Switching to User-mode using iret

I am writing a small OS that will execute some code in user mode (privilege level 3). From that user level code, I want to call an interrupt back to the OS that prints a message. Right now I don't really care how my interrupt handler takes arguments or anything like that, I really just want an interrupt handler to inform me (the user) that the code has executed.
My question is: how do I run code in user mode? I have a function that sets up a Local Descriptor Table with a code segment and data segment (both with user mode privileges). What I dont understand is how I am supposed to load these segments into cs, ss, and ds. I successfully load the my LDT, but I do not know how to actually use it. I have heard that I should use iret, but I don't understand exactly how.
Another question that I have is how my interrupt handler should work. Let's say I install an interrupt handler for vector number 0x40, which I want to print "hello, user mode!". I know how to setup an interrupt handler, but I don't exactly understand how the context will be switched when entering a kernel interrupt handler from user mode. I know that the cs register must change, since my routine will be running from the code segment specified in my IDT entry. I also understand that the stack selector probably changes as well, but I cannot be certain of this.
Could someone please explain to me what context changes are made when an interrupt gate is called?

asked Jul 31 '11 at 21:25

Alex Nichol

2 Answers

Getting to ring 3 can be done using iret because the way it works has been documented. When you receive an interrupt, the processor pushes:

1. The stack segment and pointer (ss:esp), as 4 words
2. EFLAGS
3. The return code segment and instruction pointer (cs:eip), as 4 words
4. An error code, if required.

iret works by undoing steps 1-3 (The ISR is responsible for undoing step 4 if necessary). We can use this fact to get to ring 3 by pushing the required information to the stack and issuing an iretinstruction. Make sure you have the proper CPL in your code and stack segments (the low two bits should be set in each). However, iret doesn't change any of the data segments, so you will need to change them manually. You use the mov instruction to do this, but you will not be able to read data outside the stack between doing this and switching rings.

```
cli
mov   ax, Ring3_DS
mov   ds, eax
push  dword Ring3_SS
push  dword Ring3_ESP
pushfd
or    dword [esp], 0x200 // Set IF in EFLAGS so that interrupts will be reenabled in user
mode
push  dword Ring3_CS
push  dword Ring3_EIP
iret
```

If you want a complete, working example, see this tutorial.

When an interrupt is issued, the processor reads your IDT to get the proper code segment and instruction pointer for the ISR. It then looks at your TSS to find the new stack segment and pointer. It changes ss and esp appropriately, and then pushes the old values to the new stack. It does *not*change any of the data segment registers. You must do this manually if you need to access memory in your ISR.

answered Jul 31 '11 at 23:08

ughoavgfhw

1 *Alright, you answered my question and MORE! What I am now understanding is that I will need to have a TSS that contains my kernel stack, etc. Thank you for the help. – Alex Nichol Aug 1 '11 at 0:39*

I logged in just to upvote this! Thanks! Also for anyone else out there, I recommend osdev, and the Intel ISA manuals as well. They really clear things up. – Sid Nov 13 '13 at 7:17

You can also do a retf. A far return to a less privileged code segment will cause the new ss and sp to be popped off of the privileged stack. Just make sure that you do far returns for far calls and irets for interrupts. The only difference between them is the presence of flags on the stack, but it's wise not ot mix them up.
Also, don't forget that exceptions sometimes push error codes on the stack.

answered yesterday

Raymond Jennings