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

ОТЧЕТ

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

Студент гр. 9383 _____ Чебесова И.Д.

Преподаватель _____ Ефремов М.А.

Санкт-Петербург
2021

ПОСТАНОВКА ЗАДАЧИ

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

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.

РЕЗУЛЬТАТЫ ИССЛЕДОВАНИЯ ПРОБЛЕМ

Шаг 1. Был написан и отлажен программный модуль типа .EXE, который выполняет, прописанные в задании функции.

Шаг 2. Программа была отлажена и запущена. Проверено размещение прерывания в памяти.

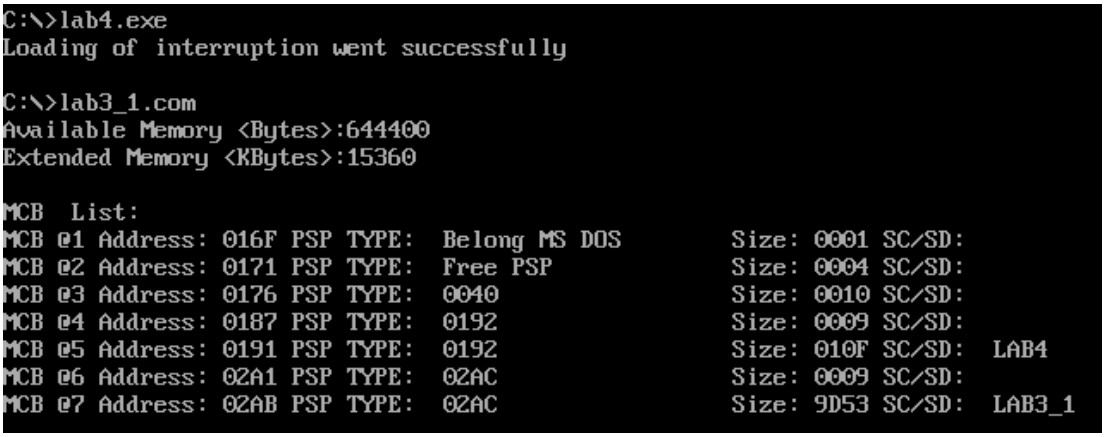

Рисунок 1 – Демонстрация корректной работы программы

Рисунок 2 – Демонстрация корректной работы счетчика

Счетчик моргает и поймать его на каждом рисунке не получилось, поэтому его корректная работа приведена на отдельном рисунке.

Шаг 3. Программа была повторна запущена, чтобы удостовериться, что программа определяет установленный обработчик прерываний.

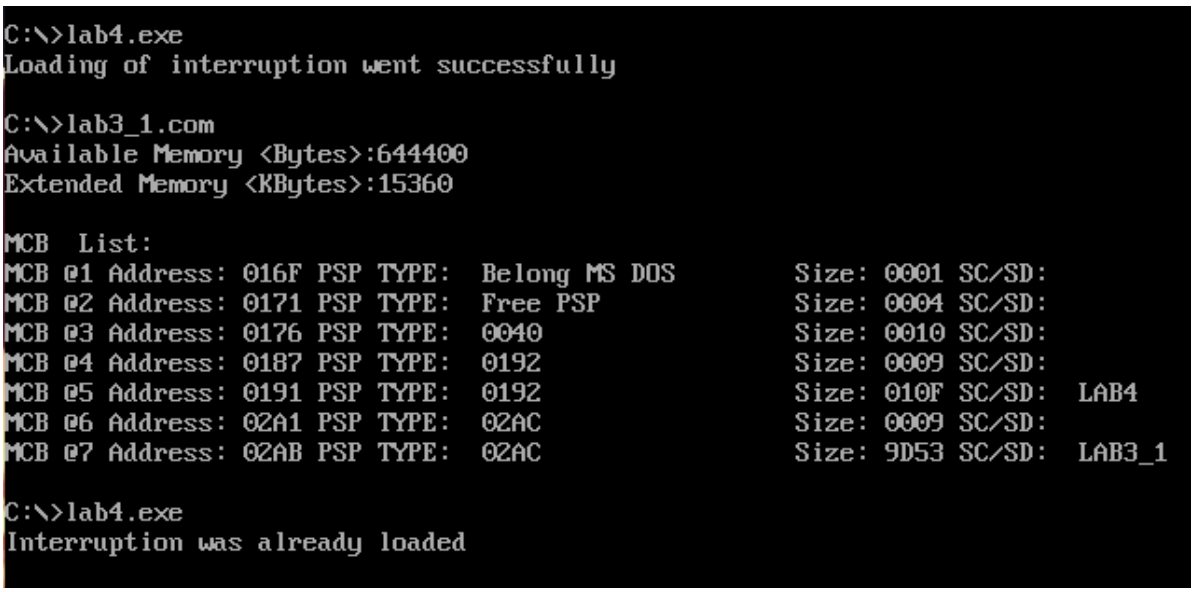

Рисунок 3 – Демонстрация корректной работы при повторном запуске программы.

Шаг 4. Программа была запущена с ключом выгрузки, чтобы убедиться, что резидентный обработчик прерывания выгружен и память, занятая резидентом освобождена.

