BEST WAY TO DO I/O PORT PROTECTION

Best way to do I/O port protection

by **NickJohnson** » Wed May 27, 2009 4:49 pm

In my kernel design, all of the I/O devices' drivers run entirely in userspace. Obviously, I need to allow these driver processes to use the I/O ports in order for them to work properly. I know that the TSS allows you to set a bitmap of allowed ports for each process, which initially sounded pretty good to me. But now that I think about it, it seems like rewriting 65536 bits of bitmap (i.e. 2048 dwords) every time I switch to or from a driver will be very time consuming. Btw, I'll probably add some info about the IOPB to the wiki... it seems very sparse on the TSS in general.

I know that in traditional microkernels for the x86, driver processes are run in ring 1, which according to the Intel manuals means they can access all of the ports. Allowing all ports is fine with me - I'm not worrying about writing to the wrong ports, unless I really should be. Are there any significant changes I have to make to my tasking system in order to run certain processes in ring 1, or do I just set up a couple of GDT entries and change the segment descriptors for those processes?

NickJohnson
Member
Posts: 1249
Joined: Tue Mar 24, 2009 8:11 pm
Location: Sunnyvale, California

Re: Best way to do I/O port protection

by **Brendan** » Wed May 27, 2009 10:10 pm

Hi,

> **NickJohnson wrote:**
> In my kernel design, all of the I/O devices' drivers run entirely in userspace. Obviously, I need to allow these driver processes to use the I/O ports in order for them to work properly. I know that the TSS allows you to set a bitmap of allowed ports for each process, which initially sounded pretty good to me. But now that I think about it, it seems like rewriting 65536 bits of bitmap (i.e. 2048 dwords) every time I switch to or from a driver will be very time consuming. Btw, I'll probably add some info about the IOPB to the wiki... it seems very sparse on the TSS in general.

There's at least 3 methods:

- - use the I/O permission bitmap in the TSS
  - emulate the I/O instructions in the general protection fault handler (e.g. when any "CPL != 0" code tries to access an I/O port it causes a GPF and the general protection fault hander checks if the cause was an I/O port access and determines if access should be allowed, and then does the I/O port access itself)
  - use kernel functions (where device drivers use the kernel API to access I/O ports instead of instructions like IN, OUT, etc)

Emulating the I/O port instructions in the general protection fault handler is the most powerful way, because it lets you protect individual bits in some I/O ports (for e.g. giving a CMOS driver access I/O port 0x70 without letting it change bit 7 and enable/disable NMI; or letting a keyboard driver access I/O port 0x64 without letting it change the A20 and reset bits). Also, it allows you to use virtual I/O ports - for example, tell the device driver that it's device uses I/O ports from 0x0000 to 0x0008, and then when the driver accesses I/O port 0x0001 you actually access I/O port 0x0123 instead (which could make it easier to write devices drivers because you could hard-code I/O ports instead of doing "mov dx,[IOportBase]" everywhere). It can be used for device virtualization (e.g. have an entirely fictional floppy drive controller, where a

Brendan
Member
Posts: 8561
Joined: Sat Jan 15, 2005 12:00 am
Location: At his keyboard!
Contact:

completely normal device driver can't tell if it's a virtual device or a real device). Lastly it's useful for debugging purposes (e.g. create a log of all I/O port accesses make by a device driver).

Using the I/O permission bitmap in the TSS probably has the least overhead, especially if you keep track of whether the I/O permission bitmap has already been cleared (to avoid clearing it when you switch between tasks that don't have access to any I/O ports).

If you're willing to accept lower security (e.g. any device driver can access any I/O ports) then you could set IOPL (in EFLAGS) during task switches. For e.g. if a task shouldn't have access to any I/O ports set IOPL to 0, and if a task should have access to all I/O ports then set IOPL to 3. Unfortunately IOPL also controls access to CLI/STI, so a dodgy device driver could lock up the computer ("cli; jmp $") in this case.

You can also use a mixture of these options. For example, you could clear the I/O permission bitmap during task switches (if it's not already clear), then if you get a general protection fault caused by I/O port access you'd check if it might be allowed and then load the correct I/O permission bitmap and retry the instruction; and if a device driver runs correctly for long enough (and/or if the administrator gives their blessing?) the kernel could start using "IOPL = 3" for that task instead.

> **NickJohnson wrote:**
> I know that in traditional microkernels for the x86, driver processes are run in ring 1, which according to the Intel manuals means they can access all of the ports.

Most micro-kernels run device drivers at CPL=3, and (except for OS/2) I don't think any OS ever used CPL=1 for anything (AFAIK the main use for CPL=1 is "ring compression" in virtual machines). Also, CPL=1 code may or may not have access to I/O ports (it depends on the how IOPL in EFLAGS is set).

> **NickJohnson wrote:**
> Allowing all ports is fine with me - I'm not worrying about writing to the wrong ports, unless I really should be.

