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 _____ Ильин Д.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.

1. Для выполнения лабораторной работы необходимо написать и отладить 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.

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. Длина кода сигнатуры должна быть достаточной, чтобы сделать случайное совпадение маловероятным.

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

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

- 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.

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 ЛР3, которая отображает карту памяти в виде списка блоков МСВ. Полученные результаты поместите в отчет.

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

4. Запустите отлаженную программу с ключом выгрузки и убедитесь, что резидентный обработчик прерывания выгружен, т.е. сообщения на экран не

выводятся, а память, занятая резидентом, освобождена. Для этого также следует запустить программу ЛР3. Полученные результаты поместите в отчет.

5. Ответьте на контрольные вопросы.

Выполнение работы.

1. Был написан .EXE модуль, в котором реализованы следующие процедуры:

- процедура MY_INT – пользовательский обработчик прерывания;

- процедура IS_FLAG – проверка нахождение в командной строке параметра /un (устанавливает в AL 0, если параметр не установлен; 1 – если установлен);

- процедура IS_LOAD – проверяет, не установлен ли пользовательский обработчик прерывания в память (проверяется при помощи сигнатуры, указанной в теле резидента; устанавливает в AL 1, если установлена; 0 – если не установлена).

- процедура MY_INT_LOAD – устанавливает пользовательский обработчик прерывания

- процедура MY_INT_UNLOAD – выгрузка обработчика прерывания.

В процедуре MAIN сначала вызывается процедура IS_ FLAG. Полученное значение сохраняется в регистр BX. Затем вызывается процедура IS_LOAD. Значение в регистре AL сравнивается с 0.

Если равно, то обработчик не установлен. Переходим на метку not_loaded. Сравниваем значение в регистре BL с 0. Если не равно, то параметр /un не был передан – выводим соответствующее сообщение. Если не равно – переходим на метку int_load. Тут вызываем процедуру MY_INT_LOAD и выводим сообщение о загрузке обработчика.

Если не равно, то обработчик уже установлен. Проверяем значение в регистре BL. Если не 0 – то вызываем процедуру MY_INT_UNLOAD и

выводим сообщение о выгрузке обработчика. Если 0 – то выводим сообщение о том, что обработчик уже установлен.

Результат вызова программы представлен на рисунке 1.

Рисунок 1 – результат первого шага

2. Для выполнения данного шага воспользуемся программой lb3_1 из предыдущей лабораторной работы. Результат представлен на рисунке 2. Как можно заметить, обработчик прерывания действительно загружен в память.

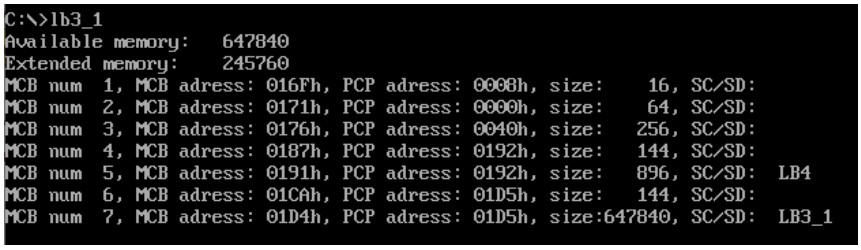

Рисунок 2 – Результаты второго шага

3. При повторном запуске программы действительно выводится сообщение о том, что обработчик уже установлен. Вывод сообщения представлен на рисунке 3.

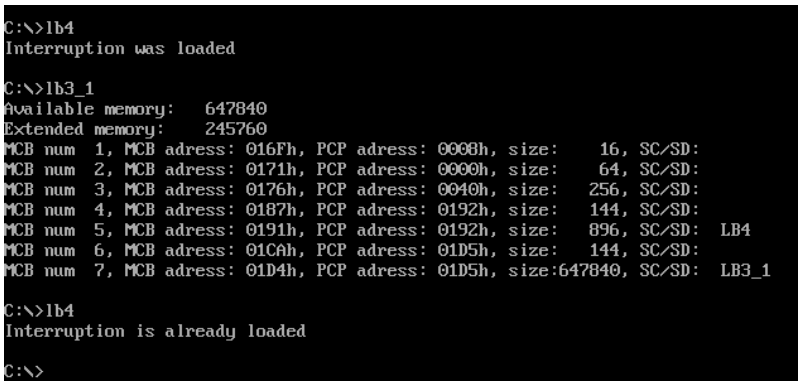

Рисунок 3 – Результаты третьего шага

4. Вызовем программу с параметром /un. Запустим программу прошлой лабораторной работы. Результат представлен на рисунке 4. Как можно заметить, сообщение перестало выводиться, а память освобождена.

