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. 9381 _____ Гурин С.Н.

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.

Порядок выполнения работы.

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.

Выполнение работы.

Был реализована программа, которая меняла вектор прерывания у таймера на написанное. В ходе работы выводится информация о количестве сигналов таймера с момента запуска программы. Программа оставалась резидентной в памяти, при попытке повторной загрузки в память выводилось сообщение о том, что прерывание уже установлено (для проверки загрузки сравнивались сигнатура у обработчика прерывания пользовательской программы и той, что загружена в память).

```
Count of  interruptions: 0254AB_1\SRC>

C:\USERS\SIMON\DESKTOP\AEE\LAB_1\SRC>

C:\USERS\SIMON\DESKTOP\AEE\LAB_1\SRC>

C:\USERS\SIMON\DESKTOP\AEE\LAB_1\SRC>

C:\USERS\SIMON\DESKTOP\AEE\LAB_1\SRC>

C:\USERS\SIMON\DESKTOP\AEE\LAB_1\SRC>

C:\USERS\SIMON\DESKTOP\AEE\LAB_1\SRC>

C:\USERS\SIMON\DESKTOP\AEE\LAB_1\SRC>

C:\USERS\SIMON\DESKTOP\AEE\LAB_1\SRC>

C:\USERS\SIMON\DESKTOP\AEE\LAB_1\SRC>LAB4.EXE
interrupt was loaded

C:\USERS\SIMON\DESKTOP\AEE\LAB_1\SRC>LAB4.EXE
interrupt is already load

C:\USERS\SIMON\DESKTOP\AEE\LAB_1\SRC>
```

Рис. 1

Для проверки того, что программа находится в памяти, используем программу, которая выводит цепочку mcb-блоков.

```
C:\USERS\SIMON\DESKTOP\AEE\LAB_1\SRC>LAB3_1.COM
available memory size: 647968 bytes
extended memory size: 246720 bytes
MCB #1: address: 016F PSP address: 0008 size: 16     SC/SD:
MCB #2: address: 0171 PSP address: 0000 size: 64     SC/SD:
MCB #3: address: 0176 PSP address: 0040 size: 256    SC/SD:
MCB #4: address: 0187 PSP address: 0192 size: 144    SC/SD:
MCB #5: address: 0191 PSP address: 0192 size: 768    SC/SD: LAB4
MCB #6: address: 01C2 PSP address: 01CD size: 144    SC/SD:
MCB #7: address: 01CC PSP address: 01CD size: 647968 SC/SD: LAB3_1
```

Рис. 2

Теперь выгрузим обработчик из памяти и снова запустим программу, которая выводит цепочку mcb-блоков.

```
C:\USERS\SIMON\DESKTOP\AEE\LAB_1\SRC>LAB4.EXE /un
interrupt unloaded

C:\USERS\SIMON\DESKTOP\AEE\LAB_1\SRC>LAB4.EXE /un
interrupt not loaded
```

Рис. 3

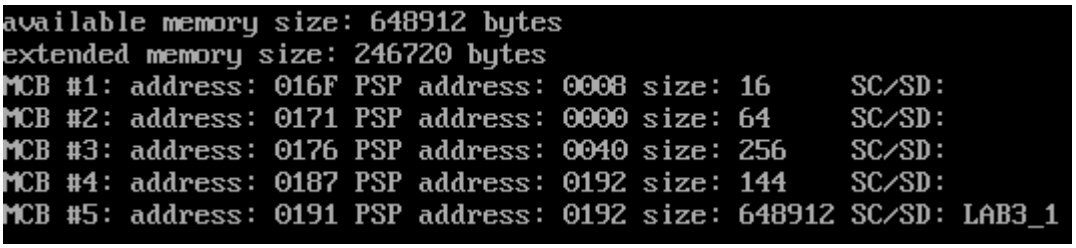

Рис. 4

Так можно убедиться, что обработчик прерываний выгружен из памяти.