Рисунок 4 – Демонстрация корректной выгрузки.

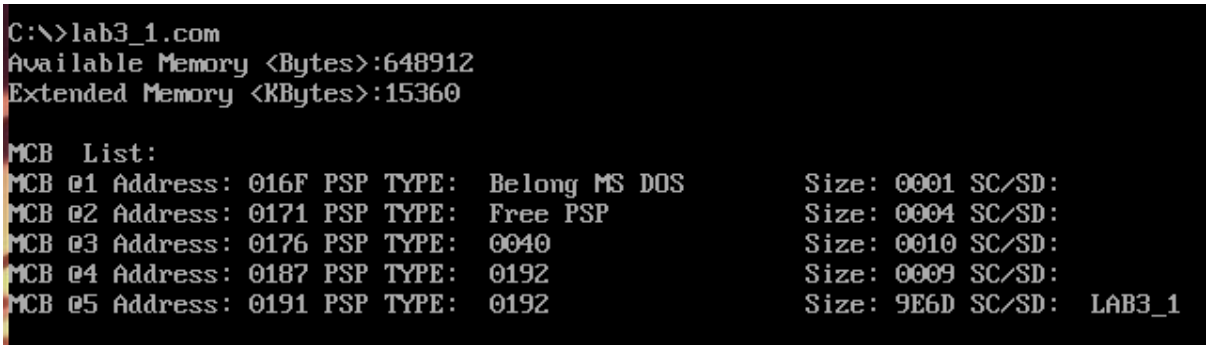

Рисунок 5 – Демонстрация освобождения памяти.

Ответы на контрольные вопросы

1). Как реализован механизм прерывания от часов?

Ответ: прерывание от часов (прерывание int 1Ch) вызывается с помощью обработчика аппаратного прерывания от системного таймера int 08h. Происходит это около 18 раз в секунду.

При вызове данного прерывания прежде всего сохраняются значения регистров IP и CS для дальнейшего возвращения в программу.

Далее по номеру источника прерывания определяется смещение (адрес) вызываемого вектора, он записывается в регистры IP и CS, после чего обработчик прерывания вызывается по этому сохраненному адресу.

В конце управление возвращается прерванной программе от обработчика прерываний.

2). Какого типа прерывания использовались в работе?

Ответ: в данной работе были использованы следующие прерывания: 1Ch – это аппаратное прерывание, а также 10h и 21h – это программные прерывания.

Выводы.

В ходе проделанный работы был построен собственный обработчик прерываний сигналов таймера. Также были получены дополнительные знания о работе с памятью (резидентный обработчик может быть загружен и выгружен из памяти).

