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. 0382 _____ Крючков А.М.

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.

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 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 4 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 этого также следует запустить программу ЛР 3. Полученные результаты поместите в отчет.

Шаг 5. Ответьте на контрольные вопросы

Выполнение работы.

Исходный код модуля представлен в приложении А.

Шаг 1.

Был написан программный модуль типа .EXE. Для проверки наличия установки пользовательского прерывания с вектором 1ch была написана процедура is_int_installed. Если прерывание не установлено, то прерывание устанавливается при помощи функции install_int. Для этого сохраняется стэк в специально выделенное место памяти. Функция прерывания была помещена в самое начало программы, а сегмент данных и стэка после сегмента кода. Это было сделано для того, чтобы в памяти оставалась только функция прерывания. Если прерывание уже установлено, то выводится соответствующее сообщение, а программа завершается. Если в командную строку было передано сообщение о выгрузке прерывания, то при помощи помощи процедуры check_console проверяется наличие этого сообщения и при помощи процедуры uninstall_int прерывание выгружается.

Прерывание организует свой стек и при помощи прерывания int 10h выводится информация на экран.

Рисунок 1 – Результат работы программы lab4.exe

Шаг 2. Был запущен код 3 лабораторной, чтобы показать размещение в памяти нашего прерывания.

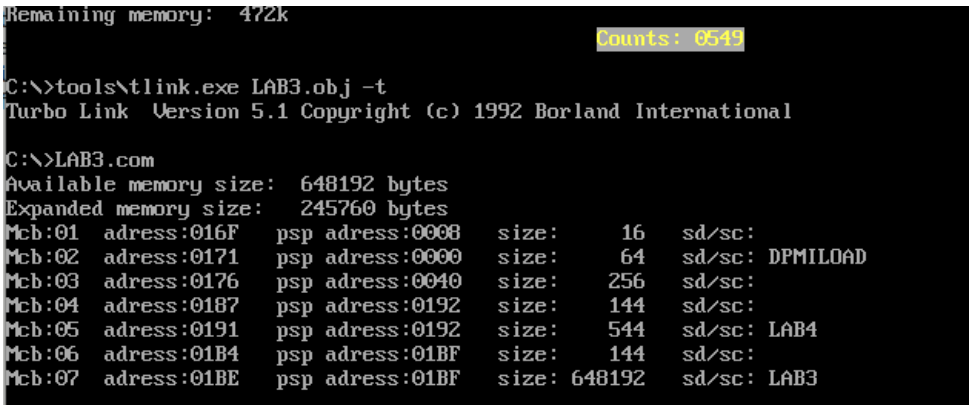

Рисунок 2 – память в виде списка блоков MCB

Шаг 3. Отлаженная программа была запущена еще раз. Можно убедиться, что программа определяет установленный обработчик прерываний.

Рисунок 3 – повторный запуск lab4.exe

Шаг 4. При запуске программы с ключом \un выводится сообщение об выгрузке прерывания. После запуска lab3.com можно увидеть что ни один блок памяти больше не содержит в себе lab4.

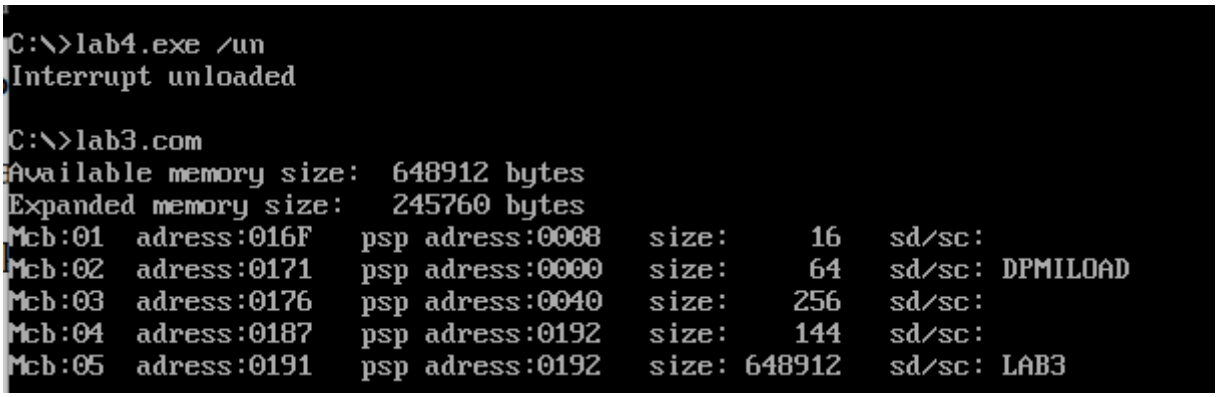

Рисунок 4 – запуск lab4.exe с ключом \un и запуск lab3.com

Контрольные вопросы.

