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. 8381 _____ Облизанов А.Д.

Преподаватель _____ Ефремов М.А.

Санкт-Петербург
2020

Цель работы.

В работе предлагается построить обработчик прерываний сигналов таймера. Эти сигналы генерируются аппаратурой через определённые интервалы времени и, при возникновении такого сигнала, возникает прерывание с определённым значением вектора.

Выполнение работы.

Написание работы производилось на базе операционной системы Windows 10 в редакторе Visual Code. Сборка, отладка производились на базе эмулятора DOSBox 0.74-3.

Был написан текст исходного .EXE модуля с именем lr4.exe. Описание процедур в программе представлено в табл. 1.

Таблица 1 – Описание процедур программы

Название	**Назначение**
PRINT_STRING	Вывод на экран строки, адрес которой содержится в DX
INTERRUPTION	Процедура обработчика прерывания. В процедуре производится: • Установка позиции курсора и ее восстановление с помощью прерывания int 10h • Увеличение значения счетчика, являющегося строкой из цифр • Вывод на экран значения счетчика с помощью прерывания int 10h
INT_CHECK	Процедура проверки установки резидента INTERRUPTION
INT_LOAD	Процедура загрузки резидентной функции INTERRUPTION
INT_UNLOAD	Процедура загрузки резидентной функции INTERRUPTION (восстановление исходного обработчика прерывания системного таймера)
CL_CHECK	Процедура проверки параметра командной строки

Вывод программы после первого запуска представлен на рис. 1.

Рисунок 1 – Выполнение lr4.exe

Вывод программы lr31.com после запуска lr4.exe представлен на рис. 2.

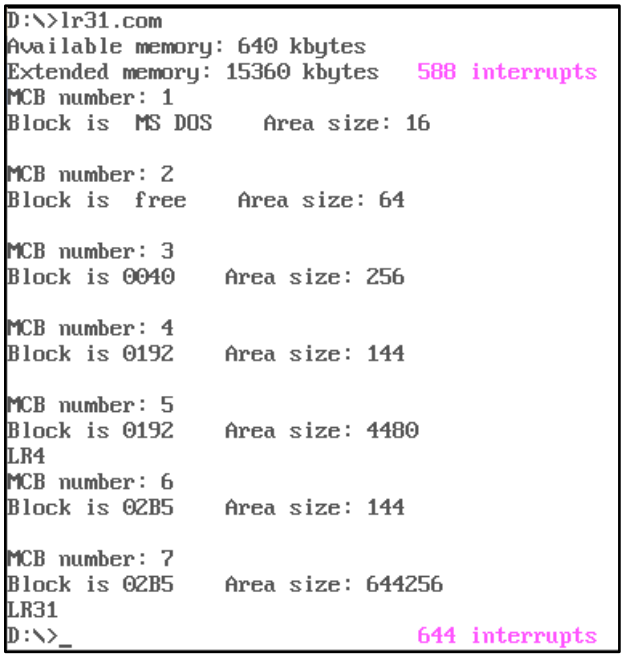

Рисунок 2 – Выполнение lr31.com после запуска lr4.exe

На рисунке видно, что процедура прерывания осталась резидентной в памяти и располагается в блоке 5.

Далее программа lr4.exe была запущена еще раз, в результате чего было выведено сообщение о том, что программа определила уже установленный обработчик прерывания, что представлено на рис. 3.

Рисунок 3 – Повторный запуск lr4.exe

Далее программа lr4.exe была запущена с параметром командной строки “/un” для выгрузки резидентного обработчика прерываний, а таже была запущена

программа lr31.com для вывода блоков MCB. Результат выполнения программы представлен на рис. 4.

