Subject: Remaining straight forward kthread API conversions...
Posted by ebiederm on Thu, 19 Apr 2007 06:52:28 GMT
View Forum Message <> Reply to Message

The following patches are against 2.6.21.rc6-mm1.
Hopefully that is enough to catch most of the recent development
activity.

I am aiming to remove all kernel threads that handle signals
from user space, to remove all calls to daemonize and kernel_thread
from non-core kernel code.

kernel thrreads handling signals from user space is a problem because
it makes the kernel thread part of the user/kernel API which make
changing things difficult and it breaks as soon as you are inside of
a pid namespace because you won't be able to see your kernel thread.

Calling kernel_thread has problems because it returns a pid_t value
which once we get to the pid namespace is context depending so it
cannot be used to globally identify a process. kernel_thread is
also a problem because it traps user space state and requires us
to call daemonize to free that state.

daemonize is a maintenance problem because every time you play with
user space state and limiting things you need to remember to update
daemonize. Occasionally it has taken years like in the case of the
mount namespace before someone realizes they need to update it.
With the kthread api we no longer need daemonize.

In addition we don't want kernel threads visible in anything but
the initial pid namespace or they will hold a reference to a
child pid namespace. However calling kernel_thread from a non-kernel
parent in a child pid namespace will give the thread a pid in
the child pid namespace, and there is nothing daemonize can do about
it. So daemonize appears impossible to support going forward, and
I choose to remove all of it's callers rather than attempt to support
it.

Eric

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

Subject: [PATCH] i386 balance_irq: Convert to the kthread api.
Posted by ebiederm on Thu, 19 Apr 2007 06:55:26 GMT

This patch just trivial converts from calling kernel_thread and daemonize
to just calling kthread_run.

Cc: Andi Kleen <ak@suse.de>
Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
---
 arch/i386/kernel/io_apic.c | 5 ++---
 1 files changed, 2 insertions(+), 3 deletions(-)

diff --git a/arch/i386/kernel/io_apic.c b/arch/i386/kernel/io_apic.c
index 24ac67c..84b412a 100644
--- a/arch/i386/kernel/io_apic.c
+++ b/arch/i386/kernel/io_apic.c
@@ -34,6 +34,7 @@
 #include <linux/msi.h>
 #include <linux/htirq.h>
 #include <linux/freezer.h>
+#include <linux/kthread.h>

 #include <asm/io.h>
 #include <asm/smp.h>
@@ -660,8 +661,6 @@ static int balanced_irq(void *unused)
  unsigned long prev_balance_time = jiffies;
  long time_remaining = balanced_irq_interval;

- daemonize("kirqd");
-
  /* push everything to CPU 0 to give us a starting point. */
  for (i = 0 ; i < NR_IRQS ; i++) {
   irq_desc[i].pending_mask = cpumask_of_cpu(0);
@@ -721,7 +720,7 @@ static int __init balanced_irq_init(void)
  }

  printk(KERN_INFO "Starting balanced_irq\n");
- if (kernel_thread(balanced_irq, NULL, CLONE_KERNEL) >= 0)
+ if (!IS_ERR(kthread_run(balanced_irq, NULL, "kirqd")))
   return 0;
  else
   printk(KERN_ERR "balanced_irq_init: failed to spawn balanced_irq");
--
1.5.0.g53756

Subject: [PATCH] i386 voyager: Convert the monitor thread to use the kthread API
Posted by ebiederm on Thu, 19 Apr 2007 06:55:27 GMT
View Forum Message <> Reply to Message

From: Eric W. Biederman <ebiederm@xmission.com> - unquoted

This patch just trivially replaces kernel_thread and daemonize
with a single call to kthread_run.

