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. 7383 _____ Медведев И.С.

Преподаватель _____ Ефремов М.А.

Санкт-Петербург

2019

Цель работы

Построить обработчик прерываний сигналов таймера. Эти сигналы генерируются аппаратурой через определенные интервалы времени и, при возникновении такого сигнала, возникает прерывание с определенным значением вектора.

Ход работы

В ходе работы были написаны некоторые функции, которые описаны в табл. 1.

Таблица 1 – Используемые функции

Название функции	Выполняемая задача
DELETE_INTERRUPT	Удаление пользовательского прерывания.
PRINT	Вывод сообщения на экран.
ROUT	Функция - обработчик прерывания от клавиатуры.
SET_INTERRUPT	Установка пользовательского прерывания.
BASE_FUNC	Функция проверяет было ли установлено пользовательское прерывание.
MAKE_STR	Добавляет символ в буфер.

Таблица 2 – Структура данных управляющей программы

Имя	Тип	Назначение
message_1	db	Вывод строки 'Resident was loaded'
message_2	db	Вывод строки 'Resident has already been loaded'
message_3	db	Вывод строки 'Resident was unloaded'

Таблица 3 – Структура данных собственного прерывания.

Имя	Тип	Назаначение
identifier	db	Идентификатор пользовательского прерывания.
KEEP_IP	dw	Переменная для сохранения значения регистра IP.
KEEP_CS	dw	Переменная для сохранения значения регистра CS.
KEEP_PSP	dw	Переменная для сохранения значения регистра PSP
flag	dw	Флаг для определения необходимости выгружать прерывание.
KEEP_SS	dw	Переменная для сохранения значения регистра SS.
KEEP_AX	dw	Переменная для сохранения значения регистра AX.
KEEP_SP	dw	Переменная для сохранения значения регистра SP
sym_code	db	Скан-код нажатия левого Alt.

Также в ходе работы был создан .EXE файл, который 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 по функции 4Ch прерывания int 21h. Если прерывание установлено, то выводится соответствующее сообщение и осуществляется выход по функции 4Ch прерывания int 21h, также осуществляет выгрузку прерывания по значению параметра в командной строке /un. Выгрузка прерывания состоит в восстановлении стандартного вектора прерываний и освобождении памяти,

занимаемой резидентом. Прерывание происходит при нажатии левого Alt. Примеры работы программы представлены на рис 1-3.

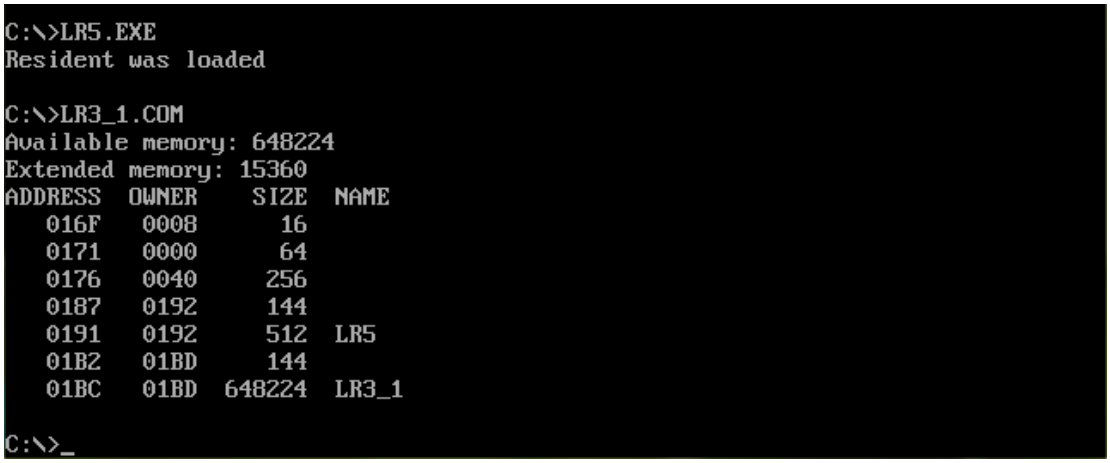

Рисунок 1 – Запуск 5-ой лабораторной работы.

Рисунок 2 – Запуск 3-ей лабораторной работы, после выгрузки прерывания.

Рисунок 3 – Повторный запуск 5-ой лабораторной работы и проверка реакции на нажатие левого Alt.

Вывод

В результате выполнения лабораторной работы был создан обработчик прерываний, встроенный в стандартный обработчик от клавиатуры.