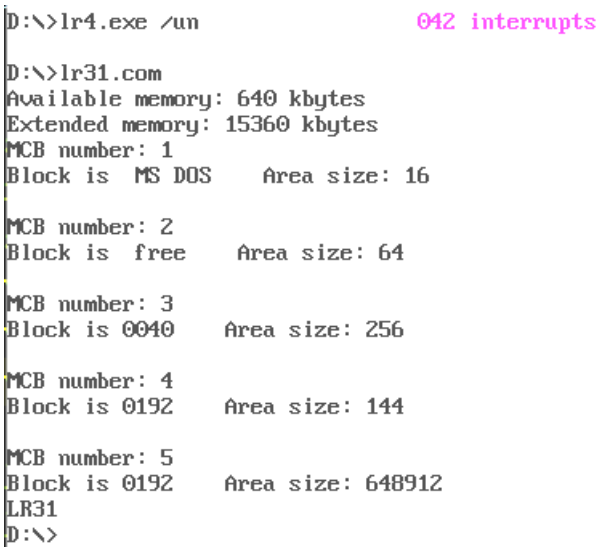

Рисунок 4 – Выгрузка обработчика и выполнение lr31.com

Из рисунка видно, что память для резидентного обработчика была освобождена (ранее он занимал блок 5). Также видно, что пользовательский обработчик прерывания прекратил работу (после выполнения lr31.com внизу командной строки больше не выводится число вызовов прерывания таймера).

Контрольные вопросы.

1. **Как реализован механизм прерывания от часов?**

1. Прерывание int 8h от системного таймера является аппаратным и срабатывает 1193180/65536 раз в секунду
2. При инициализации BIOS устанавливает свой обработчик для прерывания таймера. Этот обработчик каждый раз увеличивает на 1 текущее значение счетчика тиков таймера.
3. В конце этот обработчик прерывания вызывает прерывание int 1Ch – пользовательское прерывание по таймеру (по соответствующему адресу в таблице векторов прерываний). После инициализации системы вектор INT 1Ch указывает на команду IRET, однако в реализованной в данной работе программе вектор указывает на пользовательский обработчик,

который выполняет вывод на экран счетчика вызовов прерывания системного таймера.

4. Во время выполнения int 8h и int 1Ch все аппаратные прерывания (например, прерывания от клавиатуры) не вызываются.
5. После выполнения обработчиков осуществляется возврат к коду, выполнение которого было прервано

2. Какого типа прерывания использовались в программе?

- Был написан обработчик асинхронного прерывания от таймера, которое является аппаратным.
- Были использованы программные прерывания, например, int 21h, int 10h.

Выводы.

В ходе выполнения лабораторной работы была реализована программа загружающая и выгружающая пользовательское прерывание от системного таймера в память.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ. LR4.ASM

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

INTERRUPTION    PROC    FAR
        jmp     INT_START
    INT_DATA:
        COUNTER_INFO    DB  "000 interrupts"
        INT_CODE        DW  3158h

        KEEP_IP   DW  0
        KEEP_CS   DW  0
            KEEP_PSP    DW    0

    INT_START:
            push  AX
            push    BX
            push    CX
            push    DX
            push    SI
        push    ES
        push    DS
    STACK_SETUP:
            mov   AX, seg COUNTER_INFO
            mov   DS, AX

    SET_CURSOR:
        mov     AH, 03h
            mov     BH, 0h
            int     10h ; получение позиции курсора
        push    DX

        mov     AH, 02h
            mov     BH, 0h
            mov     DX, 1820h
            int     10h ; установка позиции курсора

      INCREASE:
            mov   AX, SEG COUNTER_INFO
            push  DS
            mov   DS, AX
            mov   SI, offset COUNTER_INFO
            add         SI, 2
            mov   CX, 3
```

```
      INT_CYCLE:
            mov   AH, [SI]
            inc   AH
            mov   [SI], AH
            cmp   AH, ':'
            jne   INT_END_CYCLE
            mov   AH, '0'
            mov   [SI], AH
            dec   SI
            loop  INT_CYCLE
      INT_END_CYCLE:
            pop   DS

