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. 8382 _____ Чирков С.А.

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 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.

Выполнение работы.

В процессе выполнения лабораторной работы был написан и отлажен программный модуль типа .EXE, выполняющий следующие функции:

- Проверяет, установлено ли пользовательское прерывание с вектором 1Ch
- Устанавливает резидентную функцию для обработки прерывания и настраивает вектор прерываний, если прерывание не установлено, и осуществляется выход по функции 4Ch прерывания int 21h.
- Если прерывание установлено, то выводится соответствующее сообщение и осуществляется выход по функции 4Ch прерывания int 21h.
- 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.

Результат работы программы в различных состояниях показан на рисунке 1. Состояние памяти при работе с обработчиком прерывания показано на рисунках 2-4.

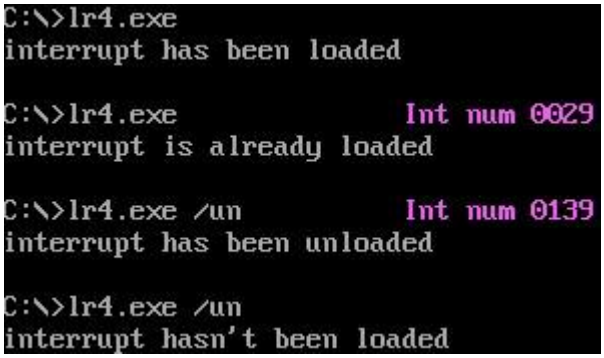

Рисунок 1. Тестирование программы при различных состояниях

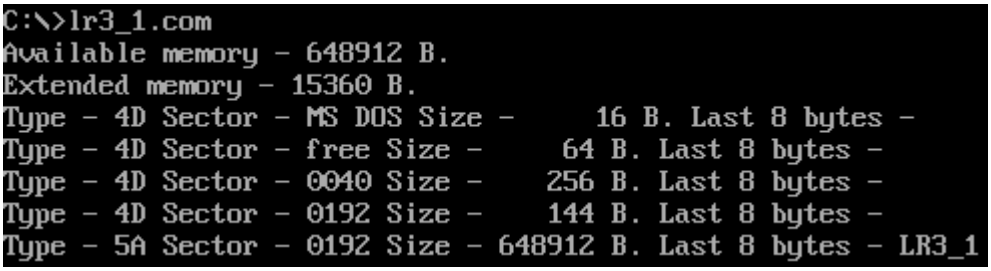

Рисунок 2. Состояние памяти до загрузки прерывания

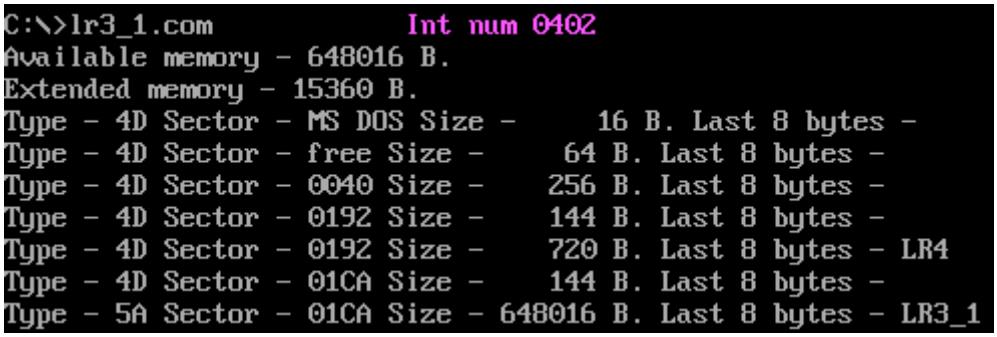

Рисунок 3. Состояние памяти после загрузки прерывания

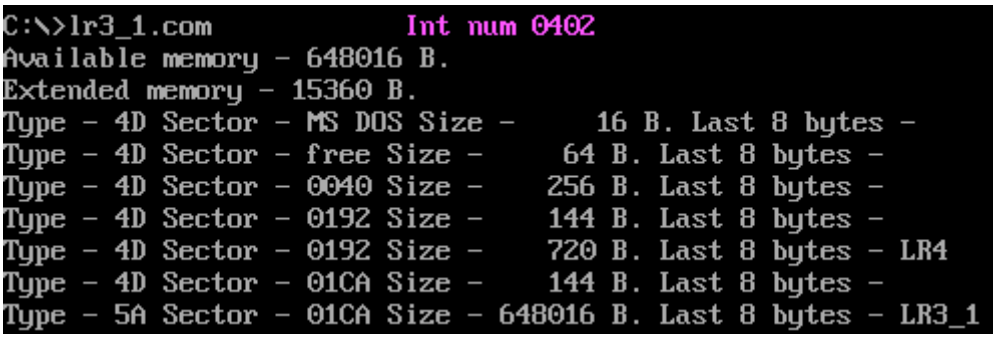

Рисунок 4. Состояние памяти после освобождения

Контрольные вопросы.

1. Как реализован механизм прерывания от часов?

Прерывание int 1Ch берет по каждому тику аппаратных часов (каждые 55 миллисекунд; приблизительно 18.2 раз в секунду), сохраняет состояние регистров, определяет смещение прерывания в таблице векторов прерываний и помещает этот адрес в CS:IP. Далее обрабатывается само прерывание и после завершения работы управление возвращается прерванной программе.

2. Какого типа прерывания использовались в работе?

