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

ОТЧЕТ

по лабораторной работе №4
по дисциплине «Операционные системы»
Тема: Обработка стандартных прерываний.

Студент гр. 8383 _____ Кормщикова А. О.

Преподаватель _____ Ефремов М. А.

Санкт-Петербург
2020

Цель работы.

Построить обработчик прерываний сигналов таймера.

Ход выполнения.

Был написан код исходного .EXE модуля (LR4.ASM представлен в приложении А), который проверяет, установлено ли пользовательское прерывание с вектором 1Ch; устанавливает резидентную функцию для обработки прерывания и настраивает вектор прерываний, если прерывание не установлено, и не осуществляется выход по 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; На рис. 1 показан запуск программы без загруженного прерывания, запуск

Код устанавливаемого прерывания сохраняет значения регистров в стеке при входе и восстанавливает их при выходе, при выполнении тела процедуры накапливает общее суммарное число прерываний и выводит их на экран с помощью прерывания 10h.

На рис.1 представлен вывод программы, которая отображает карту памяти до загрузки прерывания.

На рис.2 представлена попытка выгрузить не загруженное прерывание, загрузка прерывания, повторная загрузка прерывания.

На рис.3 представлен вывод программы, которая отображает карту памяти после загрузки прерывания.

На рис.4 программа была запущена с ключом выгрузки.

На рис.5 представлен вывод программы, которая отображает карту памяти после выгрузки прерывания.

