Interrupt Descriptor Table

From OSDev Wiki

The **Interrupt Descriptor Table** (**IDT**) is specific to the `I386` architecture. It is the Protected mode counterpart to the Real Mode Interrupt Vector Table (IVT) telling where the Interrupt Service Routines (ISR) are located. It is similar to the Global Descriptor Table in structure.

The IDT entries are called gates. It can contain Interrupt Gates, Task Gates and Trap Gates.

Contents

Location and Size

Location of IDT (address and size) is kept in the IDTR register of the CPU, which can be loaded/stored using LIDT, SIDT instructions.

IDTR

Name	Bit	Description
Limit	0..15	Defines the length of the IDT in bytes - 1 (minimum value is 100h, a value of 1000h means 200h interrupts).
Base	16..47	This 32 bits are the linear address where the IDT starts (INT 0)

This is similar to the GDT, except:

- The first entry (at zero offset) is used in the IDT.
- There are 256 interrupts (0..255), so IDT should have 256 entries, each entry corresponding to a specific interrupt.
- It can contain more or less than 256 entries. More entries are ignored. When an interrupt or exception is invoked whose entry is not present, a GPF is raised that tells the number of the missing IDT entry, and even whether it was hardware or software interrupt. There should therefore be at least enough entries so a GPF can be caught.

Structure

The table contains 8-byte Gate entries. Each entry has a complex structure:

```
struct IDTDescr{
   uint16_t offset_1; // offset bits 0..15
   uint16_t selector; // a code segment selector in GDT or LDT
   uint8_t zero;      // unused, set to 0
   uint8_t type_attr; // type and attributes, see below
   uint16_t offset_2; // offset bits 16..31
};
```

The `offset` is a 32 bit value, split in two parts. The `selector` is a 16 bit value and must point to a valid selector in your GDT.

`type_attr` is specified here:

```
 7                         0
+---+---+---+---+---+---+---+---+
| P |  DPL  | S |    GateType   |
+---+---+---+---+---+---+---+---+
```

The bit fields mean:

IDT entry, Interrupt Gates

Name	Bit	Full Name	Description
Offset	48..63	Offset 16..31	Higher part of the offset.
P	47	Present	can be set to **0** for unused interrupts or for Paging.
DPL	45,46	Descriptor Privilege Level	Gate call protection. Specifies which privilege Level the calling Descriptor minimum should have. So hardware and CPU interrupts can be protected from beeing called out of userspace.
S	44	Storage Segment	= **0** for interrupt gates.
Type	40..43	Gate Type 0..3	Possible IDT gate types : 0b0101 0x5 5 80386 32 bit Task gate 0b0110 0x6 6 80286 16-bit interrupt gate 0b0111 0x7 7 80286 16-bit trap gate 0b1110 0xE 14 80386 32-bit interrupt gate 0b1111 0xF 15 80386 32-bit trap gate
0	32..39	Unused 0..7	Have to be **0**.
Selector	16..31	Selector 0..15	Selector of the interrupt function (to make sense - the kernel's selector). The selector's descriptor's DPL field has to be **0**.

Offset	0..15	Offset 0..15	Lower part of the interrupt function's offset address (also known as pointer).

I386 Interrupt Gate

The Interrupt Gate is used to specify an interrupt service routine. When you do `INT 50` in assembly, running in protected mode, the CPU looks up the 50th entry (located at 50 * 8) in the IDT. Then the Interrupt Gates selector and offset value is loaded. The selector and offset is used to call the interrupt service routine. When the `IRET` instruction is read, it returns. If running in 32 bit mode and the specified selector is a 16 bit selector, then the CPU will go in 16 bit protected mode after calling the interrupt service routine. To return you need to do `O32 IRET`, else the CPU doesn't know that it should do a 32 bit return (reading 32 bit offset of the stack instead of 16 bit).

type_attr	Type
0b1110=0xE	32-bit interrupt gate
0b0110=0x6	16-bit interrupt gate

Here are some pre-cooked type_attr values people are likely to use (assuming DPL=0):

- 32-bit Interrupt gate: 0x8E (P=1, DPL=00b, S=0, type=1110b => type_attr=1000_1110b=0x8E)

I386 Trap Gate

When an interrupt/exception occurs that corresponds to a Trap or Interrupt Gate, the CPU places the return info on the stack (EFLAGS, CS, EIP), so the interrupt handler can resume the interrupted code by IRET.

Then, execution is transferred to the given selector:offset from the gate descriptor.

For some exceptions, an error code is also pushed on the stack, which must be POPped before doing IRET.

Trap and Interrupt gates are similar, and their descriptors are structurally the same, they differ only in the "type" field. The difference is that for interrupt gates, interrupts are automatically disabled upon entry and reenabled upon IRET which restores the saved EFLAGS.

Choosing type_attr values: (See Descriptors#type_attr)

type_attr	Type
0b1111=0xf	32-bit trap gate
0b0111=0x7	16-bit trap gate

Here are some pre-cooked type_attr values people are likely to use (assuming DPL=0):

- 32-bit Trap gate: 0x8F (P=1, DPL=00b, S=0, type=1111b => type_attr=1000_1111b=0x8F)

Thus, Trap and Interrupt gate descriptors hold the following data (other than type_attr):

- 16-bit selector of a code segment in GDT or LDT
- 32-bit offset into that segment - address of the handler, where execution will be transferred

I386 Task Gate

In the Task Gate descriptor the `offset` values are not used. Set them to 0.