CC: James Bottomley <James.Bottomley@HansenPartnership.com>
Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
---
 arch/i386/mach-voyager/voyager_thread.c | 5 ++---
 1 files changed, 2 insertions(+), 3 deletions(-)

diff --git a/arch/i386/mach-voyager/voyager_thread.c b/arch/i386/mach-voyager/voyager_thread.c
index ebfd913..ee23d9b 100644
--- a/arch/i386/mach-voyager/voyager_thread.c
+++ b/arch/i386/mach-voyager/voyager_thread.c
@@ -23,6 +23,7 @@
 #include <linux/kmod.h>
 #include <linux/completion.h>
 #include <linux/sched.h>
+#include <linux/kthread.h>
 #include <asm/desc.h>
 #include <asm/voyager.h>
 #include <asm/vic.h>
@@ -43,7 +44,7 @@ static __u8 set_timeout = 0;
 static int __init
 voyager_thread_start(void)
 {
- if(kernel_thread(thread, NULL, CLONE_KERNEL) < 0) {
+ if (IS_ERR(kthread_run(thread, NULL, "%s", THREAD_NAME))) {
    /* This is serious, but not fatal */
    printk(KERN_ERR "Voyager: Failed to create system monitor thread!!!\n");
    return 1;
@@ -122,8 +123,6 @@ thread(void *unused)

  kvoyagerd_running = 1;

- daemonize(THREAD_NAME);
-
  set_timeout = 0;

  init_timer(&wakeup_timer);
--
1.5.0.g53756

_____

Subject: [PATCH] mtd_blkdevs: Convert to use the kthread API
Posted by ebiederm on Thu, 19 Apr 2007 06:55:28 GMT
View Forum Message <> Reply to Message

From: Eric W. Biederman <ebiederm@xmission.com> - unquoted

thread_run is used intead of kernel_thread, daemonize, and mucking
around blocking signals directly.

CC: David Woodhouse <dwmw2@infradead.org>
Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
---
 drivers/mtd/mtd_blkdevs.c | 19 +++++----------
 1 files changed, 5 insertions(+), 14 deletions(-)

diff --git a/drivers/mtd/mtd_blkdevs.c b/drivers/mtd/mtd_blkdevs.c
index db7397c..ed71d5e 100644
--- a/drivers/mtd/mtd_blkdevs.c
+++ b/drivers/mtd/mtd_blkdevs.c
@@ -21,6 +21,7 @@
 #include <linux/init.h>
 #include <linux/mutex.h>
 #include <linux/freezer.h>
+#include <linux/kthread.h>
 #include <asm/uaccess.h>

 static LIST_HEAD(blktrans_majors);
@@ -84,17 +85,6 @@ static int mtd_blktrans_thread(void *arg)
  /* we might get involved when memory gets low, so use PF_MEMALLOC */
  current->flags |= PF_MEMALLOC | PF_NOFREEZE;

- daemonize("%sd", tr->name);
-
- /* daemonize() doesn't do this for us since some kernel threads
-    actually want to deal with signals. We can't just call
-    exit_sighand() since that'll cause an oops when we finally
-    do exit. */
- spin_lock_irq(&current->sighand->siglock);
- sigfillset(&current->blocked);
- recalc_sigpending();
- spin_unlock_irq(&current->sighand->siglock);
-
  spin_lock_irq(rq->queue_lock);

```
  while (!tr->blkcore_priv->exiting) {
@@ -368,6 +358,7 @@ static struct mtd_notifier blktrans_notifier = {

 int register_mtd_blktrans(struct mtd_blktrans_ops *tr)
 {
+ struct task_struct *task;
  int ret, i;

  /* Register the notifier if/when the first device type is
@@ -406,13 +397,13 @@ int register_mtd_blktrans(struct mtd_blktrans_ops *tr)
  blk_queue_hardsect_size(tr->blkcore_priv->rq, tr->blksize);
  tr->blkshift = ffs(tr->blksize) - 1;

- ret = kernel_thread(mtd_blktrans_thread, tr, CLONE_KERNEL);
- if (ret < 0) {
+ task = kthread_run(mtd_blktrans_thread, tr, "%sd", tr->name);
+ if (IS_ERR(task)) {
    blk_cleanup_queue(tr->blkcore_priv->rq);
    unregister_blkdev(tr->major, tr->name);
    kfree(tr->blkcore_priv);
    mutex_unlock(&mtd_table_mutex);
-   return ret;
+   return PTR_ERR(task);
  }

  INIT_LIST_HEAD(&tr->devs);
--
1.5.0.g53756
```

