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. 0382 _____ Санников В.А.

Преподаватель _____ Ефремов М.А.

Санкт-Петербург

2022

Цель работы.

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

В лабораторной работе № 4 предлагается построить обработчик прерываний сигналов таймера. Эти сигналы генерируются аппаратурой через определенные интервалы времени и, при возникновении такого сигнала, возникает прерывание с определенным значением вектора. Таким образом, управление будет передано функции, чья точка входа записана в соответствующий вектор прерывания.

Задание.

Шаг 1. Для выполнения лабораторной работы 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 прерываний, если прерывание не установлено, и осуществляется выход по функции 4Ch прерывания int 21h.

3) Если прерывание установлено, то выводится соответствующее сообщение и осуществляется выход по функции 4Ch прерывания int 21h.

4) 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.

Для того, чтобы проверить установку прерывания, можно поступить следующим образом. Прочитать адрес, записанный в векторе прерывания. Предположим, что этот адрес указывает на точку входа в установленный резидент. На определенном, известном смещении в теле резидента располагается сигнатура, некоторый код, который идентифицирует резидент. Сравнив известное значение сигнатуры с реальным кодом, находящимся в резиденте, можно определить, установлен ли резидент. Если значения совпадают, то резидент установлен. Длину кода сигнатуры должна быть достаточной, чтобы сделать случайное совпадение маловероятным. Программа должна содержать код устанавливаемого прерывания в виде удаленной процедуры. Этот код будет работать после установки при возникновении прерывания. Он должен выполнять следующие функции:

1) Сохраняет стек прерванной программы (регистры SS и SP) в рабочих переменных и восстановить 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. Для вывода на экран следует использовать прерывание int 10h, которое позволяет непосредственно выводить информацию на экран.

5) Функция прерывания должна содержать только переменные, которые она использует

Шаг 2. Запустите отлаженную программу и убедитесь, что резидентный обработчик прерывания 1Ch установлен. Работа прерывания должна отображаться на экране, а также необходимо проверить размещение прерывания в памяти. Для этого запустите программу ЛР 3, которая отображает

4 карту памяти в виде списка блоков MCB. Полученные результаты поместите в отчет.

Шаг 3. Запустите отлаженную программу еще раз и убедитесь, что программа определяет установленный обработчик прерываний. Полученные результаты поместите в отчет.

Шаг 4. Запустите отлаженную программу с ключом выгрузки и убедитесь, что резидентный обработчик прерывания выгружен, то есть сообщения на экран не выводятся, а память, занятая резидентом освобождена. Для этого также следует запустить программу ЛР 3. Полученные результаты поместите в отчет.

Шаг 5. Ответьте на контрольные вопросы

Ход работы.

Для выполнения лабораторной работы был написан .EXE модуль, который содержит следующие процедуры:

1)INTERRUPT — обработчик прерывания.

2)IS_LOADED — процедура, которая проверяет загрузку обработчика прерывания в память.

3)LOAD_INT — загружает обработчик прерывания в память.

4)UNLOAD_INT — процедура, которая выгружает обработчик прерывания из памяти.

5)CHECK_CMD_KEY — процедура для определения флага /nu командной строки.

Результаты выполнения пунктов задания приведены на следующих рисунках:

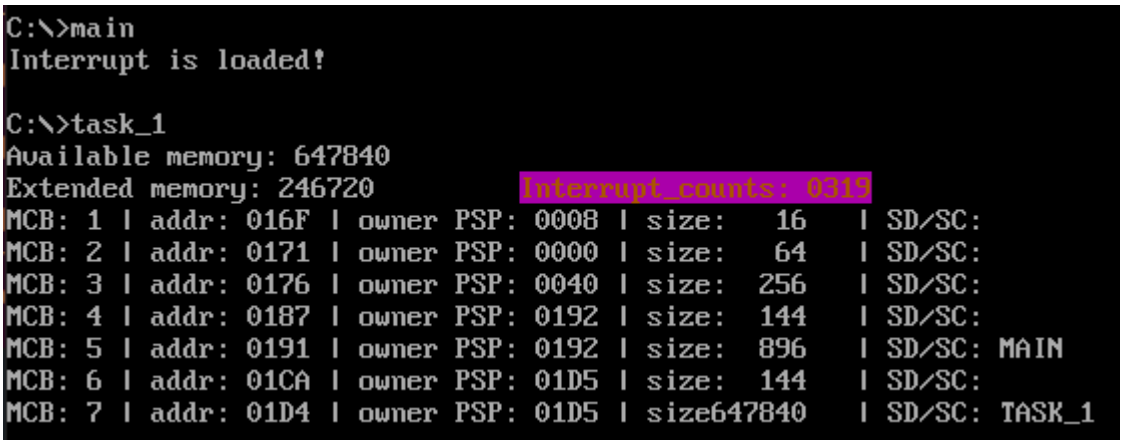

Рисунок 1 — Загрузка обработчик прерывания в память и запуск .COM модуля task_1.com

Как видно из данного рисунка, обработчик прерывания находится в памяти.

