Triple faulting on GDT, need information

Author	Message
naccyde	**Post subject:** Triple faulting on GDT, need information **Posted:** Tue Nov 22, 2016 3:38 pm

offline

Joined: Thu Aug 27, 2015 5:58 am
Posts: 4

Hello !

I am currently trying to create a custom bootloader. It does work on real mode, but when I try to enter protected mode, it triple faults on QEMU.

The bootloader is in 2 stages. The first stage load the second stage to avoid the 512 bytes limit (by the way, the stage 2 is far from the 512 bytes), then the stage 2 enter in protected mode (it tries...), and jump the the 32 bits code which is in the stage 2 binary.

Code:

```
gdt:
   .byte    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
   .byte   0xff, 0xff, 0x00, 0x00, 0x00, 0x9b, 0xdf, 0x00
   .byte   0xff, 0xff, 0x00, 0x00, 0x00, 0x93, 0xdf, 0x00
gdtptr:
   .short   0x18
   .long   (gdt + STAGE2_BOOTSEG * 0x10 + STAGE2_BOOTOFF)

_start:
   /* Initializing segments */
   movw    $STAGE2_BOOTSEG, %ax
   movw   %ax, %ds
   movw    %ax, %es
   movw    $STAGE2_STACKSEG, %ax
```

```
   movw   %ax, %ss
   movw   $STAGE2_STACKP, %sp

   /* Reset disks */
   movb   $0x0, %ah
   int   $0x13

   /* Setup GDT and enable protected mode */
   cli
   lgdt   gdtptr

   movl   %cr0, %eax
   orl   $0x00000001, %eax
   movl   %eax, %cr0

   /* Setup kernel segments */
   movl   $STAGE2_BOOTSEG, %eax
   movl   %eax, %ds
   movl    %eax, %es
   movl   %eax, %fs
   movl    %eax, %gs

   movl   $STAGE2_STACKSEG, %eax
   movl    %eax, %ss
   movl   $STAGE2_STACKP, %esp

   /* Jump to kernel */
   ljmp   $STAGE2_BOOTSEG, $test

.code32
test:
   movb   $0x41, 0xb8a00
   jmp   .
```

The variables used here are :

Code:

```
.equ STAGE2_BOOTSEG,   0x0820
.equ STAGE2_STACKSEG,   0x0860
.equ STAGE2_STACKP,   0x0400
```

There is obviously an error here I don't see  I tried debugging it with GDB and QEMU, passing options "--32 -gstabs -ggdb" to as assembler and "-d int,cpu_reset -no-reboot" to QEMU. QEMU with debug options output many of theses lines (see below), I show you the lines with the

protected mode enabled in cr0, then a pattern is repeating (cr0 with protected mode disabled, then enabled enable, then disabled...).

Code:

```
SMM: enter
EAX=00000001 EBX=0000000b ECX=02000000 EDX=02000628
ESI=00000000 EDI=02000000 EBP=00000007 ESP=00006d68
EIP=000ef2e5 EFL=00000002 [-------] CPL=0 II=0 A20=1 SMM=0
HLT=0
ES =0010 00000000 ffffffff 00cf9300 DPL=0 DS   [-WA]
CS =0008 00000000 ffffffff 00cf9b00 DPL=0 CS32 [-RA]
SS =0010 00000000 ffffffff 00cf9300 DPL=0 DS   [-WA]
DS =0010 00000000 ffffffff 00cf9300 DPL=0 DS   [-WA]
FS =0010 00000000 ffffffff 00cf9300 DPL=0 DS   [-WA]
GS =0010 00000000 ffffffff 00cf9300 DPL=0 DS   [-WA]
LDT=0000 00000000 0000ffff 00008200 DPL=0 LDT
TR =0000 00000000 0000ffff 00008b00 DPL=0 TSS32-busy
GDT=     000f7180 00000037
IDT=     000f71be 00000000
CR0=00000011 CR2=00000000 CR3=00000000 CR4=00000000
DR0=00000000 DR1=00000000 DR2=00000000 DR3=00000000
DR6=ffff0ff0 DR7=00000400
CCS=00000080 CCD=00000001 CCO=LOGICB
EFER=0000000000000000
SMM: after RSM
EAX=00000001 EBX=0000000b ECX=02000000 EDX=02000628
ESI=00000000 EDI=02000000 EBP=00000007 ESP=00006d68
EIP=000ef2e5 EFL=00000002 [-------] CPL=0 II=0 A20=1 SMM=0
HLT=0
ES =0010 00000000 ffffffff 00c09300 DPL=0 DS   [-WA]
CS =0008 00000000 ffffffff 00c09b00 DPL=0 CS32 [-RA]
SS =0010 00000000 ffffffff 00c09300 DPL=0 DS   [-WA]
DS =0010 00000000 ffffffff 00c09300 DPL=0 DS   [-WA]
FS =0010 00000000 ffffffff 00c09300 DPL=0 DS   [-WA]
GS =0010 00000000 ffffffff 00c09300 DPL=0 DS   [-WA]
LDT=0000 00000000 0000ffff 00008200 DPL=0 LDT
TR =0000 00000000 0000ffff 00008b00 DPL=0 TSS32-busy
GDT=     000f7180 00000037
IDT=     000f71be 00000000
CR0=00000011 CR2=00000000 CR3=00000000 CR4=00000000
DR0=00000000 DR1=00000000 DR2=00000000 DR3=00000000
DR6=ffff0ff0 DR7=00000400
CCS=00000000 CCD=000a7fe4 CCO=EFLAGS
EFER=0000000000000000
```

