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 _____ Черницын П.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.

```
C:\>LR4.EXE
interrupt has been loaded

C:\>LR4.EXE
interrupt is already loaded                                    Time 0050

C:\>LR4.EXE /un                                                Time 0142
interrupt has been unloaded

C:\>LR4.EXE /un
interrupt hasn't been loaded
```

Рисунок 1. Тестирование программы при различных состояниях

```
C:\>LR3_1.COM
Available memory:     648912
Extended memory size: 15360
MCB type: MS DOS;  MCB size:      16 bytes; MCB last 8 bytes:
MCB type: free;  MCB size:      64 bytes; MCB last 8 bytes:
MCB type: 0040;  MCB size:     256 bytes; MCB last 8 bytes:
MCB type: 0192;  MCB size:     144 bytes; MCB last 8 bytes:
MCB type: 0192;  MCB size: 648912 bytes; MCB last 8 bytes:LR3_1
```

Рисунок 2. Состояние памяти до загрузки прерывания

```
C:\>LR3_1.COM                                                  Time 0033
Available memory:     647760
Extended memory size: 15360
MCB type: MS DOS;  MCB size:      16 bytes; MCB last 8 bytes:
MCB type: free;  MCB size:      64 bytes; MCB last 8 bytes:
MCB type: 0040;  MCB size:     256 bytes; MCB last 8 bytes:
MCB type: 0192;  MCB size:     144 bytes; MCB last 8 bytes:
MCB type: 0192;  MCB size:     976 bytes; MCB last 8 bytes:LR4
MCB type: 01DA;  MCB size:     144 bytes; MCB last 8 bytes:
MCB type: 01DA;  MCB size: 647760 bytes; MCB last 8 bytes:LR3_1
```

Рисунок 3. Состояние памяти после загрузки прерывания

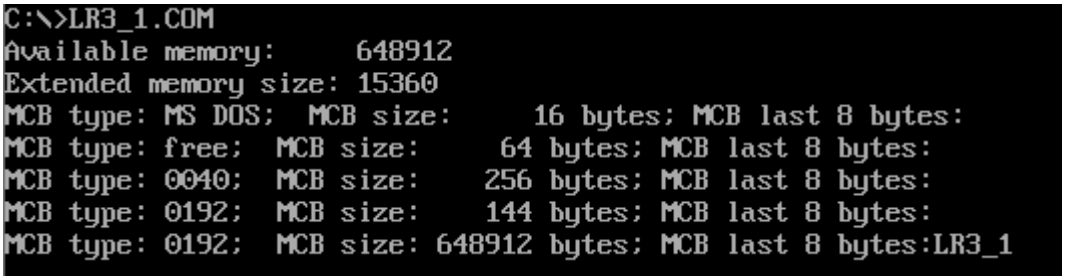

Рисунок 4. Состояние памяти после освобождения

Контрольные вопросы.

1. Как реализован механизм прерывания от часов?

Прерывание по таймеру вызывается каждые 55мс (примерно 18.2 раза в секунду). После вызова сохраняется содержимое регистров и определяется смещение в таблице векторов прерываний. Полученный адрес сохраняется в регистр CS:IP. После этого управление передается по этому адресу, т.е. выполняется запуск обработчика прерываний и происходит его выполнение. После выполнения происходит возврат управления прерванной программе.

2. Какого типа прерывания использовались в работе?

Аппаратное прерывание от системного таймера(1ch), а также программные прерывания 21h и 10h.

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

MY_INT PROC FAR
jmp my_int_begin
my_int_data:
keep_cs dw 0
keep_ip dw 0
keep_psp dw 0
keep_ax dw 0
keep_ss dw 0
keep_sp dw 0
my_int_flag dw 0BABAh
my_int_stack dw 100h dup(?)
count dw 0
count_msg db 'Time 0000'

my_int_begin:
mov keep_ss, ss
mov keep_sp, sp
mov keep_ax, ax
mov ax, seg my_int_stack
mov ss, ax
mov sp, offset my_int_stack
add sp, 120h

push ax
push bx
push cx
push dx
push si
push ds
mov ax, seg my_int_data
mov ds, ax

