REF:

https://stackoverflow.com/questions/37618111/keyboard-irq-within-an-x86-kernel

Se also:

<https://alex.dzyoba.com/blog/os-interrupts/>

# [Keyboard IRQ within an x86 kernel](https://stackoverflow.com/questions/37618111/keyboard-irq-within-an-x86-kernel)

I'm trying to program a very simple kernel for learning purposes. After reading a bunch of articles about the PIC and IRQs in the x86 architecture, I've figured out that IRQ1 is the keyboard handler. I'm using the following code to print the keys being pressed:

// ….

## **Sample Code**

You didn't provide a complete example, so I filled in some of the gaps (including a simple keyboard map) and slight change to your keyboard handler. The revised keyboard handler only displays key down events and skips over characters that had no mapping. In all cases the code drops through to the end of the handler so that the PIC is sent an EOI (End Of Interrupt). The current cursor location is a static integer that will retain its value across interrupt calls. This allows the position to advance between each character press.

My kprintd.h file is empty, and I put ALL the assembler prototypes into your port\_io.h. The prototypes should be divided properly into multiple headers. I only did it this way to reduce the number of files. My file lowlevel.asm defines all the low level assembly routines. The final code is as follows:

kernel.asm:

bits 32

section .text

;grub bootloader header

align 4

dd 0x1BADB002 ;magic

dd 0x00 ;flags

dd - (0x1BADB002 + 0x00) ;checksum. m+f+c should be zero

global start

extern kmain

start:

lgdt [gdtr] ; Load our own GDT, the GDTR of Grub may be invalid

jmp CODE32\_SEL:.setcs ; Set CS to our 32-bit flat code selector

.setcs:

mov ax, DATA32\_SEL ; Setup the segment registers with our flat data selector

mov ds, ax

mov es, ax

mov fs, ax

mov gs, ax

mov ss, ax

mov esp, stack\_space ; set stack pointer

call kmain

; If we get here just enter an infinite loop

endloop:

hlt ; halt the CPU

jmp endloop

; Macro to build a GDT descriptor entry

%define MAKE\_GDT\_DESC(base, limit, access, flags) \

(((base & 0x00FFFFFF) << 16) | \

((base & 0xFF000000) << 32) | \

(limit & 0x0000FFFF) | \

((limit & 0x000F0000) << 32) | \

((access & 0xFF) << 40) | \

((flags & 0x0F) << 52))

section .data

align 4

gdt\_start:

dq MAKE\_GDT\_DESC(0, 0, 0, 0); null descriptor

gdt32\_code:

dq MAKE\_GDT\_DESC(0, 0x00ffffff, 10011010b, 1100b)

; 32-bit code, 4kb gran, limit 0xffffffff bytes, base=0

gdt32\_data:

dq MAKE\_GDT\_DESC(0, 0x00ffffff, 10010010b, 1100b)

; 32-bit data, 4kb gran, limit 0xffffffff bytes, base=0

end\_of\_gdt:

gdtr:

dw end\_of\_gdt - gdt\_start - 1

; limit (Size of GDT - 1)

dd gdt\_start ; base of GDT

CODE32\_SEL equ gdt32\_code - gdt\_start

DATA32\_SEL equ gdt32\_data - gdt\_start

section .bss

resb 8192 ; 8KB for stack

stack\_space:

lowlevel.asm:

section .text

extern keyboard\_handler

global read\_port

global write\_port

global load\_idt

global keyboard\_handler\_int

keyboard\_handler\_int:

pushad

cld

call keyboard\_handler

popad

iretd

load\_idt:

mov edx, [esp + 4]

lidt [edx]

sti

ret

; arg: int, port number.

read\_port:

mov edx, [esp + 4]

in al, dx

ret

; arg: int, (dx)port number

; int, (al)value to write

write\_port:

mov edx, [esp + 4]

mov al, [esp + 4 + 4]

out dx, al

ret

port\_io.h:

extern unsigned char read\_port (int port);

extern void write\_port (int port, unsigned char val);

extern void kb\_init(void);

kprintf.h:

/\* Empty file \*/

keyboard\_map.h:

unsigned char keyboard\_map[128] =

{

0, 27, '1', '2', '3', '4', '5', '6', '7', '8', /\* 9 \*/

'9', '0', '-', '=', '\b', /\* Backspace \*/

'\t', /\* Tab \*/

'q', 'w', 'e', 'r', /\* 19 \*/

't', 'y', 'u', 'i', 'o', 'p', '[', ']', '\n', /\* Enter key \*/

0, /\* 29 - Control \*/

'a', 's', 'd', 'f', 'g', 'h', 'j', 'k', 'l', ';', /\* 39 \*/

'\'', '`', 0, /\* Left shift \*/

'\\', 'z', 'x', 'c', 'v', 'b', 'n', /\* 49 \*/

'm', ',', '.', '/', 0, /\* Right shift \*/

'\*',

0, /\* Alt \*/

' ', /\* Space bar \*/

0, /\* Caps lock \*/

0, /\* 59 - F1 key ... > \*/

0, 0, 0, 0, 0, 0, 0, 0,

0, /\* < ... F10 \*/

0, /\* 69 - Num lock\*/

0, /\* Scroll Lock \*/

0, /\* Home key \*/

0, /\* Up Arrow \*/

0, /\* Page Up \*/

'-',

0, /\* Left Arrow \*/

0,

0, /\* Right Arrow \*/

'+',

0, /\* 79 - End key\*/

0, /\* Down Arrow \*/

0, /\* Page Down \*/

0, /\* Insert Key \*/

0, /\* Delete Key \*/

0, 0, 0,

0, /\* F11 Key \*/

0, /\* F12 Key \*/

0, /\* All other keys are undefined \*/

};

keyb.c:

#include "kprintf.h"

#include "port\_io.h"

#include "keyboard\_map.h"

void kb\_init(void)

{

/\* This is a very basic keyboard initialization. The assumption is we have a

\* PS/2 keyboard and it is already in a proper state. This may not be the case

\* on real hardware. We simply enable the keyboard interupt \*/

/\* Get current master PIC interrupt mask \*/

unsigned char curmask\_master = read\_port (0x21);

/\* 0xFD is 11111101 - enables only IRQ1 (keyboard) on master pic

by clearing bit 1. bit is clear for enabled and bit is set for disabled \*/

write\_port(0x21, curmask\_master & 0xFD);

}

/\* Maintain a global location for the current video memory to write to \*/

static int current\_loc = 0;

/\* Video memory starts at 0xb8000. Make it a constant pointer to

characters as this can improve compiler optimization since it

is a hint that the value of the pointer won't change \*/

static char \*const vidptr = (char\*)0xb8000;

void keyboard\_handler(void)

{

signed char keycode;

keycode = read\_port(0x60);

/\* Only print characters on keydown event that have

\* a non-zero mapping \*/

if(keycode >= 0 && keyboard\_map[keycode]) {

vidptr[current\_loc++] = keyboard\_map[keycode];

/\* Attribute 0x07 is white on black characters \*/

vidptr[current\_loc++] = 0x07;

}

/\* Send End of Interrupt (EOI) to master PIC \*/

write\_port(0x20, 0x20);

}

main.c:

#include "port\_io.h"

#define IDT\_SIZE 256

#define PIC\_1\_CTRL 0x20

#define PIC\_2\_CTRL 0xA0

#define PIC\_1\_DATA 0x21

#define PIC\_2\_DATA 0xA1

void keyboard\_handler\_int();

void load\_idt(void\*);

struct idt\_entry

{

unsigned short int offset\_lowerbits;

unsigned short int selector;

unsigned char zero;

unsigned char flags;

unsigned short int offset\_higherbits;

} \_\_attribute\_\_((packed));

struct idt\_pointer

{

unsigned short limit;

unsigned int base;

} \_\_attribute\_\_((packed));

struct idt\_entry idt\_table[IDT\_SIZE];

struct idt\_pointer idt\_ptr;

void load\_idt\_entry(int isr\_number, unsigned long base, short int selector, unsigned char flags)

