**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. 9382 _____ Рыжих Р.В.

Преподаватель _____ Ефремов М.А.

Санкт-Петербург
2021

**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.

**Ход работы**

| Процедура | Описание |
|---|---|
| ROUT | Резидентное прерывание, которое загружается в память и выполняет накопление и вывод числа накопленных прерываний на экран. |
| WRITESTRING | Вывод строки на экран |
| IF_NEED_UNLOAD | Проверка на наличия флага “/un” |
| IF_LOADED | Проверка на загрузку пользовательского прерывания в память |
| LOAD_ROUT | Сохранение первоначального прерывания и загрузка пользовательского прерывания в память |
| UNLOAD_ROUT | Выгрузка пользовательского прерывания из памяти, а также освобождение памяти и восстановление первоначальных прерываний |
| MAIN | Главная функция |

**1)** Был написан программный модуль типа .EXE, который требовался на первом шаге задания. Код см. в приложении А

**2)** Запуск программы и проверка установки резидентного обработчика прерывания 1Ch. Так же по заданию была запущена программа из 3 лабораторной работы для проверки размещения в памяти. (рис.1)

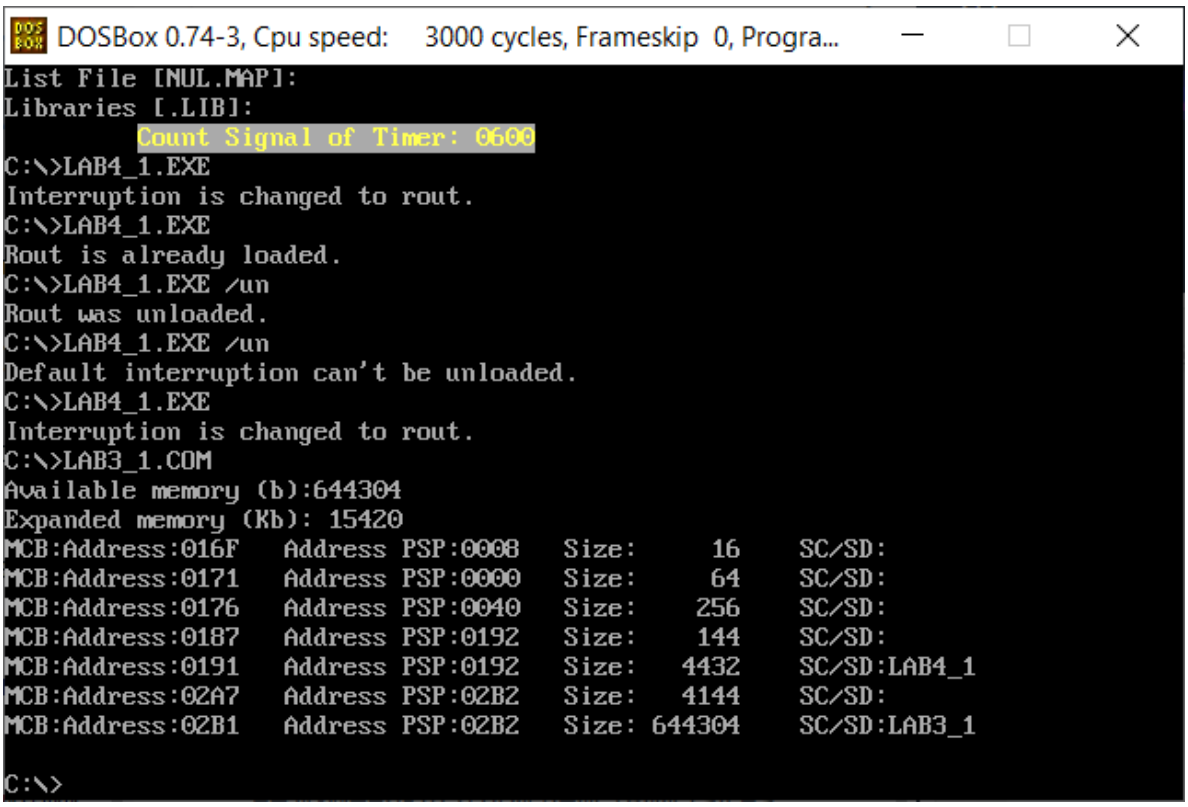


Рис.1 – работа и размещение написанной программы

**3)** Запустим программу с ключом выгрузки "/un", чтобы увидеть что обработчик прерывания выгружен и память освободилась. (рис.2)

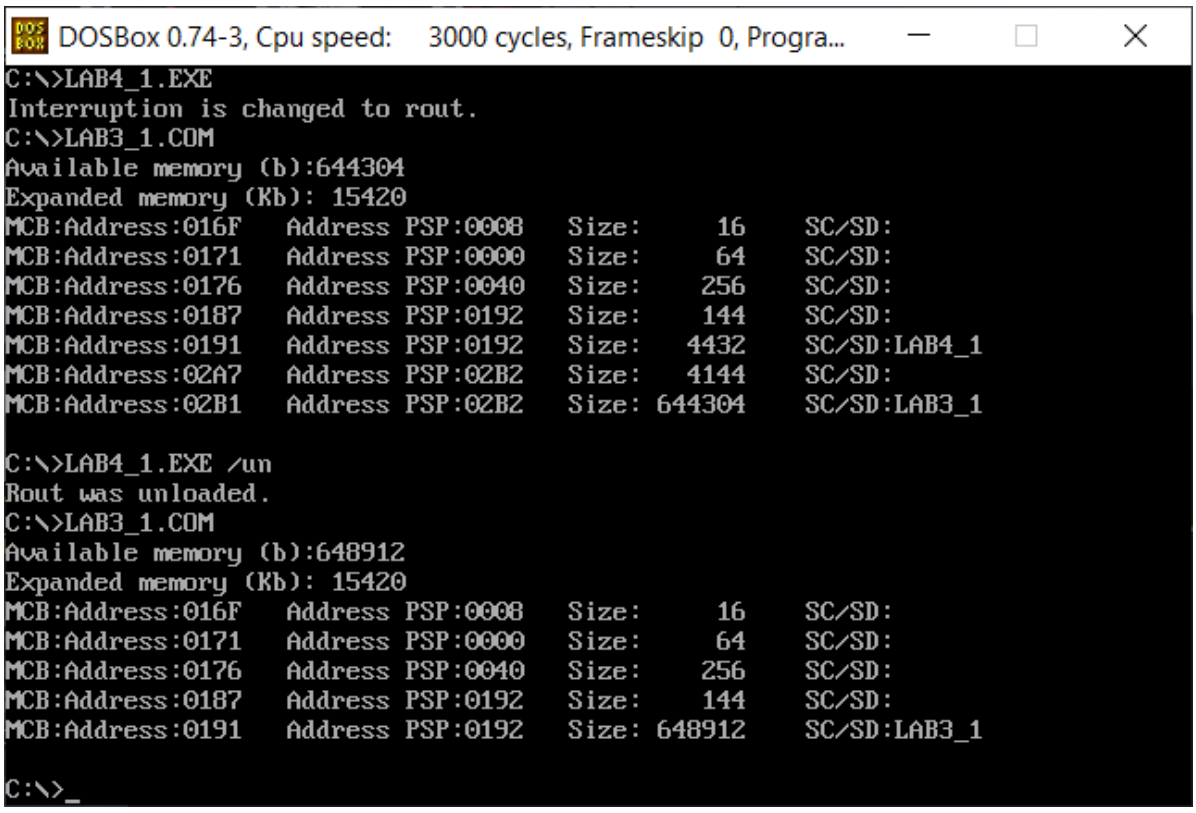


Рис.2 – Запуск с ключом '/un'

**Ответы на контрольные вопросы**

1) Как реализован механизм прерывания от часов?

