторной работы была реализована программа, которая встраивала пользовательский обработчик прерываний в стандартный обработчик от клавиатуры.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ. OS4.ASM

```
ASTACK     SEGMENT  STACK
DW 128 DUP(0)
ASTACK  ENDS

DATA       SEGMENT
INT_LOADED_NOW     db 'Resident program has been loaded', 0dh, 0ah, '$'
INT_UNLOAD         db 'Resident program has been unloaded', 0dh, 0ah,
'$'
INT_ALREADY_LOADdb 'Resident program is already loaded', 0dh, 0ah, '$'
NOT_LOADED           db 'Resident program is not loaded', 0dh, 0ah, '$'
IS_LOADED            DB 0
CHECK                     DB 0

DATA       ENDS

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

INTERRUPT proc far
     jmp  INTERRUPT_START
          INT_ID     dw 6666h
          SAVE_IP     dw 0
          SAVE_CS     dw 0
          SAVE_PSP   dw 0
          SAVE_AX    dw 0
          SAVE_SS         dw 0
          SAVE_SP         dw 0
          SAVE_BP         dw 0

          SYMB DB ?
     INTERRUPT_START:
          mov SAVE_AX, ax
          mov SAVE_BP, bp
          mov SAVE_SP, sp
          mov SAVE_SS, ss

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

          mov ax, seg SYMB
          mov ds, ax
```

```
in al, 60h ;читать ключ

irpc CASE, 23456789
cmp al, 0&CASE&h
je PRINT_&CASE&
endm

cmp al , 0Ah
je PRINT_R0

cmp al, 0Bh
je PRINT_R1

pushf
call dword ptr cs:SAVE_IP
jmp INT_END

irpc MET, 23456789
PRINT_&MET&:
     mov SYMB, 3&MET&h
     jmp DO_INT
endm

PRINT_R0:
     mov SYMB, 30h
     jmp DO_INT

PRINT_R1:
     mov SYMB, 31h
     jmp DO_INT

DO_INT:
     in al, 61h
     mov ah, al
     or al, 80h
     out 61h, al
     xchg ah,al
     out 61h, al
     mov al, 20h
     out 20h,al

PRINT_LETTER:
     mov  AH, 05h
     mov  CL, SYMB
     mov  CH, 00h
     int  16h
     or         AL, AL
     jz         INT_END
```

```
                xor ax,ax
                mov es,ax
                mov  AL, ES:[41Ah]
                mov  ES:[41Ch], AL
                jmp  PRINT_LETTER

           INT_END:
                pop ds
                pop es
                pop di
                pop  si
                pop dx
                pop cx
                pop bx
                pop ax
                mov ax, SAVE_SS
                mov ss, ax
                mov sp, SAVE_SP
                mov ax, SAVE_AX
                mov bp, SAVE_BP
                mov al, 20h
                out 20h, al


          iret
INTERRUPT ENDP
     LAST_BYTE:

PRINT proc near
     push ax
     mov ah, 09h
     int 21h
     pop ax
     ret
PRINT endp

LOAD_INTERRUPT 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 SAVE_IP, BX ; запоминание смещения
     mov SAVE_CS, ES ; и сегмента
```

```
      CLI
      push DS
      mov DX, offset INTERRUPT
      mov AX, seg INTERRUPT
      mov DS, AX
      mov AH, 25H
      mov AL, 09H
      int 21H ; восстанавливаем вектор
      pop DS
      STI

      mov DX, offset LAST_BYTE
      mov cl, 4h
      shr dx, cl
      add dx, 10fh
      inc DX ; размер в параграфах
      xor AX, AX
      mov AH, 31h
      int 21h

      pop ES
      pop DS
      pop DX
      pop CX
      pop BX
      pop AX
      ret
LOAD_INTERRUPT ENDP

UNLOAD_INTERRUPT 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 SAVE_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
UNLOAD_INTERRUPT ENDP

CHECK_UN PROC near
      push AX
      push ES

      mov AX, SAVE_PSP
      mov ES, AX
      cmp byte ptr ES:[82h], '/'
      jne END_OF_CHECK
      cmp byte ptr ES:[83h], 'u'
      jne END_OF_CHECK
      cmp byte ptr ES:[84h], 'n'
      jne END_OF_CHECK
      mov CHECK , 1

      END_OF_CHECK:
            pop ES
            pop AX
            ret
CHECK_UN ENDP

CHECK_09H PROC near
      push AX
      push BX
      push SI
```

```
      mov AH, 35h
      mov AL, 09h
      int 21h
      mov SI, offset INT_ID
      sub SI, offset INTERRUPT
      mov AX, ES:[BX+SI]
      cmp AX, 6666h
      jne END_OF_CHECK_09H
      mov IS_LOADED, 1

      END_OF_CHECK_09H:
            pop SI
            pop BX
            pop AX
            ret
CHECK_09H ENDP

MAIN PROC FAR
      push DS
      xor ax, ax
     push ax
      mov AX, DATA
      mov DS, AX
      mov SAVE_PSP, ES

      call CHECK_09H
      call CHECK_UN


      cmp CHECK, 1
      je UNLOAD

      mov al, IS_LOADED
      cmp al, 1
      jne LOAD


      mov DX, offset INT_ALREADY_LOAD
      call PRINT
      jmp ENDD

      LOAD:
            mov DX, offset INT_LOADED_NOW
            call PRINT
            call LOAD_INTERRUPT
            jmp ENDD

      UNLOAD:
            cmp IS_LOADED, 1
```

```
            jne IF_09H_NOT_SET
            call UNLOAD_INTERRUPT
            mov DX, offset INT_UNLOAD
            call PRINT
            jmp ENDD

      IF_09H_NOT_SET:
            mov DX, offset NOT_LOADED
            call PRINT

      ENDD:
            xor AL, AL
            mov AH, 4Ch
            int 21h
MAIN ENDP
CODE ENDS


END MAIN
```