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 _____ Субботин М.О.

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 выполнения:

Был написан программный модуль типа EXE. Он устанавливал пользовательское прерывание. Во время работы программы на консоль выводилась информация о текущем количестве тиков таймера.

Рисунок 1. Счетчик в программе.

При повторном запуске программы, выводилась информация о том, что резидентный обработчик прерывания уже установлен. Чтобы выгрузить прерывание, использовался параметр “/un” при запуске программы.

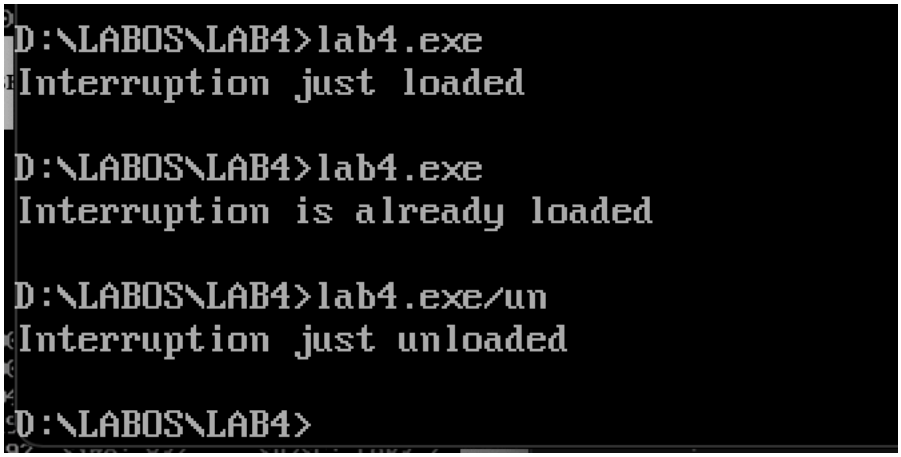

Рисунок 2. Повторный запуск программы и запуск программы с флагом для выгрузки прерывания

Для проверки того, что программа действительно находится в памяти была использована программы из предыдущей лабораторной работы.