When an interrupt/exception occurs whose entry is a Task Gate, a task switch results.

> "A task gate in the IDT references a TSS descriptor in the GDT. A switch to the handler task is handled in the same manner as an ordinary task switch. (..) The link back to the interrupted task is stored in the previous task link field of the handler task's TSS. If an exception caused an error code to be generated, this error code is copied to the stack of the new task."
>
> —Intel manual (vol.3 p.5-19)

> "*NOTE* Because IA-32 tasks are not re-entrant, an interrupt-handler task must disable interrupts between the time it completes handling the interrupt and the time it executes the IRET instruction. This action prevents another interrupt from occurring while the interrupt task's TSS is still marked busy, which would cause a general-protection (#GP) exception."
>
> —Intel manual

Choosing type_attr values: (See

type_attr	Type
0b0101=0x5	task gate

For DPL=0, type_attr=0x85=0b0101

Thus, a TSS selector is the only custom piece of data you need for a Task Gate descriptor.

Advantages over using trap/interrupt gates:

- The entire context of the interrupted task is saved automatically (no need to worry about registers)
- The handler can be isolated from other tasks in a separate address space in LDT.
- *"A new tss permits the handler to use a new privilege level 0 stack when handling the exception or interrupt. If an exception or interrupt occurs when the current privilege level 0 stack is corrupted, accessing the handler through a task gate can prevent a system crash by providing the handler with a new privilege level 0 stack"* --Intel manual

Disadvantage:

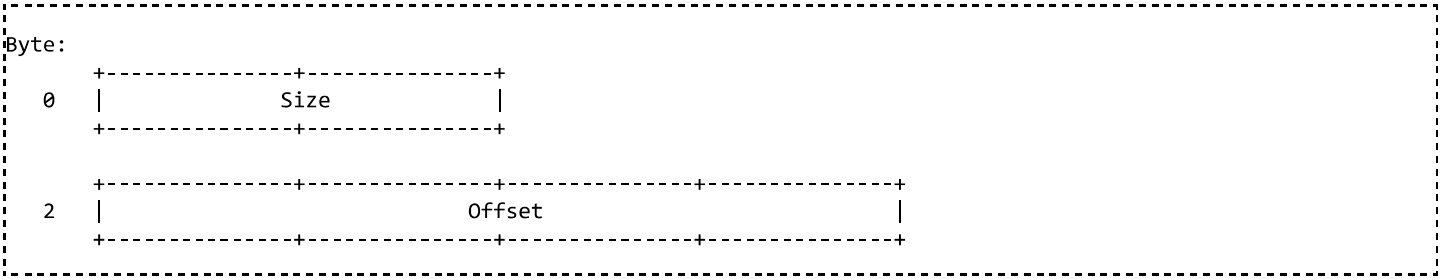
- Saving the entire task context into TSS is slower than using a trap/interrupt gate (where the

handler can save only what it needs).

- Is it that much faster if the handler does PUSHAD or pushes registers one by one?
- Does it make a difference, considering a non-dummy, non-trivial handler?

Loading/Storing

The IDT is loaded using the `LIDT` assembly instruction. It expects the location of a IDT description structure:



The `offset` is the virtual address of the table itself. The `size` is the size of the table subtracted by 1. This structure can be stored to memory again with the `SIDT` instruction.

IDT in IA-32e Mode (64-bit IDT)

When in long or compatibility mode (once the EFER.LME flag has been set) the IDT's structure changes slightly. The IDTR structure's (used by `LIDT` and `SITD`) `base` field changes to 64-bits to allow the IDT to reside anywhere in memory, and each entry in the IDT grows by 64-bits. The first 32-bit value is the high bits of the address, while the second is zero.

IDTR

Offset	Size	Description
0	2	Limit - Maximum addressable byte in table
2	8	Offset - Linear (paged) base address of IDT

IDT Descriptor

Offset	Size	Description
0	2	Offset low bits (0..15)
2	2	Selector (Code segment selector)
4	1	Zero
5	1	Type and Attributes (same as before)
6	2	Offset middle bits (16..31)
8	4	Offset high bits (32..63)
12	4	Zero

In your interrupt handler routines, remember to use IRETQ instead of IRET, as nasm won't translate that for you. Many 64bit IDT related problems on the forum are caused by that missing 'Q'. Don't let this happen to you.

See Also

Articles

- GDT
- IDT problems

External references

- Michal Ludvig's Intel 80386 Programmer's Reference Manual chapter 9 (http://www.logix.cz/michal/doc/i386/chp09-00.htm)

Retrieved from "http://wiki.osdev.org/index.php?title=Interrupt_Descriptor_Table&oldid=17410"
Categories: X86 CPU | Interrupts

- This page was last modified on 1 January 2015, at 13:11.
- This page has been accessed 113,745 times.