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

ОТЧЕТ

по лабораторной работе №4

по дисциплине «Операционные системы»

Тема: Обработка стандартных прерываний

Студент гр. 8383 _____ Костарев К.В.

Преподаватель _____ Ефремов М.А.

Санкт-Петербург

2020

Цель работы.

Построить обработчик прерываний сигналов таймера.

Основные теоретические сведения.

В архитектуре компьютера существуют стандартные прерывания, за которыми закреплены определённые вектора прерываний. Вектор прерываний хранит адрес подпрограммы обработчика прерываний. При возникновении прерывания, аппаратура компьютера передаёт управление по соответствующему адресу вектора прерывания. Обработчик прерываний получает управление и выполняет соответствующие действия.

Резидентные обработчики прерываний - это программные модули, которые вызываются при возникновении прерываний определенного типа (сигнал таймера, нажатие клавиши и т.д.), которым соответствуют определенные вектора прерывания. Когда вызывается прерывание, процессор переключается на выполнение кода обработчика, а затем возвращается на выполнение прерванной программы. Адрес возврата в прерванную программу (CS:IP) запоминается в стеке вместе с регистром флагов. Затем в CS:IP загружается адрес точки входа программы обработки прерывания и начинает выполняться его код. Обработчик прерывания должен заканчиваться инструкцией IRET (возврат из прерывания).

Вектор прерывания имеет длину 4 байта. В первом хранится значение IP, во втором - CS. Младшие 1024 байта памяти содержат 256 векторов. Вектор для прерывания 0 начинается с ячейки 0000:0000, для прерывания 1 - с ячейки 0000:0004 и т.д.

Выполнение работы.

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

1) проверяет, установлено ли прерывание с вектором 1CH;

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;
4) выгружает прерывание, если в командной строке был объявлен ключ выгрузки “/un” (т.е. восстанавливает стандартный вектор прерываний и освобождает память, занятую резидентом);

Результат работы программы представлен на рис. 1, исходный код программы в Приложении А.

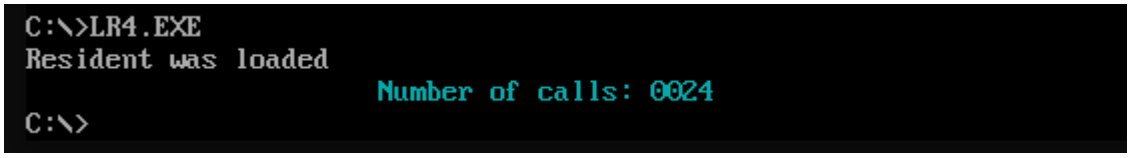

Рисунок 1 – Результат работы программы

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

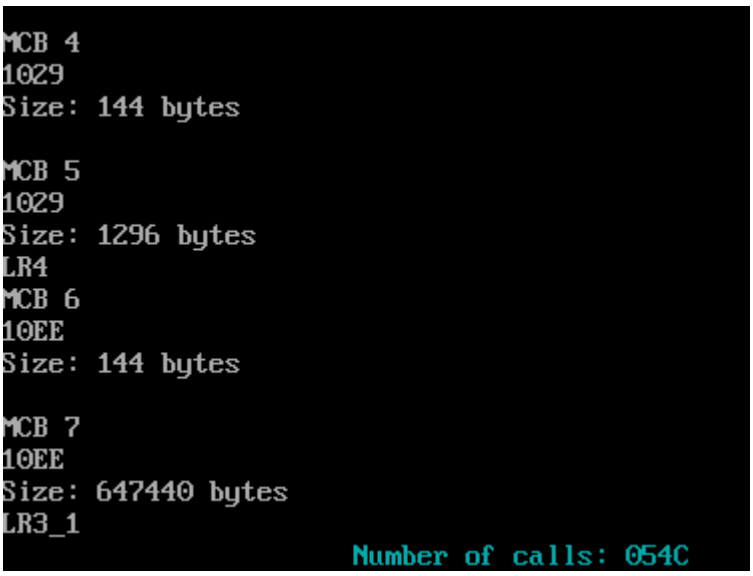

Рисунок 2 – Проверка размещения прерывания в памяти

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, а на рис. 5 – освобождена память, занятая резидентом.

