

## Re: [patch 08/13] syslets: x86, add move\_user\_context() method

---

*Source:* <http://linux.derkeiler.com/Mailing-Lists/Kernel/2007-02/msg07926.html>

---

- *From:* Ingo Molnar <mingo@xxxxxxx>
  - *Date:* Thu, 22 Feb 2007 00:20:17 +0100
- 

\* Davide Libenzi <davidel@xxxxxxxxxxxxxxxxxx> wrote:

On Wed, 21 Feb 2007, Ingo Molnar wrote:

From: Ingo Molnar <mingo@xxxxxxx>

add the move\_user\_context() method to move the user-space context of one kernel thread to another kernel thread.

User-space might notice the changed TID, but execution, stack and register contents (general purpose and FPU) are still the same.

Also signal handling should/must be maintained, on top of TID. You don't want the user to be presented with a different signal handling after an sys\_async\_exec call.

right now CLONE\_SIGNAL and CLONE\_SIGHAND is used for new async threads, so they should inherit and share all the signal settings.

one area that definitely needs more work is that the ptrace parent (if any) should probably follow the 'head' context. gdb at the moment copes surprisingly well, but some artifacts are visible every now and then.

```
+ *new_regs = *old_regs;
+ /*
+ * Flip around the FPU state too:
+ */
+ tmp = new_task->thread.i387;
+ new_task->thread.i387 = old_task->thread.i387;
+ old_task->thread.i387 = tmp;
+ }
```

Re: [patch 08/13] syslets: x86, add move\_user\_context() method

This is not going to work in this case (already posted twice in other emails):

i'm really sorry – i still have a huge email backlog.

So NTSK loads a non up2date FPUo, instead of the FPUc that was the "dirty" context to migrate (since TS\_USEDFPU was set). I think you need an early \_\_unlazy\_fpu() in that case, that would turn the above into:

yes. My plan is to to avoid all these problems by having a special-purpose sched\_yield\_to(old\_task, new\_task) function.

this, besides being even faster than the default scheduler (because the runqueue balance does not change so no real scheduling decision has to be done – the true scheduling decisions happen later on at async-wakeup time), should also avoid all the FPU races: the FPU just gets flipped between old\_task and new\_task (and TS\_USEDFPU needs to be moved as well, etc.). No intermediate task can come inbetween.

can you see a hole in this sched\_yield\_to() method as well?

Ingo

–

To unsubscribe from this list: send the line "unsubscribe linux-kernel" in the body of a message to majordomo@xxxxxxxxxxxxxxxxx

More majordomo info at <http://vger.kernel.org/majordomo-info.html>

Please read the FAQ at <http://www.tux.org/lkml/>