This might be more important than you think. For your OS, where are the device drivers going to come from, and can you guarantee that none of them will contain malicious (or just plain buggy) code? There's several approaches to this - one approach is to protect the OS from malicious device drivers (which allows anyone to write device drivers for the OS with very little risk to end users). Another approach is to have no protection at all, spend 10 years trying to write open source device drivers yourself (and try to pretend that "many eyes" will catch malicious or buggy code), and then get all worried about "binary blobs" when someone like NVidia goes out of their way to write a closed source device driver for your OS... 😉

> **NickJohnson wrote:**
> Are there any significant changes I have to make to my tasking system in order to run certain processes in ring 1, or do I just set up a couple of GDT entries and change the segment descriptors for those processes?

To use CPL=1 you'd need GDT entries. To allow CPL=3 code to call device drivers (e.g. with a software interrupt or call gate) you'd need to set "SS1:ESP1" in the TSS (the same way you need to set "SS0:ESP0" in the TSS to allow CPL=3 code to call the kernel). That's about it I think.

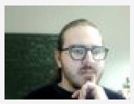
Cheers,

Brendan

For all things; perfection is, and will always remain, impossible to achieve in practice. However; by striving for perfection we create things that are as perfect as practically possible. Let the pursuit of perfection be our guide.

Re: Best way to do I/O port protection

by **xenos** » Thu May 28, 2009 1:38 am



I don't think you have to actually re-write the whole I/O bitmap on a task switch. 2048 dwords is two pages, so if your bitmap is page-aligned, you can just map another I/O bitmap for each process. So there is no overhead at all, since switching to a different address space should switch the page directory, anyway.

Programmers' Hardware Database // GitHub user: xenos1984; OS project: NOS

xenos
Member
Posts: 1117
Joined: Thu Aug 11, 2005 11:00 pm
Libera.chat IRC: xenos1984
Location: Tartu, Estonia
Contact:

Re: Best way to do I/O port protection

by **NickJohnson** » Thu May 28, 2009 7:54 am

I like the idea of emulating port accesses - it does seem like the most flexible way - but how much overhead does that create? It seems like it would be equally effective to have a system call that does port accesses. Then you could have the kernel read from a list of requested port accesses instead of just doing one per system call, which is probably much faster for devices that need multiple port accesses in a row.

> **Brendan wrote:**
>
> > **NickJohnson wrote:**
> > I know that in traditional microkernels for the x86, driver processes are run in ring 1, which according to the Intel manuals means they can access all of the ports.
>
> Most micro-kernels run device drivers at CPL=3, and (except for OS/2) I don't think any OS ever used CPL=1 for anything (AFAIK the main use for CPL=1 is "ring compression" in virtual machines). Also, CPL=1 code may or may not have access to I/O ports (it depends on the how IOPL in EFLAGS is set).

I thought that MINIX ran drivers in ring 1 and 2 though. I'm using it as at least one of my reference designs, because I have the book.

NickJohnson
Member
Posts: 1249
Joined: Tue Mar 24, 2009 8:11 pm
Location: Sunnyvale, California

Re: Best way to do I/O port protection

by **whowhatwhere** » Thu May 28, 2009 7:55 am

iirc rings were essentially removed on newer processors (starting with x86_64 afaik) and the only rigs left are 1 and 3.

whowhatwhere
Member
Posts: 199
Joined: Sat Jun 28, 2008 6:44 pm

Re: Best way to do I/O port protection

by **jal** » Thu May 28, 2009 8:32 am

> **syntropy wrote:**
> iirc rings were essentially removed on newer processors (starting with x86_64 afaik) and the only rigs left are 1 and 3.

0 and 3, obviously.

JAL

jal
Member
Posts: 1385
Joined: Wed Oct 31, 2007 9:09 am

Re: Best way to do I/O port protection

by **jal** » Thu May 28, 2009 8:34 am

> **XenOS wrote:**
> I don't think you have to actually re-write the whole I/O bitmap on a task switch. 2048 dwords is two pages, so if your bitmap is page-aligned, you can just map another I/O bitmap for each process. So there is no overhead at all, since switching to a different address space should switch the page directory, anyway.

I have no time to check, but iirc, the TSS is at a physical address, and not subject to paging. You are right though that you don't need to rewrite the whole I/O bitmap: you can just clear the

jal
Member
Posts: 1385
Joined: Wed Oct 31, 2007 9:09 am

permissions of the previous task, and set the permissions of the current one. Since devices usually have a fairly limited range of I/O ports they use, this should be rather quick.

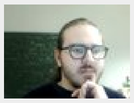
JAL

Re: Best way to do I/O port protection

by **xenos** » Thu May 28, 2009 10:17 am

I'm pretty sure the TSS is at a virtual address, not a physical one... I will check that, though.

