JohnBurger:Demo/x86/TSS

From OSDev Wiki
Revision as of 00:30, 10 April 2014 by Johnburger (Talk | contribs)
(diff) ← Older revision | Latest revision (diff) | Newer revision → (diff)

Although the Task State Segment is (as described) a Segment, it's actually merely the definition for the *beginning* of a Segment. Intel wisely decided to let (nearly*) the rest of the Segment be used for other stuff relating to the Task that it holds the state for - a perfect example is the current Floating Point Unit state.

This Demonstrator doesn't handle the Floating Point execution Unit (FPU) at all - adding that is a project for the reader! There are two strategies:

- On every task switch, just before executing the `JMP` that switches Tasks, first dump the current FPU's state to RAM just above the TSS structure. And on Task resume, recover the FPU state and let it resume.

This has the advantage of simplicity, but a bigger disadvantage: many Floating Point operations take a while to complete, and before an FPU store operation can start it needs to finish what it's doing. Worse, often there aren't many Tasks even using the FPU - it may very well be that the Task resuming operation of the FPU was the very one that dumped its state a few Task Switches ago!

- To help this latter case, Intel defined a Flag (`TS` in `CR0`) and an Exception. On every hardware Task Switch, the Flag is set to indicate that a Task has been switched. On every Floating Point operation that flag is tested: if it is set, the CPU raises an `INT 7` - Coprocessor Not Available exception. That fault handler can quickly determine if it's inside a different Task than what was previously using the FPU: if not, do nothing; if so, only *now* save the FPU state and recover the new Task's FPU state.

---

* And what did I mean by (nearly) above? One of the things that a TSS can do is to define which I/O ports a Task can access. If it is a Supervisor Task, it can (and should) access all available ports - for example, the Interrupt Acknowledge ports on the PICs. A User Task, however, can do immeasurable damage if it could access any port it liked. To that end, a TSS can have a bitmap (`.IOMap` below) that defines which bits the Task is allowed to access - anything else will result in a General Protection Fault. That bitmap is defined as part of the TSS - an array of bits from `.IOMap` to the Limit of the TSS.

Demo/x86/TSS.inc

```
;
; x86/TSS.inc
;

; This module defines the Task State Segment (the 32-bit one, anyway).

x86.TSS.Flags.Trap  EQU         0000_0000_0000_0001b

                    STRUC       x86.TSS
```

```
.Back           RESW    1               ; Back-link to calling TS
                RESW    1
.ESP0           RESD    1               ; Stack 0 pointer
.SS0            RESW    1
                RESW    1
.ESP1           RESD    1               ; Stack 1 pointer
.SS1            RESW    1
                RESW    1
.ESP2           RESD    1               ; Stack 2 pointer
.SS2            RESW    1
                RESW    1
.CR3            RESD    1               ; Page Directory Base Reg
.EIP            RESD    1
.EFlags         RESD    1
.EAX            RESD    1
.ECX            RESD    1
.EDX            RESD    1
.EBX            RESD    1
.ESP            RESD    1
.EBP            RESD    1
.ESI            RESD    1
.EDI            RESD    1
.ES             RESW    1
                RESW    1
.CS             RESW    1
                RESW    1
.SS             RESW    1
                RESW    1
.DS             RESW    1
                RESW    1
.FS             RESW    1
                RESW    1
.GS             RESW    1
                RESW    1
.LDT            RESW    1
                RESW    1
.Flags          RESW    1
.IOMap          RESW    1
                ENDSTRUC
```