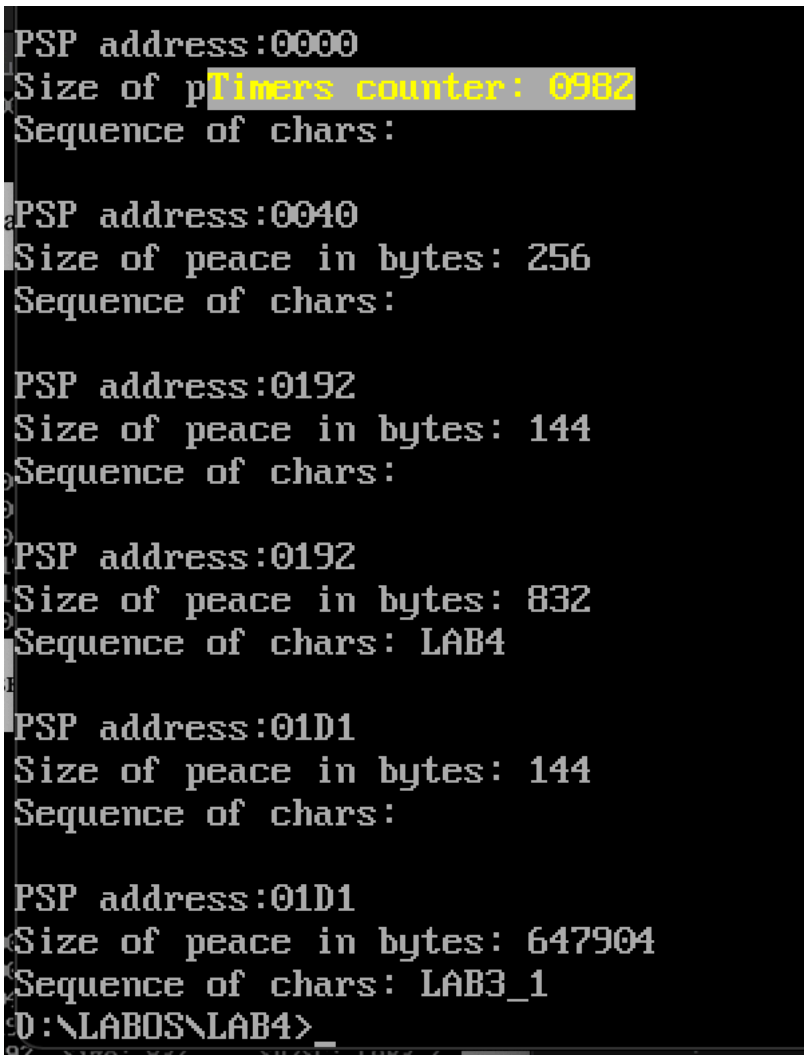

Рисунок 3. Запуск программы “lab3_1.com” при работающей программы “lab4.exe”.

Как видно, обработчик прерывания действительно находится в памяти компьютера.

После выгрузки обработчик прерывания получается следующий результат:

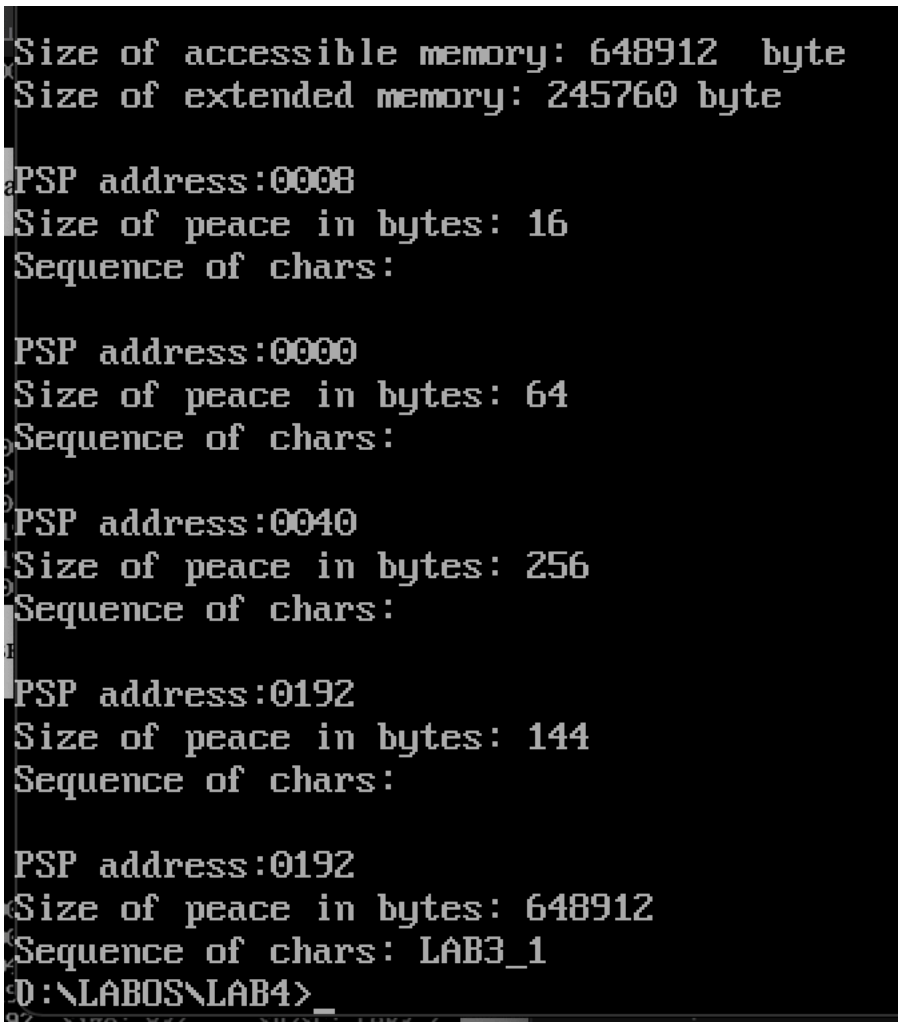

Рисунок 4. Запуск программы “lab3_1.com” после выгрузки обработчика прерывания.

