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

Студентка гр. 0382 _____ Деткова А.С.

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.

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.

Функции, используемые в программе:

1. *_print* — напечатать строку, адрес смещения до которой лежит в регистре DX.
2. *my_interuption* - собственный обработчик прерывания.
3. *load_int* — загрузка прерывания в память.
4. *unload_int* — выгрузка прерывания из памяти.
5. *find_cmd_key* — считать ключ командной строки, если есть — флаг = 1.
6. *is_loaded_int* — проверка, загружено ли прерывание в память.
7. *MAIN* — вызывающая функция.

Шаг 1.

На первом шаге был написан .EXE модуль, который выполняет все требования

Шаг 2.

На втором шаге был запущен модуль .EXE, а также была запущена программа ЛР3, которая показала, что прерывание действительно загружено в память. Результаты работы программ см. Рис. 1, 2 и 3.

Рисунок 1: Работа счетчика.

Рисунок 2: Прерывание загружено.

```
Turbo Link  Version 5.1 Copyright (c) 1992 Borland International

C:\>lab3_1.com
Size of available memory = 647648.
Size of extended memory = 15728640.
MCB:  1; adress: 016FH; PSP: 0008H; size in bytes:     16; SC\SD:
MCB:  2; adress: 0171H; PSP: 0000H; size in bytes:     64; SC\SD: DPMILOAD
MCB:  3; adress: 0176H; PSP: 0040H; size in bytes:    256; SC\SD:
MCB:  4; adress: 0187H; PSP: 0192H; size in bytes:    144; SC\SD:
MCB:  5; adress: 0191H; PSP: 0192H; size in bytes:   1088; SC\SD: LAB4
MCB:  6; adress: 01D6H; PSP: 01E1H; size in bytes:   1144; SC\SD:
MCB:  7; adress: 01E0H; PSP: 01E1H; size in bytes: 647648; SC\SD: LAB3_1
```

Рисунок 3: Результат запуска модуля из ЛР3.

После запуска модуля из ЛР3 видно, что прерывание действительно занимает память (строка MCB №5).

Шаг 3.

Программа была запущена еще раз. Действительно, программа определяет установленный обработчик прерываний. Результат см. на рис. 4.

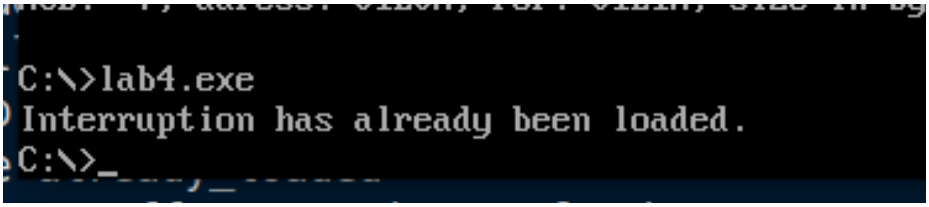

Рисунок 4: Результат повторного запуска модуля.

Шаг 4.

На четвертом шаге программа запущена с ключом выгрузки. Результат см. на рис. 5. А также была вновь запущена программа, выводящая ячейки памяти. Результат см. на рис. 6. Также программа была запущена еще раз, и обработчик прерываний вновь был загружен в память. См. Рис. 7.

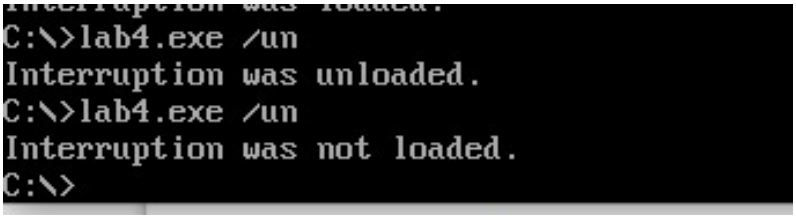

Рисунок 5: Прерывание выгружено из памяти.

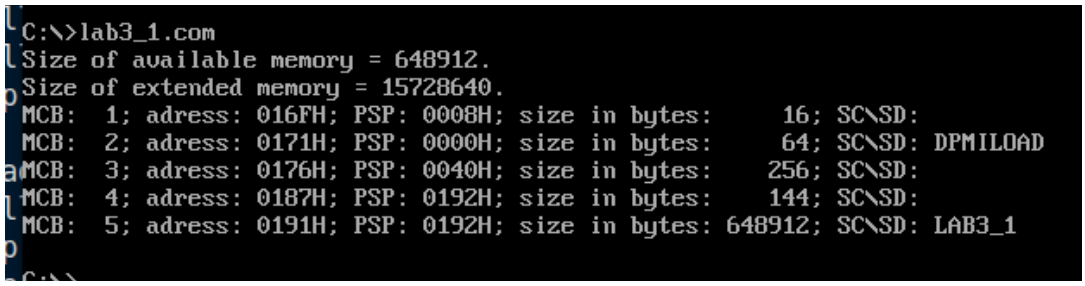

Рисунок 6: После выгрузки прерывания из памяти, модуля в памяти больше нет.

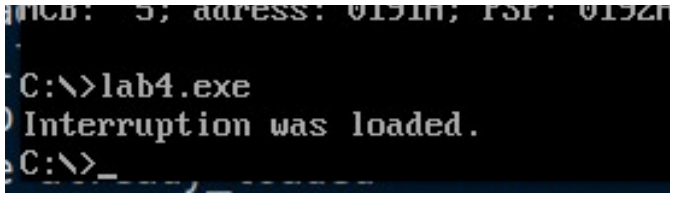

Рисунок 7: Повторная загрузка в память.

Ответы на контрольные вопросы.

1. Как реализован механизм прерывания от часов?

