**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. 7381 | Минуллин М.А. |
| Преподаватель | Ефремов М.А. |

Санкт-Петербург

2019

**Цель работы.**

В архитектуре компьютера существуют стандартные прерывания, за которыми закреплены определённые вектора прерываний. Вектор прерываний хранит адрес подпрограммы обработчика прерываний. При возникновении прерывания, аппаратура компьютера передаёт 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 определённые интервалы времени и, при возникновении такого сигнала, возникает прерывание с определённым значением вектора. Таким образом, управление будет передано функции, чья точка входа записана в соответствующий вектор прерывания.

**Необходимые сведения для составления программы.**

Резидентные обработчик прерываний – это программные модули, которые вызываются при возникновении прерываний определённого типа (сигнал таймера, нажатие клавиши и т. д.), которым соответствуют определённые вектора прерывания. Когда вызывается прерывание, процессор переключается на выполнение кода обработчика, а затем возвращается на выполнение прерванной программы. Адрес возврата в прерванную программу (CS:IP) запоминается в стеке вместе с регистром флагов. Затем в CS:IP загружается адрес точки входа программы обработки прерывания и начинает выполняться его код. Обработчик прерывания должен заканчиваться инструкцией iret (возврат из прерывания).

Вектор прерывания имеет длину 4 байта. В первом хранится значение IP, во втором – CS. Младшие 1024 байта памяти содержат 256 векторов. Вектор для прерывания 0 начинается с ячейки 0000:0000, для прерывания 1 – с ячейки 0000:0004 и т. д.

**Ход работы.**

Был написан и отлажен .EXE модуль, выполняющий следующие функции:

1. Проверяется, установлено ли пользовательское прерывание с вектором 1Ch.
2. Устанавливается резидентная функция для обработки прерывания и настраивается вектор прерываний, если прерывание не установлено, осуществляется выход через функцию 2Ch прерывания 21h.
3. Если прерывание установлено, то выводится соответствующее сообщение и осуществляется выход через функцию 2Ch прерывания 21h.
4. Выгружается прерывание по соответствующему значению параметра в командной строке /un. Выгрузка прерывания состоит в восстановлении стандартного вектора прерываний и освобождения памяти, занимаемой резидентом. Осуществляется выход через функцию 4Ch прерывания 21h.

Была запущена полученная программа (рис. 1).



Рисунок 1 – Резидентная программа лабораторной работы №4.

Для проверки размещения прерывания в памяти была запущена программа из лабораторной работы №3, отображающей карту памяти в виде блоков MCB (рис. 2).

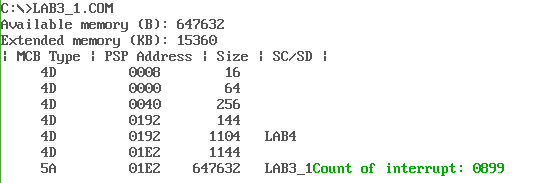


Рисунок 2 – Резидентная программа на карте памяти.

Произведена попытка повторного запуска программы лабораторной работы №4 (рис. 3). Выводится сообщение о том, что программа уже была запущена.



Рисунок 3 – Попытка запустить программу дважды.

Выполнен запуск программы лабораторной работы №4 с ключом выгрузки (рис. 4).



Рисунок 4 – Выгрузка резидентной программы.

Проверка отсутствия программы-резидента на карте памяти с помощью повторного запуска программы из лабораторной работы №3 (рис. 5).

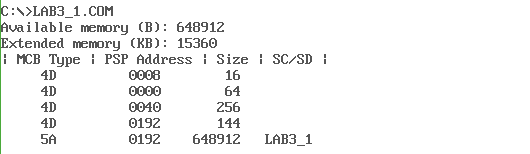


Рисунок 5 – Проверка отсутствия резидентной программы.

**Контрольные вопросы.**

В: Как реализован механизм прерывания от часов?

О: Прерывание по таймеру вызывается автоматически по каждому тику аппаратных часов каждые 55 миллисекунд. После вызова сохраняется содержимое регистров и определяется источник прерывания, по номеру которого определяется смещение в таблице векторов прерываний. Полученный адрес сохраняется в регистр CS:IP. После этого управление передаётся по этому адресу, т. е. выполняется запуск обработчика прерываний и происходит его выполнение. После выполнения происходит возврат управления прерванной программе.

В: Какого типа прерывания использовались в работе?

О: В данной лабораторной работе использовались аппаратные прерывания, прерывания функций MS DOS и прерывания функций BIOS.

**Выводы.**

В ходе выполнения лабораторной работы была написана программа, в которой был реализован обработчик прерываний сигналов таймера, а также были изучены дополнительный функции для работы с памятью: установка и выгрузка из памяти программы-резидента.

**ПРИЛОЖЕНИЕ А**

**ИСХОДНЫЙ ТЕКСТ .EXE МОДУЛЯ**

MY\_STACK segment STACK

    DW 64 dup (?)