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: [PATCH] cpci_hotplug: Convert to use the kthread API
Posted by ebiederm on Thu, 19 Apr 2007 06:55:29 GMT
View Forum Message <> Reply to Message

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.

```
CC: Scott Murray <scottm@somanetworks.com>
Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
---
 drivers/pci/hotplug/cpci_hotplug_core.c |   22 +++++++----------------
 1 files changed, 7 insertions(+), 15 deletions(-)

diff --git a/drivers/pci/hotplug/cpci_hotplug_core.c b/drivers/pci/hotplug/cpci_hotplug_core.c
index 6845515..c620c7e 100644
--- a/drivers/pci/hotplug/cpci_hotplug_core.c
+++ b/drivers/pci/hotplug/cpci_hotplug_core.c
@@ -33,6 +33,7 @@
 #include <linux/init.h>
 #include <linux/interrupt.h>
 #include <linux/smp_lock.h>
+#include <linux/kthread.h>
 #include <asm/atomic.h>
 #include <linux/delay.h>
 #include "cpci_hotplug.h"
@@ -521,17 +522,13 @@ 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))
+  if (thread_finished)
    break;
   do {
    rc = check_slots();
@@ -562,12 +559,8 @@ poll_thread(void *data)
 {
  int rc;

- lock_kernel();
- daemonize("cpci_hp_polld");
- unlock_kernel();
-
  while (1) {
-  if (thread_finished || signal_pending(current))
+  if (thread_finished)
```

```
    break;
   if (controller->ops->query_enum()) {
    do {
@@ -592,7 +585,7 @@ poll_thread(void *data)
 static int
 cpci_start_thread(void)
 {
- int pid;
+ struct task_struct *task;

  /* initialize our semaphores */
  init_MUTEX_LOCKED(&event_semaphore);
@@ -600,14 +593,13 @@ cpci_start_thread(void)
  thread_finished = 0;

  if (controller->irq)
-  pid = kernel_thread(event_thread, NULL, 0);
+  task = kthread_run(event_thread, NULL, "cpci_hp_eventd");
  else
-  pid = kernel_thread(poll_thread, NULL, 0);
- if (pid < 0) {
+  task = kthread_run(poll_thread, NULL, "cpci_hp_polld");
+ if (IS_ERR(task)) {
   err("Can't start up our thread");
   return -1;
  }
- dbg("Our thread pid = %d", pid);
  return 0;
 }

--
1.5.0.g53756
```

Subject: [PATCH] ibmphp: Convert to use the kthreads API
Posted by ebiederm on Thu, 19 Apr 2007 06:55:30 GMT
View Forum Message <> Reply to Message

From: Eric W. Biederman <ebiederm@xmission.com> - unquoted

kthread_run replaces kernel_thread and dameonize.

allow_signal is unnecessary and has been removed.

tid_poll was unused and has been removed.

Cc: Jyoti Shah <jshah@us.ibm.com>
Cc: Greg Kroah-Hartman <gregkh@suse.de>
Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
---
 drivers/pci/hotplug/ibmphp_hpc.c | 14 +++++---------
 1 files changed, 5 insertions(+), 9 deletions(-)

diff --git a/drivers/pci/hotplug/ibmphp_hpc.c b/drivers/pci/hotplug/ibmphp_hpc.c
index 46abaa8..27e12f1 100644
--- a/drivers/pci/hotplug/ibmphp_hpc.c
+++ b/drivers/pci/hotplug/ibmphp_hpc.c
@@ -34,6 +34,7 @@
 #include <linux/pci.h>
 #include <linux/init.h>
 #include <linux/mutex.h>
+#include <linux/kthread.h>