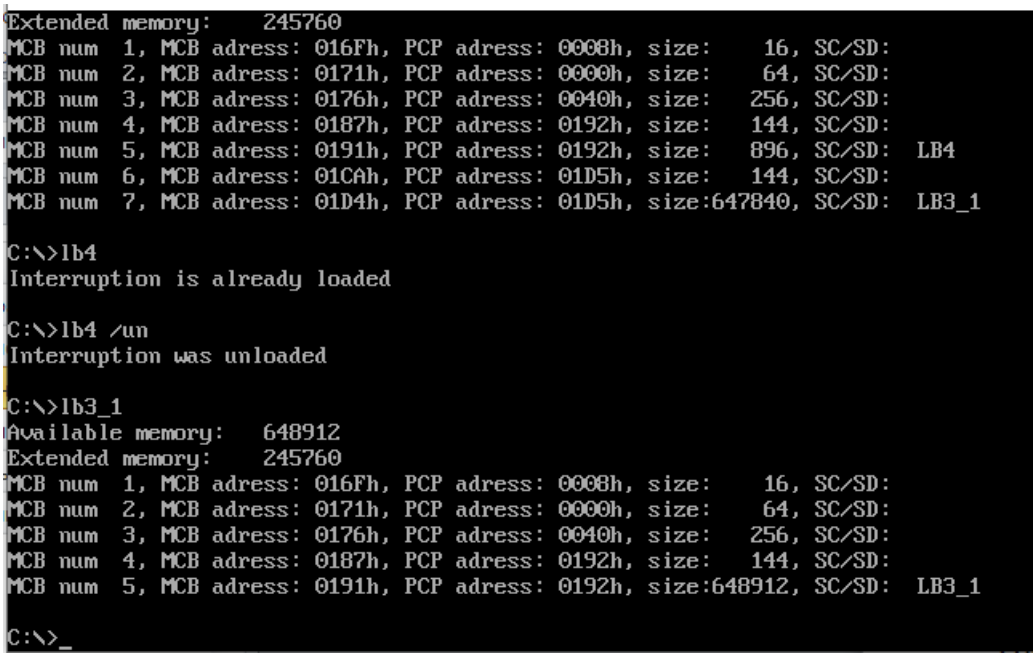

Рисунок 4 – Результаты четвертого шага

Исходный код программы см. в приложении А.

Ответы на вопросы.

1. Как реализован механизм прерывания от часов?

Механизм реализован следующим образом – каждые 55 миллисекунд (примерно 18.2 раза в секунду) вызывается прерывание 1Ch. Можно также заменить обработчик данного прерывания на пользовательский. В таком случае после, каждого вызова прерывания, будет выполняться он.

2. Какого типа прерывания использовались в работе?

В работе использовались аппаратные (1Ch) и программные (10h, 21h) прерывания.

Выводы.