MY\_STACK ends

DATA segment

    message\_1 db 'Resident was loaded', 13, 10, '$'

    message\_2 db 'Resident has already been loaded', 13, 10, '$'

    message\_3 db 'Resident was unloaded', 13, 10, '$'

DATA ends

CODE segment

    assume cs:CODE, ds:DATA, es:DATA, ss:MY\_STACK

setCurs proc

    push ax

    push bx

    push cx

    mov ah, 02h

    mov bh, 00h

    int 10h

    pop cx

    pop bx

    pop ax

    ret

setCurs endp

getCurs proc

    push ax

    push bx

    push cx

    mov ah, 03h

    mov bh, 00h

    int 10h

    pop cx

    pop bx

    pop ax

    ret

getCurs endp

printSTR proc

    push es

    push bp

    mov ax, SEG COUNT

    mov es, ax

    mov ax, offset COUNT

    mov bp, ax

    mov ah, 13h

    mov al, 00h

    mov cx, 25

    mov bh, 0

    mov bl, 13

    int 10h

    pop bp

    pop es

    ret

printSTR endp

ROUT proc far

    jmp ROUT\_START

    identifier db '0000'

    KEEP\_IP dw 0

    KEEP\_CS dw 0

    KEEP\_PSP dw 0

    flag db 0

    KEEP\_SS dw 0

    KEEP\_AX dw 0

    KEEP\_SP dw 0

    COUNT db 'Count of interrupt: 0000 $'

    inter\_stack dw 64 dup (?)

    end\_stack dw 0

ROUT\_COUNT:

    push si

    push cx

    push ds

    mov ax, SEG COUNT

    mov ds, ax

    mov si, offset COUNT

    add si, 23

    mov ah, [si]

    add ah, 1

    mov [si], ah

    cmp ah, 58

    jne END\_COUNT

    mov ah, 48

    mov [si], ah

    mov bh, [si - 1]

    add bh, 1

    mov [si-1], bh

    cmp bh, 58

    jne END\_COUNT

    mov bh, 48

    mov [si - 1], bh

    mov ch, [si - 2]

    add ch, 1

    mov [si - 2], ch

    cmp ch, 58

    jne END\_COUNT

    mov ch, 48

    mov [si - 2], ch

    mov dh, [si - 3]

    add dh, 1

    mov [si - 3], dh

    cmp dh, 58

    jne END\_COUNT

    mov dh, 48

    mov [si - 3], dh

END\_COUNT:

pop ds

pop cx

    pop si

    call printSTR

    pop dx

    call setCurs

    jmp END\_ROUT

ROUT\_START:

    mov KEEP\_AX, ax

    mov KEEP\_SS, ss

    mov KEEP\_SP, sp

    mov ax, cs

    mov ss, ax

    mov sp, offset end\_stack

    mov ax, KEEP\_AX

    push dx

    push ds

    push es

    cmp flag, 1

    je ROUT\_REC

    call getCurs

    push dx

    mov dh, 22

    mov dl, 39

    call setCurs

    jmp ROUT\_COUNT

ROUT\_REC:

    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

END\_ROUT:

    pop es

    pop ds

    pop dx

    mov ss, KEEP\_SS

    mov sp, KEEP\_SP

    mov ax, KEEP\_AX

    iret

ROUT endp

SET\_INTERRUPT 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 message\_1

    call PRINT

    pop dx

    ret

SET\_INTERRUPT endp

BASE\_FUNC proc

    mov ah, 35h

    mov al, 1Ch

    int 21h

    mov si, offset identifier

    sub si, offset ROUT

    mov ax, '00'

    cmp ax, es:[bx + si]

    jne NOT\_LOADED

    cmp ax,es:[bx + si + 2]

    jne NOT\_LOADED

    jmp LOADED

NOT\_LOADED:

    call SET\_INTERRUPT

    mov dx, offset LAST\_BYTE

    mov cl, 4

    shr dx, cl

    inc dx

    add dx, CODE

    sub dx, KEEP\_PSP

    xor al, al

    mov ah, 31h

    int 21h

LOADED:

    push es

    push ax

    mov ax, KEEP\_PSP

    mov es, ax

    mov al, es:[81h + 1]

    cmp al,'/'

    jne NOT\_UNLOAD

    mov al, es:[81h + 2]

    cmp al,'u'

    jne NOT\_UNLOAD

    mov al, es:[81h + 3]

    cmp al,'n'

    je UNLOAD

NOT\_UNLOAD:

    pop ax

    pop es

    mov dx, offset message\_2

    call PRINT

    ret

UNLOAD:

    pop ax

    pop es

    mov byte ptr es:[bx + si + 10], 1

    mov dx, offset message\_3

    call PRINT

    ret

BASE\_FUNC 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 KEEP\_PSP,es

    call BASE\_FUNC

    xor al,al

    mov ah,4Ch

    int 21H

LAST\_BYTE:

MAIN endp

CODE ends

END MAIN