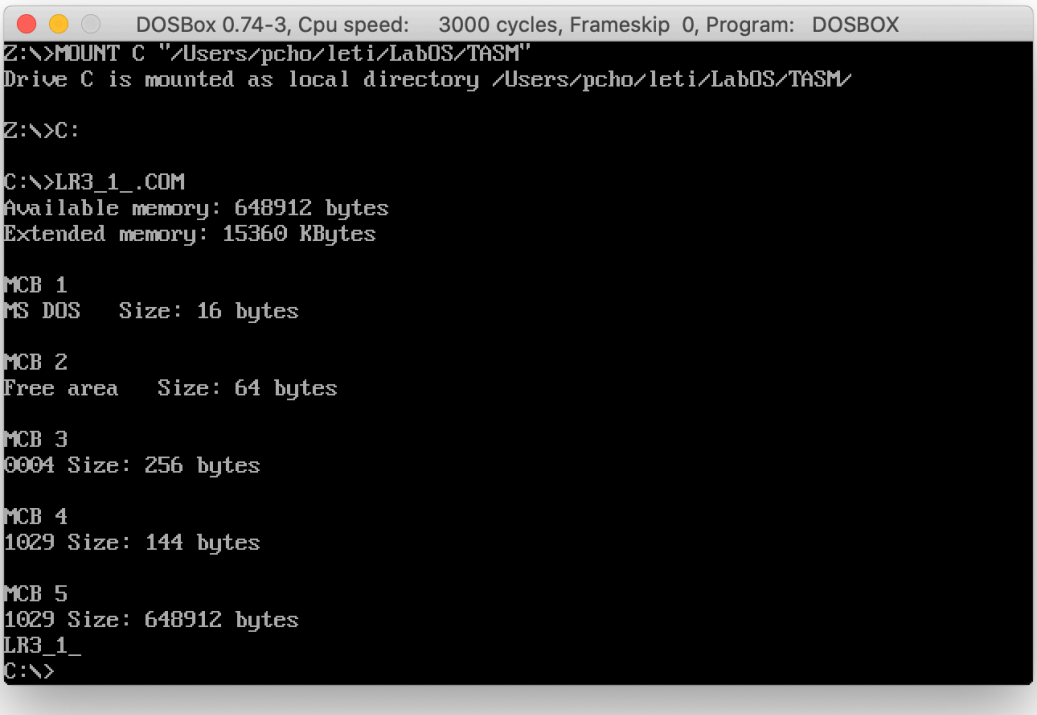

Рисунок 1 - Вывод карты памяти до загрузки прерывания

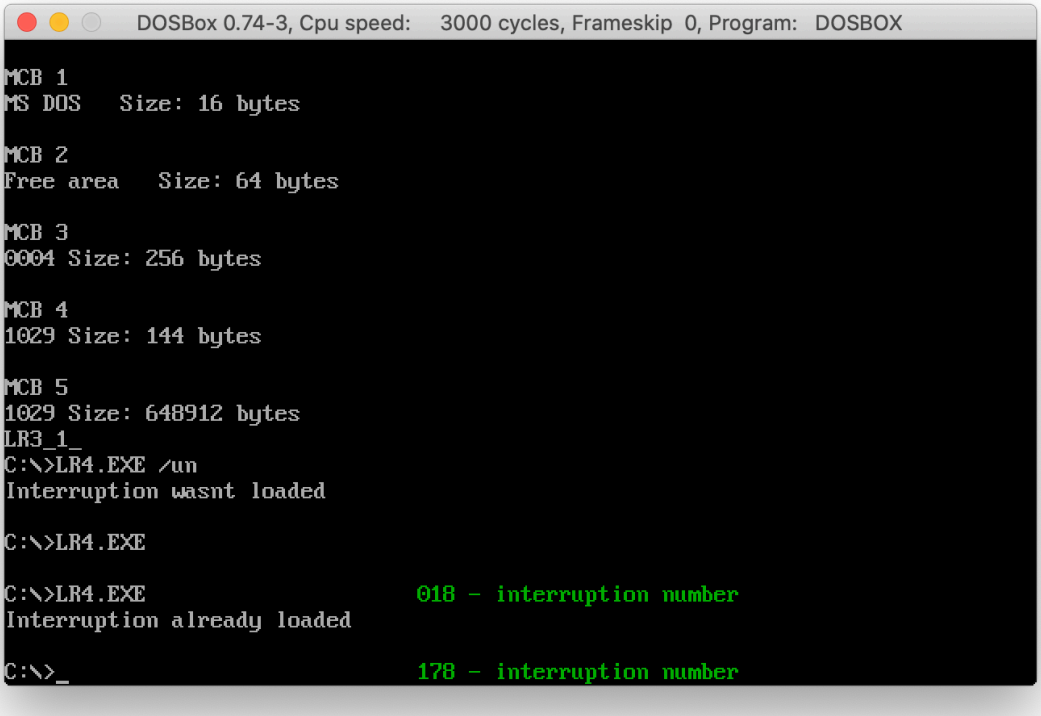

Рисунок 2 - Результат выполнения программы

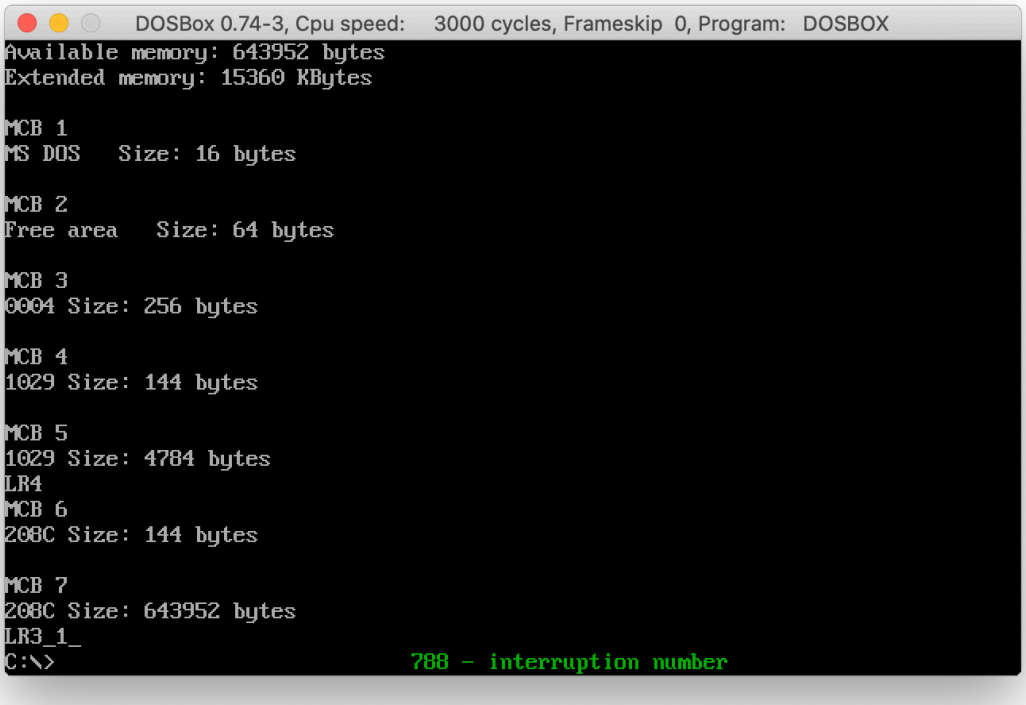

Рисунок 3 - Вывод карты памяти после загрузки прерывания

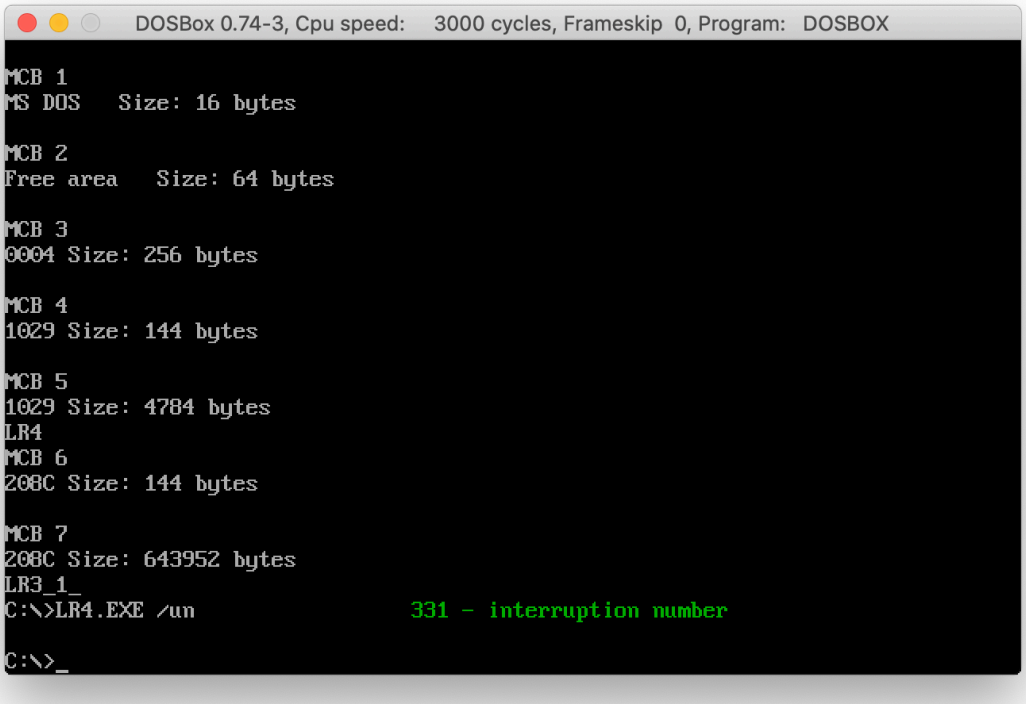

Рисунок 4 - Запуск программы с ключом выгрузки

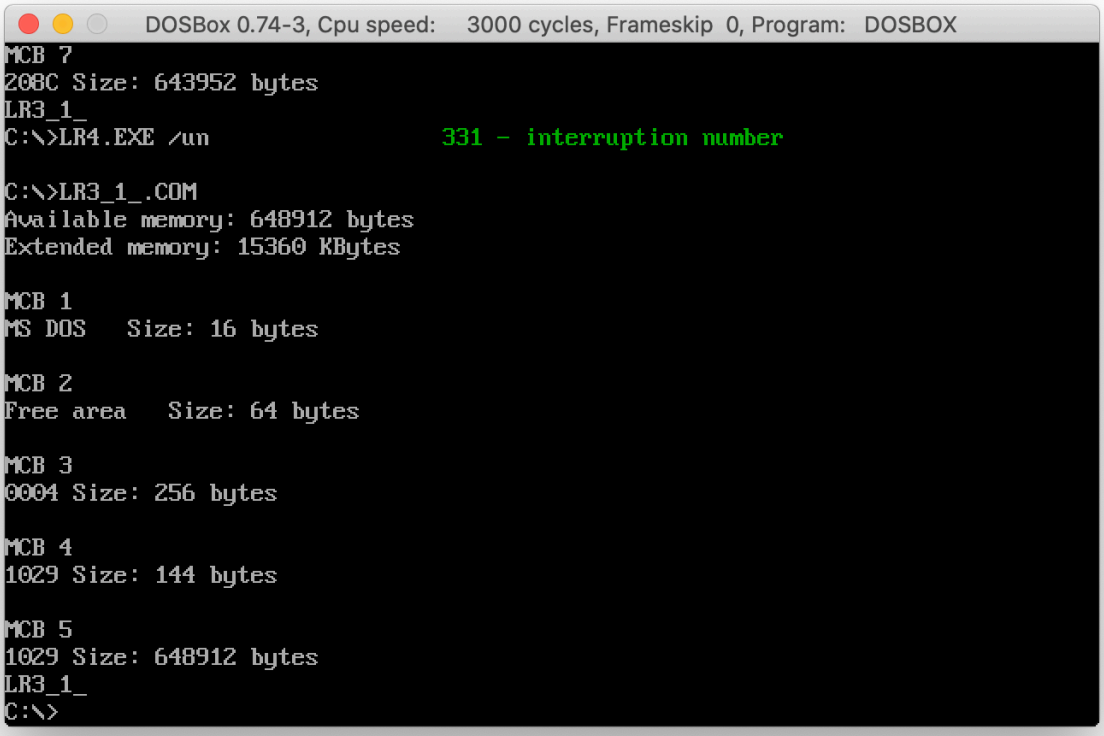

Рисунок 5 - Вывод карты памяти после выгрузки прерывания

Контрольные вопросы:

Как реализован механизм прерывания от часов?