;count++ and cast to str
inc count
mov ax, count
xor dx, dx
mov si, offset count_msg
add si, 8
call WRD_TO_DEC

;save cursor
mov ah, 3h
mov bh, 0h
int 10h
push dx

;set cursor
mov ah, 02h
```

```
mov bh, 0h
mov dx, 1845h     ;dh = row, dl = col
int 10h

;print counter
push es
push bp
mov ax, seg count_msg
mov es, ax
mov bp, offset count_msg
mov ah, 13h
mov al, 1
mov bh, 0
mov bl, 10          ;pomenyat'
mov cx, 9
int 10h
pop bp
pop es

;reset cursor
pop dx
mov ah, 2h
mov bh, 0h
int 10h

pop ds
pop si
pop dx
pop cx
pop bx
pop ax

mov sp, keep_sp
mov ax, keep_ss
mov ss, ax
mov ax, keep_ax
mov al, 20h
out 20h, al
iret
MY_INT ENDP

WRD_TO_DEC proc near
;ax содержит исходное слово
;si адрес поля младшей цифры
push ax
push cx
push dx
mov cx,10
loop_wd:
div cx
or dl,30h
mov [si],dl
dec si
xor dx,dx
cmp ax,0
jne loop_wd
```

```
end_l1:
pop dx
pop cx
pop ax
ret
WRD_TO_DEC ENDP
MY_INT_END:

WriteMsg PROC near
push ax
mov ah,09h
int 21h
pop ax
ret
WriteMsg ENDP

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

CHECK_MY_INT_LOADED PROC
push ax
push si
; get int's segment
mov ah, 35h
mov al, 1ch
int 21h
; get signature's offset
mov si, offset my_int_flag
sub si, offset MY_INT
mov ax, es:[bx+si]
cmp ax, 0BABAh
jne check_load_end
mov load_flag, 1
check_load_end:
pop si
pop ax
ret
CHECK_MY_INT_LOADED ENDP

LOAD_MY_INT PROC
push ax
```

```
push bx
push es
push dx
push es
push cx

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

;set new int
push ds
mov dx, offset MY_INT
mov ax, seg MY_INT
mov ds, ax
mov ah, 25h
mov al, 1ch
int 21h
pop ds

;make resident
mov dx, offset MY_INT_END
add dx, 10fh
mov cl, 4
shr dx, cl
inc dx
xor ax, ax
mov ah, 31h
int 21h

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

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

;get int's seg
mov ah, 35h
mov al, 1ch
int 21h

;get int's data offset
```

```
mov si, offset keep_cs
sub si, offset MY_INT

mov ax, es:[bx+si]                      ;cs
mov dx, es:[bx+si+2]        ;ip

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

;free mem
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
UNLOAD_MY_INT ENDP

BEGIN PROC
mov ax, DATA
mov ds, ax
mov keep_psp, es
call CHECK_MY_INT_LOADED
call CHECK_MY_INT_UNLOADED
cmp unload_flag, 1
je unload
cmp load_flag, 0
je load
lea dx, int_exist_msg
call WriteMsg
jmp _end
unload:
cmp load_flag, 0
je not_exist
call UNLOAD_MY_INT
lea dx, int_unload_msg
call WriteMsg
jmp _end
not_exist:
lea dx, int_not_exist_msg
call WriteMsg
jmp _end
```

```
load:
lea dx, int_load_msg
call WriteMsg
call LOAD_MY_INT

_end:
xor al, al
mov ah, 4ch
int 21h
BEGIN ENDP

CODE ENDS

AStack SEGMENT STACK
DW 100h DUP(?)
AStack ENDS

DATA SEGMENT
load_flag                       db 0
unload_flag             db 0
int_load_msg            db 'interrupt
has been loaded',    13, 10, '$'
int_exist_msg           db 'interrupt is
already loaded',  13, 10, '$'
int_unload_msg          db 'interrupt
has been unloaded',  13, 10, '$'
int_not_exist_msg       db "interrupt
hasn't been loaded", 13, 10, '$'
DATA ENDS

END BEGIN
```