I Cant Get Interrupts Working

From OSDev Wiki

This page is a sort of TroubleShooting manual to help you getting through common interrupts framework problems encountered by guests and members of the forum

Make sure you collected enough information about your own situation (for instance running your code in Bochs).

Contents

- 1 ISR problems
  - 1.1 My handler doesn't get called!? (ASM)
  - 1.2 My Handler doesn't get called (C) !?
  - 1.3 My handler is called but it doesn't return !?
- 2 IRQ problems
  - 2.1 I'm receiving EXC9 instead of IRQ1 when striking a key ?!
  - 2.2 I'm receiving a double fault after enabling interrupts
  - 2.3 I'm not receiving any IRQ
  - 2.4 I can only receive one IRQ
  - 2.5 When I try to enable the PIT, the keyboard doesn't work anymore
  - 2.6 I keep getting an IRQ7 for no apparent reason
  - 2.7 what does "shift operator may only be applied to scalar values" mean ?
- 3 Assembly Examples
  - 3.1 NASM
  - 3.2 GNU Assembler
- 4 Problems with IDTs
- 5 Problems
- 6 IDT problems in Assembly
  - 6.1 FASM notice
- 7 See also

ISR problems

My handler doesn't get called!? (ASM)

For this test, you need to call the interrupt yourself, by software. Don't try to get IRQ handled right from the start before you're sure your IDT setup is correct. You need to have:

- your IDT loaded and filled properly.
- your IDT's *linear* address loaded in a structure together with the table's size (in bytes, iirc). Be especially cautious if you have a Higher Half Kernel design or did not set up identity paging.
- a valid Code selector and offset in the descriptor, proper type, etc.
- a handling code at the defined offset.

see test code below

My Handler doesn't get called (C) !?

If you're programming the IDT setup in C, make sure the IDTR structure has been correctly understood by your compiler. As Intel's 6 bytes structures infringe most compiler's packing rules, you'll need to use either *bitfields* or *packing pragmas*. Use `sizeof()` and `OFFSETOF()` macros to make sure the expected definition is used (a runtime test would be fine)

My handler is called but it doesn't return !?

Try to run it in the BOCHS and see if you get any exception report. Program all your exception to have the same kind of behavior as the example, but displaying a character indicating the fault. Exceptions occurring at the end of an interrupt handler are usually due to a wrong stack operation within the handler.

- don't try to return from an exception (unless you solved its cause). Returning from a division by zero, for instance, makes no sense at all
- pops everything you push, but no more
- make sure you didn't forget the CPU-pushed error code (for exceptions 8,10 and 14 at least)
- make sure your handler doesn't trash unexpected registers. For exceptions and hardware IRQ handlers, no registers *at all* should be modified.

Another common source of error at this point comes from misimplementation of ISR in C. Check the InterruptServiceRoutines page for enlightenment ...

IRQ problems

Now that you're sure an interrupt can be called and can return, you're ready to enable hardware interrupts. As a first step, you're suggested to enable the _keyboard handler only_, as you'll have almost complete control of what it does. Use the mask feature of the PIC to enable/disable some handlers.

```
    outb(0x21,0xfd);
    outb(0xa1,0xff);
    enable(); // asm("sti");
```

I'm receiving EXC9 instead of IRQ1 when striking a key ?!

You missed the PIC vector reprogramming step. Check Can I remap the PIC? page. Note that if you remap the PIC vectors out of the IDT you'll get a GPF exception instead of any interrupt.

I'm receiving a double fault after enabling interrupts

Different symptom for the same error as above. This time caused by an timer interrupt calling vector 8.

I'm not receiving any IRQ

Make sure you receive software interrupts first. Also make sure you enabled the IRQ of your interest on the PIC mask and that you enabled the cascading line (bit #2 of the master) if you're waiting for a slave IRQ.

I can only receive one IRQ

Each IRQ needs to be acknowledged to the PIC manually by sending an EOI. You need to have

```
outb(0x20,0x20)
```

within any master handler and any

```
outb(0x20,0x20); outb(0xa0,0x20);
```

within any slave handler.

When I try to enable the PIT, the keyboard doesn't work anymore

A common mistake is that people reload the mask with `0xFE` when they want to add timer, but doing this actually enables *only* the timer and disables the keyboard (bit #1 of 0xFE is set!) The correct value for enabling *both* keyboard and timer is 0xFC.

I keep getting an IRQ7 for no apparent reason

This is a known problem that cannot be prevented from happening, although there is a workaround. When any IRQ7 is received, simply read the In-Service Register

```
outb(0x20, 0x0B); unsigned char irr = inb(0x20);
```

and check if bit 7

```
irr & 0x80
```

is set. If it isn't, then return from the interrupt without sending an EOI.

