How are interrupts handled on SMP?

How are interrupts handled on SMP (Symmeteric multiprocessor/multicore) machines? Is there only one memory management unit or more?

Say two threads, A and B running on different cores touch a memory page (at the same time) which is not there in the page table, in which case there will be a page fault and a new page is brought in from the memory.

What is the sequence of events which will happen? If there is one memory management unit, to which core is the page fault forwarded to? How does the kernel handle it? Are there multiple instances of the kernel, each one running on a different core? If so, how do they synchronize on such events as page fault handling?

c linux x86-64

edited Apr 19 '12 at 19:22

asked Apr 19 '12 at 19:04
user1018562
3,245 5 34 73

5 Answers

On multicore/multiprocessor architectures, an APIC is used to route interrupts to cores/processors. As the name implies, APICs can be programmed to do the routing as desired.

Regarding the synchronization of the kernel: This depends on the kernel/OS. You can either use a scheme with locking (although IPIs might be necessary on non-cachecoherent architectures) or you can also use your suggested approach of running a kernel on every core and use some kind of explicit inter-kernel communication.

Barrelfish is an example of an OS running multiple kernels. If you are interested in that kind of architecture, you might want to read the paper "The Multikernel: A new OS architecture for scalable multicore systems"

answered Apr 19 '12 at 19:32
mensi
4,411 12 30

Well, it depends on the specific architecture, but from what I can remember from the Intel docs...

There are two main sources of interrupts:

- Internal: These are generated by the CPU itself. Includes faults, traps, software interrupts, etc.
- External: These are hardware interrupts generated by peripherals.

The internal interrupts are always delivered to the CPU that generated it. The external ones are sent to an arbirary core.

In modern models, interrupts can also be delivered using a bus-like system instead the old interrupt driven one, but I ignore if this model is being used in any current OS.

About the MMU, each core has its own, of course, but they are usually forced the same segments by the OS, so they can be used symmetrically. Note that most of the mapping between physical and virtual memory are actually in memory, and that is always shared.

About the sequence in your example:

- The page fault is forwarded to the core that generated it.
- The kernel updates its MMU tables, that are protected by a shared lock or similar.
- No, there is only one kernel, usually, unless you apply a model of virtualization.
- They synchronize using a shared lock or similar structure. If both cores happen to fault at the same page at the same time... well it's not a big deal, actually.

answered Apr 19 '12 at 19:32
rodrigo
36.5k 2 30 66

Again, please have a look at Barrelfish as an example for multiple kernels. A monolithic kernel approach does not scale well beyond hundreds of cores. – mensi Apr 20 '12 at 9:34

@mensi - Of course, but I felt that the OP referred more to mainstreams OSes, particularly Linux, as he included that tag. And AFAIK hundreds of cores are not yet a scalability issue... – rodrigo Apr 20 '12 at 16:40

Each processor has its own memory management unit with a translation lookaside buffer. This is necessary because each core could be executing a different process which has a different address space.

The multiple cores can independently handle interrupts/exceptions at the same time. So there can be multiple concurrent interrupt contexts executing in a kernel at the same time.

An exception such as a page fault or division by zero will always be handled on the same processor on which it occured, since it pertains to what that processor is doing.

External interrupts typically go through some kind of switching fabric that allows them to be mapped to processors in some way, statically or dynamically. E.g. the "APIC" on PC type hardware.

If the fabric is sufficiently sophisticated, then interrupts can be reprogrammed target a different core on the fly.

It depends on the architecture. A simplistic architecture could, for instance, tie all external interrupts to one core. That would not be very symmetric though; it would not allow for IRQ load balancing.

(Also note that it's useful to have certain external interrupts happen on all processors. An example of this is the timer interrupt. If every core has its own interrupt timer, then handling time slicing in the scheduler is symmetric: there is no special case handling for one main core versus the others. An interrupt goes off, the core runs the scheduler code, and if the current task's time quantum is up, it chooses another task to run on that core.)

edited Apr 20 '12 at 21:59

answered Apr 19 '12 at 19:35
Kaz
20k 2 21 52

Cool; i like how you call it "the fabric". Any details on this for ARM arch (like OMAP3 / 4)? Thanks. – kaiwan Jun 28 '12 at 1:47

Each logical CPU (i.e. CPU core) has it's own `cr3` register, handling pointer to paging structures. The two page faults can occur either:

- in threads of the same process
- in threads of different processes

If those are threads of different processes, then it's no problem. I don't know what is specific Linux's implementation of this (yes, I know it's tagged as "linux"), but there are two general algorithms to manage virtual memory in SMP environment:

- each core holds it's own list of free physical pages and requests some more when all of pages on it's own list are allocated
- all cores use the same list of free pages, protected by some kind of lock (usually spinlock is enough in such case), which is of course slower solution

The same code (#PF handler) can execute simultaneously on two different cores, that's not a problem. If threads use two different VASes[1], then their page faults are just handled symmetrically. If the page faults occurs within single VAS, it's still no problem, until the #PFs are caused by access to the same page. In such case, each page in VAS should be protected by a spinlock (or just #PF in given VAS can be protected by single lock - this way reduces memory overhead, but removes the possibility to run two #PF handlers simultaneously).

According to this answer, only in NUMA systems, each CPU core has it's own MMU; in other systems, every physical processor has it's own MMU, as well as TLB to handle different paging structures referenced by different values of `cr3` register.

1. VAS = Virtual Address Space

answered Apr 19 '12 at 19:34
Griwes
5,298 2 12 50

I didn't get the "multiple instances of the kernel", usually the kernel rules em' all. meaning that it has no instance, instead one should think about the kernel as a global reactive system, which provides services to applications.

As far as I know, the memory handing is one unit (though each core has its own interrupt vector), the pages are locked using `page_table_lock` , so the page retrieving is executed only once, in the order of locking.

** EDIT: After seeing the other comments, my answer might be out-dated: anyway you should check: http://www.xml.com/ldd/chapter/book/ch13.html

answered Apr 19 '12 at 19:34
Guy L
687 5 18

By "multiple instances of the kernel", he means "multiple #PF handlers running simultaneously". – Griwes Apr 19 '12 at 19:35

For an example of "multiple instances of the kernel" see the paper I linked in my answer – mensi Apr 19 '12 at 19:36