      PRINT:
            push  ES
            push  BP
        mov     AX, SEG COUNTER_INFO
            mov     ES, AX
            mov     BP, offset COUNTER_INFO
            mov     AH, 13h
            mov     AL, 1h
            mov   BL, 2h
            mov     BH, 0
            mov     CX, 14
            int     10h ; вывод на экран

            pop         BP
            pop         ES

        pop     DX
        mov     AH, 02h ; восстановление курсора
            mov     BH, 0h
            int     10h

            pop     DS
            pop     ES
            pop         SI
            pop     DX
            pop     CX
            pop     BX
            pop         AX

            mov     AL, 20h
            out     20h, AL
      iret
INTERRUPTION    ENDP
    LAST_BYTE:
```

```
INT_CHECK       PROC
            push    AX
            push    BX
            push    SI

            mov     AH, 35h
            mov     AL, 1Ch
            int     21h
            mov     SI, offset INT_CODE
            sub     SI, offset INTERRUPTION
            mov     AX, ES:[BX + SI]
            cmp       AX, INT_CODE
            jne     INT_CHECK_END
            mov     INT_LOADED, 1

      INT_CHECK_END:
            pop     SI
            pop     BX
            pop     AX
      ret
INT_CHECK       ENDP

INT_LOAD        PROC
        push    AX
            push    BX
            push    CX
            push    DX
            push    ES
            push    DS

        mov     AH, 35h
            mov     AL, 1Ch
            int     21h
            mov     KEEP_CS, ES
        mov     KEEP_IP, BX
        mov     AX, seg INTERRUPTION
            mov     DX, offset INTERRUPTION
            mov     DS, AX
            mov     AH, 25h
            mov     AL, 1Ch
            int     21h
            pop         DS

        mov     DX, offset LAST_BYTE
            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
INT_LOAD        ENDP

INT_UNLOAD      PROC
        CLI
            push    AX
            push    BX
            push    DX
            push    DS
            push    ES
            push    SI

            mov     AH, 35h
            mov     AL, 1Ch
            int     21h
            mov   SI, offset KEEP_IP
            sub   SI, offset INTERRUPTION
            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     BX
            pop     AX

      ret
INT_UNLOAD      ENDP

CL_CHECK        PROC
        push    AX
            push    ES

            mov     AX, KEEP_PSP
            mov     ES, AX
            cmp     byte ptr ES:[82h], '/'
            jne     CL_CHECK_END
            cmp     byte ptr ES:[83h], 'u'
            jne     CL_CHECK_END
            cmp     byte ptr ES:[84h], 'n'
            jne     CL_CHECK_END
            mov     UN_CL, 1

      CL_CHECK_END:
            pop     ES
            pop     AX
            ret
CL_CHECK        ENDP

PRINT_STRING    PROC    NEAR
        push    AX
        mov     AH, 09h
        int     21h
        pop     AX
    ret
PRINT_STRING    ENDP

MAIN PROC
            push    DS
            xor     AX, AX
            push    AX
            mov     AX, DATA
```

```
            mov     DS, AX
            mov     KEEP_PSP, ES

            call    INT_CHECK
            call    CL_CHECK
            cmp     UN_CL, 1
            je      UNLOAD
            mov     AL, INT_LOADED
            cmp     AL, 1
            jne     LOAD
            mov     DX, offset WAS_LOADED_INFO
            call    PRINT_STRING
            jmp     MAIN_END
      LOAD:
            call    INT_LOAD
            jmp     MAIN_END
      UNLOAD:
            cmp     INT_LOADED, 1
            jne     NOT_EXIST
            call    INT_UNLOAD
            jmp     MAIN_END
      NOT_EXIST:
            mov     DX, offset NOT_LOADED_INFO
            call    PRINT_STRING
      MAIN_END:
            xor   AL, AL
            mov   AH, 4Ch
            int   21h
      MAIN ENDP

CODE    ENDS

ASTACK  SEGMENT STACK
    DW  128 dup(0)
ASTACK  ENDS

DATA    SEGMENT
      WAS_LOADED_INFO     DB  "Interruption was already loaded", 10, 13,"$"
      NOT_LOADED_INFO         DB  "Interruption is not loaded", 10, 13,"$"
    INT_LOADED          DB  0
    UN_CL               DB  0
DATA    ENDS

END   MAIN
```