Software Task-Switching

From alt.os.development

All lines preceded by a ">" were written by *sefirot*. All other lines were written by *Kovacs Viktor Peter*.

```
> hello,
> would you explain the way of software task-switching
> ( namely, switching without using tss ) on x86 in detail ?

It's simple:
  -push all regs to the stack
  -load a new value into ss:esp (switch kernel stacks)
  -pop all regs from the new stack
  -load a new value into cr3 (pd base reg)
  -load a new ss:esp value into the system tss (patch it)
  -iret (when the task switcher is placed into an interrupt handler)

 OR even simplier: (faster; only for microkernels)
  -save all regs to the thread data struct
  -load all regs from the new thread data struct
  -reload cr3 (on process switches only)
  -iret

    Viktor

ps1:
 It can be as fast as 64 mov-s in case of a thread switch.
 (about 32 cycles on a P6 core /Ppro-PIII/)

 The 1st method modifies the tss. The x86 reads the ss0:esp values from
 the tss on a ring3->ring0 switch. This is the only required field.
 (Using one kernel stack per cpu saves the patch and the reload, but makes
 the kernel ring0 code uninterruptable. Actually... who wants to
 interrupt the task switcher in a critical section?)

ps2: the code in unoptimized inline gcc assembly:


asm("_kernel_lbl_int_00:                ");
asm("    pushl   $0                     ");
asm("    jmp     _kernel_lbl_to_kernel  ");
[...]
asm("_kernel_lbl_int_40:                ");
asm("    pushl   $0                     "); /* filler for the err code */
asm("    pushl   $64                    "); /* irq number */
asm("    jmp     _kernel_lbl_to_kernel  ");
[...]
asm("_kernel_lbl_to_kernel:             ");
asm("    pushl   %gs            "); /* could be optimized */
asm("    pushl   %fs                    "); /* to one instr. */
asm("    pushl   %es            ");
asm("    pushl   %ds            ");
asm("    pushl   %ebp           ");
asm("    pushl   %edi           ");
asm("    pushl   %esi           ");
asm("    pushl   %edx           ");
asm("    pushl   %ecx           ");
asm("    pushl   %ebx           ");
asm("    pushl   %eax           ");

asm("    movl    %esp, %ebx             ");
asm("    movl    $_kernel_stack_top, %esp  "); /* single kstack mode */
asm("    pushl   %ebx           ");
```

```
asm("    movw    $16, %ax               "); /* KERNEL_DATA_SEG!!!! */
asm("    movw    %ax, %ds               ");
asm("    movw    %ax, %es               ");
asm("    movw    %ax, %fs               ");
asm("    movw    %ax, %gs               ");
asm("    call    _kernel_entry          "); /* => eax: retval */
asm("    popl    %ebx           ");
asm("    movl    %ebx, %esp             ");

asm("    popl    %eax           ");
asm("    popl    %ebx           ");
asm("    popl    %ecx           ");
asm("    popl    %edx           ");
asm("    popl    %esi           ");
asm("    popl    %edi           ");
asm("    popl    %ebp           ");
asm("    popl    %ds                    ");
asm("    popl    %es                    ");
asm("    popl    %fs                    ");
asm("    popl    %gs                    ");
asm("    addl    $8, %esp               "); /* drop err and irq num */
asm("    iretl                  ");

[...]
uint kernel_entry(uint* regs)
{
        /* put your microkernel here */

        return(0); /* ignored now, could be used to flag new cr3 (pid) value */
}
```

The original thread can be read here.