МИНОБРНАУКИ РОССИИ

САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ

ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)

Кафедра МО ЭВМ

ОТЧЁТ

по лабораторной работе №4

по дисциплине «Операционные системы»

Тема: Обработка стандартных прерываний

Студент гр.8382 _____ Фильцин И.В.

Преподаватель _____ Ефремов М.А..

Санкт-Петербург

2020

Цель работы

В архитектуре компьютера существуют стандартные прерывания, за которыми закреплены определенные вектора прерываний. Вектор прерываний хранит адрес подпрограммы обработчика прерываний. При возниковении 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 возниковении такого сигнала возникает прерывание с определенным значением вектора. Таким образом, управление будет передано функции, чья точка входа записана в соответствующий вектор прерывания.

Ход работы

В ходе лабораторной программы был написан программный модуль типа .EXE, устанавливающий собственный обработчик прерывания сигналов таймера.

После запуска программа запоминает старый обработчик прерывания, устанавливает новый и завершается не освобождая занятую память.

Размещение программы в памяти см. на рис. 1

После повторного запуска программа корректно определяет, что обработчик прерывания был изменен, и не делает этого снова.

При запуске программы с ключом $/un$ происходит выгрузка обработчика прерывания и восстановления старого. При этом занята память освобождается.

Состояние памяти см. на рис. 2

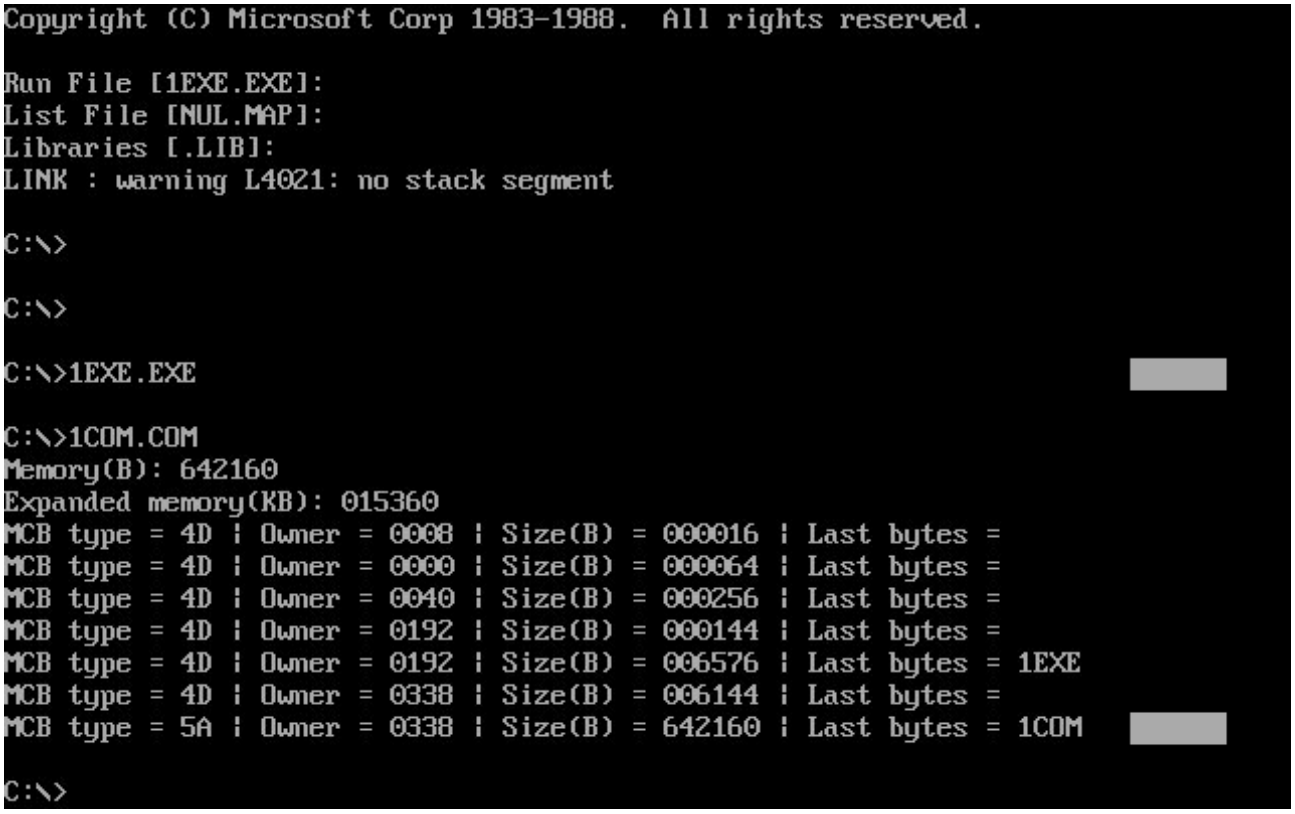
Copyright (C) Microsoft Corp 1983-1988. All rights reserved.
Run File [1EXE.EXE]:
List File [NUL.MAP]:
Libraries [.LIB]:
LINK : warning L4021: no stack segment
C:\>
C:\>
C:\>1EXE.EXE
C:\>1COM.COM
Memory(B): 642160
Expanded memory(KB): 015360
MCB type = 4D ¦ Owner = 0008 ¦ Size(B) = 000016 ¦ Last bytes =
MCB type = 4D ¦ Owner = 0000 ¦ Size(B) = 000064 ¦ Last bytes =
MCB type = 4D ¦ Owner = 0040 ¦ Size(B) = 000256 ¦ Last bytes =
MCB type = 4D ¦ Owner = 0192 ¦ Size(B) = 000144 ¦ Last bytes =
MCB type = 4D ¦ Owner = 0192 ¦ Size(B) = 006576 ¦ Last bytes = 1EXE
MCB type = 4D ¦ Owner = 0338 ¦ Size(B) = 006144 ¦ Last bytes =
MCB type = 5A ¦ Owner = 0338 ¦ Size(B) = 642160 ¦ Last bytes = 1COM
C:\>

