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.

Студентка гр. 9383 _____ Лысова А.М.

Преподаватель _____ Ефремов М.А.

Санкт-Петербург
2021

Цель работы.

В данной лабораторной 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. Таким образом, управление будет передано функции, чья точка входа записана в соответствующий вектор прерывания.

Задание.

Необходимо написать и отладить программный модуль типа .EXE, который выполняет следующие функции:

1. проверяет, установлено ли пользовательское прерывание с вектором 1Ch;

2. устанавливает резидентную функцию для обработки прерывания и настраивает вектор прерывания, если прерывание не установлено;

3. если прерывание установлено, то выводится соответствующее сообщение;

4. выгружает прерывания по соответствующему значению параметра в командные строки ‘/un’.

Программа должна содержать код устанавливаемого прерывания в виде удалённой процедуры. Этот код будет работать после установки при возникновении прерывания. Он должен выполнять следующие функции:

1. сохранять стек прерванной программы в рабочих переменных и восстанавливать при выходе;

2. организовать свой стек;

3. сохранить значения регистров в стеке при входе и восстановить их при выходе;

4. при выполнении тела процедуры накапливать общее суммарное число прерываний и выводить на экран.

Запустить отлаженную программу и убедиться, что резидентный обработчик прерывания 1Ch установлен. Запустить отлаженную программу ещё раз и убедиться, что программа определяет установленный обработчик прерываний.

Выполнение работы.

Написан и отлажен программный модуль типа .EXE, который выполняет следующие функции:

- Проверяет, установлено ли пользовательское прерывание с вектором 1Ch.
- 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.
- 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.

Используемые функции.

- MY_INTERRUPT — код резидентного обработчика прерываний 1Ch.
- LOAD — функция для загрузки обработчика прерываний.
- UNLOAD — функция для выгрузки обработчика прерываний.
- LOAD_FLAG — функция для установки флага(ключа) выгрузки.
- IS_LOAD — функция для проверки установки обработчика прерываний.
- PRINT_STRING — функция для вывода строки на экран.

Резидентная функция в начале своего выполнения сохраняет в переменных регистры SS, SP и AX (последний используется в создании собственного стека, поэтому значение сохраняется в переменную), создаёт свой стек и сохраняет все регистры в него.

Затем запоминает позицию курсора, чтобы в конце можно было восстановить его, и устанавливает его на позицию (0; 0). В строке, являющейся по совместительству счётчиком, увеличивается число обработанных прерываний, а затем строка выводится на экран. Курсор возвращается на первоначальное местоположение, все регистры восстанавливаются.

Рисунок 1: Загрузка прерывания в память.