Принимается сигнал прерывания (приходит примерно каждые 54 мс), запоминаются содержимые регистров, по номеру источника прерывания в таблице векторов определяется смещение, запоминается адрес 2 байта в IP и 2 байта в CS. Дальше выполняется прерывание по сохранённому адресу и далее восстанавливается информация прерванного процесса и управление возвращается прерванной программе.

2) Какого типа прерывания использовались в работе?

Int 10h – видео сервис BIOS

Int 21h – сервисы DOS

Пользовательское прерывание с вектором 1ch int 21h

**Выводы.**

В ходе лабораторной работы была исследована обработка стандартных прерываний, а также построен обработчик прерываний сигналов таймера, которые генерируются аппаратурой через определённые интервалы времени. Программа загружает и выгружает резидент, а также производится проверка флагов и загрузки прерывание в память. С помощью rout выполняет накопление и вывод числа накопленных прерываний на экран.

# ПРИЛОЖЕНИЕ А
# ИСХОДНЫЙ КОД ПРОГРАММЫ

## Lab4.asm:

```
ASTACK    SEGMENT  STACK
          DW 64 DUP(?)
ASTACK    ENDS

DATA  SEGMENT
     ROUT_LOADED DB "ROUT IS ALREADY LOADED.$"
     ROUT_IS_LOADING DB "INTERRUPTION IS CHANGED TO ROUT.$"
     ROUT_IS_NOT_LOADED DB "DEFAULT INTERRUPTION CAN'T BE UNLOADED.$"
     ROUT_IS_UNLOADED DB "ROUT WAS UNLOADED.$"
DATA  ENDS

CODE SEGMENT
     ASSUME CS:CODE, DS:DATA, SS:ASTACK
;---------------------------------------
WRITESTRING PROC NEAR
     PUSH AX
     MOV AH, 9H
     INT 21H
     POP AX
     RET
WRITESTRING ENDP
;---------------------------------------
START_ROUT:
ROUT PROC FAR
     JMP START_PROC
     KEEP_PSP DW 0
     KEEP_IP DW 0
     KEEP_CS DW 0
     KEEP_SS DW 0
     KEEP_SP DW 0
     KEEP_AX DW 0

     ROUT_INDEX DW 00AAH
     TIMER_COUNTER DB 'COUNT SIGNAL OF TIMER: 0000$'
     BSTACK DW 64 DUP(?)
```

```
START_PROC:
     MOV KEEP_SP, SP
     MOV KEEP_AX, AX
     MOV AX, SS
     MOV KEEP_SS, SS

     MOV AX, KEEP_AX

     MOV SP, OFFSET START_PROC

     MOV AX, SEG BSTACK
     MOV SS, AX

     PUSH BX
     PUSH CX
     PUSH DX

     MOV AH,3H
     MOV BH,0H
     INT 10H
     PUSH DX


     PUSH SI
     PUSH CX
     PUSH DS
     PUSH AX
     PUSH BP


     MOV AX, SEG TIMER_COUNTER
     MOV DS,AX
     MOV SI, OFFSET TIMER_COUNTER

     ADD SI, 22
     MOV CX, 4

TIMER_INC:
```

```
      MOV BP, CX
      MOV AH, [SI+BP]
      INC AH
      CMP AH, 3AH
      JL TIMER_INC_END
      MOV AH, 30H
      MOV [SI+BP], AH

      LOOP TIMER_INC

TIMER_INC_END:
      MOV [SI+BP], AH

      POP BP
      POP AX
      POP DS
      POP CX
      POP SI

      PUSH ES
      PUSH BP

      MOV AX, SEG TIMER_COUNTER
      MOV ES,AX
      MOV AX, OFFSET TIMER_COUNTER
      MOV BP,AX
      MOV AH,13H
      MOV AL,00H
      MOV DH,02H
      MOV DL,09H
      MOV CX,27
      MOV BH,0
      INT 10H

      POP BP
      POP ES

      ;RETURN CURSOR
```