В ходе работы был написан собственный обработчик прерываний сигналов таймера, а также была реализована установка и выгрузка данного обработчика.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: lb4.asm

```
AStack SEGMENT STACK
 DW 128 DUP(?)
AStack ENDS

DATA SEGMENT
     NOT_LOAD db 'Interruption did not load', 0DH, 0AH, '$'
     LOAD db 'Interruption was loaded', 0DH, 0AH, '$'
     UNLOAD db 'Interruption was unloaded', 0DH, 0AH, '$'
     ALREADY_LOAD db 'Interruption is already loaded', 0DH,
0AH, '$'
DATA ENDS

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

;-----------------------------------
; ПРОЦЕДУРЫ
GET_CURS PROC near

     mov AH, 03h
     mov BH, 0
     int 10h

     ret
GET_CURS ENDP
;-----------------------------------
SET_CURS PROC near

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

     ret
SET_CURS ENDP
;-----------------------------------
PRINT PROC near
     push AX
     mov AH, 09h
     int 21h
     pop AX
     ret
PRINT ENDP
;-----------------------------------
MY_INT PROC far
     jmp handle
     counter db 'Interruptions count: 0000$' ;26 или 22
```

```
     PSP dw 0
     KEEP_IP dw 0
     KEEP_CS dw 0
     KEEP_SS dw 0
     KEEP_SP dw 0
     KEEP_AX dw 0
     signature dw 9871h
     IStack db 50 dup(" ")
handle:
     mov KEEP_AX, AX
     mov AX, SS
     mov KEEP_SS, AX
     mov KEEP_SP, SP
     mov AX, seg IStack
     mov SS, AX
     mov SP, offset handle

     push CX
     push DX

     call GET_CURS
     push DX

     mov DH, 0
     mov DL, 0
     call SET_CURS
     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_int:
     mov BP, CX
     mov AH, [SI+BP]
     inc AH
     mov [SI+BP], AH
     cmp AH, 3Ah
     jne print_msg
     mov AH, 30h
     mov [SI+BP], AH
     loop loop_int
print_msg:
     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, 0
     mov CX, 26
     mov BH, 0
     int 10h

     pop BP
     pop ES

     pop DX
     call SET_CURS

     pop DX
     pop CX

     mov SP, KEEP_SP
     mov AX, KEEP_SS
     mov SS, AX
     mov AX, KEEP_AX
     mov AL, 20h
     out 20h, AL
     iret
end_int:
MY_INT ENDP
;---------------------------------
MY_INT_LOAD PROC near
     mov PSP, ES
     mov AH, 35h
     mov AL, 1Ch
     int 21h
     mov KEEP_IP, BX
     mov KEEP_CS, ES

     push DS
     mov DX, offset MY_INT
     mov AX, seg MY_INT
     mov DS, AX
     mov AH, 25h
     mov AL, 1Ch
     int 21h
     pop DS
```

```
     mov DX, offset end_int
     mov CL, 4
     shr DX, CL
     inc DX
     mov AX, CS
     sub AX, PSP
     add DX, AX
     mov AL, 0
     mov AH, 31h
     int 21h
     ret
MY_INT_LOAD ENDP
;-----------------------------------
MY_INT_UNLOAD PROC near
     CLI
     push DS
     mov AX, ES:[KEEP_CS]
     mov DS, AX
     mov DX, ES:[KEEP_IP]
     mov AH, 25h
     mov AL, 1Ch
     int 21h
     pop DS
     STI

     mov AX, ES:[PSP]
     mov ES, AX
     push ES
     mov AX, ES:[2Ch]
     mov ES, AX
     mov AH, 49h
     int 21h
     pop ES
     int 21h
     ret
MY_INT_UNLOAD ENDp
;-----------------------------------
IS_LOADED PROC near
     push BX
     push ES
     mov AH, 35h
     mov AL, 1Ch
     int 21h

     mov AX, ES:[signature]
     cmp AX, 9871h
     je loaded
     mov AL, 0h
     jmp end_isloaded
loaded:
```

```
     mov AL, 01h
end_isloaded:
     pop ES
     pop BX
     ret
IS_LOADED ENDP
;-----------------------------------
IS_FLAG PROC near
     push BP
     mov BP, 0082h

     mov AL, ES:[BP]
     cmp AL, '/'
     jne not_good

     mov AL, ES:[BP+1]
     cmp AL, 'u'
     jne not_good

     mov AL, ES:[BP+2]
     cmp AL, 'n'
     jne not_good

     mov AL, 01h
     jmp good
not_good:
     mov AL, 0h
good:
     pop BP
     ret
IS_FLAG endp
;-----------------------------------
MAIN PROC far
     mov ax, data
     mov ds, ax

     call IS_FLAG
     mov BX, AX

     call IS_LOADED
     cmp AL, 0h
     je not_loaded
     cmp BL, 0h
     jne int_unload
     mov DX, offset ALREADY_LOAD
     call PRINT
     jmp end_main

not_loaded:
     cmp BL, 0h
     je int_load
```

```
     mov DX, offset NOT_LOAD
     call PRINT
     jmp end_main
int_load:
     mov DX, offset LOAD
     call PRINT
     call MY_INT_LOAD
     jmp end_main
int_unload:
     mov AH, 35h
     mov AL, 1Ch
     int 21h
     mov DX, offset UNLOAD
     call PRINT
     call MY_INT_UNLOAD

end_main:
     xor AL, AL
     mov AH, 4Ch
     int 21h
MAIN ENDP
TESTPC ENDS
END MAIN
```