Task state segment

From Lowlevel

The **Task State Segment** (TSS for short) is required by the CPU to store information about a task. Originally used for hardware multitasking , it is also needed for software multitasking, in order to pass the CPU from a lower privileged ring to a higher one in the case of a ring switch . In order to be able to use a TSS, a corresponding descriptor is created in the GDT pointing to it. The selector of this descriptor must then be loaded into the task register of the CPU with *ltr <selector>* .

Construction & # 150; TSS descriptor

Task State Segment Descriptor

31-24	23	22-21	20	19-16	15	14-13	12	11-8	7-0
Base address 31-24	G	0	AVL	Limit 19-16	P	DPL	0	grade	Base address 16-23

31-16	15-0
Base address 15-0	Limit 15-0

- *Type* & # 150; Descriptor type & # 150; Here, the value 1001b must be entered. The back zero is also used as a busy flag. This is set to 1 as soon as the task is loaded into the system and started. Tasks are not reentrant. That means they can not call themselves. This is prevented with the busy flag.

Construction TSS

You can divide the TSS fields into two categories:

1. **Static fields** are assigned values once and are then loaded by the CPU into the corresponding registers at each task change, but never changed automatically. Static fields include IOMap Base Address, LDT Segment Selector, CR3, ESP0-2 and SS0-2. The CR3 field need not be initialized if paging is not active, as well as the LDT selector if you are not using LDTs. With the IOMap you can mask specific ports to which the task should not have access. IOMap Base Address can be set to zero if you do not want to use this function of the CPU.
2. **Dynamic fields** must also be initialized, but are then always assigned the corresponding changed register values by the CPU when the task is changed. The dynamic fields include EAX-EDX, CS-GS / SS, EFLAGS, ESI / EDI / EBP / ESP and Previous Task Link.

Even if hardware multitasking is not used, at least one TSS must have been created and its selector stored in the task register. Because the CPU loads a new stack in a ring change so less privileged software does not bring higher priviledged software through a stackfault to crash. The CPU loads the

Construction & # 150; Task state segment

offset	31-16	0-15
0x00		Previous Task Link
0x04	ESP0	
0x08		SS0
0x0C	ESP1	
0x10		SS1
0x14	ESPN2	
0x18		SS2
0x1C	CR3	
0x20	EIP	
0x24	EFLAGS	
0x28	EAX	
0x2C	ECX	
0x30	EDX	
0x34	EBX	

fields ss0 (or ss1 or ss2) as well as esp0 (or esp1 or esp2) as a new stack. A stack change occurs when one program passes control to another by ...

- an interrupt
- a call gate
- a task gate

Retrieved from " http://www.lowlevel.eu/w/index.php?title=Task_State_Segment&oldid=9428 "

0x38	ESP	
0x3C	EBP	
0x40	IT I	
0x44	EDI	
0x48		IT
0x4C		CS
0x50		SS
0x54		DS
0x58		FS
0x5C		GS
0x60		LDT segment selector
0x64	I / O Map Base Address	