So, questions are:
* Why is there so many lines on QEMU output ? Also it looks like protected mode is enabled, disabled, etc...

* What hint could lead me to the solution ?

If you have got hints apart the GDT problem, about the flags, code conventions or else, do not hesitate to inform me
Thank you

Top profile

Octocontrabass

offline

Member

Joined: Mon Mar 25, 2013 7:01 pm
Posts: 883

Post subject: Re: Triple faulting on GDT, need information
Posted: Tue Nov 22, 2016 3:55 pm

I'm not too fluent in AT&T syntax, but I'm pretty sure you need to use a protected mode segment selectors once you enable protected mode. It looks like you're using real mode segments.

Top profile

bzt

offline

Joined: Thu Oct 13, 2016 4:55 pm
Posts: 5

Post subject: Re: Triple faulting on GDT, need information
Posted: Tue Nov 22, 2016 7:41 pm

> **naccyde wrote:**
>
> So, questions are:
> * Why is there so many lines on QEMU output ? Also it looks like protected mode is enabled, disabled, etc...
> * What hint could lead me to the solution ?

Well, the hint is that there are so many lines That suggests me that your code is running in a loop, in which it enables and disables protmode.

Other than that, the output says that EIP is somewhere outside of your code, so I'd assume there's a bad jump somewhere.
For debug, you can
* use a breakpoint at the cli instruction
* use bochs and xchg bx,bx
and do a step-by-step execution from there to pinpoint the faulty code.

Happy bug hunting,
bzt

Top profile

naccyde

offline

Joined: Thu Aug 27, 2015 5:58 am
Posts: 4

Post subject: Re: Triple faulting on GDT, need information
Posted: Wed Nov 23, 2016 9:23 am

Hello !

Thank you for your help. I now use Bochs instead of QEMU (but I patch it and build bochs from sources, otherwise it does not works on Fedora 24).

@Octocontrabass:
I am no fluent in AT&T syntax either, that's why I use it  And you where right, I now mov segment selectors inside registers.

@bzt:
What do you mean by "xchg bx, bx", I saw this instruction exchange memory / register or register / register content, how could it help me ?

It looks likes the problem was the one suggested by Octocontrabass, the bootloader does not triple fault on the mov %ax, %ss anymore.

To bo honest, after this problem I was stuck on the ljmp $STAGE2_BOOTSEG, $test, it was then replaced by ljmp 0x08, $test. But it did not work either, the solution was to use the address of my 32bits label and add 0x8200, because I set ".org 0x0" and the code is loaded in 0x8200 !

Thank you for your help !

Top profile

FusT

offline

Member

Joined: Wed Sep 19, 2012 3:43 am
Posts: 42
Location: The Netherlands

Post subject: Re: Triple faulting on GDT, need information
Posted: Thu Nov 24, 2016 1:21 am

> **Quote:**
> What do you mean by "xchg bx, bx"

the xchg bx, bx instruction is a magic breakpoint in bochs:
http://wiki.osdev.org/Bochs#Magic_Breakpoint

Top profile

Mikumiku747

offline

Post subject: Re: Triple faulting on GDT, need information
Posted: Thu Nov 24, 2016 1:26 am

> **naccyde wrote:**

Member

Joined: Thu Apr 16, 2015 7:37 am
Posts: 56

> What do you mean by "xchg bx, bx", I saw this instruction exchange memory / register or register / register content, how could it help me ?