Рисунок 4 – Результат работы программы с ключом выгрузки

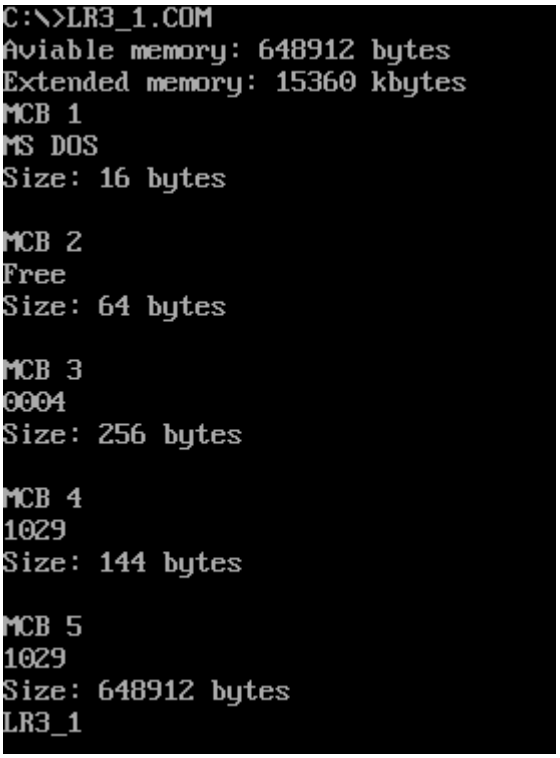

Рисунок 5 – Проверка освобождения памяти

1) *Как реализован механизм прерывания от часов?*

   После вызова прерывания сохраняется содержимое регистров (записываются в переменные) и определяется источник прерывания. Далее по номеру источника определяется смещение в таблице векторов прерываний. Данный адрес сохраняется в регистр CS:IP. После этого управление передаётся по этому адресу, т. е. выполняется запуск обработчика прерываний и происходит его выполнение. После выполнения происходит возврат управления прерванной программе. И так

происходит каждые 55 мс – именно с такой периодичностью вызывается прерывание от таймера.

2) *Какого типа прерывания использовались в работе?*

   Аппаратное (реализуемое прерывание 1CH) и программные (прерывания 21H и 10H).

Выводы.

В ходе выполнения данной лабораторной работы был изучен принцип работы прерываний и реализованы резидентный обработчик прерывания по сигналу таймера и восстановление стандартного вектора прерываний с освобождением памяти.

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    100 dup (?)
     SIGNATURE  dw    01984H
    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
     NUMBER_OF_CALL  db 'Number of calls:    0000$'
START_ROUT:
    mov KEEP_AX, AX
     mov KEEP_SS, SS
     mov KEEP_SP, SP
     mov AX, seg INT_STACK
     mov SS, AX
     mov SP, 0
     mov AX, KEEP_AX
     push AX
     push BP
     push DX
     push DI
    push DS
    push ES
     mov AX, CS
     mov DS, AX
     mov ES, AX
     mov AX, CS:COUNT
     add AX, 1
     mov CS:COUNT, AX
```

```
      mov DI, offset NUMBER_OF_CALL + 20
      call WRD_TO_HEX
      mov BP, offset NUMBER_OF_CALL
      call outputBP
    pop ES
    pop DS
      pop DI
      pop DX
      pop BP
      pop AX
      mov AL, 20H
      out 20H, AL
    mov AX, KEEP_SS
      mov SS, AX
      mov AX, KEEP_AX
      mov SP, KEEP_SP
      iret
ROUT ENDP

TETR_TO_HEX     PROC near
      and       AL, 0Fh
      cmp       AL,09
      jbe       NEXT
      add       AL,07
NEXT:
    add         AL,30h
      ret
TETR_TO_HEX     ENDP

BYTE_TO_HEX     PROC near
      push    CX
      mov       AH,AL
      call  TETR_TO_HEX
      xchg  AL,AH
      mov         CL,4
      shr         AL,CL
      call  TETR_TO_HEX
      pop         CX
      ret
BYTE_TO_HEX     ENDP

WRD_TO_HEX PROC near
      push  BX
      mov         BH,AH
      call  BYTE_TO_HEX
      mov         [DI],AH
      dec         DI
      mov         [DI],AL
      dec         DI
      mov         AL,BH
```

```
      call  BYTE_TO_HEX
      mov         [DI],AH
      dec         DI
      mov         [DI],AL
      pop         BX
      ret
WRD_TO_HEX ENDP

outputBP PROC near
      push  ax
      push  bx
      push  dx
      push  cx
      mov   ah, 13h
      mov   al, 0
      mov   bl, 03h
      mov   bh, 0
      mov   dh, 23
      mov   dl, 22
      mov   cx, 21
      int   10h
      pop   cx
      pop   dx
      pop   bx
      pop   ax
      ret
outputBP ENDP
END_ROUT:

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, 1CH
      int 21H
      mov SI, offset SIGNATURE
      sub SI, offset ROUT
      mov AX, 01984H
      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, 1CH
      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, 1CH
      int 21H
      pop DS
      mov DX, offset END_ROUT
      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, 1CH
      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, 1CH
      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
```