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 _____ Кривенцова Л.С.

Преподаватель _____ Губкин А.Ф.

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.

Выполнение работы.

Исходный код содержится в файле lb4.asm.

Написан корректный 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 выводит соответствующее сообщение и осуществляет выход по функции 4Ch прерывания int 21h.

4) Выгружает 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.

- Функции *setCurs* и *getCurs* отвечают за установку и возврат положения курсора соответственно.
- Функция *ROUTE* является обработчиком прерывания. В ней вычисляется число вызовов от таймера и выводится в терминал.
- Функция *Control* проверяет загрузку пользовательского прерывания.
- Функция *print* печатает информацию в терминал.
- *Int_Setting* устанавливает новый обработчик прерывания, запоминая данные для восстановления предыдущего.

Далее осуществлялась загрузка программы

Воспользуемся программой lb3_1 из предыдущей лабораторной работы.

```
D:\>lb3.com
Amount of available memory : 648912 bytes;
Extended memory size : 15360 kbytes;
MCB Type: 4D, MCB Adress: 016F, Owner: 0008h, Size:     16 b, Name:
MCB Type: 4D, MCB Adress: 0171, Owner: 0000h, Size:     64 b, Name: DPMILOA
MCB Type: 4D, MCB Adress: 0176, Owner: 0040h, Size:    256 b, Name:
MCB Type: 4D, MCB Adress: 0187, Owner: 0192h, Size:    144 b, Name:
MCB Type: 5A, MCB Adress: 0191, Owner: 0192h, Size: 648912 b, Name: LB3
D:\>S
```

Рис.1. - Результат запуска lb3_part1.com

```
D:\>lb4.exe
User interrupt set
 quantity of calls: 00026
```

Рис.2. - Результат запуска lb4.exe

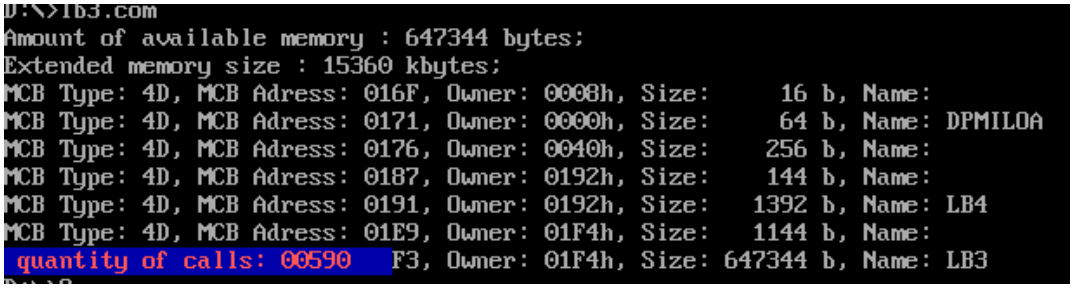

Рис.3. - Результат повторного запуска lb3_part1.com после программы с обработкой пользовательского прерывания

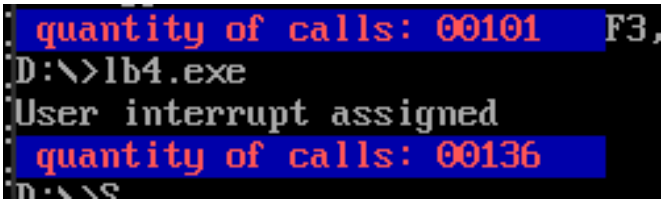

Рис.4. - Результат повторного запуска lb4.exe

Рис.5. - Результат запуска lb4.exe с ключом выгрузки

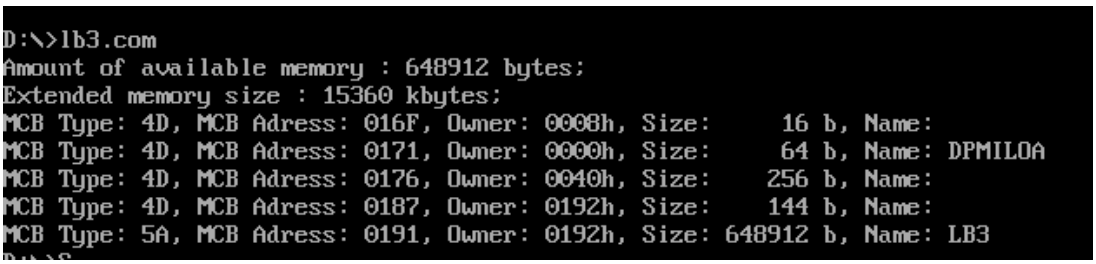

Рис.6. - Результат запуска lb3_part1.com после выгрузки прерывания

Таким образом, по результатам вывода программы lb3_part1.com после загрузки программы с обработкой пользовательского прерывания очевидно, что в памяти зафиксированы данные обработчика прерывания lb3.exe, т.е. оно было успешно загружено в память.

После запуска программы lb4.exe с ключом выгрузки результат запуска lb3_part1.com означает, что память, где хранилось пользовательское прерывание удалена, т.е. пользовательское прерывание было выгружено.

Контрольные вопросы.

1. Как реализован механизм прерывания от часов?

С определенным интервалом (тиком аппаратных часов) вызывается прерывание 1Ch. Фиксируется содержимое регистров и CS:IP для дальнейшего возврата. Затем управление передаётся функции, точка входа которой записана в соответствующем векторе прерывания. После обработки прерывания управление возвращается к прерванной ранее текущей программе.

2. Какого типа прерывания использовались в работе?

Аппаратные (1Ch) и программные (10h, 21h).

Выводы.

