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. 9382 _____ Голубева В.П.

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.

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 lCh.

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

1) Сохранить значения регистров в стеке при входе и восстановить их при выходе.

2) 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.

Шаг 2. Запустите отлаженную программу и убедитесь, что резидентный обработчик прерывания lCh 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 МСВ. 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 того также следует запустить программу ЛР 3. Полученные результаты поместите в отчет.

Шаг 5. Ответьте на контрольные вопросы.

Выполнение работы.

Был создан exe модуль, который менял вектор прерывания у таймера на написанное. В ходе его работы выводилась 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 сравнивалсь сигнатура у обработчика прерывания пользовательской программы и той, что загружена в память). Результаты работы можно посмотреть в Рисунке 1.

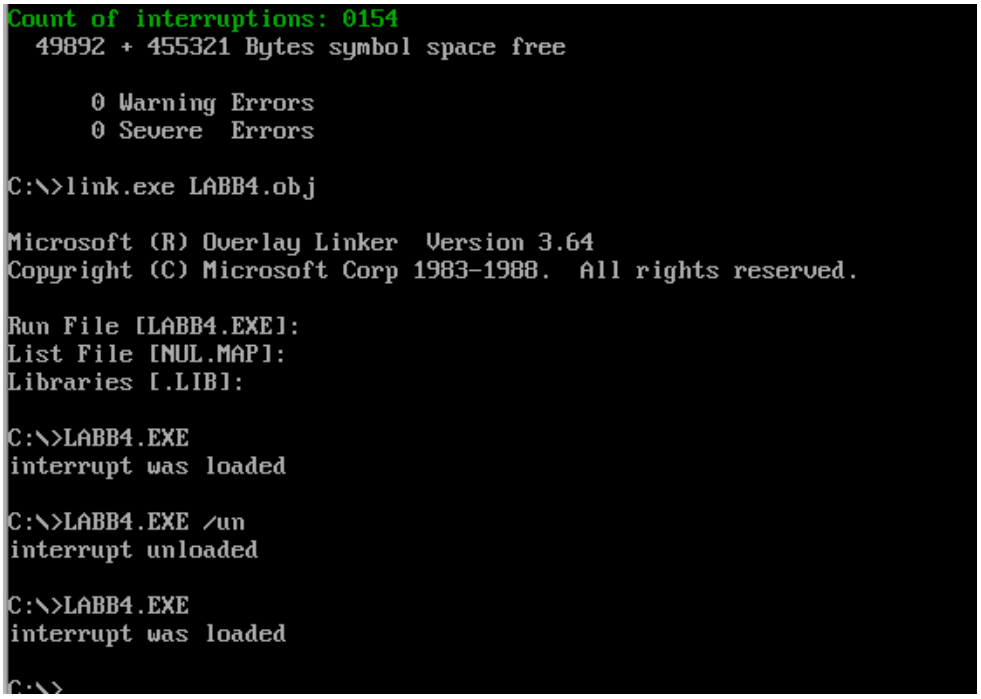

Рисунок 1. Результат загрузки программы в память

Для того, чтобы проверить, что программа находится в памяти, использовалась программа из лабораторной работы номер три, которая выводит цепочку mcb-блоков. Запустим эту программу и увидим, что обработчик прерывания находится в памяти. Результаты можно посмотреть в Рисунке 2.