Ответы на контрольные вопросы

1) Какого типа использовались прерывания в работе?

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

2) Чем отличается скан-код от кода ASCII?

Скан-код - код, присвоенный каждой клавише, с помощью которого драйвер клавиатуры распознает, какая клавиша была нажата. ASCII код - это код для представления символов в виде чисел, в соответствии со стандартной кодировочной таблицей.

ПРИЛОЖЕНИЕ А. КОД ПРОГРАММЫ

```
CODE SEGMENT
      ASSUME CS:CODE, DS:DATA, ES:DATA, SS:MY_STACK

ROUT PROC FAR
      jmp ROUT_START

      ;DATA
      identifier db '0000'
      KEEP_PSP dw 0
      KEEP_IP dw 0
      KEEP_CS dw 0
      KEEP_SS dw 0
      KEEP_SP dw 0
      KEEP_AX dw 0
      sym_code db 38h
      inter_stack dw 64 dup (?)
      end_stack dw 0

ROUT_START:
      mov KEEP_AX, ax
      mov KEEP_SS, ss
      mov KEEP_SP, sp
      mov ax, cs
      mov ss, ax
      mov sp, offset end_stack
      mov ax, KEEP_AX
      push ax
      push dx
      push ds
      push es


      in al, 60H
      cmp al, sym_code
      je DO_REQ
      pushf
      call dword ptr cs:KEEP_IP
      jmp ROUT_END

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

      mov cl, 'A'
      call MAKE_STR
```

```
      mov cl, 'l'
      call MAKE_STR
      mov cl, 't'
      call MAKE_STR


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


DELETE_INTERRUPT PROC
      push ax
      push ds
      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]
      mov ds, ax
      mov ah, 25h
      mov al, 09h
      int 21h
      pop ds
      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
      STI
      pop ax
      ret
DELETE_INTERRUPT ENDP

MAKE_STR PROC
      mov ah, 05h
      mov ch, 00h
```

```
      int 16h
      or al, al
      jz end_m
      mov ax, 0040h
      mov es, ax
      mov ax,es:[1Ah]
      mov es:[1Ch],ax
      jmp make_str
end_m:
      ret

MAKE_STR ENDP

SET_INTERRUPT PROC
      push ax
      push dx
      push ds
      mov ah,35h
      mov al,09h
      int 21h
      mov KEEP_IP,bx
      mov KEEP_CS,es
      mov dx,offset ROUT
      mov ax,seg ROUT
      mov ds,ax
      mov ah,25h
      mov al,09h
      int 21h
      pop ds
      mov dx,offset message_1
      call PRINT
      pop dx
      pop ax
      ret
SET_INTERRUPT ENDP


BASE_FUNC PROC
      mov ah,35h
      mov al,09h
      int 21h

      mov si, offset identifier
      sub si, offset ROUT

      mov ax,'00'
      cmp ax,es:[bx+si]
      jne NOT_LOADED
      cmp ax,es:[bx+si+2]
      jne NOT_LOADED
      jmp LOADED

NOT_LOADED:
      call SET_INTERRUPT
      mov dx,offset LAST_BYTE
```

```
      mov cl,4
      shr dx,cl
      inc dx
      add dx,CODE
      sub dx,KEEP_PSP
      xor al,al
      mov ah,31h
      int 21h

LOADED:
      push es
      push ax
      mov ax, KEEP_PSP
      mov es, ax
      mov al, es:[81h+1]
      cmp al, '/'
      jne NOT_UNLOAD
      mov al, es:[81h+2]
      cmp al, 'u'
      jne NOT_UNLOAD
      mov al, es:[81h+3]
      cmp al, 'n'
      je UNLOAD

NOT_UNLOAD:
      pop ax
      pop es
      mov dx,offset message_2
      call PRINT
      ret

UNLOAD:
      pop ax
      pop es
      call DELETE_INTERRUPT
      mov dx, offset message_3
      call PRINT
      ret
BASE_FUNC ENDP

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

MAIN PROC Far
      mov ax,DATA
      mov ds,ax
      mov KEEP_PSP, es
      call BASE_FUNC
      xor al,al
      mov ah,4Ch
```

```
      int 21H
MAIN ENDP
CODE ENDS

MY_STACK SEGMENT STACK
      DW 64 DUP (?)
MY_STACK ENDS

DATA SEGMENT
      message_1 db 'Resident was loaded', 13, 10, '$'
      message_2 db 'Resident has been already loaded', 13, 10, '$'
      message_3 db 'Resident was unloaded', 13, 10, '$'
DATA ENDS

END MAIN
```