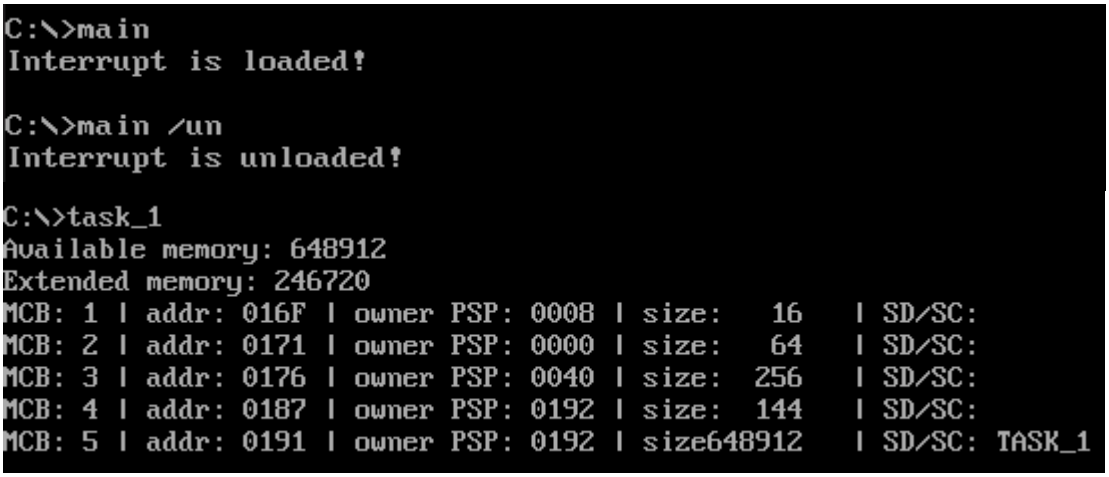

Рисунок 2 — Загрузка обработчик прерывания в память, его выгрузка и запуск task_1.com

На данном шаге мы загрузили, а затем выгрузили обработчик прерываний из памяти. На рисунке видно, что он действительно не занимает память.

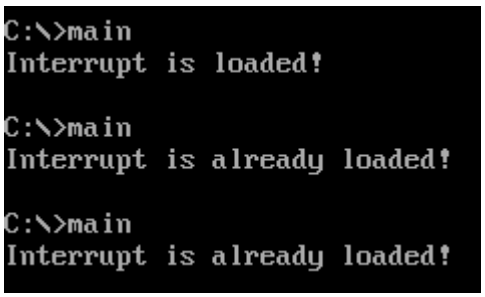

Рисунок 3 — Результат загрузки обработчика прерываний в память несколько раз.

В данном случае была произведена загрузка обработчика в память несколько раз, программа отвечает тем, что обработчик уже загружен.

Исходный код программы см в приложении А.

Ответы на контрольные вопросы.

1. Как реализован механизм прерывания от часов?

Аппаратура генерирует сигналы через определённые временные интервалы (18.2 раз в секунду). После каждого сигнала происходит прерывание с номером 1Ch. При замене обработчика этого прерывания в таблице на пользовательскую функцию, после каждого сигнала будет выполняться именно она.

2. Какого типа прерывания использовались в работе?

Программные (21h — функции DOS, 10h — видеосервис функция DOS) и аппаратные (1Ch).

Вывод.

В ходе работы был построен обработчик прерываний таймера, который выводит количество прерываний на экран. Также были приведены наглядные примеры с выгрузкой и загрузкой обработчика в память.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Файл main.asm

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

DATA SEGMENT
     FLAG db 0
     LOADED_MES db 'Interrupt is loaded!', 0DH, 0AH,'$'
     UNLOADED_MES db 'Interrupt is unloaded!', 0DH, 0AH,'$'
     ALREADY_LOADED_MES db 'Interrupt is already loaded!', 0DH,
0AH,'$'
DATA ENDS

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

     INTERRUPT PROC FAR
          jmp int_start
          PSP DW ?
          KEEP_CS DW ?
          KEEP_IP DW ?
          KEEP_SS DW ?
          KEEP_SP DW ?
          INT_ID DW 1f17h
          COUNTER DB 'Interrupt_counts: 0000$'
          INT_STACK DB 128 dup(?)

