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. 8383 _____ Костарев К.В.

Преподаватель _____ Ефремов М.А.

Санкт-Петербург

2020

Цель работы.

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

Выполнение работы.

Для выполнения лабораторной работы был написан и отлажен программный модуль типа .EXE, который:

1) проверяет, установлено ли прерывание с вектором 09H;
2) устанавливает резидентную функцию для обработки прерывания и настраивает вектор прерывания, если прерывание не установлено;
3) выдает соответствующее сообщение о том, что прерывание уже установлено и осуществляет выход через функцию 4CH прерывания 21H;
4) выгружает прерывание, если в командной строке был объявлен ключ выгрузки “/un” (т.е. восстанавливает стандартный вектор прерываний и освобождает память, занятую резидентом).

Некоторые функции, такие как проверка установки прерывания и его выгрузка, были реализованы еще в предыдущей лабораторной работе № 4 и остались без видимых изменений, кроме вектора прерываний.

После установки пользовательского обработчика при нажатии на клавиатуре клавиши «0» выводится эмодзи. Результат работы программы представлен на рис. 1, исходный код программы в Приложении А.

Рисунок 1 – Результат работы программы

Далее было проверено размещение прерывания в памяти. Для этого была запущена программа из лабораторной работы №3. Результат проверки можно видеть на рис. 2.

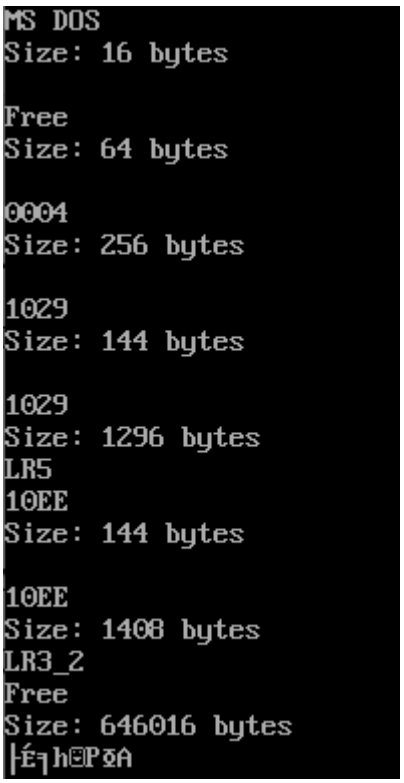

Рисунок 2 – Проверка размещения прерывания в памяти

Далее программа с обработчиком прерывания была запущена еще раз. Как мы можем видеть на рис. 3, программа понимает, что обработчик прерываний уже было установлен.

Рисунок 3 – Повторный запуск программы

Далее программа была запущена с ключом выгрузки “/un”. На рис. 4 можно видеть, что обработчик прерывания выгружен и эмодзи вместо нуля больше не печатаются, а на рис. 5 – что освобождена память, занятая резидентом.

Рисунок 4 – Результат работы программы с ключом выгрузки

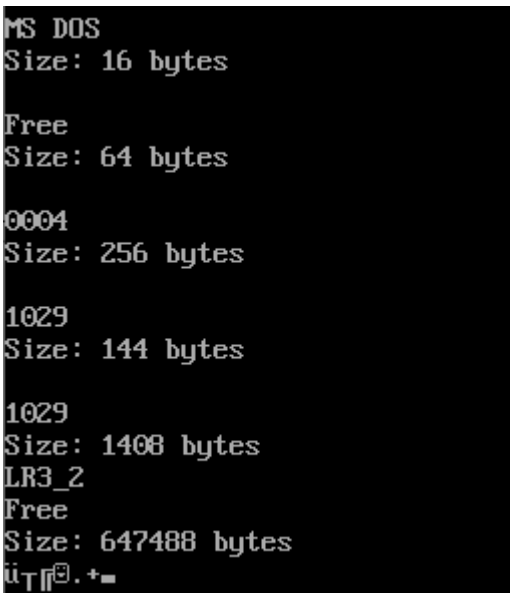

Рисунок 5 – Проверка освобождения памяти

1) *Какого типа прерывания использовались в работе?*

   Аппаратное (реализуемое прерывание 09H) и программные (прерывания 21H и 10H).

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

   В IBM-совместимых компьютерах скан-код – это код, присвоенный каждой клавише, с помощью которого драйвер клавиатуры распознает ее, а код ASCII – это код символа в кодировочной таблице.

Выводы.