Любой компьютер содержит устройство, называемое системным таймером. Это устройство подключено к линии запроса на прерывание IRQ0 и вырабатывает прерывание INT 8h приблизительно 18,2 раза в секунду. При инициализации BIOS устанавливает свой обработчик для прерывания таймера. Этот обработчик каждый раз увеличивает на 1 текущее значение четырехбайтовой переменной, располагающейся в области данных BIOS по адресу 0000:046Ch - счетчик тиков таймера. Если этот счетчик переполняется (прошло более 24 часов с момента запуска таймера), в ячейку 0000:0470h заносится 1.Обработчик прерывания таймера также вызывает прерывание int 1Ch. После инициализации системы вектор INT 1Ch указывает на команду IRET, т.е. ничего не выполняется. Программа может установить собственный обработчик этого прерывания для того чтобы выполнять какие-либо

периодические действия (в реализованной программе вектор указывает на обработчик, который выводит на экран значение счетчика вызовов прерываний системного таймера).

2) Какого типа прерывания использовались в работе?

int 1Ch -пользовательское прерывание по таймеру. Аппаратное прерывание

int 10h - видео сервис. Программное прерывание

int 21h - сервис DOS. Программное прерывание

Выводы.

В ходе выполнения лабораторной работы были построен обработчик прерываний сигналов таймера.