В этот раз в памяти программы “lab4” обнаружено не было.

Ответы на контрольные вопросы.

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

   По сигналу часов, сохраняются состояния регистров (чтобы можно было вернуться обратно в программу). Определяется источник прерывания и из вектора прерывания считываются CS и IP. Затем прерывание обрабатывается и управление возвращается прерванной программе.

2. **Какого типа прерывания использовались в работе?**

   Аппаратное(1ch) и пользовательские(21h,10h).

Выводы.

Была исследована обработка стандартных прерываний, также был построен обработчик прерывания сигналов таймера. Написанная программа производит загрузку и выгрузку резидента.

ПРИЛОЖЕНИЕ А

КОД ПРОГРАММ

Lab4.asm:

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

CUSTOM_INTERRUPTION PROC FAR
        JMP START_CUSTOM_INTERRUPTION

        PSP DW ?
        KEEP_IP DW 0
        KEEP_CS DW 0
        KEEP_SS DW ?
        KEEP_SP DW ?
        KEEP_AX DW ?

        INTERRUPTION_INDEX DW 1234H
        TIMER DB 'TIMERS COUNTER: 0000$'
        INTERRUPTION_STACK DW 200 DUP (?)
        END_STACK DW ?

START_CUSTOM_INTERRUPTION:
        MOV KEEP_SS,SS
        MOV KEEP_SP,SP
        MOV KEEP_AX,AX

        MOV AX,CS
        MOV SS,AX
        MOV SP, OFFSET END_STACK

        PUSH BX
        PUSH CX
        PUSH DX

        ;GETTING CURSOR
        MOV AH,3H
        MOV BH,0H
        INT 10H

        PUSH DX ; CURSOR POSITION (ROW,COLUMN) IN DX
```

```
        ;SETTING CURSOR
        MOV AH,02H
        MOV BH,0H
        MOV DH,02H
        MOV DL,09H
        INT 10H

        PUSH SI
        PUSH CX
        PUSH DS
        PUSH BP

        MOV AX, SEG TIMER
        MOV DS,AX
        MOV SI, OFFSET TIMER
        ADD SI,15

        MOV CX,4

INTERRUPTION_LOOP:
        MOV BP,CX
        MOV AH,[SI+BP]
        INC AH
        MOV [SI+BP],AH
        CMP AH,3AH
        JNE TIMER_WRITER
        MOV AH,30H
        MOV [SI+BP],AH

        LOOP INTERRUPTION_LOOP

TIMER_WRITER:
        POP BP

        POP DS
        POP CX
        POP SI

        ;WRITE TIMER
        PUSH ES
        PUSH BP

        MOV AX, SEG TIMER
        MOV ES,AX
```

```
        MOV AX, OFFSET TIMER
        MOV BP,AX
        MOV AH,13H
        MOV AL,00H
        MOV CX,20
        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 AX, KEEP_SS
        MOV SS,AX
        MOV AX, KEEP_AX
        MOV SP, KEEP_SP

