Tutorial: Entering User mode 4 months ago

by Aaron

If you are creating your own operating system or are interested in how to get from Ring 0 (Kernel mode) to Ring 3 (user mode) then the following tutorial is for you.

Required Knowledge

In order to use this tutorial a knowledge of operating systems is assumed, in addition to this your kernel will need to have a working GDT, IDT, and a video driver of some description.

Whats covered?

This tutorial covers getting to user mode (ring 3) but does not yet cover system calls so you will only be able to switch to ring 3 (user mode) but you will not be able to switch back, in addition to ring switching we will cover creating and installing a TSS.

Creating the TSS (Task State Segment)

First off we need to create a TSS (Task State-Segment) this is just a special entry in the GDT that allows the CPU to jump back to ring 0, for now you can just use the TSS bellow if you donâ€™t have one.

```
void install_tss(int cpu_no){
    // now fill each value
    // set values necessary
    sys_tss.ss0 = 0x10;
              // now set the IO bitmap (not necessary, so set above limit)

    sys_tss.iomap = ( unsigned short ) sizeof( tss_struct );
}
```

And the TSS structure that you will need is: (this should go in a header file e.g. tss.h)

```
typedef volatile struct strtss{
    unsigned short   link;
    unsigned short   link_h;
    unsigned long   esp0;
    unsigned short   ss0;
    unsigned short   ss0_h;
    unsigned long   esp1;
    unsigned short   ss1;
    unsigned short   ss1_h;
    unsigned long   esp2;
    unsigned short   ss2;
    unsigned short   ss2_h;
    unsigned long   cr3;
    unsigned long   eip;
    unsigned long   eflags;
    unsigned long   eax;
    unsigned long   ecx;
    unsigned long   edx;
    unsigned long    ebx;
    unsigned long   esp;
    unsigned long   ebp;
    unsigned long   esi;
    unsigned long   edi;
    unsigned short   es;
    unsigned short   es_h;
    unsigned short   cs;
    unsigned short   cs_h;
    unsigned short   ss;
    unsigned short   ss_h;
    unsigned short   ds;
    unsigned short   ds_h;
    unsigned short   fs;
    unsigned short   fs_h;
    unsigned short   gs;
    unsigned short   gs_h;
    unsigned short   ldt;
    unsigned short   ldt_h;
    unsigned short   trap;
    unsigned short   iomap;
}__attribute__((packed)) tss_struct;

tss_struct sys_tss; //Define the TSS as a global structure
```

You will then need to set up some user mode GDT code segment and data segment like so:

```
gdt_set_gate(3, 0, 0xFFFFFFFF, 0xFA, 0xCF); // User mode code segment gdt_set
_gate(4, 0, 0xFFFFFFFF, 0xF2, 0xCF); // User mode data segment
```

Then set the GDT entry for the TSS:

```
unsigned long addr=(unsigned long)tss;
int size = sizeof(tss_struct)+1;  gdt_set_gate(5,addr,addr+size,0x89,0xCF);
```

Getting to user mode

Now you should have a valid TSS and a running kernel so we can now make the jump to User Mode.

```
void switch_to_user_mode() {
    // Set up a stack structure for switching to user mode.
    asm volatile("  \
        cli; \
        mov $0x23, %ax; \
        mov %ax, %ds; \
        mov %ax, %es; \
        mov %ax, %fs; \
        mov %ax, %gs; \
              \
        mov %esp, %eax; \
        pushl $0x23; \
        pushl %eax; \
        pushf; \
        mov $0x200, %eax; \
        push %eax; \
        pushl $0x1B; \
        push $1f; \
        iret; \    1: \
    ");
}
```

All this code does is setup the CPU for jumping into user mode and then jump to the address at the end of this code, once it has done that you are in user mode! However any interrupts will cause a exception in you kernel at some level depending on how its setup.

Whats Next?

You may wish to add some systemcalls so that your user mode code can place some text on the screen or even take user input.