 #include "ibmphp.h"

@@ -101,7 +102,6 @@ static int to_debug = 0;
 // global variables
 //--
 static int ibmphp_shutdown;
-static int tid_poll;
 static struct mutex sem_hpcaccess; // lock access to HPC
 static struct semaphore semOperations; // lock all operations and
     // access to data structures
@@ -137,7 +137,6 @@ void __init ibmphp_hpc_initvars (void)
  init_MUTEX_LOCKED (&sem_exit);
  to_debug = 0;
  ibmphp_shutdown = 0;
- tid_poll = 0;

  debug ("%s - Exit\n", __FUNCTION__);
 }
@@ -1060,12 +1059,8 @@ static int hpc_poll_thread (void *data)
 {
  debug ("%s - Entry\n", __FUNCTION__);

- daemonize("hpc_poll");
- allow_signal(SIGKILL);
-
  poll_hpc ();

- tid_poll = 0;
  debug ("%s - Exit\n", __FUNCTION__);

```
  return 0;
 }
@@ -1078,17 +1073,18 @@ static int hpc_poll_thread (void *data)
 *-----------------------------------------------------------------------*/
 int __init ibmphp_hpc_start_poll_thread (void)
 {
+ struct task_struct *task;
  int rc = 0;

  debug ("%s - Entry\n", __FUNCTION__);

- tid_poll = kernel_thread (hpc_poll_thread, NULL, 0);
- if (tid_poll < 0) {
+ task = kthread_run(hpc_poll_thread, NULL, "hpc_poll");
+ if (IS_ERR(task)) {
   err ("%s - Error, thread not started\n", __FUNCTION__);
   rc = -1;
  }

- debug ("%s - Exit tid_poll[%d] rc[%d]\n", __FUNCTION__, tid_poll, rc);
+ debug ("%s - Exit rc[%d]\n", __FUNCTION__, rc);
  return rc;
 }

--
1.5.0.g53756
```

Subject: [PATCH] cpqphp: Convert to use the kthread API
Posted by ebiederm on Thu, 19 Apr 2007 06:55:31 GMT
View Forum Message <> Reply to Message

From: Eric W. Biederman <ebiederm@xmission.com> - unquoted

This patch changes cpqphp to use kthread_run and not
kernel_thread and daemonize to startup and setup
the cpqphp thread.

Cc: Greg Kroah-Hartman <gregkh@suse.de>
Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
---
 drivers/pci/hotplug/cpqphp_ctrl.c | 12 ++++++--------
 1 files changed, 4 insertions(+), 8 deletions(-)

```
diff --git a/drivers/pci/hotplug/cpqphp_ctrl.c b/drivers/pci/hotplug/cpqphp_ctrl.c
index 79ff6b4..c2c06c4 100644
--- a/drivers/pci/hotplug/cpqphp_ctrl.c
+++ b/drivers/pci/hotplug/cpqphp_ctrl.c
@@ -37,6 +37,7 @@
 #include <linux/smp_lock.h>
 #include <linux/pci.h>
 #include <linux/pci_hotplug.h>
+#include <linux/kthread.h>
 #include "cpqphp.h"

 static u32 configure_new_device(struct controller* ctrl, struct pci_func *func,
@@ -1746,10 +1747,6 @@ static void pushbutton_helper_thread(unsigned long data)
 static int event_thread(void* data)
 {
  struct controller *ctrl;
- lock_kernel();
- daemonize("phpd_event");
-
- unlock_kernel();

  while (1) {
   dbg("!!!!event_thread sleeping\n");
@@ -1771,7 +1768,7 @@ static int event_thread(void* data)

 int cpqhp_event_start_thread(void)
 {
- int pid;
+ struct task_struct *task;

  /* initialize our semaphores */
  init_MUTEX(&delay_sem);
@@ -1779,12 +1776,11 @@ int cpqhp_event_start_thread(void)
  init_MUTEX_LOCKED(&event_exit);
  event_finished=0;

- pid = kernel_thread(event_thread, NULL, 0);
- if (pid < 0) {
+ task = kthread_run(event_thread, NULL, "phpd_event");
+ if (IS_ERR(task)) {
   err ("Can't start up our event thread\n");
   return -1;
  }
- dbg("Our event thread pid = %d\n", pid);
  return 0;
 }
```