Programmers' Hardware Database // GitHub user: xenos1984; OS project: NOS



xenos
Member

Posts: 1117
Joined: Thu Aug 11, 2005 11:00 pm
Libera.chat IRC: xenos1984
Location: Tartu, Estonia
Contact:

Re: Best way to do I/O port protection

by **jal** » Thu May 28, 2009 12:41 pm

[quote="XenOS"]I'm pretty sure the TSS is at a virtual address, not a physical one... I will check that, though.[/qu

You are right of course, as the TSS is an entry in the GDT. Duh! *slaps head*

JAL

jal
Member

Posts: 1385
Joined: Wed Oct 31, 2007 9:09 am

Re: Best way to do I/O port protection

by **Troy Martin** » Thu May 28, 2009 4:18 pm

> **jal wrote:**
>
> > **syntropy wrote:**
> > iirc rings were essentially removed on newer processors (starting with x86_64 afaik) and the only rigs left are 1 and 3.
>
> 0 and 3, obviously.

Even so, the removal of rings 1 and 2 makes no sense. That breaks a chunk of protected mode operating systems, since many use rings 1 and/or 2 as driver layers.

Image

> **Solar wrote:**
> It keeps stunning me how friendly we - as a community - are towards people who start programming "their first OS" who don't even have a solid understanding of pointers, their compiler, or how a OS is structured.

I wish I could add more tex

Troy Martin
Member

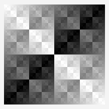
Posts: 1686
Joined: Fri Apr 18, 2008 4:40 pm
Location: Langley, Vancouver, BC, Canada
Contact:

Re: Best way to do I/O port protection

by **NickJohnson** » Thu May 28, 2009 4:26 pm

Would it also be possible to just set the IOPL to 3 for driver processes exclusively, and stay out of the whole segmentation thing altogether? Would that be safe?

I would give init I/O privileges, then give fork() a way to permanently remove it for child processes, thereby keeping things safe.



NickJohnson
Member

Posts: 1249
Joined: Tue Mar 24, 2009 8:11 pm
Location: Sunnyvale, California

Re: Best way to do I/O port protection

by **pcmattman** » Thu May 28, 2009 5:07 pm

> That breaks a chunk of protected mode operating systems, since many use rings 1 and/or 2 as driver layers.

> (starting with x86_64 afaik)

x86_64 is a whole new ballpark

I personally agree with Brendan:

> emulate the I/O instructions in the general protection fault handler (e.g. when any "CPL != 0" code tries to access an I/O port it causes a GPF and the general protection fault hander checks if the cause was an I/O port access and determines if access should be allowed, and then does the I/O port access itself)

Rather than using 64 KB for the IOPB, you can then run control who gets port access to an extremely fine degree thanks to the GPF handler.

> I would give init I/O privileges, then give fork() a way to permanently remove it for child processes, thereby keeping things safe.

It could work - you'd need to figure out a way of differentiating drivers from normal applications, so that they get I/O privileges.

Pedigree | GitHub | Twitter | LinkedIn

pcmattman
Member
Posts: 2566
Joined: Sun Jan 14, 2007 9:15 pm
Libera.chat IRC: miselin
Location: Sydney, Australia (I come from a land down under!)
Contact:

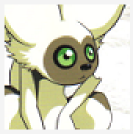
Re: Best way to do I/O port protection

by **Combuster** » Thu May 28, 2009 5:13 pm

> Rather than using 64 KB for the IOPB

at full size, its only 8kb (8192 * 8 = 65536)

"Certainly avoid yourself. He is a newbie and might not realize it. You'll hate his code deeply a few years down the road." - Sortie
[My OS] [VDisk/SFS]



Combuster
Member
Posts: 9301
Joined: Wed Oct 18, 2006 3:45 am
Libera.chat IRC: [com]buster
Location: On the balcony, where I can actually keep 1½m distance
Contact:

Re: Best way to do I/O port protection

by **pcmattman** » Thu May 28, 2009 5:14 pm

Err, sorry, I read "65536 bits" as "65536 *bytes*".

Pedigree | GitHub | Twitter | LinkedIn

pcmattman
Member
Posts: 2566
Joined: Sun Jan 14, 2007 9:15 pm
Libera.chat IRC: miselin
Location: Sydney, Australia (I come from a land down under!)
Contact:

Re: Best way to do I/O port protection

by **jal** » Fri May 29, 2009 3:06 am

> **Troy Martin wrote:**

jal
Member
Posts: 1385
Joined: Wed Oct 31, 2007 9:09 am

> Even so, the removal of rings 1 and 2 makes no sense. That breaks a chunk of protected mode operating systems, since many use rings 1 and/or 2 as driver layers.

Please get your facts straight.
1) No commercial operating system uses rings 1 and 2, except OS/2. Reasons for this are limited functional use and portability (no other CPU has more than 2 protection levels comparable to the x86 rings).
2) Removal of these rings is *only* for long mode. That breaks nothing, since these older OSes don't run in long mode.

So removing this rings to keep things simple makes all the sense in the world.

JAL

POST REPLY

29 posts 1 2

Return to "OS Development"

JUMP TO

Board index Delete cookies