Task state segment

From Wikipedia, the free encyclopedia

The **task state segment** (**TSS**) is a special structure on x86-based computers which holds information about a task. It is used by the operating system kernel for task management. Specifically, the following information is stored in the TSS:

- Processor register state
- I/O port permissions
- Inner-level stack pointers
- Previous TSS link

All this information should be stored at specific locations within the TSS as specified in the IA-32 manuals.

Contents

Location of the TSS

The TSS may reside anywhere in memory. A special segment register called the task register (TR) holds a segment selector that points to a valid TSS segment descriptor which resides in the GDT (a TSS descriptor may not reside in the LDT). Therefore, to use a TSS the following must be done by the operating system kernel:

1. Create a TSS descriptor entry in the GDT
2. Load the TR with a segment selector for that segment
3. Add information to the TSS in memory as needed

For security purposes, the TSS should be placed in memory that is accessible only to the kernel.

Task register

The TR register is a 16-bit register which holds a segment selector for the TSS. It may be loaded through the LTR instruction. LTR is a privileged instruction and acts in a manner similar to other segment register loads. The task register has two parts: a portion visible and accessible by the programmer and an invisible one that is automatically loaded from the TSS descriptor

Register states

The TSS may contain saved values of all the x86 registers. This is used for task switching. The operating system may load the TSS with the values of the registers that the new task needs and after executing a hardware task switch (such as with an IRET instruction) the x86 CPU will load the saved values from the TSS into the appropriate registers. Note that some modern operating systems such as Windows and Linux[1] do not use these fields in the TSS as they implement software task switching.

I/O port permissions

The TSS contains a 16-bit pointer to I/O port permissions bitmap for the current task. This bitmap, usually set up by the operating system when a task is started, specifies individual ports to which the program should have access. The I/O bitmap is a bit array of port access permissions; if the program has permission to access a port, a "0" is stored at the corresponding bit index, and if the program does not have permission, a "1" is stored there. The feature operates as follows: when a program issues an x86 I/O port instruction such as IN or OUT (see x86 instruction listings), the hardware will do an I/O privilege level (IOPL) check to see if the program has access to all I/O ports. If the CPL of the program is numerically greater than the IOPL (the program is less-privileged than what the IOPL specifies), the program does not have I/O port access to all ports. The hardware will then check the I/O permissions bitmap in the TSS to see if that program can access the specific port in the IN or OUT instruction. If the bit in the I/O port permissions bitmap is clear, the program is allowed access to this port, and the instruction is allowed to execute. If the bit is set, the program does not have access and the processor generates a general protection fault. This feature allows operating systems to grant selective port access to user programs.

Inner-level stack pointers

The TSS contains 6 fields for specifying the new stack pointer when a privilege level change happens. The field SS0 contains the stack segment selector for CPL=0, and the field ESP0/RSP0 contains the new ESP/RSP value for CPL=0. When an interrupt happens in protected (32-bit) mode, the x86 CPU will look in the TSS for SS0 and ESP0 and load their values into SS and ESP respectively. This allows for the kernel to use a different stack than the user program, and also have this stack be unique for each user program.

A new feature introduced in the AMD64 extensions is called the Interrupt Stack Table (IST). This also resides in the TSS and contains logical (segment+offset) stack pointers. An interrupt descriptor table may specify an IST entry to use (there are 8). If that is the case, the processor will load the new stack from the IST instead. This allows known-good stacks to be used in case of serious errors (NMI or Double fault for example). Previously, to do this, the entry for the exception or interrupt in the IDT pointed to a task gate. This cause the processor to switch to the task that is pointed by the task gate. The original register values were saved in the TSS current at the time the interrupt or exception occurred, and

the processor then set the registers, including SS:ESP, to a known value specified in the TSS and saved the selector to the previous TSS. The problem here is that hardware task switching is not supported on AMD64.

Previous TSS link

This is a 16-bit selector which allows linking this TSS with the previous one. This is only used for hardware task switching. See the IA-32 manuals for details.

Use of TSS in Linux

Although a TSS could be created for each task running on the computer, Linux kernel only creates one TSS for each CPU and uses them for all tasks. This approach was selected as it provides easier portability to other architectures (for example, the AMD64 architecture does not support hardware task switches), and improved performance and flexibility. Linux only uses the I/O port permission bitmap and inner stack features of the TSS; the other features are only needed for hardware task switches, which the Linux kernel does not use.[2]

Exceptions related to the TSS

The x86 exception vector 10 is called the Invalid TSS exception (#TS). It is issued by the processor whenever something goes wrong with the TSS access. For example, if an interrupt happens in CPL=3 and is transferring control to CPL=0, the TSS is used to extract SS0 and ESP0/RSP0 for the stack switch. If the task register holds a bad TSS selector, a #TS fault will be generated. The Invalid TSS exception should never happen during normal operating system operation and is always related to kernel bugs or hardware failure.

For more details on TSS exceptions, see Volume 3a, Chapter 6 of the IA-32 manual.[3]

TSS in x86-64 mode

The x86-64 architecture does not support hardware task switches. However the TSS can still be used in a machine running in the 64 bit extended modes. In these modes the TSS is still useful as it stores:

1. The stack pointer addresses for each privilege level.
2. Pointer Addresses for the Interrupt Stack Table (The inner-level stack pointer section above, discusses the need for this).
3. Offset Address of the IO permission bitmap.

Also, the task register is expanded in these modes to be able to hold a 64-bit base address.

References

1. ^ Bovet, Daniel Pierre; Cesatí, Marco (2006). *Understanding the Linux Kernel, Third Edition* (http://books.google.com/?id=h0lltXyJ8aIC&lpg=PA104&dq=Linux%20hardware%20TSS&pg=PA104#v=onepage&q=Linux%20hardware%20TSS). O'Reilly Media. p. 104. ISBN 978-0-596-00565-8. Retrieved 2009-11-23.

2. ^ Daniel P. Bovet; Marco Cesati (2006). "Understanding the Linux Kernel" (http://books.google.com/books?id=h0lltXyJ8aIC&pg=PT122&lpg=PT122&dq=tss+linux&source=bl&ots=gN8oIT4eMV&sig=pEfnVyqpJCd233hVWBCO-pX55oQ&hl=en&sa=X&ei=DxkMU92-LtHJsga7w4DIAQ&redir_esc=y#v=onepage&q=tss%20linux&f=false). *books.google.com*. O'Reilly. p. 104. Retrieved 2014-02-25.
3. ^ "Intel 64 and IA-32 Architectures Software Developer's Manual Volume 3a" (http://www.intel.com/content/www/us/en/processors/architectures-software-developer-manuals.html). Retrieved 21 May 2012.

External links

- Demonstration program for TSS (http://wiki.osdev.org/JohnBurger:Demo)

Retrieved from "http://en.wikipedia.org/w/index.php?title=Task_state_segment&oldid=639965441"

Categories: X86 architecture

- This page was last modified on 28 December 2014, at 17:31.
- Text is available under the Creative Commons Attribution-ShareAlike License; additional terms may apply. By using this site, you agree to the Terms of Use and Privacy Policy. Wikipedia® is a registered trademark of the Wikimedia Foundation, Inc., a non-profit organization.