--
1.5.0.g53756

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: [PATCH] pnpbios: Conert to use the kthread API.
Posted by ebiederm on Thu, 19 Apr 2007 06:55:32 GMT
View Forum Message <> Reply to Message

From: Eric W. Biederman <ebiederm@xmission.com> - unquoted

This patches modifies the pnpbios kernel thread to start
with ktrhead_run not kernel_thread and deamonize. Doing
this makes the code a little simpler and easier to maintain.

Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
---
 drivers/pnp/pnpbios/core.c | 16 +++++++---------
 1 files changed, 7 insertions(+), 9 deletions(-)

diff --git a/drivers/pnp/pnpbios/core.c b/drivers/pnp/pnpbios/core.c
index c2ed53f..3a201b7 100644
--- a/drivers/pnp/pnpbios/core.c
+++ b/drivers/pnp/pnpbios/core.c
@@ -62,6 +62,7 @@
 #include <linux/delay.h>
 #include <linux/acpi.h>
 #include <linux/freezer.h>
+#include <linux/kthread.h>

 #include <asm/page.h>
 #include <asm/desc.h>
@@ -159,9 +160,7 @@ static int pnp_dock_thread(void * unused)
 {
  static struct pnp_docking_station_info now;
  int docked = -1, d = 0;
- daemonize("kpnpbiosd");
- allow_signal(SIGKILL);
- while(!unloading && !signal_pending(current))
+ while (!unloading)
  {
   int status;

@@ -170,11 +169,8 @@ static int pnp_dock_thread(void * unused)

```
    */
   msleep_interruptible(2000);

-  if(signal_pending(current)) {
-   if (try_to_freeze())
-    continue;
-   break;
-  }
+  if (try_to_freeze())
+   continue;

   status = pnp_bios_dock_station_info(&now);

@@ -582,6 +578,7 @@ subsys_initcall(pnpbios_init);

 static int __init pnpbios_thread_init(void)
 {
+ struct task_struct *task;
 #if defined(CONFIG_PPC_MERGE)
  if (check_legacy_ioport(PNPBIOS_BASE))
   return 0;
@@ -590,7 +587,8 @@ static int __init pnpbios_thread_init(void)
   return 0;
 #ifdef CONFIG_HOTPLUG
  init_completion(&unload_sem);
- if (kernel_thread(pnp_dock_thread, NULL, CLONE_KERNEL) > 0)
+ task = kthread_run(pnp_dock_thread, NULL, "kpnpbiosd");
+ if (!IS_ERR(task))
   unloading = 0;
 #endif
  return 0;
--
1.5.0.g53756
```

Subject: [PATCH] sas_scsi_host: Convert to use the kthread API
Posted by ebiederm on Thu, 19 Apr 2007 06:55:33 GMT
View Forum Message <> Reply to Message

From: Eric W. Biederman <ebiederm@xmission.com> - unquoted

This patch modifies the sas scsi host thread startup
to use kthread_run not kernel_thread and deamonize.

kthread_run is slightly simpler and more maintainable.