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

      POP DX
      POP CX
      POP BX

      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
END_ROUT:
ROUT ENDP
;----------------------------------------
IF_LOADED PROC NEAR
      PUSH AX
      PUSH SI

      PUSH ES
      PUSH DX

      MOV AH,35H
      MOV AL,1CH
      INT 21H

      MOV SI, OFFSET ROUT_INDEX
      SUB SI, OFFSET ROUT
      MOV DX,ES:[BX+SI]
      CMP DX, ROUT_INDEX
      JNE END_IF_LOADED
```

```
      MOV CH,1H

END_IF_LOADED:
      POP DX
      POP ES
      POP SI
      POP AX
      RET
IF_LOADED ENDP
;---------------------------------------
IF_NEED_UNLOAD PROC NEAR
      PUSH AX
      PUSH ES

      MOV AL,ES:[81H+1]
      CMP AL,'/'
      JNE END_IF_NEED_UNLOAD

      MOV AL,ES:[81H+2]
      CMP AL,'U'
      JNE END_IF_NEED_UNLOAD

      MOV AL,ES:[81H+3]
      CMP AL,'N'
      JNE END_IF_NEED_UNLOAD

      MOV CL,1H

END_IF_NEED_UNLOAD:
      POP ES
      POP AX
      RET
IF_NEED_UNLOAD ENDP
;---------------------------------------
UNLOAD_ROUT PROC NEAR
      PUSH AX
      PUSH SI
```

```
      CLI
      PUSH DS
      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]
      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 SI
      POP AX
      RET
UNLOAD_ROUT ENDP
;---------------------------------------
LOAD_ROUT PROC NEAR
      PUSH AX
      PUSH DX
```

```
      MOV KEEP_PSP, ES

      MOV AH,35H
      MOV AL,1CH
      INT 21H
      MOV KEEP_IP, BX
      MOV KEEP_CS, ES

      PUSH DS
      LEA DX, ROUT
      MOV AX, SEG ROUT
      MOV DS,AX
      MOV AH,25H
      MOV AL,1CH
      INT 21H
      POP DS

      LEA DX, END_ROUT
      MOV CL,4H
      SHR DX,CL
      INC DX
      ADD DX,100H
      XOR AX, AX
      MOV AH,31H
      INT 21H

      POP DX
      POP AX
      RET
LOAD_ROUT ENDP
;-----------------------------------------
MAIN PROC FAR
      PUSH  DS
      PUSH  AX
      MOV   AX,DATA
      MOV   DS,AX

      CALL IF_NEED_UNLOAD
```

```
      CMP CL, 1H
      JE NEED_UNLOAD

      CALL IF_LOADED
      CMP CH, 1H
      JE PRINT_ROUT_IS_ALREADY_SET
      MOV DX, OFFSET ROUT_IS_LOADING
      CALL WRITESTRING
      CALL LOAD_ROUT
      JMP EXIT

NEED_UNLOAD:
      CALL IF_LOADED
      CMP CH, 1H
      JNE PRINT_ROUT_CANT_BE_UNLOADED
      CALL UNLOAD_ROUT
      MOV DX, OFFSET ROUT_IS_UNLOADED
      CALL WRITESTRING
      JMP EXIT

PRINT_ROUT_CANT_BE_UNLOADED:
      MOV DX, OFFSET ROUT_IS_NOT_LOADED
      CALL WRITESTRING
      JMP EXIT
PRINT_ROUT_IS_ALREADY_SET:
      MOV DX, OFFSET ROUT_LOADED
      CALL WRITESTRING
      JMP EXIT

EXIT:
      MOV AH, 4CH
      INT 21H
MAIN ENDP
CODE ENDS
END MAIN
```