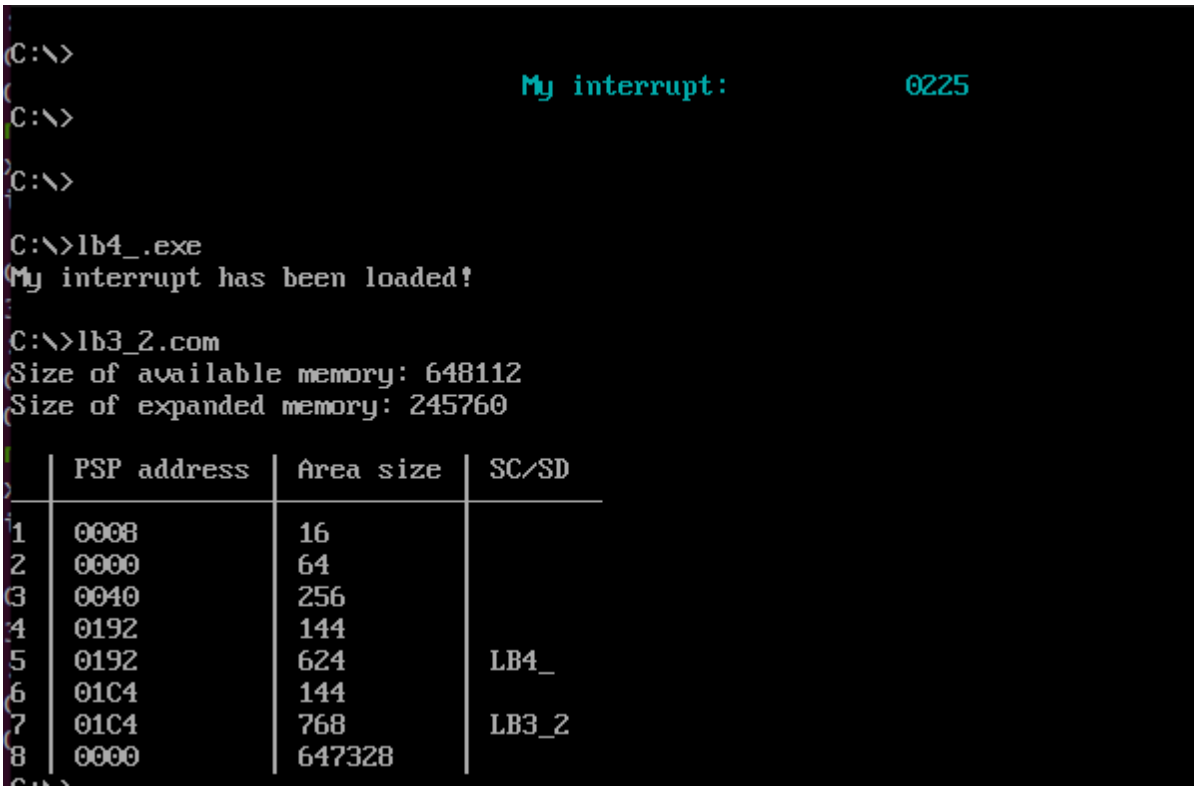

Рисунок 2: Запуск программы lb3_2 параллельно прерыванию

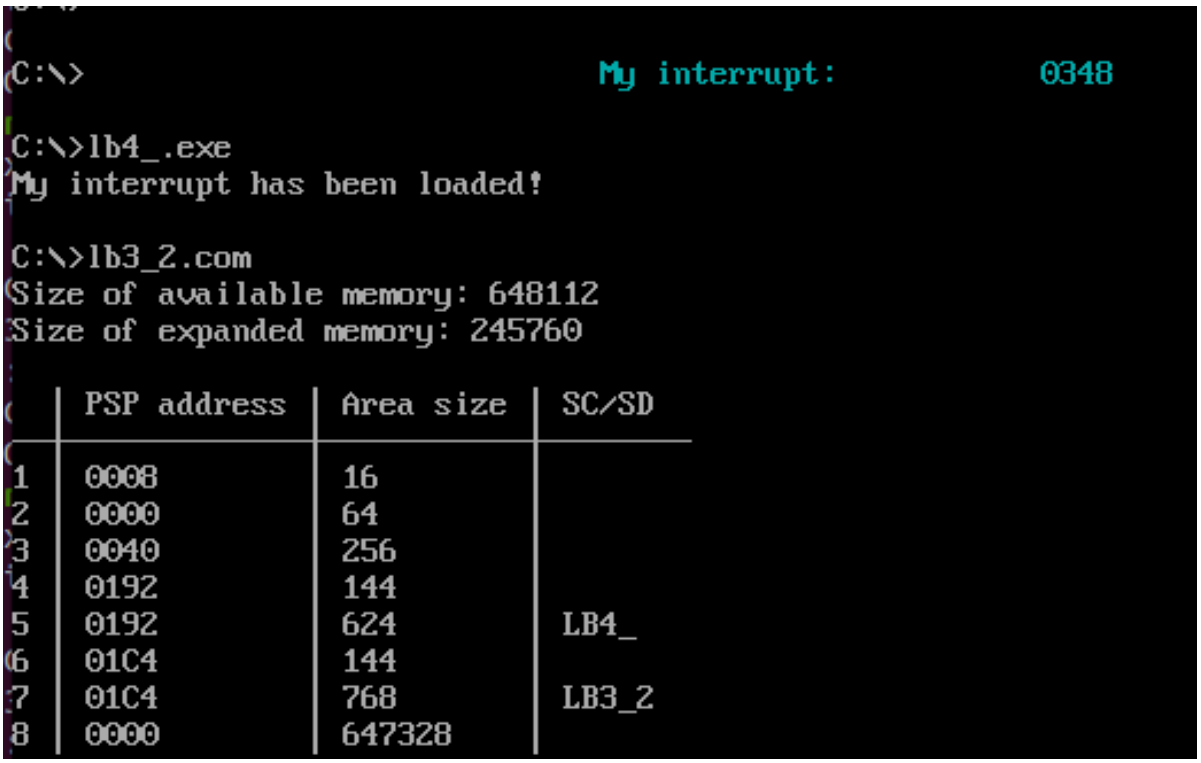

Рисунок 3: Повторная загрузка прерывания в память.

Контрольные вопросы.

1) Как реализован механизм обработчика прерывания от часов?

Каждый такт из таймера вычитается определённое значение. Когда значение достигает 0, возникает прерывание от таймера. При возникновении прерывания процессор запоминает в стеке адрес возврата (CS:IP) и регистр флагов. Затем в CS:IP загружается адрес обработчика прерывания и выполняется его код. В конце регистры восстанавливаются, и процессор возвращается на выполнение прерванной программы.

2) Какого типа прерывания использовались в работе?

В данной работе использовались аппаратное прерывание 21h с вектором 1Ch, а также пользовательские прерывания 10h и 21h.

Выводы.

В ходе работы были исследованы структуры обработчиков стандартных прерываний, был построен обработчик прерываний сигналов таймера.

ПРИЛОЖЕНИЕ А.
ИСХОДНЫЙ КОД.