Рис. 1

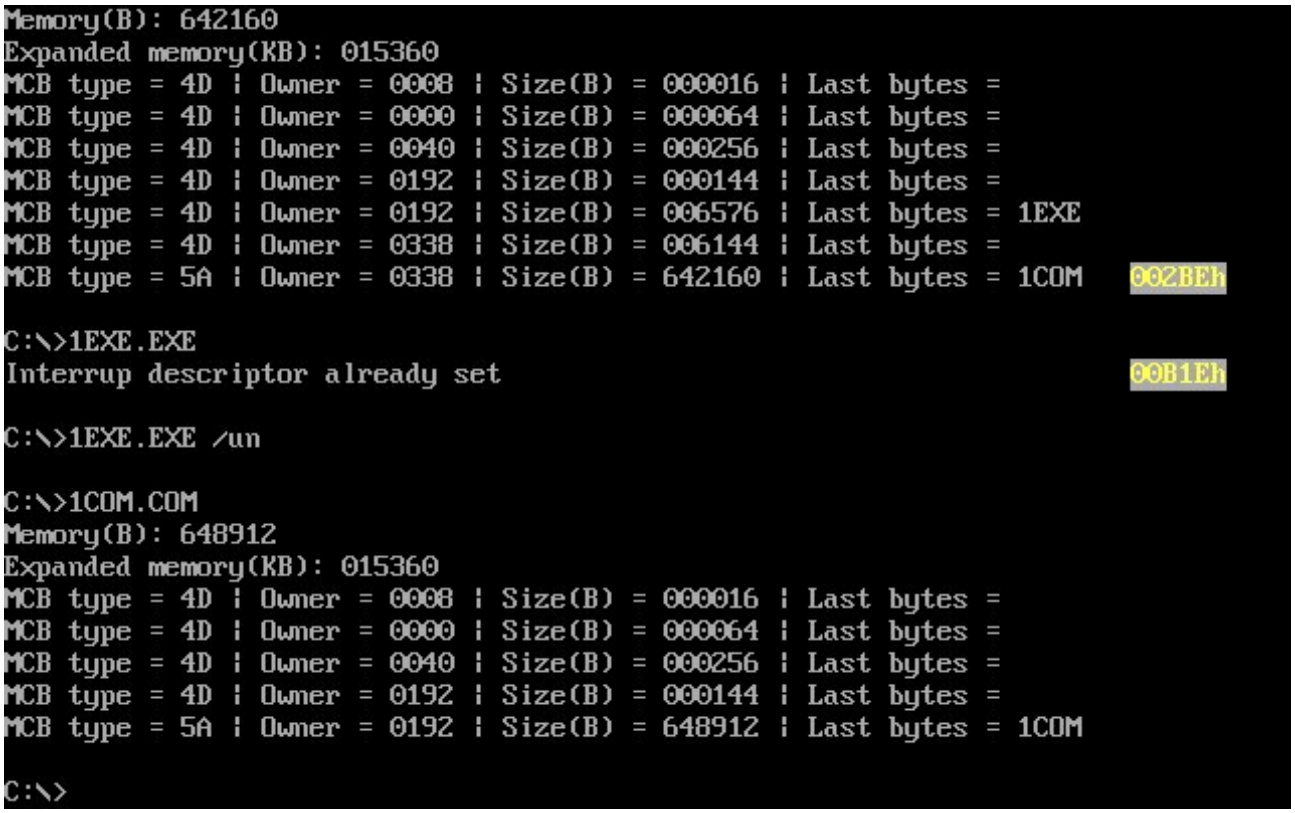
Memory(B): 642160
Expanded memory(KB): 015360
MCB type = 4D ¦ Owner = 0008 ¦ Size(B) = 000016 ¦ Last bytes =
MCB type = 4D ¦ Owner = 0000 ¦ Size(B) = 000064 ¦ Last bytes =
MCB type = 4D ¦ Owner = 0040 ¦ Size(B) = 000256 ¦ Last bytes =
MCB type = 4D ¦ Owner = 0192 ¦ Size(B) = 000144 ¦ Last bytes =
MCB type = 4D ¦ Owner = 0192 ¦ Size(B) = 006576 ¦ Last bytes = 1EXE
MCB type = 4D ¦ Owner = 0338 ¦ Size(B) = 006144 ¦ Last bytes =
MCB type = 5A ¦ Owner = 0338 ¦ Size(B) = 642160 ¦ Last bytes = 1COM 002BEh
C:\>1EXE.EXE
Interrup descriptor already set 00B1Eh
C:\>1EXE.EXE /un
C:\>1COM.COM
Memory(B): 648912
Expanded memory(KB): 015360
MCB type = 4D ¦ Owner = 0008 ¦ Size(B) = 000016 ¦ Last bytes =
MCB type = 4D ¦ Owner = 0000 ¦ Size(B) = 000064 ¦ Last bytes =
MCB type = 4D ¦ Owner = 0040 ¦ Size(B) = 000256 ¦ Last bytes =
MCB type = 4D ¦ Owner = 0192 ¦ Size(B) = 000144 ¦ Last bytes =
MCB type = 5A ¦ Owner = 0192 ¦ Size(B) = 648912 ¦ Last bytes = 1COM
C:\>

Рис. 2

Контрольные вопросы

1) Как реализован механизм прерывания от часов?

Системный таймер подключен к линии запроса на прерывание IRQ0 и вызывает прерывание 8h приблизительно 18.2 раза в секунду.

Обработчик прерывания 8h выполнят "полезную"работу (увеличивает на 1 текущее значение переменной, располагающейся в BIOS).

Затем обработчик прерывания таймера вызывает прерывание 1ch. По умолчанию вектор 1ch указывает на iret. При этом прерывание 1ch вызывается обработчиком 8h до сброса контроллера прерывания, поэтому во время выполнения все аппаратные прерывания запрещены.

2) Какого типа прерывания использовались в работе?

Аппаратные - 8h (неявно)

Программные - 1ch, 10h, 21h

Вывод

В ходе лабораторной работы был реализован обработчик прерываний сигналов таймера.