{

idt\_table[isr\_number].offset\_lowerbits = base & 0xFFFF;

idt\_table[isr\_number].offset\_higherbits = (base >> 16) & 0xFFFF;

idt\_table[isr\_number].selector = selector;

idt\_table[isr\_number].flags = flags;

idt\_table[isr\_number].zero = 0;

}

static void initialize\_idt\_pointer()

{

idt\_ptr.limit = (sizeof(struct idt\_entry) \* IDT\_SIZE) - 1;

idt\_ptr.base = (unsigned int)&idt\_table;

}

static void initialize\_pic()

{

/\* ICW1 - begin initialization \*/

write\_port(PIC\_1\_CTRL, 0x11);

write\_port(PIC\_2\_CTRL, 0x11);

/\* ICW2 - remap offset address of idt\_table \*/

/\*

\* In x86 protected mode, we have to remap the PICs beyond 0x20 because

\* Intel have designated the first 32 interrupts as "reserved" for cpu exceptions

\*/

write\_port(PIC\_1\_DATA, 0x20);

write\_port(PIC\_2\_DATA, 0x28);

/\* ICW3 - setup cascading \*/

write\_port(PIC\_1\_DATA, 0x00);

write\_port(PIC\_2\_DATA, 0x00);

/\* ICW4 - environment info \*/

write\_port(PIC\_1\_DATA, 0x01);

write\_port(PIC\_2\_DATA, 0x01);

/\* Initialization finished \*/

/\* mask interrupts \*/

write\_port(0x21 , 0xff);

write\_port(0xA1 , 0xff);

}

void idt\_init()

{

initialize\_pic();

initialize\_idt\_pointer();

load\_idt(&idt\_ptr);

}

void kmain(void)

{

//Using grub bootloader..

idt\_init();

load\_idt\_entry(0x21, (unsigned long) keyboard\_handler\_int, 0x08, 0x8e);

kb\_init();

while(1) \_\_asm\_\_("hlt\n\t");

}

In order to link this kernel I use a file link.ld with this definition:

/\*

\* link.ld

\*/

OUTPUT\_FORMAT(elf32-i386)

ENTRY(start)

SECTIONS

{

. = 0x100000;

.text : { \*(.text) }

.rodata : { \*(.rodata) }

.data : { \*(.data) }

.bss : { \*(.bss) }

}

I compile and link this code using a *GCC* i686 [cross compiler](http://wiki.osdev.org/GCC_Cross-Compiler) with these commands:

nasm -f elf32 -g -F dwarf kernel.asm -o kernel.o

nasm -f elf32 -g -F dwarf lowlevel.asm -o lowlevel.o

i686-elf-gcc -g -m32 -c main.c -o main.o -ffreestanding -O3 -Wall -Wextra -pedantic

i686-elf-gcc -g -m32 -c keyb.c -o keyb.o -ffreestanding -O3 -Wall -Wextra -pedantic

i686-elf-gcc -g -m32 -Wl,--build-id=none -T link.ld -o kernel.elf -ffreestanding -nostdlib lowlevel.o main.o keyb.o kernel.o -lgcc

The result is a kernel called kernel.elf with debug information. I prefer an optimization level of -O3 rather than a default of -O0. Debug information makes it easier to debug with *QEMU* and *GDB*. The kernel can be debugged with these commands:

qemu-system-i386 -kernel kernel.elf -S -s &

gdb kernel.elf \

-ex 'target remote localhost:1234' \

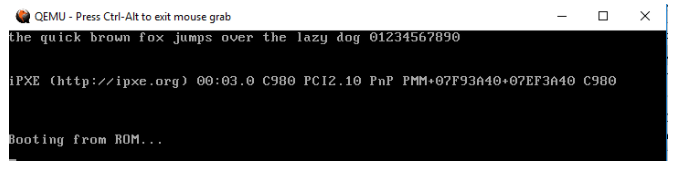
-ex 'layout src' \

-ex 'layout regs' \

-ex 'break kmain' \

-ex 'continue'

If you wish to debug at the assembly code level replace layout src with layout asm. When run with the input the quick brown fox jumps over the lazy dog 01234567890 *QEMU* displayed this:



[Ask Question](https://stackoverflow.com/questions/ask)