ПРИЛОЖЕНИЕ А

```
SSTACK SEGMENT STACK
               DW 128
SSTACK  ENDS

DATA SEGMENT

      LOADED db 'Interruption already loaded', 0DH, 0AH, '$'
      NOTLOAD db 'Interruption wasnt loaded', 0DH, 0AH, '$'
      LOAD  db 0
      UN db 0


DATA ENDS

CODE SEGMENT

ASSUME CS:CODE, DS:DATA, SS:SSTACK


ROUT PROC FAR ; обработчик прерываний

      jmp INTERRUPT
      COUNTER db  "000 - interruption number"
      SIGN dw  1000h
      KEEP_IP dw  0
      KEEP_CS dw  0
      KEEP_PSP dw  0
      KEEP_SS dw  0
      KEEP_SP dw  0
      KEEP_AX dw  0
      INT_STACK dw 128 dup(0)

INTERRUPT:
    mov     KEEP_AX, AX
      mov   KEEP_SP, SP
      mov   KEEP_SS, SS
      mov   AX, SEG INT_STACK
      mov   SS, AX
      mov   AX, offset INT_STACK
      add   AX, 256
      mov   SP, AX

      push  AX
      push    BX
      push    CX
      push    DX
      push    SI
    push    ES
    push    DS
      mov   AX, seg COUNTER
      mov   DS, AX

;cursor
      mov ah, 03h
      mov bh, 0h
      int 10h
      push dx

      mov ah, 02h
      mov dx, 1820h ;18 строка, 20 столбец
      mov bh, 0h
      int 10h
```

```
      mov ax, seg COUNTER
      push ds
      mov ds, ax
      mov si, offset COUNTER
      add si, 2
      mov cx, 3
loopa:
      mov ah, [si]
      inc ah
      mov [si], ah
      cmp ah, ':'
      jnz END_loopa
      mov ah, '0'
      mov [si], ah
      dec si
      loop loopa
END_loopa:
      pop ds

;print
      push es
      push bp

      mov ax, seg COUNTER
      mov es, ax
      mov bp, offset COUNTER
      mov ah, 13h
      mov al, 01h
      mov bh, 00h
      mov bl, 02h
      mov cx, 25
      int 10h

      pop bp
      pop es

      pop dx
      mov ah, 02h
      mov bh, 00h
      int 10h

      pop ds
      pop es
      pop si
      pop dx
      pop cx
      pop bx
      mov ax, KEEP_SS
      mov ss, ax
      mov ax, KEEP_AX
      mov sp, KEEP_SP

      mov al, 20h
      out 20h, al
      IRET
ROUT ENDP
END_ROUT:

CHECK PROC
      push ax
      push bx
      push si

      mov ah, 35h
```

```
      mov al, 1ch
      int 21h
      mov si, offset SIGN
      sub si, offset ROUT
      mov ax, es:[bx+si]
      cmp ax, SIGN
      jnz check_end
      mov LOAD, 1

check_end:
      pop si
      pop bx
      pop ax
      ret
CHECK ENDP

CHECK_UN PROC
      push ax
      push es

      mov ax, KEEP_PSP
      mov es, ax
      cmp byte ptr ES:[82H], '/'
      jnz CHECK_UN_END
      cmp byte ptr ES:[83H], 'u'
      jnz CHECK_UN_END
      cmp byte ptr ES:[84H], 'n'
      jnz CHECK_UN_END

      mov UN, 1

CHECK_UN_END:

      pop es
      pop ax
      ret
CHECK_UN ENDP



LOAD_I PROC
      push ax
      push bx
      push cx
      push dx
      push es
      push ds

      mov ah, 35h
      mov al, 1ch
      int 21h

      mov KEEP_IP, bx
      mov KEEP_CS, es

      mov ax, seg ROUT
      mov dx, offset ROUT
      mov ds, ax
      mov ah, 25h
      mov al, 1ch
      int 21h

      pop ds

      mov dx, offset END_ROUT
```

```
      mov cl, 4h
      shr dx, cl
      add dx, 10fh
      inc dx
      xor ax, ax
      mov ah, 31h
      int 21h

      pop es
      pop dx
      pop cx
      pop bx
      pop ax

      ret
LOAD_I ENDP


LOAD_UN PROC
      CLI
      push ax
      push bx
      push cx
      push dx
      push ds
      push es
      push si

      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]
      push ds
      mov ds, ax
      mov ah, 25h
      mov al, 1ch
      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 cx
      pop bx
      pop ax
      ret
LOAD_UN ENDP

MAIN PROC FAR
```

```
      push ds
      xor ax,ax
      push ax
      mov ax, DATA
      mov ds, ax
      mov KEEP_PSP, es
      call CHECK
      call CHECK_UN
      cmp UN, 1
      je UNLOAD
      cmp LOAD, 1
      jnz LOAD_
      mov dx, offset LOADED

      push ax
      mov ah, 09h
      int 21h
      pop ax

      jmp MEND
LOAD_:
      call LOAD_I
      jmp MEND

UNLOAD:
      cmp LOAD, 1
      jnz NOT_LOAD
      call LOAD_UN
      jmp MEND

NOT_LOAD:
      mov dx, offset NOTLOAD
      push ax
      mov ah, 09h
      int 21h
      pop ax

MEND:
      xor al, al
      mov ah, 4ch
      int 21h


MAIN      ENDP
CODE ENDS
END MAIN
```