          int_start:
               mov KEEP_SS, SS
               mov KEEP_SP, SP
               mov SP, SEG INTERRUPT
               mov SS, SP
               mov SP, OFFSET int_start
               push DS
               push ES
               push AX
               push BX
               push CX
               push DX
               push SI
               push BP

               mov AH, 03h
               mov BH, 0
               int 10h
               push DX
```

```
mov AH, 02h
mov BH, 0
mov DL, 20h
mov DH, 10h
int 10h

mov SI, SEG COUNTER
mov DS, SI
mov SI, OFFSET COUNTER
add SI, 17

mov CX, 4
num_loop:
     mov BP, CX
     mov AH, [SI+BP]
     inc AH
     mov [SI+BP], AH
     cmp AH, 3Ah
     jne num_loop_end
     mov AH, 30h
     mov [SI+BP], AH
     loop num_loop
num_loop_end:

mov BP, SEG COUNTER
mov ES, BP
mov BP, OFFSET COUNTER
mov AH, 13h
mov AL, 1
mov BH, 0
mov CX, 22
int 10h

mov AH, 02h
mov BH, 0
pop DX
int 10h

pop BP
pop SI
pop DX
pop CX
pop BX
pop AX
pop ES
pop DS
mov SP, KEEP_SS
mov SS, SP
mov SP, KEEP_SP
mov AL, 20h
out 20h, AL
iret
```

```
     int_end:
INTERRUPT ENDP

IS_LOADED PROC NEAR
     push AX
     push BX
     push DX
     push SI

     mov FLAG, 1
     mov AH, 35h
     mov AL, 1Ch
     int 21h
     mov SI, OFFSET INT_ID
     sub SI, OFFSET INTERRUPT
     mov DX, ES:[BX+SI]
     cmp DX, 1f17h
     je if_loaded
     mov FLAG, 0

     if_loaded:
          pop SI
          pop DX
          pop BX
          pop AX
     ret
IS_LOADED ENDP

LOAD_INT PROC NEAR
     push AX
     push BX
     push CX
     push DX
     push DS
     push ES

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

     mov DX, offset INTERRUPT
     mov AX, seg INTERRUPT
     mov DS, AX
     mov AH, 25h
     mov AL, 1Ch
     int 21h

     mov DX, offset int_end
     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 ES
      pop DS
      pop DX
      pop CX
      pop BX
      pop AX
      ret
LOAD_INT ENDP

UNLOAD_INT PROC NEAR
      push AX
      push BX
      push DX
      push DS
      push ES

      cli
      mov AH, 35h
      mov AL, 1Ch
      int 21h
      mov DX, ES:[offset KEEP_IP]
      mov AX, ES:[offset KEEP_CS]
      mov DS, AX
      mov AH, 25h
      mov AL, 1Ch
      int 21h

      mov AX, ES:[offset PSP]
      mov ES, AX
      mov DX, ES:[2Ch]
      mov AH, 49h
      int 21h
      mov ES, DX
      mov AH, 49h
      int 21h
      sti

      pop ES
      pop DS
      pop DX
      pop BX
      pop AX
      ret
UNLOAD_INT ENDP
```

```
CHECK_CMD_KEY PROC NEAR
      push AX

      mov FLAG, 0
      mov AL, ES:[82h]
      cmp AL, '/'
      jne if_no_key
      mov AL, ES:[83h]
      cmp AL, 'u'
      jne if_no_key
      mov AL, ES:[84h]
      cmp AL, 'n'
      jne if_no_key
      mov FLAG, 1

      if_no_key:
            pop AX
            ret
CHECK_CMD_KEY ENDP

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

main PROC FAR
      push DS
      xor AX, AX
      mov AX, DATA
      mov DS, AX

      mov PSP, ES
      call CHECK_CMD_KEY
      cmp FLAG, 1
      je int_unload

      call IS_LOADED
      cmp FLAG, 0
      je int_load
      mov DX, OFFSET ALREADY_LOADED_MES
      call PRINT
      jmp exit

      int_load:
            mov DX, OFFSET LOADED_MES
            call PRINT
            call LOAD_INT
            jmp exit
```

```
            int_unload:
                  call IS_LOADED
                  cmp FLAG, 0
                  je unloaded
                  call UNLOAD_INT

            unloaded:
                  mov DX, OFFSET UNLOADED_MES
                  call PRINT

            exit:
                  pop DS
                  mov AX, 4C00h
                  int 21h
      main ENDP
CODE ENDS
      END main
```