

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

Чирков С.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 цифрой (кроме нуля) – её код меняется на букву из словосочетания “hello world”. То есть при загруженном прерывании можно написать это словосочетание только клавишами цифр на клавиатуре. 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:\>hello world
Illegal command: hello.

C:\>lr5.exe
interrupt is already loaded

C:\>lr5.exe /un
interrupt has been unloaded

C:\>lr5.exe /un
interrupt hasn't been loaded

C:\>12334 56789
Illegal command: 12334.
```

Рисунок 1. Тестирование программы при различных состояниях

```
C:\>lr3_1.com
Available memory - 648912 B.
Extended memory - 15360 B.
Type - 4D Sector - MS DOS Size - 16 B. Last 8 bytes -
Type - 4D Sector - free Size - 64 B. Last 8 bytes -
Type - 4D Sector - 0040 Size - 256 B. Last 8 bytes -
Type - 4D Sector - 0192 Size - 144 B. Last 8 bytes -
Type - 5A Sector - 0192 Size - 648912 B. Last 8 bytes - LR3_1
```

Рисунок 2. Состояние памяти до загрузки прерывания

```

C:\>lr3_1.com
Available memory - 647952 B.
Extended memory - 15360 B.
Type - 4D Sector - MS DOS Size -      16 B. Last 8 bytes -
Type - 4D Sector - free Size -       64 B. Last 8 bytes -
Type - 4D Sector - 0040 Size -      256 B. Last 8 bytes -
Type - 4D Sector - 0192 Size -      144 B. Last 8 bytes -
Type - 4D Sector - 0192 Size -      784 B. Last 8 bytes - LR5
Type - 4D Sector - 01CE Size -      144 B. Last 8 bytes -
Type - 5A Sector - 01CE Size - 647952 B. 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 B.
Extended memory - 15360 B.
Type - 4D Sector - MS DOS Size -      16 B. Last 8 bytes -
Type - 4D Sector - free Size -       64 B. Last 8 bytes -
Type - 4D Sector - 0040 Size -      256 B. Last 8 bytes -
Type - 4D Sector - 0192 Size -      144 B. Last 8 bytes -
Type - 5A Sector - 0192 Size - 648912 B. 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, присвоенный каждой клавише, с помощью которого драйвер клавиатуры распознает, какая клавиша была нажата. Скан-коды жёстко привязаны к каждой клавише на аппаратном уровне и не зависят от их состояния.

ASCII – название таблицы (кодировки, набора), в которой некоторым распространённым печатным и непечатным символам сопоставлены числовые коды. В таблице ASCII намного больше символов, чем клавиш на клавиатуре, вследствие чего используются скан-коды.

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
```

```
MYINT PROC FAR
jmp myintcode
intdata:
key db 0
int_flag dw 1919h
keep_ip dw 0
keep_cs 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

in al, 60h
cmp al, 2
je one
cmp al, 3
je two
cmp al, 4
```

```
je three
cmp al, 5
je four
cmp al, 6
je five
cmp al, 7
je six
cmp al, 8
je seven
cmp al, 9
je eight
cmp al, 0ah
je nine
pushf
call dword ptr cs:keep_ip
jmp endint
one:
mov key, 'h'
jmp secure
two:
mov key, 'e'
jmp secure
three:
mov key, 'l'
jmp secure
four:
mov key, 'o'
jmp secure
five:
mov key, 'w'
jmp secure
six:
mov key, 'o'
jmp secure
seven:
mov key, 'r'
jmp secure
eight:
mov key, 'l'
jmp secure
nine:
mov key, 'd'
```

```

secure:
in al, 61h
mov ah, al
or al, 80h
out 61h, al
xchg al, al
out 61h, al
mov al, 20h
out 20h, al
buffer:
mov ah, 5
mov cl, key
mov ch, 0
int 16h
or al, al
jz endint
push es
mov ax, 0040h
mov es, ax
mov ax, es:[1ah]
mov es:[1ch], ax
pop es
jmp buffer
endint:
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

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, 09h
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, 09h
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, 09h
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, 09h
int 21h
mov si, offset keep_ip
sub si, offset MYINT
mov dx, es:[bx+si]
mov ax, es:[bx+si+2]

```

```

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