Был построен обработчик прерываний сигналов таймера. Реализована установка и выгрузка данного обработчика.

ПРИЛОЖЕНИЕ А.

Исходный код.

Lb4.asm:

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

DATA SEGMENT
SET db 'User interrupt set' , 0DH, 0AH, '$'
ASSIGN   db 'User interrupt assigned', 0DH, 0AH, '$'
DIS db 'User interrupt discharged' , 0DH, 0AH, '$'
DATA ENDS

CODE SEGMENT
    ASSUME CS:CODE, DS:DATA, ES:DATA, SS:AStack

print PROC
     push AX
     mov AH, 09h
     int 21h
     pop AX
     ret
print ENDP

setCurs PROC
     push AX
     push BX
     push DX
     push CX
     mov AH,02h
     mov BH,0
     int 10h
     pop CX
     pop DX
     pop BX
     pop AX
     ret
```

```
setCurs ENDP

; 03H читать позицию и размер курсора
;  вход: BH = видео страница
; выход: DH,DL = текущие строка, колонка курсора
;        CH,CL = текущие начальная, конечная строки курсора
getCurs PROC
     push AX
     push BX
     push CX
     mov AH,03h
     mov BH,0
     int 10h
     pop CX
     pop BX
     pop AX
     ret
getCurs ENDP

ROUT PROC FAR
    jmp _ROUT

     SIGN db '0000'
     KEEP_CS dw 0    ; для хранения сегмента
     KEEP_IP dw 0    ; и смещения прерывания
     KEEP_PSP dw 0
     VAL db 0
     _STACK     dw 100 dup (0)
     KEEP_SS dw 0
     KEEP_AX    dw 0
     KEEP_SP dw 0
     NUMBER db ' quantity of calls: 00000    ','$'

_ROUT:
     mov KEEP_SS, SS
```

```
      mov KEEP_AX, AX
      mov KEEP_SP, SP
      mov AX, seg _STACK
      mov SS, AX
      mov SP, 0
      mov AX, KEEP_AX
      push AX
      push DX
      push DS
      push ES
      cmp VAL, 1
      je RES
      call getCurs
      push DX
      mov DH, 23
      mov DL, 0
      call setCurs
ROUT_SUM:
      push SI
      push CX
      push DS
      push AX
      mov AX, seg NUMBER
      mov DS, AX
      mov BX, offset NUMBER
      add BX, 21
      mov SI, 3
loop_for_route:
      mov AH,[BX+SI]
      add AH, 1
      cmp AH, 58
      jne ROUT_NEXT
      mov AH, 48
      mov [BX+SI], AH
      sub SI, 1
      cmp SI, 0
      jne loop_for_route
ROUT_NEXT:
      mov [BX+SI], AH
```

```
        pop DS
        pop SI
        pop BX
        pop AX
        push ES
        push BP
        mov AX, seg NUMBER
        mov ES, AX
        mov AX, offset NUMBER
        mov BP, AX
        mov AH, 13h
        mov AL, 1
        mov CX, 28
        mov BH, 0
        int 10h
        pop BP
        pop ES
        pop DX
        call setCurs
        jmp The_end
RES:
        cli
        mov DX, KEEP_IP
        mov AX, KEEP_CS
        mov DS, AX
        mov AH, 25h
        mov AL, 1CH
        int 21H
        mov ES, KEEP_PSP
        mov ES, ES:[2Ch]
        mov AH, 49h
        int 21h
        mov ES, KEEP_PSP
        mov AH, 49h
        int 21h
        sti
The_end:
        pop ES
        pop DS
```

```
      pop DX
      pop AX
      mov AX, KEEP_SS
      mov SS, AX
      mov SP, KEEP_SP
      mov AX, KEEP_AX

      mov AL, 20H
      out 20H,AL

      iret
ROUT ENDP

Control PROC
      mov AH, 35h
      mov AL, 1Ch
      int 21h
      mov SI, offset SIGN
      sub SI, offset ROUT
      mov AX,'00'
      cmp AX, ES:[BX+SI]
      jne Upload
      cmp AX, ES:[BX+SI+2]
      je Download
Upload:
      call Int_Setting
      mov DX, offset Size_in_bytes
      mov CL, 4
      shr DX, CL
      inc DX
      add DX, CODE
      sub DX, KEEP_PSP
      xor AL, AL
      mov AH, 31h
      int 21h
Download:
      push ES
      push AX
      mov AX, KEEP_PSP
```

```
      mov ES, AX
      cmp byte ptr ES:[82h],'/'
      jne stay
      cmp byte ptr ES:[83h],'u'
      jne stay
      cmp byte ptr ES:[84h],'n'
      je _Upload
stay:
      pop AX
      pop ES
      mov DX, offset ASSIGN
      call print
      ret
_Upload:
      pop AX
      pop ES
      mov byte ptr ES:[BX+SI+10], 1
      mov DX, offset DIS
      call print
      ret
Control ENDP

Int_Setting PROC
      push DX
      push DS
      mov AH, 35h
      mov AL, 1Ch
      int 21h
      mov KEEP_IP, BX
      mov KEEP_CS, ES
      mov dx, offset ROUT
      mov ax, seg ROUT
      mov DS, AX
      mov AH, 25h
      mov AL, 1Ch
      int 21h
      pop DS
      mov DX, offset SET
      call PRINT
```

```
      pop DX
      ret
Int_Setting ENDP

Main PROC FAR
      mov AX, DATA
      mov DS, AX
      mov KEEP_PSP, ES
      call Control
      xor AL, AL
      mov AH, 4Ch
      int 21h
Main ENDP

      Size_in_bytes:
CODE ENDS
    END Main
```