**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра математического обеспечения и применения ЭВМ**

**ОТЧЕТ  
по лабораторной работе № 4**

**по дисциплине «Операционные системы»**

**Тема: «Обработка стандартных исключений»**

|  |  |  |
| --- | --- | --- |
| Студент гр. 8304 |  | Сергеев А.Д. |
| Преподаватель |  | Губкин А.Ф. |

Санкт-Петербург

2020

**Цель работы.**

Изучение обработки прерываний сигналов таймера, которые генерируются аппаратурой примерно 18.2 раза в секунду, вызывая прерывание с соответствующим вектором.

**Постановка задачи:**

**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 освобождения 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 определённом, 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 удалённой процедуры. Этот код будет работать после установки при возникновении прерывания. Он должен выполнять следующие функции:

1. Сохранить значение 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 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 отчёт.

**Шаг 3.** Запустите отлаженную программу ещё раз и убедитесь, что программа определяет установленный обработчик прерываний. Полученные результаты поместите в отчёт.

**Шаг 2.** 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 отчёт.

**Необходимые сведения для составления программы.**

Резидентные обработчики 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.

Для установки написанного прерывания используется функция 25H прерывания 21H, которая устанавливает вектор прерывания на указанный адрес.

Программа, выгружающая обработчик прерываний, должна восстанавливать оригинальные векторы прерываний. Функция 35 прерывания 21H позволяет восстановить значение вектора прерывания, помещая значение сегмента в ES, а смещение в BX.

Для того, чтобы оставить процедуру прерывания резидентной в памяти, следует воспользоваться функцией DOS 31H прерывания int 21H. Эта функция оставляет память, размер которой указывается в качестве параметра, занятой, а остальную память освобождает, и осуществляет выход в DOS.

Вывод на экран информации обработчиком прерываний осуществляется с помощью функций прерывания 10H.

**Описание программы.**

В результате выполнения лабораторной работы была написана программа, описание функций которой представлено ниже.

- INTERRUPT - резидентный обработчик прерывания, сохраняет и запоминает количество обработанных прерываний;

- LOAD - загрузка резидентного обработчика INTERRUPTION;

- UNLOAD - выгрузка резидентного обработчика INTERRUPTION;

- PRINT\_STRING - вывод строки из DX на экран;

- INT\_CHECK - проверка того, установлен ли резидентный обработчик INTERRUPTION;

- TAIL\_CHECK - проверка того, содержат ли аргументы, с которыми была вызвана программа /un.

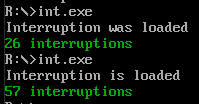
**Ход работы**

Написание исходного кода производилось в редакторе Atom на базе операционной системы Windows 10, сборка и отладка производились в эмуляторе DOSBox.

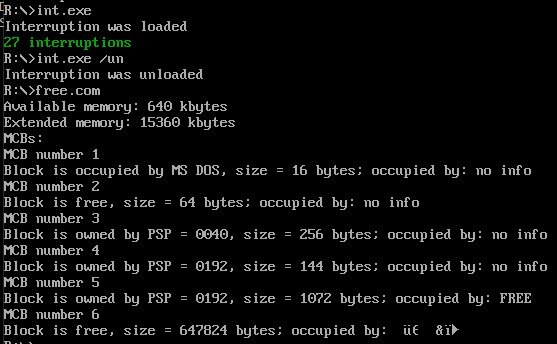
Рисунок 1 — Вывод программы int.exe после первого запуска

Рисунок 2 — Вывод программы free.com после выполнения int.exe

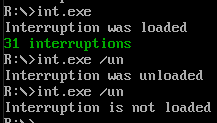
Как видно из рисунка, процедура прерывания осталось резидентной в памяти.

Рисунок 3 — Вывод программы int.exe при повторном запуске

На рисунке 3 показано, что при повторном запуске программа выводит сообщение о том, что резидентный обработчик уже загружен.