Ответы на контрольные вопросы

1) Как реализован механизм прерывания от часов?

Когда происходит сигнал часов, то сохраняется состояние регистров для того, чтобы вернуться в текущую программу, затем определяется источник прерывания, из вектора прерывания считываются CS и IP обработчика прерывания, прерывание обрабатывается, затем управление возвращается прерванной программе.

2) Какого типа прерывания использовались в работе?

Аппаратные: 1Ch; программные: 10h, 21h

Вывод

Был изучен механизм обработки прерываний в DOS, написана программа, которая заменяет текущий обработчик прерываний от таймера на пользовательский.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: lab4.asm

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

     ASSUME     CS:CODE, DS:DATA, SS:SSTACK

CODE SEGMENT

INT_COUNT_FUNC  PROC FAR
                     JMP  RUN

KEEP_CS    DW        0
KEEP_IP    DW        0
NOWPSP          DW        0
MEMADRPSP  DW        0
COUNT_INT  DW        0FEDCH
KEEP_SS    DW        0
KEEP_SP    DW        0
KEEP_AX    DW        0
COUNT_MES  DB        'COUNT OF INTERRUPTIONS: 0000 $'
NEWSTACK   DW        64 DUP(?)
RUN:

                     MOV  KEEP_SP, SP
                     MOV  KEEP_AX, AX
                     MOV  KEEP_SS, SS
                     MOV  SP, OFFSET RUN
                     MOV  AX, SEG NEWSTACK
                     MOV  SS, AX

                     PUSH AX
                     PUSH BX
                     PUSH CX
                     PUSH DX
```

```
            MOV  AH, 3H
            MOV  BH, 0H
            INT  10H

            PUSH DX

            MOV  AH, 2H
            MOV  BH, 0H
            MOV  BL, 2H
            MOV  DX, 0H
            INT  10H

            PUSH SI
            PUSH CX
            PUSH DS

            MOV  AX, SEG COUNT_MES
            MOV  DS, AX
            MOV  SI, OFFSET COUNT_MES
            ADD  SI, 27
            MOV  CX, 4

LOOP_M:
            MOV  AH,[SI]
            INC  AH
            MOV  [SI], AH
            CMP  AH, 3AH
            JNE  END_INTERRUPT
            MOV  AH, 30H
            MOV  [SI], AH
            DEC  SI
       LOOP LOOP_M

END_INTERRUPT:
            POP  DS
            POP  CX
            POP  SI
            PUSH ES
```

```
                    PUSH BP

                    MOV  AX, SEG COUNT_MES
                    MOV  ES, AX
                    MOV  AX, OFFSET COUNT_MES
                    MOV  BP, AX
                    MOV  AH, 13H
                    MOV  AL, 00H
                    MOV  CX, 28
                    MOV  BH, 0
                    INT  10H

                    POP  BP
                    POP  ES

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

                    POP  DX
                    POP  CX
                    POP  BX
                    POP  AX

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

                    IRET
INT_COUNT_FUNC  ENDP

ISBOOTFUNC          PROC NEAR
                    PUSH BX
                    PUSH DX
                    PUSH ES

                    MOV  AH, 35H
```

```
                    MOV  AL, 1CH
                    INT  21H

                    MOV  DX, ES:[BX + 11]
                    CMP  DX, 0FEDCH
                    JE        INT_IS_SET
                    MOV  AL, 00H
                    JMP  END_CHECK_BOOT

          INT_IS_SET:
                    MOV  AL, 01H
                    JMP  END_CHECK_BOOT

          END_CHECK_BOOT:
                    POP  ES
                    POP       DX
                    POP  BX

                    RET
ISBOOTFUNC          ENDP

          SSIZE:

CHECK_UNBOOT   PROC NEAR
                    PUSH ES

                    MOV  AX, NOWPSP
                    MOV  ES, AX

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

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

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

```
                    CMP  AL, 'N'
                    JNE  END_CHECK
                    MOV  AL, 1H
          END_CHECK:
                    POP  ES

                    RET
CHECK_UNBOOT   ENDP

LOADFUNC       PROC NEAR
                    PUSH AX
                    PUSH BX
                    PUSH DX
                    PUSH ES

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

                    PUSH DS

                    MOV  DX, OFFSET INT_COUNT_FUNC
                    MOV  AX, SEG INT_COUNT_FUNC
                    MOV  DS, AX
                    MOV  AH, 25H
                    MOV  AL, 1CH
                    INT  21H

                    POP  DS

                    MOV  DX, OFFSET STR_LOAD

                    PUSH AX
                    MOV  AH, 09H
                    INT       21H
                    POP  AX
```

```
                    POP  ES
                    POP  DX
                    POP  BX
                    POP  AX

                    RET
LOADFUNC        ENDP

UNBOOTFUNC          PROC NEAR
                    PUSH AX
                    PUSH BX
                    PUSH DX
                    PUSH ES

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

                    PUSH DS

                    MOV  DX, ES:[BX + 5]
                    MOV  AX, ES:[BX + 3]
                    MOV  DS, AX
                    MOV  AH, 25H
                    MOV  AL, 1CH
                    INT  21H

                    POP  DS

                    STI
                    MOV  DX, OFFSET STR_UNLOAD

                    PUSH AX
                    MOV  AH, 09H
                    INT       21H
                    POP  AX
```

```
                    PUSH ES

                    MOV  CX, ES:[BX + 7] ;NOWPSP
                    MOV  ES, CX
                    MOV  AH, 49H
                    INT  21H

                    POP  ES

                    MOV  CX, ES:[BX + 9] ;MEMADRPSP
                    MOV  ES, CX
                    INT  21H

                    POP  ES
                    POP  DX
                    POP  BX
                    POP  AX

                    RET
UNBOOTFUNC          ENDP

MAIN          PROC FAR
                    MOV  BX, 02CH
                    MOV  AX, [BX]
                    MOV  MEMADRPSP, AX
                    MOV  NOWPSP, DS
                    XOR  AX, AX
                    XOR  BX, BX

                    MOV  AX, DATA
                    MOV  DS, AX

                    CALL CHECK_UNBOOT
                    CMP  AL, 01H
                    JE        UNLOAD_MARK

                    CALL ISBOOTFUNC
                    CMP  AL, 01H
```

```
            JNE   INTERRUPTION_IS_NOT_LOADED

            MOV   DX, OFFSET STR_ALR_LOADED

            PUSH  AX
            MOV   AH, 09H
            INT         21H
            POP   AX

            JMP   EEND

            MOV   AH,4CH
            INT   21H

INTERRUPTION_IS_NOT_LOADED:
            CALL  LOADFUNC

            MOV   DX, OFFSET SSIZE
            MOV   CL, 04H
            SHR   DX, CL
            ADD   DX, 1BH
            MOV   AX, 3100H
            INT   21H

UNLOAD_MARK:
            CALL  ISBOOTFUNC
            CMP   AL, 00H
            JE          NOT_SET
            CALL  UNBOOTFUNC
            JMP   EEND

NOT_SET:
            MOV   DX, OFFSET STR_NOT_LOADED

            PUSH  AX
            MOV   AH, 09H
            INT         21H
            POP   AX
```

```
                    JMP   EEND

          EEND:
                    MOV   AH, 4CH
                    INT   21H
MAIN            ENDP

CODE            ENDS

DATA            SEGMENT
STR_NOT_LOADED  DB        "INTERRUPT NOT LOADED", 0DH, 0AH, '$'
STR_UNLOAD           DB        "INTERRUPT UNLOADED", 0DH, 0AH, '$'
STR_ALR_LOADED  DB        "INTERRUPT IS ALREADY LOAD", 0DH, 0AH, '$'
STR_LOAD        DB        "INTERRUPT WAS LOADED", 0DH, 0AH, '$'
DATA            ENDS

END             MAIN
```