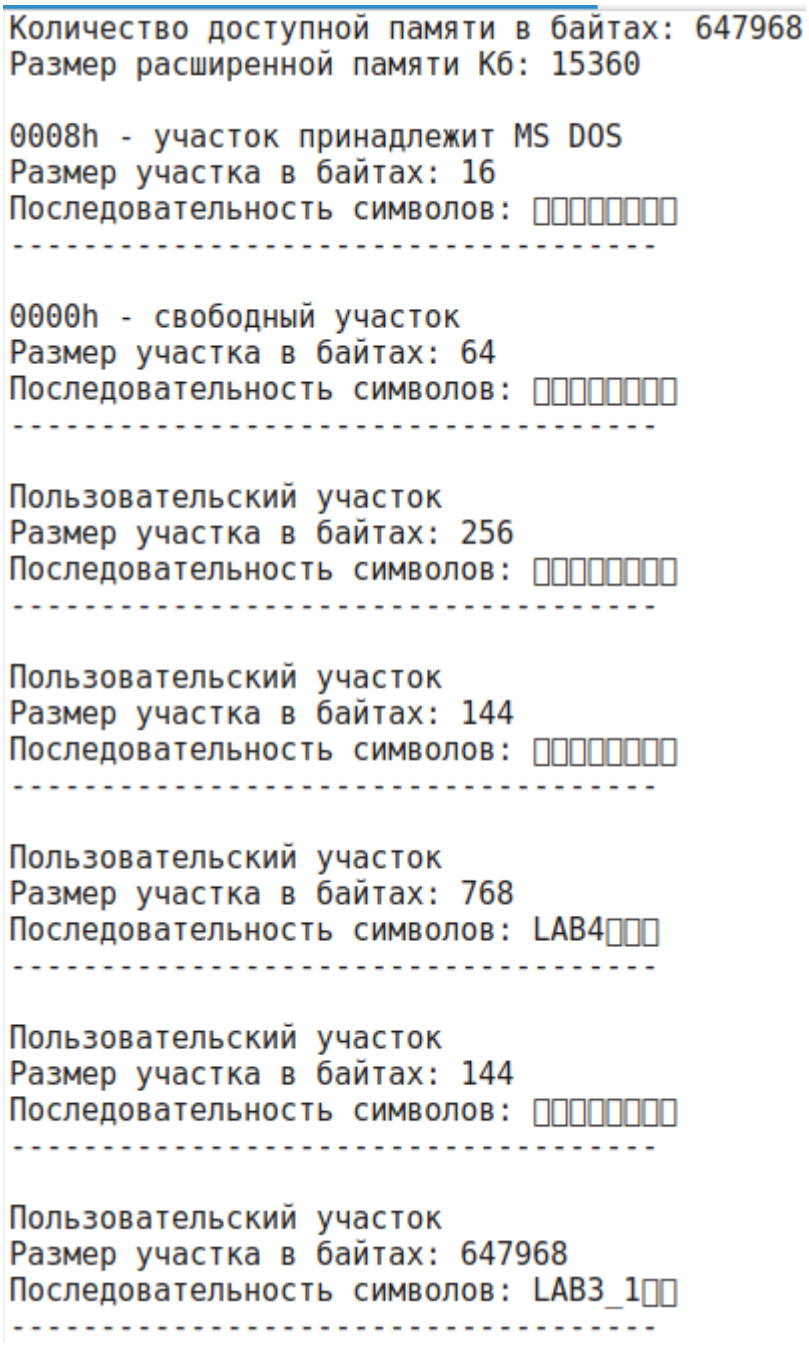

Рисунок 2. Результат работы программы LAB_1.COM из лабораторной работы номер 3

Теперь выгрузим обработчик из памяти и снова запустим программу лабораторной работы номер три. Результаты можно посмотреть в Рисунке 3 и Рисунке 4.

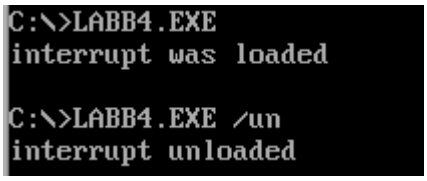

Рисунок 3. Выгрузка обработчика прерывания из памяти

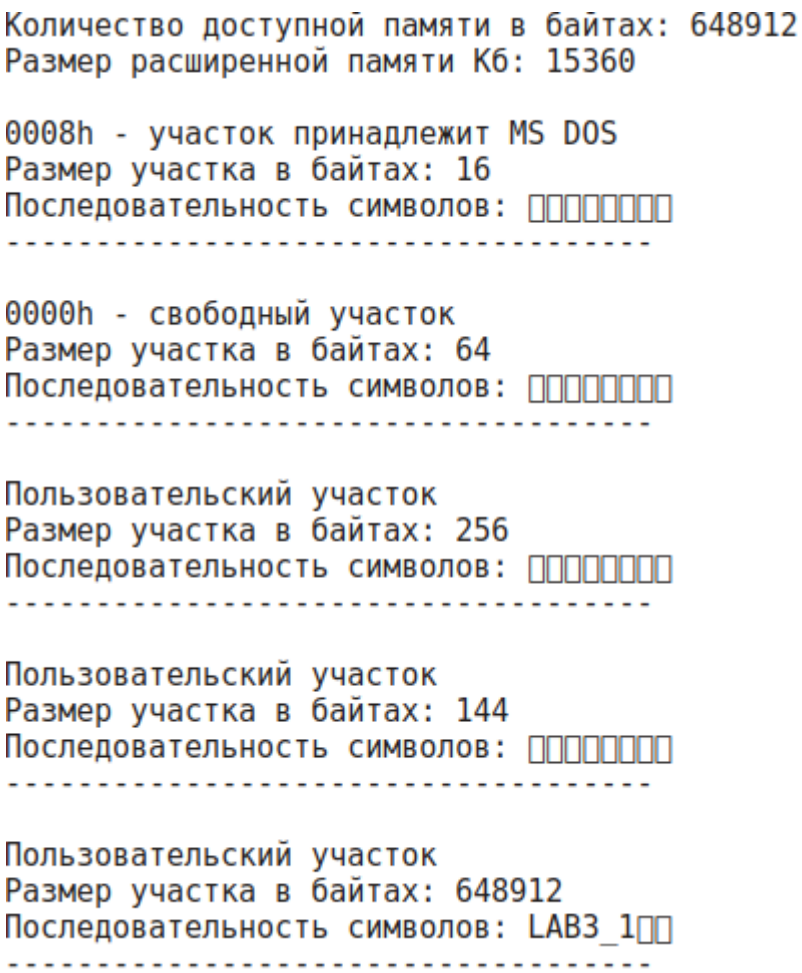

Рисунок 4. Демонстрация освобождения памяти после выгрузки программы

Таким образом можно убедиться, что обработчик прерывания был успешно выгружен из памяти.

Ответы на контрольные вопросы

Контрольные вопросы по лабораторной работе №4

1) Как реализован механизм прерывания от часов?

Ответ: 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 обрабатывется, затем управление возвращается прерванной программе

2) Какого типа прерывания использовались в работе?

Ответ: аппаратные(1Ch) и программные(int 10h, int 21h)

Выводы.

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

```
stacks segment stack
     dw 64 dup(?)
stacks ends

assume cs:code, ds:data, ss:stacks

code segment

int_count_func proc far
     jmp run

     keep_cs dw 0
     keep_ip dw 0
     nowPsp dw 0
     memAdrPsp dw 0
     count_interruptions 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

sizee:

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
      call print_str

      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
      call print_str
```

```
      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

print_str proc near
      push ax

      mov ah, 09h
      int  21h

      pop ax

      ret
print_str 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_already_load
      call print_str
      jmp eeend

      mov ah,4ch
      int 21h

interruption_is_not_loaded:
      call loadfunc

      mov dx, offset sizee
      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 eeend
```

```
not_set:
      mov dx, offset str_not_loaded
      call print_str
     jmp eeend

eeend:
      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_already_load db "interrupt is already load", 0DH, 0AH,
'$'
      str_load db "interrupt was loaded", 0DH, 0AH, '$'
data ends

end main
```