Рисунок 4 - Вывод программы free.com после выполнения int.exe с ключом выгрузки

Из рисунка 4 видно, что после выгрузки резидентного обработчика из памяти вся занятая им память была освобождена.

Рисунок 5 — Вывод программы int.exe при повторном запуске с ключом выгрузки

Как видно из рисунка 5, при выгрузке резидентного обработчика было выведено сообщение, а также при запросе повторной выгрузки было показано, что резидентный обработчик не загружен.

**Вывод.**

В результате выполнения данной лабораторной работы была изучена работа прерываний от системного таймера, а также механизм загрузки и выгрузки резидентных обработчиков.

**Контрольные вопросы.**

**Как реализован механизм прерывания от часов:**

Аппаратное прерывание int 8h срабатывает 1193180/65536 раз в секунду. Стандартный обработчик этого прерывания увеличивает счётчик и вызывает другое прерывание — 1Ch. По умолчанию оно указывает на команду IRET. Во время выполнения этих двух прерываний не вызываются другие.

**Какого типа прерывания использовались в программе:**

В программе использовались по большей части программные прерывания, такие как int 21h и 10h. Написанный обработчик применялся к асинхронному аппаратному прерыванию, 1Ch, прерыванию от таймера.

**ПРИЛОЖЕНИЕ А**

**ИСХОДНЫЙ КОД ПРОГРАММЫ. INT.ASM**

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

CODE SEGMENT

INTERRUPT PROC FAR

jmp INT\_START

INT\_DATA:

SUB\_STACK dw 128 dup(0)

INTERRUPTIONS\_INFO db " interruptions "

INT\_CODE dw 42025

INTERRUPTIONS dw 0

KEEP\_IP dw 0

KEEP\_CS dw 0

KEEP\_PSP dw 0

KEEP\_SS dw 0

KEEP\_SP dw 0

INT\_START:

mov KEEP\_SS, ss

mov KEEP\_SP, sp

mov ss, SUB\_STACK

mov sp, 0

push ax

push bx

push cx

push dx

push si

push es

push ds

push bp

INT\_SETUP:

mov ax, seg INTERRUPT

mov ds, ax

mov es, ax

INT\_SAVE\_CURSOR:

mov ah, 03h

mov bh, 0h

int 10h

push dx

INT\_ADD:

mov si, offset INTERRUPTIONS

mov ah, [si]

mov al, [si + 1]

inc ax

mov [si], ah

mov [si + 1], al

INT\_PREP\_TO\_DEC:

xor dx, dx

mov bx, 10

xor cx, cx

INT\_TO\_DEC\_CYCLE:

div bx

push dx

xor dx, dx

inc cx

cmp ax, 0h

jnz INT\_TO\_DEC\_CYCLE

mov ah, 2

mov bh, 0

mov dh, 23

mov dl, 0

int 10h

INT\_PRINT\_NUM:

pop ax

or al, 30h

push cx

mov ah, 09h

mov bl, 2h

mov bh, 0

mov cx, 1

int 10h

mov ah, 2

mov bh, 0

add dx, 1

int 10h

pop cx

loop INT\_PRINT\_NUM

INT\_PRINT\_STRING:

mov bp, offset INTERRUPTIONS\_INFO

mov ah, 13h

mov al, 1h

mov bl, 2h

mov bh, 0

mov cx, 19

int 10h

INT\_LOAD\_CURSOR:

pop dx

mov ah, 02h

mov bh, 0h

int 10h

INT\_END:

pop bp

pop ds

pop es

pop si

pop dx

pop cx

pop bx

pop ax

mov ss, KEEP\_SS

mov sp, KEEP\_SP

mov al, 20h

out 20h, al

iret

INTERRUPT ENDP

LAST\_BYTE:

LOAD PROC

push ax

push bx

push cx

push dx

push es

push ds

mov ah, 35h

mov al, 1Ch

int 21h

mov KEEP\_IP, bx

mov KEEP\_CS, es

mov dx, offset INTERRUPT

mov ax, seg INTERRUPT

mov ds, ax

mov ah, 25h

mov al, 1Ch

int 21h

pop ds

mov dx, offset LAST\_BYTE

add dx, 100h

mov cl, 4h

shr dx, cl

inc dx

mov ah, 31h

int 21h

pop es

pop dx

pop cx

pop bx

pop ax

ret

LOAD ENDP

UNLOAD PROC

push ax

push bx

push dx

push ds

push es

push si

mov ah, 35h

mov al, 1Ch

int 21h

mov si, offset KEEP\_IP

sub si, offset INTERRUPT

mov dx, es:[bx + si]

mov si, offset KEEP\_CS

sub si, offset INTERRUPT

mov ax, es:[bx + si]

push ds

mov ds, ax

mov ah, 25h

mov al, 1Ch

int 21h

pop ds

mov si, offset KEEP\_PSP

sub si, offset INTERRUPT

mov ax, es:[bx + si]

mov es, ax

push es

mov ax, es:[2Ch]

mov es, ax

mov ah, 49h

int 21h

pop es

mov ah, 49h

int 21h

pop si

pop es

pop ds

pop dx

pop bx

pop ax

sti

ret

UNLOAD ENDP

INT\_CHECK PROC

push ax

push bx

push si

mov ah, 35h

mov al, 1Ch

int 21h

mov si, offset INT\_CODE

sub si, offset INTERRUPT

mov ax, es:[bx + si]

cmp ax, INT\_CODE

jne INT\_CHECK\_END

mov INT\_LOADED, 1

INT\_CHECK\_END:

pop si

pop bx

pop ax

ret

INT\_CHECK ENDP

TAIL\_CHECK PROC

cmp byte ptr es:[82h], '/'

jne CL\_CHECK\_END

cmp byte ptr es:[83h], 'u'

jne CL\_CHECK\_END

cmp byte ptr es:[84h], 'n'

jne CL\_CHECK\_END

mov TAIL\_UN, 1

CL\_CHECK\_END:

ret

TAIL\_CHECK ENDP

PRINT\_STRING PROC NEAR

push ax

mov ah, 09h

int 21h

pop ax

ret

PRINT\_STRING ENDP

MAIN PROC

push ds

xor ax, ax

push ax

mov ax, DATA

mov ds, ax

mov KEEP\_PSP, es

call TAIL\_CHECK

call INT\_CHECK

cmp TAIL\_UN, 1

je INT\_UNLOAD

cmp INT\_LOADED, 1

jne INT\_LOAD

mov dx, offset IS\_LOADED\_INFO

call PRINT\_STRING

jmp MAIN\_END

INT\_LOAD:

mov dx, offset LOADED\_INFO

call PRINT\_STRING

call LOAD

jmp MAIN\_END

INT\_UNLOAD:

cmp INT\_LOADED, 1

jne NOT\_EXIST

call UNLOAD

mov dx, offset NOT\_LOADED\_INFO

call PRINT\_STRING

jmp MAIN\_END

NOT\_EXIST:

mov dx, offset IS\_NOT\_LOADED\_INFO

call PRINT\_STRING

MAIN\_END:

xor al, al

mov ah, 4Ch

int 21h

MAIN ENDP

CODE ENDS

ASTACK SEGMENT STACK

dw 128 dup(0)

ASTACK ENDS

DATA SEGMENT

LOADED\_INFO db "Interruption was loaded", 10, 13, "$"

IS\_LOADED\_INFO db "Interruption is loaded", 10, 13, "$"

NOT\_LOADED\_INFO db "Interruption was unloaded$"

IS\_NOT\_LOADED\_INFO db "Interruption is not loaded$"

INT\_LOADED db 0

TAIL\_UN db 0

DATA ENDS

END MAIN