Active MM

List: linux-kernel
Subject: Re: active_mm
From: Linus Torvalds <torvalds () transmeta ! com>
Date: 1999-07-30 21:36:24

Cc'd to linux-kernel, because I don't write explanations all that often,
and when I do I feel better about more people reading them.

On Fri, 30 Jul 1999, David Mosberger wrote:
>
> Is there a brief description someplace on how "mm" vs. "active_mm" in
> the task_struct are supposed to be used? (My apologies if this was
> discussed on the mailing lists---I just returned from vacation and
> wasn't able to follow linux-kernel for a while).

Basically, the new setup is:

- we have "real address spaces" and "anonymous address spaces". The
  difference is that an anonymous address space doesn't care about the
  user-level page tables at all, so when we do a context switch into an
  anonymous address space we just leave the previous address space
  active.

  The obvious use for a "anonymous address space" is any thread that
  doesn't need any user mappings - all kernel threads basically fall into
  this category, but even "real" threads can temporarily say that for
  some amount of time they are not going to be interested in user space,
  and that the scheduler might as well try to avoid wasting time on
  switching the VM state around. Currently only the old-style bdflush
  sync does that.

- "tsk->mm" points to the "real address space". For an anonymous process,
  tsk->mm will be NULL, for the logical reason that an anonymous process
  really doesn't _have_ a real address space at all.

- however, we obviously need to keep track of which address space we
  "stole" for such an anonymous user. For that, we have "tsk->active_mm",
  which shows what the currently active address space is.

  The rule is that for a process with a real address space (ie tsk->mm is
  non-NULL) the active_mm obviously always has to be the same as the real
  one.

  For a anonymous process, tsk->mm == NULL, and tsk->active_mm is the
  "borrowed" mm while the anonymous process is running. When the
  anonymous process gets scheduled away, the borrowed address space is
  returned and cleared.

To support all that, the "struct mm_struct" now has two counters: a
"mm_users" counter that is how many "real address space users" there are,
and a "mm_count" counter that is the number of "lazy" users (ie anonymous
users) plus one if there are any real users.

Usually there is at least one real user, but it could be that the real
user exited on another CPU while a lazy user was still active, so you do
actually get cases where you have a address space that is _only_ used by
lazy users. That is often a short-lived state, because once that thread
gets scheduled away in favour of a real thread, the "zombie" mm gets
released because "mm_count" becomes zero.

Also, a new rule is that _nobody_ ever has "init_mm" as a real MM any
more. "init_mm" should be considered just a "lazy context when no other
context is available", and in fact it is mainly used just at bootup when
no real VM has yet been created. So code that used to check

```
	if (current->mm == &init_mm)
```

should generally just do

```
	if (!current->mm)
```

instead (which makes more sense anyway - the test is basically one of "do
we have a user context", and is generally done by the page fault handler
and things like that).

Anyway, I put a pre-patch-2.3.13-1 on ftp.kernel.org just a moment ago,

because it slightly changes the interfaces to accommodate the alpha (who would have thought it, but the alpha actually ends up having one of the ugliest context switch codes - unlike the other architectures where the MM and register state is separate, the alpha PALcode joins the two, and you need to switch both together).

(From http://marc.info/?l=linux-kernel&m=93337278602211&w=2)