В ходе выполнения данной лабораторной работы была изучена возможность встраивания пользовательского обработчика прерываний в стандартный обработчик от клавиатуры и реализован такой обработчик прерывания по нажатию клавиши клавиатуры.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

```
AStack     SEGMENT    STACK
           db    512   dup(0)
AStack ENDS

DATA SEGMENT
     RESIDENT_LOAD db 'Resident was loaded', 13, 10, '$'
     RESIDENT_UNLOAD db 'Resident was unloaded', 13, 10, '$'
     RESIDENT_ALR_LOAD db 'Resident is already loaded', 13, 10, '$'
     RESIDENT_NOT_LOAD db 'Resident not yet loaded', 13, 10, '$'
DATA ENDS

CODE SEGMENT
           ASSUME CS:CODE, DS:DATA, ES:NOTHING, SS:AStack

ROUT PROC FAR
     jmp START_ROUT
    INT_STACK   dw    128 dup (?)
     SIGNATURE  dw    7373H
    COUNT       dw    0
    KEEP_AX           dw    ?
     KEEP_PSP   dw    0
     KEEP_IP    dw    0
     KEEP_CS    dw    0
     KEEP_SS    dw    0
     KEEP_SP    dw    0
     ZERO_CODE   db  0BH
START_ROUT:
    mov KEEP_AX, AX
     mov KEEP_SS, SS
     mov KEEP_SP, SP
     mov AX, seg INT_STACK
     mov SS, AX
     mov AX, offset INT_STACK
     add AX, 256
     mov SP, AX
     mov AX, KEEP_AX
     push AX
     push BP
     push DX
     push DI
    push DS
    push ES
     in AL, 60H
     cmp AL, ZERO_CODE
     je DO_REQ
     pushf
```

```
      call dword ptr CS:KEEP_IP
      jmp END_ROUT
DO_REQ:
      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
ADD_TO_BUFF:
      mov AH, 05H
      mov CL, 02H
      mov CH, 00H
      int 16H
      or AL, AL
      jz END_ROUT
      mov AX, 0040H
      mov ES, AX
      mov SI, 001AH
      mov AX, ES:[SI]
      mov SI, 001CH
      mov ES:[SI], AX
      jmp ADD_TO_BUFF
END_ROUT:
      pop ES
      pop DS
      pop DI
      pop DX
      pop BP
      pop AX
      mov AX, KEEP_SS
      mov SS, AX
      mov AX, KEEP_AX
      mov SP, KEEP_SP
      mov AL, 20H
      out 20H, AL
      iret
ROUT ENDP
LAST_BYTE:

WRITE_STRING PROC near
      push AX
      mov AH, 09H
      int  21H
      pop AX
      ret
```

```
WRITE_STRING ENDP

CHECK_ROUT PROC
     mov AH, 35H
     mov AL, 09H
     int 21H
     mov SI, offset SIGNATURE
     sub SI, offset ROUT
     mov AX, 7373H
     cmp AX, ES:[BX+SI]
     je   IS_LOADED
     call SET_ROUT
IS_LOADED:
     call DEL_ROUT
     ret
CHECK_ROUT ENDP

SET_ROUT PROC
     mov AX, KEEP_PSP
     mov ES, AX
     cmp byte ptr ES:[80H], 0
     je   LOAD_ROUT
     cmp byte ptr ES:[82H], '/'
     jne LOAD_ROUT
     cmp byte ptr ES:[83H], 'u'
     jne LOAD_ROUT
     cmp byte ptr ES:[84H], 'n'
     jne LOAD_ROUT
     lea DX, RESIDENT_NOT_LOAD
     call WRITE_STRING
     jmp  END_OF_SET
LOAD_ROUT:
     mov AH, 35H
    mov AL, 09H
     int 21H
     mov KEEP_CS, ES
     mov KEEP_IP, BX
     lea  DX, RESIDENT_LOAD
     call WRITE_STRING
     push DS
     mov DX, offset ROUT
    mov AX, seg ROUT
     mov DS, AX
     mov AH, 25H
     mov AL, 09H
     int 21H
     pop DS
     mov DX, offset LAST_BYTE
     mov CL, 4
     shr DX, CL
```

```
      inc DX
      add DX,    CODE
      sub DX,    KEEP_PSP
      sub AL, AL
      mov AH, 31H
      int 21H
END_OF_SET:
      sub AL, AL
      mov AH, 4CH
      int 21H
SET_ROUT ENDP

DEL_ROUT PROC
    push AX
      push DX
      push DS
      push ES
      mov AX, KEEP_PSP
      mov ES, AX
      cmp byte ptr ES:[80h], 0
      je   ALR_LOAD
      cmp byte ptr ES:[82h], '/'
      jne ALR_LOAD
      cmp byte ptr ES:[83h], 'u'
      jne ALR_LOAD
      cmp byte ptr ES:[84h], 'n'
      jne ALR_LOAD
      lea  DX, RESIDENT_UNLOAD
      call WRITE_STRING
      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
      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
      mov AH, 49H
      int 21H
```

```
      jmp END_OF_DEL
ALR_LOAD:
      mov DX, offset RESIDENT_ALR_LOAD
      call WRITE_STRING
END_OF_DEL:
      pop ES
      pop  DS
     pop DX
      pop AX
      ret
DEL_ROUT ENDP

MAIN PROC NEAR
      mov AX, DATA
      mov DS, AX
      mov KEEP_PSP, ES
      call CHECK_ROUT
      mov AX, 4C00H
      int 21H
      ret
MAIN ENDP
CODE ENDS

END MAIN
```