Сегментный адрес недоступной памяти:

1. Как реализован механизм прерывания от часов?

Прерывание INT 1Ch является программным и вызывается обработчиком аппаратного прерывания от таймера INT 08h приблизительно 18,2 раза в секунду. Таким образом после установки пользовательского прерывания программа вызывает необходимое прерывание 18,2 раза в секунду.

2. Какого типа прерывания использовались в работе?

Программного типа: int 10h(видеосервис — функция BIOS), 21h(функции DOS). Например, int 21h по вектором прерывания 1ch.

Выводы.

В процессе выполнения данной лабораторной работы был изучен механизм работы аппаратного таймера, проведена работа по созданию резидентного пользовательского обработчика прерывания.

ПРИЛОЖЕНИЕ А.

Исходный код

lab4.asm:

```
code segment
assume cs:code, ds:data, ss:astack

interrupt proc far
jmp int_start
psp             dw ?
keep_cs    dw ?
keep_ip    dw ?
keep_ss    dw ?
keep_sp    dw ?
int_id     dw 0abcdh
counter         db 'counts: 0000'
int_stack  db 128 dup(?)

int_start:
;запомнили стэк программы
mov keep_ss, ss
mov keep_sp, sp
;поменяли стэк программы на стэк прерывания
mov sp, seg interrupt
mov ss, sp
mov sp, offset int_start
push ds
push es
push ax
push bx
push cx
push dx
push si
push bp
;запоминаем текущую позицию курсора
mov ah, 03h
mov bh, 0
int 10h
push dx

;меняем положение курсора
mov ah, 02h
mov bh, 0
mov dl, 30h
mov dh, 4h
int 10h

mov si, seg counter
mov ds, si
mov si, offset counter
add si, 7

mov cx, 4
num_loop:
mov bp, cx
mov ah, [si+bp]
```

```
inc ah
mov [si+bp], ah
cmp ah, 3ah
jne num_loop_end
mov ah, 30h
mov [si+bp], ah
loop num_loop
num_loop_end:

mov bp, seg counter
mov es, bp
mov bp, offset counter
mov ah, 13h
mov al, 1
mov bh, 0
mov cx, 12
int 10h


;восстанавливаем изначальное положение курсора
mov ah, 02h
mov bh, 0
pop dx
int 10h

pop bp
pop si
pop dx
pop cx
pop bx
pop ax
pop es
pop ds
mov sp, keep_ss
mov ss, sp
mov sp, keep_sp
mov al, 20h
out 20h, al
iret
LAST_BYTE:
interrupt endp

is_int_installed proc near
push ax
push bx
push dx
push si

mov int_installed, 1
mov ah, 35h
mov al, 1ch
int 21h
mov si, offset int_id
sub si, offset interrupt
mov dx, es:[bx+si]
cmp dx, 0abcdh
je loaded
mov int_installed, 0

loaded:
```

```
pop si
pop dx
pop bx
pop ax
ret
is_int_installed endp

install_int proc near
push ds
push es
push ax
push bx
push cx
push dx

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

;сохраняем наше прерывание до LAST_BYTE
mov dx, offset LAST_BYTE
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 dx
pop cx
pop bx
pop ax
pop es
pop ds
ret
install_int endp

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

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
mov dx, es:[2ch]
mov ah, 49h
int 21h
mov es, dx
mov ah, 49h
int 21h
sti

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

check_console proc near
push ax

mov int_installed, 0
mov al, es:[82h]
cmp al, '/'
jne no_key
mov al, es:[83h]
cmp al, 'u'
jne no_key
mov al, es:[84h]
cmp al, 'n'
jne no_key
mov int_installed, 1

no_key:
pop     ax
ret
check_console endp

print proc near
push ax
mov ah, 09h
int 21h
pop ax
ret
print endp

main proc far
push ds
xor ax, ax
mov ax, data
mov ds, ax

mov psp, es
```

```
call check_console

cmp int_installed, 1
je int_unload

call is_int_installed
cmp int_installed, 0
je int_load
mov dx, offset already_loaded_msg
call print
jmp exit

int_load:
mov dx, offset loaded_msg
call print
call install_int
jmp exit

int_unload:
call is_int_installed
cmp int_installed, 0
je unloaded
call uninstall_int
unloaded:
mov dx, offset unloaded_msg
call print

exit:
pop ds
mov ah, 4ch
int 21h
main endp
code ends

astack segment stack
dw 128 dup(?)
astack ends

data segment
int_installed                db 0
loaded_msg         db 'Interrupt loaded',0dh,0ah,'$'
unloaded_msg       db 'Interrupt unloaded',0dh,0ah,'$'
already_loaded_msg db 'Interrupt already loaded',0dh,0ah,'$'
data ends
end main
```