

МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №5

по дисциплине «Операционные системы»

**Тема: Сопряжение стандартного и пользовательского обработчиков
прерываний**

Студент гр. 8382

Черницын П.А.

Преподаватель

Ефремов М.А.

Санкт-Петербург

2020

Цель работы.

Исследование возможности встраивания пользовательского обработчика прерываний в стандартный обработчик от клавиатуры. Пользовательский обработчик прерывания получает управление по прерыванию (int 09h) при нажатии клавиши на клавиатуре. Он обрабатывает скан-код и осуществляет определенные действия, если скан-код совпадает с определенными кодами, которые он должен обрабатывать. Если скан-код не совпадает с этими кодами, то управление передается стандартному прерыванию.

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 09h
- 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.

Программа обрабатывает скан-код, полученный при нажатии клавиши на клавиатуре. Если полученная клавиша является клавишей f1, то она заменится знаком амперсанта &. На остальные скан-коды программа не реагирует.

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

```
C:\>LR5.EXE
interrupt has been loaded

C:\>LR5.EXE
interrupt is already loaded

C:\>qwe&&&rtty
Illegal command: qwe&&&rtty.

C:\>LR5.EXE /un
interrupt has been unloaded

C:\>LR5.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:\>LR5.EXE
interrupt has been loaded

C:\>LR3_1.COM
Available memory:      647808
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:     928 bytes; MCB last 8 bytes:LR5
MCB type: 01D7; MCB size:     144 bytes; MCB last 8 bytes:
MCB type: 01D7; MCB size: 647808 bytes; MCB last 8 bytes:LR3_1

```

Рисунок 3. Состояние памяти после загрузки прерывания

```

C:\>LR5.EXE /un
interrupt has been unloaded

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 4. Состояние памяти после освобождения

Контрольные вопросы.

1. Какого типа прерывания использовались в работе?

Аппаратные (09h) и программные (16h,21h) прерывания.

2. Чем отличается скан-код от кода ASCII?

Скан-код – код клавиши, позволяющий опознавать нажатые клавиши драйверу клавиатуры.

ASCII – это уникальный код для каждого символа.

Скан-код характеризует клавишу, а код ANSCII – символ.

Выводы.

В ходе работы была исследована возможность встраивания пользовательского обработчика прерываний в стандартный обработчик от клавиатуры.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ LR1COM.ASM

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

```
MY_INT PROC FAR
jmp my_int_begin
my_int_data:
keep_ip dw 0
keep_cs dw 0
keep_psp dw 0
keep_ax dw 0
keep_ss dw 0
keep_sp dw 0
my_int_flag dw 0BABAh
REQ_KEY db 3bh ;change
my_int_stack dw 100h dup(?)
```

```
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 cx
push es
mov ax, seg my_int_data
mov ds, ax
```

```
in al, 60h
cmp al, REQ_KEY
je do_req
pushf
call dword ptr cs:keep_ip
jmp int_end
do_req:
;hardware interrupt handling
push ax
in al, 61h
mov ah, al
or al, 80h
out 61h, al
xchg ah, al
out 61h, al
mov al, 20h
out 20h, al
pop ax
write_buff:
mov ah, 05h
mov cl, '&'
```

```

xor ch, ch
int 16h
or al, al
jnz skip
jmp int_end
skip:
;clear buff and try
mov ax, 0040h
mov es, ax
mov ax, es:[1ah]
mov es:[1ch], ax
jmp write_buff

```

```

int_end:
pop es
pop cx
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_END:
MY_INT ENDP

```

```

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, 09h
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, 09h
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, 09h
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, 09h
int 21h
```

```
;get int's data offset
mov si, offset keep_ip
sub si, offset MY_INT
```

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

```
push ds
mov ds, ax
mov ah, 25h
mov al, 09h
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

```