Аппаратные (1Ch) и программные (10h, 21h) прерывания.

Выводы.

В ходе работы был построен обработчик прерываний сигналов таймера, получены навыки работы написания резидентного обработчика прерывания.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ LR4.ASM

```
CODE SEGMENT
      ASSUME
SS:AStack,DS:DATA,CS:CODE

MYINT PROC FAR
jmp myintcode
intdata:
count dw 0
countstr db 'Int num 0000'
int_flag dw 1919h
keep_cs dw 0
keep_ip dw 0
keep_psp dw 0
keep_ax dw 0
keep_ss dw 0
keep_sp dw 0
int_stack dw 80h dup(?)
myintcode:
mov keep_ax, ax
mov keep_ss, ss
mov keep_sp, sp
mov ax, seg int_stack
mov ss, ax
mov sp, offset int_stack
add sp, 100h
push bx
push cx
push dx
push si
push ds
mov ax, seg intdata
mov ds, ax

inc count
mov ax, count
mov dx, 0
mov si, offset countstr
add si, 11
call WRD_TO_DEC

mov ah, 3
mov bh, 0
int 10h
push dx
```

```
mov ah, 2
mov bh, 0
mov dx, 1819h
int 10h

push es
push bp
mov ax, seg countstr
mov es, ax
mov bp, offset countstr
mov al, 1
mov bh, 0
mov bl, 13
mov cx, 12
mov ah, 13h
int 10h
pop bp
pop es

pop dx
mov ah, 2
mov bh, 0
int 10h

pop ds
pop si
pop dx
pop cx
pop bx
mov sp, keep_sp
mov ax, keep_ss
mov ss, ax
mov ax, keep_ax
mov al, 20h
out 20h, al
iret
MYINT ENDP

WRD_TO_DEC PROC NEAR
      push  cx
      push  dx
      mov   cx,10
loop_b: div      cx
      or         dl,30h
      mov   [si],dl
      dec   si
      xor   dx,dx
      cmp   ax,0
```

```
      jnz   loop_b
endl: pop   dx
      pop   cx
      ret
WRD_TO_DEC ENDP

endmyint:

CHECKTOUNLOAD PROC
push ax
push es
mov ax, keep_psp
mov es, ax
cmp byte ptr es:[82h], '/'
jne checkunend
cmp byte ptr es:[83h], 'u'
jne checkunend
cmp byte ptr es:[84h], 'n'
jne checkunend
mov tounload, 1
checkunend:
pop es
pop ax
ret
CHECKTOUNLOAD ENDP

CHECKINTLOADED PROC
push bx
push si
push ax
mov ah, 35h
mov al, 1ch
int 21h
mov si, offset int_flag
sub si, offset MYINT
mov ax, es:[bx+si]
cmp ax, 1919h
jne checkintend
mov loaded, 1
checkintend:
pop ax
pop si
pop bx
ret
CHECKINTLOADED ENDP

LOADINT PROC
push ax
push bx
```

```
push cx
push dx
push es

mov ah, 35h
mov al, 1ch
int 21h
mov keep_cs, es
mov keep_ip, bx

push ds
mov dx, offset MYINT
mov ax, seg MYINT
mov ds, ax
mov ah, 25h
mov al, 1ch
int 21h
pop ds

mov dx, offset endmyint
add dx, 10fh
mov cl, 4
shr dx, cl
inc dx
xor ax, ax
mov ah, 31h
int 21h

pop es
pop dx
pop cx
pop bx
pop ax
ret
LOADINT ENDP

UNLOADINT PROC
cli
push ax
push bx
push dx
push es
push si

mov ah, 35h
mov al, 1ch
int 21h
mov si, offset keep_cs
sub si, offset MYINT
```

```
mov ax, es:[bx+si]
mov dx, es:[bx+si+2]

push ds
mov ds, ax
mov ah, 25h
mov al, 1ch
int 21h
pop ds

mov es, es:[bx+si+4]
push es
mov es, es:[2ch]
mov ah,49h
int 21h
pop es
mov ah, 49h
int 21h

pop si
pop es
pop dx
pop bx
pop ax
sti
ret
UNLOADINT ENDP
BEGIN PROC
mov ax, DATA
mov ds, ax
mov keep_psp, es
call CHECKINTLOADED
call CHECKTOUNLOAD
cmp tounload, 1
je unload
cmp loaded, 1
jne load
mov dx, offset intexist
mov ah, 09h
int 21h
jmp endlr
unload:
cmp loaded, 1
jne nothingtounload
call UNLOADINT
mov dx, offset intunloaded
mov ah, 09h
int 21h
jmp endlr
```

```
nothingtounload:
mov dx, offset intnotexist
mov ah, 09h
int 21h
jmp endlr
load:
mov dx, offset intloaded
mov ah, 09h
int 21h
call LOADINT
endlr:
xor AL,AL
mov AH,4Ch
int 21H
BEGIN      ENDP
CODE    ENDS

AStack SEGMENT STACK
'STACK'
DW 80h DUP(?)
AStack ENDS

DATA SEGMENT
loaded db 0
tounload db 0
intloaded db 'interrupt
has been loaded', 13, 10,
'$'
intexist db 'interrupt is
already loaded', 13, 10,
'$'
intunloaded db 'interrupt
has been unloaded', 13,
10, '$'
intnotexist db "interrupt
hasn't been loaded", 13,
10, '$'

DATA ENDS

END     BEGIN
```