Subject: Re: [PATCH] cpci_hotplug: Convert to use the kthread API
Posted by Scott Murray on Fri, 27 Apr 2007 22:07:49 GMT
View Forum Message <> Reply to Message

On Sun, 22 Apr 2007, Christoph Hellwig wrote:

> On Thu, Apr 19, 2007 at 12:55:29AM -0600, Eric W. Biederman wrote:
> > From: Eric W. Biederman <ebiederm@xmission.com> - unquoted
> >
> > kthread_run replaces the kernel_thread and daemonize calls
> > during thread startup.
> >
> > Calls to signal_pending were also removed as it is currently
> > impossible for the cpci_hotplug thread to receive signals.
>
> This drivers thread are a bit of a miss, although a lot better than
> most other pci hotplug drivers :)
>
> Below is more complete conversion to the kthread infrastructure +
> wake_up_process to wake the thread. Note that we had to keep
> a thread_finished variable because the existing one had dual use.

Sorry, it took me a few days to get to testing this out. It looks good,
but I had to make a couple of tweaks to avoid a hang when rmmod'ing a
board driver. The board drivers do:

cpci_hp_stop()
cpci_hp_unregister_controller(controller)

to shutdown, and the check in cpci_hp_unregister_controller if the thread
is running wasn't working due to a bit too much code being excised. The
result was kthread_stop being called twice, which hangs. I've indicated
my changes to avoid this inline below.

Scott

> Signed-off-by: Christoph Hellwig <hch@lst.de>

Acked-by: Scott Murray <scottm@somanetworks.com>

> Index: linux-2.6/drivers/pci/hotplug/cpci_hotplug_core.c
> ===
> --- linux-2.6.orig/drivers/pci/hotplug/cpci_hotplug_core.c 2007-04-22 12:54:17.000000000 +0200
> +++ linux-2.6/drivers/pci/hotplug/cpci_hotplug_core.c 2007-04-22 13:01:42.000000000 +0200
> @@ -35,6 +35,7 @@
> #include <linux/smp_lock.h>
> #include <asm/atomic.h>
> #include <linux/delay.h>

```
> +#include <linux/kthread.h>
>  #include "cpci_hotplug.h"
>
>  #define DRIVER_AUTHOR "Scott Murray <scottm@somanetworks.com>"
> @@ -59,9 +60,8 @@ static int slots;
>  static atomic_t extracting;
>  int cpci_debug;
>  static struct cpci_hp_controller *controller;
> -static struct semaphore event_semaphore; /* mutex for process loop (up if something to
process) */
> -static struct semaphore thread_exit;  /* guard ensure thread has exited before calling it quits */
> -static int thread_finished = 1;
> +static struct task_struct *cpci_thread;
> +static int thread_finished;
>
>  static int enable_slot(struct hotplug_slot *slot);
>  static int disable_slot(struct hotplug_slot *slot);
> @@ -357,9 +357,7 @@ cpci_hp_intr(int irq, void *data)
>   controller->ops->disable_irq();
>
>   /* Trigger processing by the event thread */
> - dbg("Signal event_semaphore");
> - up(&event_semaphore);
> - dbg("exited cpci_hp_intr");
> + wake_up_process(cpci_thread);
>   return IRQ_HANDLED;
>  }
>
> @@ -521,17 +519,12 @@ event_thread(void *data)
>  {
>   int rc;
>
> - lock_kernel();
> - daemonize("cpci_hp_eventd");
> - unlock_kernel();
> -
>   dbg("%s - event thread started", __FUNCTION__);
>   while (1) {
>    dbg("event thread sleeping");
> -  down_interruptible(&event_semaphore);
> -  dbg("event thread woken, thread_finished = %d",
> -      thread_finished);
> -  if (thread_finished || signal_pending(current))
> +  set_current_state(TASK_INTERRUPTIBLE);
> +  schedule();
> +  if (kthread_should_stop())
>     break;
>    do {
```

```
>     rc = check_slots();
> @@ -541,18 +534,17 @@ event_thread(void *data)
>     } else if (rc < 0) {
>      dbg("%s - error checking slots", __FUNCTION__);
>      thread_finished = 1;
> -    break;
> +    goto out;
>     }
> -  } while (atomic_read(&extracting) && !thread_finished);
> -  if (thread_finished)
> +  } while (atomic_read(&extracting) && !kthread_should_stop());
> +  if (kthread_should_stop())
>     break;
>
>    /* Re-enable ENUM# interrupt */
>    dbg("%s - re-enabling irq", __FUNCTION__);
>    controller->ops->enable_irq();
>   }
> - dbg("%s - event thread signals exit", __FUNCTION__);
> - up(&thread_exit);
> + out:
>   return 0;
>  }
>
> @@ -562,12 +554,8 @@ poll_thread(void *data)
>  {
>   int rc;
>
> - lock_kernel();
> - daemonize("cpci_hp_polld");
> - unlock_kernel();
> -
>   while (1) {
> -  if (thread_finished || signal_pending(current))
> +  if (kthread_should_stop() || signal_pending(current))
>     break;
>    if (controller->ops->query_enum()) {
>     do {
> @@ -578,48 +566,34 @@ poll_thread(void *data)
>      } else if (rc < 0) {
>       dbg("%s - error checking slots", __FUNCTION__);
>       thread_finished = 1;
> -     break;
> +     goto out;
>      }
> -  } while (atomic_read(&extracting) && !thread_finished);
> +  } while (atomic_read(&extracting) && !kthread_should_stop());
>    }
```

```
>    msleep(100);
>   }
> - dbg("poll thread signals exit");
> - up(&thread_exit);
> + out:
>   return 0;
>  }
>
>  static int
>  cpci_start_thread(void)
>  {
> - int pid;
> -
> - /* initialize our semaphores */
> - init_MUTEX_LOCKED(&event_semaphore);
> - init_MUTEX_LOCKED(&thread_exit);
> - thread_finished = 0;
> -
>   if (controller->irq)
> -  pid = kernel_thread(event_thread, NULL, 0);
> +  cpci_thread = kthread_run(event_thread, NULL, "cpci_hp_eventd");
>   else
> -  pid = kernel_thread(poll_thread, NULL, 0);
> - if (pid < 0) {
> +  cpci_thread = kthread_run(poll_thread, NULL, "cpci_hp_polld");
> + if (IS_ERR(cpci_thread)) {
>    err("Can't start up our thread");
> -  return -1;
> +  return PTR_ERR(cpci_thread);
>   }
> - dbg("Our thread pid = %d", pid);
```

There still needs to be a:

```
        thread_finished = 0;
```

here, so that things work if a board driver is insmod'ed again after a
rmmod (i.e. cpci_hp_start after a cpci_hp_stop).

```
>   return 0;
>  }
>
>  static void
>  cpci_stop_thread(void)
>  {
> - thread_finished = 1;
> - dbg("thread finish command given");
> - if (controller->irq)
```

> - up(&event_semaphore);
> - dbg("wait for thread to exit");
> - down(&thread_exit);
> + kthread_stop(cpci_thread);

As well, there still needs to be a:

        thread_finished = 1;

here to make the check in cpci_hp_unregister_controller work.

> }
>
> int

--
Scott Murray
SOMA Networks, Inc.
Toronto, Ontario
e-mail: scottm@somanetworks.com