Прерывание от часов генерируется примерно 18 раз в секунду, вызывается с помощью аппаратно-генерируемого прерывания 1CH — системного таймера. При замене вектора прерывания на написанный нами обработчик прерываний при генерации прерывания будет вызываться наша фукция.

2. Какого типа прерывания использовались в работе?

1CH — аппаратное прерывание. 10H и 21H — программные прерывания.

Выводы.

В ходе работы были изучены структуры обработчиков стандартных прерываний, был реализован обработчик прерываний сигналов таймера.

ПРИЛОЖЕНИЕ А

КОД МОДУЛЕЙ

Название файла: lab4.asm

```
AStack SEGMENT  STACK
          DB 100H DUP('!')
AStack ENDS


DATA SEGMENT

    flag DB 0
    msg_is_load DB 'Interruption was loaded.$'
    msg_is_in_memory DB 'Interruption has already been loaded.$'
    msg_is_unload DB 'Interruption was unloaded.$'
    msg_is_not_load DB 'Interruption was not loaded.$'

DATA ENDS


CODE SEGMENT

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


my_interuption  PROC FAR

    jmp handle

    PSP DW ?
    keep_CS DW 0
    keep_IP DW 0
    keep_SS DW 0
    keep_SP DW 0
    keep_AX DW 0

    int_indicator DW 0AAAAH
    counter DB 'Counter:    0000'
    counter_len = $ - counter
    int_stack DB 100H dup (?)
   end_stack:

  handle:
    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
  push DS
  push ES
  push SI
  push DI
  push BP

  mov AH,03H
  mov BH,0
  int 10H
  push DX

  mov AH,02H
  mov BH,0
  mov DX,0
  int 10H

  push BP
  push DS
  push SI
  mov DX,seg counter
  mov DS,DX
  mov SI,offset counter
  mov CX,5

lp:
  mov BP,CX
  dec BP
  mov AL,[SI+BP+11]
  inc AL
  mov [SI+BP+11],AL
  cmp AL,3AH
  jne end_int
  mov AL,30H
  mov [SI+BP+11],AL
  loop lp

end_int:
  pop SI
  pop DS

  push ES
  mov DX,seg counter
  mov ES,DX
  mov BP,offset counter
  mov AH,13H
  mov AL,1
  mov BH,0
  mov CX,counter_len
  mov DX,0
  int 10H
  pop ES
  pop BP

  mov AH,02H
```

```
    mov BH,0
    pop DX
    int 10H

    pop BP
    pop DI
    pop SI
    pop ES
    pop DS
    pop DX
    pop CX
    pop BX

    mov AX,keep_SS
    mov SS,AX
    mov SP,keep_SP
    mov AX,keep_AX
    mov AL,20H
    out 20H,AL
    iret

  int_end:

my_interuption  ENDP


load_int PROC NEAR

    push AX
    push BX
    push CX
    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 my_interuption
    mov AX,seg my_interuption
    mov DS,AX
    mov AH,25H
    mov AL,1CH
    int 21H
    pop DS

    mov DX,offset int_end
    mov CL,4
    shr DX,CL
    inc DX
    mov AX,CS
    sub AX,PSP
```

```
    add DX,AX
    xor AX,AX
    mov AH,31H
    int 21H

    pop ES
    pop DX
    pop CX
    pop BX
    pop AX
    ret

load_int ENDP


unload_int PROC NEAR

    push AX
    push BX
    push CX
    push DX
    push ES
    push DS

    cli
    mov AH,35H
    mov AL,1CH
    int 21H
    mov DX,ES:[offset keep_IP]
    mov AX,ES:[offset keep_CS]
    mov DS,AX
    mov AH,25H
    mov AL,1CH
    int 21H

    mov AX,ES:[offset PSP]
    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 DS
    pop ES
    pop DX
    pop CX
    pop BX
    pop AX
    ret
```

```
unload_int ENDP


find_cmd_key PROC NEAR

    push AX
    push SI

    mov SI,82H
    mov AL,ES:[SI]
    cmp AL,'/'
    jne end_check

    inc SI
    mov AL,ES:[SI]
    cmp AL,'u'
    jne end_check

    inc SI
    mov AL,ES:[SI]
    cmp AL,'n'
    jne end_check
    mov flag,1

  end_check:
    pop SI
    pop AX
    ret

find_cmd_key ENDP


is_loaded_int PROC NEAR

    push AX
    push DX
    push SI

    mov flag,1
    mov AH,35H
    mov AL,1CH
    int 21H

    mov SI,offset int_indicator
    sub SI,offset my_interuption
    mov DX,ES:[BX+SI]
    cmp DX,0AAAAH
    je is_loaded
    mov flag,0

  is_loaded:
    pop SI
    pop DX
    pop AX
    ret
```

```
is_loaded_int ENDP


_print PROC NEAR

    push AX

    mov AH,09H
    int 21H

    pop AX
    ret

_print ENDP


MAIN   PROC  FAR

    mov AX,DATA
    mov DS,AX
    mov PSP,ES
    mov flag,0
    call find_cmd_key
    cmp flag,1
    je unload

    call is_loaded_int
    cmp flag,0
    je not_loaded
    mov DX,offset msg_is_in_memory
    call _print
    jmp final

  not_loaded:
    mov DX,offset msg_is_load
    call _print
    call load_int
    jmp final

  unload:
    call is_loaded_int
    cmp flag,0
    jne already_loaded
    mov DX,offset msg_is_not_load
    call _print
    jmp final

  already_loaded:
    call unload_int
    mov DX,offset msg_is_unload
    call _print

  final:
    mov AH,4CH
```

```
    xor AL,AL
    int 21H

Main      ENDP
CODE      ENDS
          END MAIN
```