For more information, including a more detailed explanation, see Brendan's post in this thread.

what does "shift operator may only be applied to scalar values" mean ?

You're trying to load a 16-bits field (a part of the IDT descriptor) with a reference to a 32-bit label that is subject to relocation. Try to replace

```
isr_label:
```

```
    iret
bad_stuff dw isr_label & 0xFFFF
          dw 0xdead
          dw 0xbeef
          dw isr_label >> 16
```

by something that extracts a 'pure value' from the address (e.g. the difference of two addresses are a pure value and `$$` means to NASM the start of the section)

```
 %define BASE_OF_SECTION SOME_CONSTANT_YOU_SHOULD_KNOW
isr_label:
   iret
 good_stuff dw (BASE_OF_SECTION + isr_label - $$) & 0xFFFF
            dw 0xcafe
            dw 0xbabe
            dw (BASE_OF_SECTION + isr_label - $$) >> 16
```

The role of

```
BASE_OF_SECTION
```

is to adjust the pure offset to the real situation (usually as defined in your linker script), e.g. if your kernel get loaded at 1MB, you'll set it to 0x100000 to keep the CPU happy.

Assembly Examples

NASM

This example is made for x86 CPUs running in IA32 mode (32-bit).

```
int_handler:
   mov ax, LINEAR_DATA_SELECTOR
   mov gs, ax
   mov dword [gs:0xB8000],') : '
   hlt

idt:
   resd 50*2

idtr:
   dw (50*8)-1
   dd LINEAR_ADDRESS(idt)

test1:
   lidt [idtr]
   mov eax,int_handler
   mov [idt+49*8],ax
   mov word [idt+49*8+2],CODE_SELECTOR
```

```
    mov word [idt+49*8+4],0x8E00
    shr eax,16
    mov [idt+49*8+6],ax
    int 49
```

should display a smiley on the top-left corner ... then the CPU is halted indefinitely.

GNU Assembler

This example sets up an interrupt handler in long mode.

```
.text
int_handler:
    movq $0x123abc, 0x0 // this places magic value "0x123abc" at the begi
    hlt

.p2align 4
idt:
    .skip 50*16

  idtr:
    .short (50*16)-1
    .quad idt

.globl do_test
do_test:
    lidt idtr
    movq $int_handler, %rax
    mov %ax, idt+49*16
    movw $0x20, idt+49*16+2 // replace 0x20 with your code section selct
    movw $0x8e00, idt+49*16+4
    shr $16, %rax
    mov %ax, idt+49*16+6
    shr $16, %rax
    mov %rax, idt+49*16+8
    int $49
```

This example differs from the previous one: it will not touch the screen, but will write the value "0x123abc" to 0x0 memory address and halt. It may be useful when there's no screen or BIOS available.

Problems with IDTs

Many of us while OS dev'ing will encounter a problem with IDT's. Here are some solved problems with IDT's

This is for solved problems. The unsolved ones can be found here on the Forum (http://forum.osdev.org/viewtopic.php?f=1&t=24805)

Problems

Please post ***Completed*** problems here.

First of all, check your GDT. Keep in mind padding issues. In C this goes like:

```
// GCC
struct IDT_reg {
 //struct here
} __attribute__((packed));

struct GDT_reg {
 //struct here
} __attribute__((packed));

// Visual C++
#pragma pack(push, 1)
struct IDT_reg {
 //struct here
};

struct GDT_reg {
 //struct here
};
#pragma pack(pop)
```

IDT problems in Assembly

Make sure the structre is correct and you are using linear addresses.

FASM notice

Since fasm doesn't accept the normal way as described above, I will describe it. Fasm does, however, support shl and shr, so to describe the higher part of an ISR address, we just use *label shl 0x10* where label is the name of the ISR. To define the higher part, we need to write a little more, since fasm use 64 bit, before compiling. This means that IF we just shl and shr, it will be that same as before. This is how we are supposed to do: (label shl 0x30) shr 0x30 Here is a little example, so you can see how it works:

```
idt:
 dw  ((isr1 shl 0x30) shr 0x30)        ; the low part of the address
 dw   0x8    ; selector
 db   0
 db   010001110b  ; type
 dw (isr1 shr 0x10) the high part of the address

isr1:
 mov ax,0xdead
```

See also

- IDT
- IDT problems

Retrieved from "http://wiki.osdev.org/index.php?title=I_Cant_Get_Interrupts_Working&oldid=17029"
Categories: Troubleshooting | FAQ | Interrupts