Приложение А. Исходный код программы

```
.model small

.code

db 100h dup(0)
istack label word

count dw 0
count_string db ’00000h’
keep_ss dw 0
keep_sp dw 0

tetr_to_hex proc near
  and al, 0fh
  cmp al, 09
  jbe next
  add al, 07
  next:
    add al, 30h
    ret
tetr_to_hex endp

byte_to_hex proc near
  push cx
  mov ah, al
  call tetr_to_hex
```

```
  xchg al, ah
  mov cl, 4
  shr al, cl
  call tetr_to_hex
  pop cx
  ret
byte_to_hex endp

wrd_to_hex proc near
  push bx
  mov bh, ah
  call byte_to_hex
  mov [di], ah
  dec di
  mov [di], al
  dec di
  mov al, bh
  call byte_to_hex
  mov [di], ah
  dec di
  mov [di], al
  pop bx
  ret
wrd_to_hex endp

resident_begin proc far
  push ax ; уж возьмём 2 байта из пользовательского
```

```
стека
mov keep_ss, ss
mov keep_sp, sp
mov ax, @code

cli
mov ss, ax
mov sp, offset istack
sti

push ds
push dx
push es
push bp
push cx
push di

mov ds, ax
mov es, ax
mov bp, offset count_string

inc count
mov di, offset count_string
add di, 4
mov ax, count
call wrd_to_hex
```

```
  mov ah, 013h
  mov al, 0
  mov bh, 0
  mov dh, 22
  mov dl, 70
  mov cx, 6
  int 10h

  pop di
  pop cx
  pop bp
  pop es
  pop dx
  pop ds

  cli
  mov ss, keep_ss
  mov sp, keep_sp
  sti

  mov al, 020h
  out 020h, al

  pop ax

  iret
resident_begin endp
```

```
last_byte label word

unload_handler proc near
  push ds
  push ax
  push es
  push bx
  push dx


  mov dx, 0
  mov ah, 025h
  mov al, vec_set_info
  int 21h

  mov ah, 035h
  mov al, 01ch
  int 21h

  mov ah, 035h
  mov al, vec_csip_info
  int 21h

  cli
  mov dx, es
  mov ds, dx
```

```
  mov dx, bx
  mov ah, 025h
  mov al, 01ch
  int 21h
  sti

  mov ah, 035h
  mov al, vec_seg_info
  int 21h

  mov ah, 049h
  int 21h

  mov es, bx
  mov ah, 049h
  int 21h


  pop dx
  pop bx
  pop es
  pop ax
  pop ds
  ret
unload_handler endp

load_handler proc near
```

```
push ax
push dx
push es
push bx
push dx

mov al, vec_set_info
mov dx, 1
mov ah, 025h
int 21h

push ds
mov dx, es:[02ch]
mov ax, es
mov ds, ax
mov al, vec_seg_info
mov ah, 025h
int 21h
pop ds

mov ah, 035h
mov al, 01ch
int 21h

mov dx, es
mov ds, dx
mov dx, bx
```

```
  mov ah, 025h
  mov al, vec_csip_info
  int 21h

  cli
  mov ax, @code
  mov ds, ax
  mov dx, offset resident_begin
  mov ah, 025h
  mov al, 01ch
  int 21h
  sti

  pop dx
  pop bx
  pop es
  pop dx
  pop ax
  ret
load_handler endp

check_handler proc near
  push bx
  push es

  mov ah, 035h
  mov al, vec_set_info
```

```
  int 21h

  cmp bx, 1
  je handler_is_set
  mov ax, 0
  jmp finish_check_handler

  handler_is_set:
    mov ax, 1

  finish_check_handler:

  pop es
  pop bx
  ret
check_handler endp

already_label db 'Interrup descriptor already set', 13,
   10, '$'
unset_option db '/un'
unset_size equ 3
vec_set_info equ 0ffh ; в этом векторе сохраним инфу
  стоит ли наше прерывание
vec_csip_info equ 0feh ; в этом векторе сохраним инфу
  о прошлом хендлере
vec_seg_info equ 0fdh ; а в этом адрес среды для
  освобождения
```

```
transient_begin:
  mov ax, @code

  cli
  mov ss, ax
  mov sp, offset istack
  sti

  mov ax, @code
  mov ds, ax

  mov cl, es:[080h]
  cmp cl, 4

  mov cx, unset_size
  mov si, offset unset_option
  mov di, 082h
  repe cmpsb

  jne unset_command_not_set

  unset_command_set:
    call unload_handler
    jmp finish

  unset_command_not_set:
```

```
    call check_handler
    cmp ax, 1
    jne handler_unloaded

  handler_loaded:
    mov dx, offset already_label
    mov ah, 09h
    int 21h
    jmp finish

  handler_unloaded:
    call load_handler
    mov dx, offset last_byte
    mov ah, 031h
    int 21h

  finish:
    mov ah, 04ch
    int 21h

end transient_begin
```