There's a special feature of bochs where if it encounters an xchg bx, bx instruction, it will halt the virtual machine for debugging. It also needs to be enabled in the config (I think it's called "Magic Breakpoint") or something like that. It's useful because it can be used to effectively place a breakpoint in your code without knowing what address that code is at, effectively a "Breakpoint" machine instruction (helpful if you don't have debugging symbols or can't use them, such as bootstrapping a higher half kernel or, like in your case, bootstrapping protected mode).

I'm not 100% sure on this, but I think they chose xchg bx bx because it's effectively a no-op (Exchanging something with itself doesn't really do much normally apart from maybe effecting some flags), and most sane compilers / generators / programmers wouldn't use it unless they specifically wanted to do something obtuse, but there could be a deeper reason as well. I think it might also work with the 32-bit PM and long mode variants of the xchg instruction as well, but I'm not too sure on that either, you'd have to check the docs to be sure.

As an example, although I'm sure you understand already, if you wanted to make sure the processor state was sane before jumping into your kernel, you could do the following:

Code:

```
/* Jump to kernel */
   xchgw %bx, %bx   /* Breakpoint so we can check state before entering kernel. */
   ljmp   $STAGE2_BOOTSEG, $test
```

- Mikumiku747

Top profile

MichaelFarthing

offline

Member

Joined: Thu Mar 10, 2016 7:35 am

Post subject: Re: Triple faulting on GDT, need information
Posted: Thu Nov 24, 2016 2:12 am

For information: xchg does not affect any flags either. It is like mov in that regard. This is a deliberate choice so as to preserve the flags in the middle of multi precision arithmetic operations (which thankfully, except for very specialist cases, is now a thing of the past).

Posts: 69
Location: Lancaster, England

Top profile

iansjack

offline

Member

Joined: Sat Mar 31, 2012 3:07 am
Posts: 2437
Location: Chichester, UK

Top profile

Post subject: Re: Triple faulting on GDT, need information
Posted: Thu Nov 24, 2016 3:14 am

Note that you can achieve the same result with gdb/qemu by inserting a "jmp ." instraction, manually breaking into the program during execution, and incrementing the IP register by 2.

sebihepp

offline

Member

Joined: Tue Aug 26, 2008 11:24 am
Posts: 149

Post subject: Re: Triple faulting on GDT, need information
Posted: Thu Nov 24, 2016 5:36 am

I see many mistakes in the code.

1. How do you assemble/compile/link the code? Because this code doesn't specify a offset to use. (org directive)
2. Your gdt comes first, so these byte values will be executed first with unwanted results. So I assume you link the code with a linker.
2. In the initial state, registers are not defined. So you don't know if ip=0 and cs=0x0072. It could be any other combination, like ip=0x720 and cs=0
3. You don't have a Bott Parameter Block - some BIOSes overwrite that BPB and if you have code there, they overwrite your code.
4. You don't read the Second Stage from floppy. (Do you use Floppy, cdrom or something else?)
5. You don't enable the A20 Gate. For this small test it doesn't matter, but don't forget it later on.
6. You need to disable interrupts just before enabling PMode, as in PMode interrupts are handled totally different.
7. Segment registers after enabling PMode need to be selectors to the GDT. You can't use addresses there.
8. You cannot assign a 32 bit Value to a segment register, as they are only 16 bit wide.
9. After enabling PMode you need to load the code segment so you are really in 32 bit mode. Therefore you need to make a long jump, but again with gdt selectors and not REAL Mode segments.

Top profile

Display posts from previous: All posts Sort by Post time Ascending Go

newtopic postreply **Page 1 of 1** [9 posts]

Board index » **Operating System Development** » **OS Development** All times are UTC - 6 hours

Who is online

Users browsing this forum: omarrx024 and 10 guests

You **cannot** post new topics in this forum
You **cannot** reply to topics in this forum
You **cannot** edit your posts in this forum
You **cannot** delete your posts in this forum
You **cannot** post attachments in this forum

Search for: Go Jump to: OS Development Go