Название файла: lb4.asm

```
AStack     SEGMENT STACK
     DB 256 DUP(?)
AStack ENDS

DATA  SEGMENT

     flag DB 0
     INT_LOAD DB 'My interrupt has been loaded!', 0DH, 0AH, '$'
     INT_UNLOAD DB 'My interrupt has been unloaded!', 0DH, 0AH, '$'
     INT_NOT_LOAD DB 'My interrupt has not been loaded!', 0DH, 0AH, '$'
     INT_ALREADY_LOAD DB 'My interrupt has already been loaded!', 0DH,
0AH, '$'

DATA  ENDS

CODE  SEGMENT
     ASSUME CS:CODE, DS:DATA, SS:AStack

MY_INTERRUPT    PROC far
     jmp start_interrupt

     SIGN DW 7777h
     PSP DW ?

     KEEP_IP DW 0
     KEEP_CS DW 0
     KEEP_SS DW 0
     KEEP_SP DW 0
     KEEP_AX DW 0

     INT_COUNTER DB 'My interrupt: 0000$'

     INT_STACK DW 128 DUP (?)
END_INT_STACK:
```

```
start_interrupt:
      mov KEEP_SS, SS
      mov KEEP_SP, SP
      mov KEEP_AX, AX

      mov AX, CS
      mov SS, AX
      mov SP, OFFSET END_INT_STACK

      push BX
      push CX
      push DX

      ; get cursor

      mov AH, 3h
      mov BH, 0h
      int 10h
      push DX

      ; set cursor

      mov AH, 2h
      mov BH, 0h
      mov DH, 2h
      mov DL, 5h
      int 10h

      ; inc counter

      push BP
      push SI
      push CX
      push DS

      mov AX, SEG INT_COUNTER
      mov DS, AX
      mov SI, OFFSET INT_COUNTER
```

```
      add SI, 14
      mov CX, 4

int_loop:
      mov BP, CX

      ;dec BP

      mov AH, [SI+BP]
      inc AH
      mov [SI+BP], AH
      cmp AH, 3Ah
      jne succes
      mov AH, 30h
      mov [SI+BP], AH

      loop int_loop

succes:
      pop DS
      pop CX
      pop SI

      push ES
      mov DX, SEG INT_COUNTER
      mov ES, DX
      mov BP, OFFSET INT_COUNTER
      mov AH, 13h
      mov AL, 0h
      mov CX, 18
      mov DX, 0h
      int 10h

      pop ES
      pop BP

      ; return cursor

      mov AH, 2h
```

```
      mov BH, 0h
      pop DX
      int 10h

      ; end resident program

      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
end_interrupt:
MY_INTERRUPT    ENDP

;-------------------------------------------------------------------

LOAD  PROC near

      push    AX
      push    CX
      push    DX

      ; storing offset and segment

      mov     AH, 35h
      mov     AL, 1Ch
      int     21h
      mov     KEEP_IP, BX
      mov     KEEP_CS, ES

      ; Interrupt setting
```

```
        push    DS
        mov     DX, OFFSET MY_INTERRUPT
        mov     AX, SEG MY_INTERRUPT
        mov     DS, AX
        mov     AH, 25h
        mov     AL, 1Ch
        int     21h
        pop     DS

      ; Resident program preservation

        mov     DX, OFFSET END_INT_STACK
        mov     CL, 4
        shr     DX, CL
        inc     DX
        mov     AX, CS
        sub     AX, PSP
        add     DX, AX
        xor     AX, AX
        mov     AH, 31h
        int     21h
        pop     DX
        pop     CX
        pop     AX
        ret

LOAD  ENDP

;-----------------------------------------------------

UNLOAD     PROC near

        push    AX
        push    DX
        push    SI
        push    ES

      ; Recovery offset and segment
```

```
        cli
        push    DS
        mov     AH, 35h
        mov     AL, 1Ch
        int     21h
        mov     SI, OFFSET KEEP_CS
        sub     SI, OFFSET MY_INTERRUPT
        mov     DX, ES:[BX+SI+2]
        mov     AX, ES:[BX+SI]
        mov     DS, AX
        mov     AH, 25h
        mov     AL, 1Ch
        int     21h
        pop     DS
        mov     AX, ES:[BX+SI-2]
        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     ES
        pop     SI
        pop     DX
        pop     AX
        ret

UNLOAD    ENDP

;-----------------------------------------------------------

LOAD_FLAG  PROC near

        push    AX
        mov     AL, ES:[82h]
```

```
        cmp     AL, '/'
        jne     end_load_flag
        mov     AL, ES:[83h]
        cmp     AL, 'u'
        jne     end_load_flag
        mov     AL, ES:[84h]
        cmp     AL, 'n'
        jne     end_load_flag
        mov     flag, 1
end_load_flag:
     pop     AX

LOAD_FLAG  ENDP

;-------------------------------------------------------------------

IS_LOAD    PROC near

        push    AX
        push    DX
        push    SI
        mov     flag, 1
        mov     AH, 35h
        mov     AL, 1Ch
        int     21h
        mov     SI, OFFSET SIGN
        sub     SI, OFFSET MY_INTERRUPT
        mov     DX, ES:[BX+SI]
        cmp     DX, 7777h
        je      loading

        mov     flag, 0
loading:
     pop     SI
        pop     DX
        pop     AX
        ret

IS_LOAD    ENDP
```

```
;-------------------------------------------------

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

;-------------------------------------------------------

MAIN  PROC far

        mov     AX, DATA
        mov     DS, AX
        mov     PSP, ES
        mov     flag, 0
        call    LOAD_FLAG
        cmp     flag, 1
        je      un

     ; loading

        call    IS_LOAD
        cmp     flag, 0
        je      notld
        mov     DX, OFFSET INT_ALREADY_LOAD
        call    PRINT_STRING
        jmp     fin

notld:  mov     DX, OFFSET INT_LOAD
        call    PRINT_STRING
        call    LOAD
        jmp     fin

     ; unloading
```

```
un:     call    IS_LOAD
        cmp     flag, 0
        jne     alrld
        mov     DX, OFFSET INT_NOT_LOAD
        call    PRINT_STRING
        jmp     fin

        ; already loading

alrld:  call    UNLOAD
        mov     DX, OFFSET INT_UNLOAD
        call    PRINT_STRING

fin:    mov     AX, 4Ch
        int     21h

MAIN  ENDP
CODE  ENDS
      END   MAIN
```