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

lab4.asm:

```
ASTACK SEGMENT STACK
   DW 200 DUP(?)
ASTACK ENDS

DATA SEGMENT
    ALREADY_LOAD_STR DB 'INTERRUPTION WAS ALREADY LOADED', 0DH, 0AH,
'$'
    SUCCESS_LOAD_STR DB 'LOADING OF INTERRUPTION WENT SUCCESSFULLY',
0DH, 0AH, '$'
    NOT_LOAD_STR DB 'INTERRUPTION ISNT LOAD', 0DH, 0AH, '$'
    RESTORED_STR DB 'INTERRUPTION IS RESTORED NOW', 0DH, 0AH, '$'
DATA ENDS

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

;-------------------------------------
    PRINT_MESSAGE  PROC  NEAR
          PUSH AX
          MOV AH, 9
          INT 21H
          POP AX
          RET
     PRINT_MESSAGE  ENDP
;-------------------------------------

;-------------------------------------
    SET_CURSOR PROC NEAR
        MOV AH, 02H
        MOV BH, 0H
        MOV DH, 0H
        MOV DL, 0H
        INT 10H
```

```
        RET
    SET_CURSOR ENDP

    GET_CURSOR PROC NEAR
        MOV AH, 03H
        MOV BH, 0
        INT 10H
        RET
    GET_CURSOR ENDP
;-----------------------------------

;-----------------------------------
    MY_INT PROC FAR
        JMP BEGIN_PROC
        COUNTER DB 'INTERRUPTION COUNTER: 0000$'
        SIGNATURE DW 7777H
        KEEP_IP DW 0
        KEEP_CS DW 0
        KEEP_SS DW 0
        KEEP_SP DW 0
        KEEP_AX DW 0
          ADDRESS_OF_PSP DW ?
        MY_STACK DW 16 DUP(?)

    BEGIN_PROC:
        MOV KEEP_SP, SP
        MOV KEEP_AX, AX
        MOV AX, SS
        MOV KEEP_SS, AX
        MOV AX, KEEP_AX
        MOV SP, OFFSET BEGIN_PROC
        MOV AX, SEG MY_STACK
        MOV SS, AX
        PUSH AX
        PUSH CX
        PUSH DX

        CALL GET_CURSOR
```

```
    PUSH DX
    CALL SET_CURSOR
    PUSH SI
     PUSH CX
     PUSH DS
     PUSH BP
    MOV AX, SEG COUNTER
 MOV DS, AX
 MOV SI, OFFSET COUNTER
 ADD SI, 21
      MOV CX, 4

LOOP_FOR_COUNT:
    MOV BP, CX
    MOV AH, [SI+BP]
    INC AH
    MOV [SI+BP], AH
    CMP AH, 3AH
    JNE FOR_PRINT
    MOV AH, 30H
    MOV [SI+BP], AH
       LOOP LOOP_FOR_COUNT

FOR_PRINT:
       POP BP
       POP DS
       POP CX
       POP SI
 PUSH ES
 PUSH BP
 MOV AX, SEG COUNTER
 MOV ES,AX
 MOV AX, OFFSET COUNTER
 MOV BP,AX
 MOV AH, 13H
 MOV AL, 00H
 MOV CX, 26
 MOV BH,0
```

```
      INT 10H
      POP BP
      POP ES
      POP DX
      MOV AH,02H
      MOV BH,0H
      INT 10H

         POP DX
         POP CX
         POP AX
         MOV KEEP_AX, AX
         MOV SP, KEEP_SP
         MOV AX, KEEP_SS
         MOV SS, AX
         MOV AX, KEEP_AX
         MOV AL, 20H
         OUT 20H, AL
         IRET
    MY_INT_LAST:
    MY_INT ENDP
;-------------------------------------

;-------------------------------------
    CHECK_UPLOAD_KEY PROC NEAR
           PUSH AX
         PUSH BP
         MOV CL, 0H
         MOV BP, 81H
           MOV AL, ES:[BP + 1]
           CMP AL, '/'
           JNE EXIT
           MOV AL, ES:[BP + 2]
           CMP AL, 'U'
           JNE EXIT
           MOV AL, ES:[BP + 3]
           CMP AL, 'N'
           JNE EXIT
```

```
      MOV CL, 1H

EXIT:
    POP BP
      POP AX
      RET
CHECK_UPLOAD_KEY ENDP


IF_ALREADY_LOAD PROC NEAR
    PUSH AX
    PUSH DX
    PUSH ES
    PUSH SI
    MOV CL, 0H
    MOV AH, 35H
    MOV AL, 1CH
    INT 21H
    MOV SI, OFFSET SIGNATURE
    SUB SI, OFFSET MY_INT
    MOV DX, ES:[BX+SI]
    CMP DX, SIGNATURE
    JNE IF_END
    MOV CL, 1H

IF_END:
    POP SI
    POP ES
    POP DX
    POP AX
    RET
IF_ALREADY_LOAD ENDP


LOAD PROC NEAR
      PUSH AX
    PUSH CX
      PUSH DX
```

```
      CALL IF_ALREADY_LOAD
      CMP CL, 1H
      JE ALREADY_LOAD
   MOV ADDRESS_OF_PSP, ES
      MOV AH, 35H
 MOV AL, 1CH
 INT 21H
   MOV KEEP_CS, ES
    MOV KEEP_IP, BX
 PUSH ES
   PUSH BX
      PUSH DS
      LEA DX, MY_INT
      MOV AX, SEG MY_INT
      MOV DS, AX
      MOV AH, 25H
      MOV AL, 1CH
      INT 21H
      POP DS
   POP BX
   POP ES
   MOV DX, OFFSET SUCCESS_LOAD_STR
      CALL PRINT_MESSAGE
      LEA DX, MY_INT_LAST
      MOV CL, 4H
      SHR DX, CL
      INC DX
      ADD DX, 100H
      XOR AX,AX
      MOV AH, 31H
      INT 21H
   JMP END_LOAD

ALREADY_LOAD:
 MOV DX, OFFSET ALREADY_LOAD_STR
   CALL PRINT_MESSAGE

END_LOAD:
```

```
      POP DX
    POP CX
      POP AX
      RET
LOAD ENDP


UNLOAD PROC NEAR
      PUSH AX
      PUSH SI
      CALL IF_ALREADY_LOAD
      CMP CL, 1H
      JNE NOT_LOAD
    CLI
    PUSH DS
    PUSH ES
    MOV AH, 35H
    MOV AL, 1CH
    INT 21H
    MOV SI, OFFSET KEEP_IP
     SUB SI, OFFSET MY_INT
     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+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
    POP ES
```

```
        POP DS
        STI
        MOV DX, OFFSET RESTORED_STR
          CALL PRINT_MESSAGE
        JMP END_UNLOAD

    NOT_LOAD:
        MOV DX, OFFSET NOT_LOAD_STR
        CALL PRINT_MESSAGE

    END_UNLOAD:
          POP SI
          POP AX
          RET
    UNLOAD ENDP
;-------------------------------------

;-------------------------------------
    MAIN PROC FAR
          MOV AX, DATA
          MOV DS, AX
        CALL CHECK_UPLOAD_KEY
        CMP CL, 0H
        JNE UPLOAD_KEY
        CALL LOAD
        JMP END_MAIN

    UPLOAD_KEY:
        CALL UNLOAD

    END_MAIN:
        XOR AL, AL
        MOV AH, 4CH
        INT 21H
    MAIN ENDP

CODE ENDS
END MAIN
```