        IRET
INTERRUPTION_ENDED:
CUSTOM_INTERRUPTION ENDP

LOAD_FLAG PROC NEAR
        PUSH AX

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

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

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

```
        JNE LOAD_FLAG_END

        MOV FLAG,1H

LOAD_FLAG_END:
        POP AX
        RET
LOAD_FLAG ENDP


IS_LOADED PROC NEAR
        PUSH AX
        PUSH SI

    ;BY 35H GETTING INTERRUPTION'S ADDRESS
        MOV AH,35H
        ;1CH -- NUMBER OF INTERRUPTION
        MOV AL,1CH
        INT 21H

        MOV SI, OFFSET INTERRUPTION_INDEX
        SUB SI, OFFSET CUSTOM_INTERRUPTION
        MOV DX,ES:[BX+SI]
        CMP DX, INTERRUPTION_INDEX
        JNE IS_LOADED_END
        MOV FLAG_LOAD,1H

IS_LOADED_END:
        POP SI
        POP AX
        RET
IS_LOADED ENDP


WRITE_STRING PROC NEAR
        PUSH AX
        MOV AH,09H
        INT 21H
        POP AX
        RET
WRITE_STRING ENDP


LOAD_INTERRUPTION PROC NEAR
```

```
        PUSH AX
        PUSH DX

        ;CHECKING IF INTERRUPTION IS ALREADY LOADED
        CALL IS_LOADED
        CMP FLAG_LOAD,1H
        JE CUSTOM_ALREADY_LOADED
        JMP STARTING_TO_LOAD

CUSTOM_ALREADY_LOADED:
        LEA DX, INTERRUPTION_ALREADY_LOADED_SEQ
        CALL WRITE_STRING
        JMP END_LOADED

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

        PUSH DS
        LEA DX, CUSTOM_INTERRUPTION
        MOV AX, SEG CUSTOM_INTERRUPTION
        MOV DS,AX
        MOV AH,25H
        MOV AL,1CH
        INT 21H
        POP DS
        LEA DX, INTERRUPTION_JUST_LOADED_SEQ
        CALL WRITE_STRING

        LEA DX, INTERRUPTION_ENDED
        MOV CL,4H
        SHR DX,CL
        INC DX
        MOV AX,CS
        SUB AX, PSP
        ADD DX,AX
        XOR AX,AX
        MOV AH,31H
        INT 21H

END_LOADED:
```

```
        POP DX
        POP AX
        RET
LOAD_INTERRUPTION ENDP


UNLOAD_INTERRUPTION PROC NEAR
        PUSH AX
        PUSH SI

        CALL IS_LOADED
        CMP FLAG_LOAD,1H
        JE START_UNLOAD

        LEA DX, INTERRUPTION_NOT_LOADED_SEQ
        CALL WRITE_STRING
        JMP UNLOAD_END

START_UNLOAD:
        CLI
        PUSH DS
        MOV AH,35H
        MOV AL,1CH
        INT 21H

        MOV SI, OFFSET KEEP_IP
        SUB SI, OFFSET CUSTOM_INTERRUPTION
        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

        LEA DX, INTERRUPTION_UNLOADED_SEQ
        CALL WRITE_STRING
UNLOAD_END:
        POP SI
        POP AX
        RET
UNLOAD_INTERRUPTION ENDP


MAIN     PROC  FAR
        PUSH  DS
        XOR   AX,AX
        PUSH  AX
        MOV   AX,DATA
        MOV   DS,AX

        CALL LOAD_FLAG
        CMP FLAG, 1H
        JE IF_UNLOADED
        CALL LOAD_INTERRUPTION
        JMP THE_END

IF_UNLOADED:
        CALL UNLOAD_INTERRUPTION

THE_END:
        MOV AH,4CH
        INT 21H
MAIN     ENDP
CODE     ENDS

ASTACK    SEGMENT  STACK
  DW 200 DUP(?)
ASTACK    ENDS

DATA     SEGMENT

        FLAG_LOAD DB 0
```

```
        FLAG DB 0

        INTERRUPTION_JUST_LOADED_SEQ  DB 'INTERRUPTION JUST
LOADED', 0AH, 0DH,'$'
        INTERRUPTION_UNLOADED_SEQ  DB 'INTERRUPTION JUST
UNLOADED', 0AH, 0DH,'$'
        INTERRUPTION_NOT_LOADED_SEQ  DB 'INTERRUPTION ISNT
LOADED', 0AH, 0DH,'$'
        INTERRUPTION_ALREADY_LOADED_SEQ  DB 'INTERRUPTION IS
ALREADY LOADED', 0AH, 0DH,'$'

DATA      ENDS
        END MAIN
```