Cc: Darrick J. Wong <djwong@us.ibm.com>
Cc: James Bottomley <James.Bottomley@SteelEye.com>
Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
---
 drivers/scsi/libsas/sas_scsi_host.c | 11 +++++++-----
 1 files changed, 6 insertions(+), 5 deletions(-)

```
diff --git a/drivers/scsi/libsas/sas_scsi_host.c b/drivers/scsi/libsas/sas_scsi_host.c
index 46ba3a7..7a38ac5 100644
--- a/drivers/scsi/libsas/sas_scsi_host.c
+++ b/drivers/scsi/libsas/sas_scsi_host.c
@@ -40,6 +40,7 @@
 #include <linux/blkdev.h>
 #include <linux/scatterlist.h>
 #include <linux/freezer.h>
+#include <linux/kthread.h>

 /* ---------- SCSI Host glue ---------- */

@@ -870,7 +871,6 @@ static int sas_queue_thread(void *_sas_ha)
  struct sas_ha_struct *sas_ha = _sas_ha;
  struct scsi_core *core = &sas_ha->core;

- daemonize("sas_queue_%d", core->shost->host_no);
  current->flags |= PF_NOFREEZE;

  complete(&queue_th_comp);
@@ -891,19 +891,20 @@ static int sas_queue_thread(void *_sas_ha)

 int sas_init_queue(struct sas_ha_struct *sas_ha)
 {
- int res;
  struct scsi_core *core = &sas_ha->core;
+ struct task_struct *task;

  spin_lock_init(&core->task_queue_lock);
  core->task_queue_size = 0;
  INIT_LIST_HEAD(&core->task_queue);
  init_MUTEX_LOCKED(&core->queue_thread_sema);

- res = kernel_thread(sas_queue_thread, sas_ha, 0);
- if (res >= 0)
+ task = kthread_run(sas_queue_thread, sas_ha,
+       "sas_queue_%d", core->shost->host_no);
+ if (!IS_ERR(task))
   wait_for_completion(&queue_th_comp);
```

```
- return res < 0 ? res : 0;
+ return IS_ERR(task) ? PTR_ERR(task) : 0;
 }

 void sas_shutdown_queue(struct sas_ha_struct *sas_ha)
--
1.5.0.g53756
```

Subject: [PATCH] sparc64/power.c: Convert to use the kthread API
Posted by ebiederm on Thu, 19 Apr 2007 06:55:34 GMT
View Forum Message <> Reply to Message

From: Eric W. Biederman <ebiederm@xmission.com> - unquoted

This starts the sparc64 powerd using kthread_run
instead of kernel_thread and daemonize. Making the
code slightly simpler and more maintainable.

In addition the unnecessary flush_signals is removed.

Cc: David S. Miller <davem@davemloft.net>
Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
---
 arch/sparc64/kernel/power.c | 8 ++++----
 1 files changed, 4 insertions(+), 4 deletions(-)

```
diff --git a/arch/sparc64/kernel/power.c b/arch/sparc64/kernel/power.c
index 699b24b..03feb8b 100644
--- a/arch/sparc64/kernel/power.c
+++ b/arch/sparc64/kernel/power.c
@@ -13,6 +13,7 @@
 #include <linux/interrupt.h>
 #include <linux/pm.h>
 #include <linux/syscalls.h>
+#include <linux/kthread.h>

 #include <asm/system.h>
 #include <asm/auxio.h>
@@ -81,15 +82,12 @@ static int powerd(void *__unused)
  char *argv[] = { "/sbin/shutdown", "-h", "now", NULL };
  DECLARE_WAITQUEUE(wait, current);
```

```
- daemonize("powerd");
-
  add_wait_queue(&powerd_wait, &wait);
 again:
  for (;;) {
   set_task_state(current, TASK_INTERRUPTIBLE);
   if (button_pressed)
    break;
-  flush_signals(current);
   schedule();
  }
  __set_current_state(TASK_RUNNING);
@@ -128,7 +126,9 @@ static int __devinit power_probe(struct of_device *op, const struct
of_device_id
  poweroff_method = machine_halt;  /* able to use the standard halt */

  if (has_button_interrupt(irq, op->node)) {
-  if (kernel_thread(powerd, NULL, CLONE_FS) < 0) {
+  struct task_struct *task;
+  task = kthread_urn(powerd, NULL, "powerd");
+  if (IS_ERR(task)) {
    printk("Failed to start power daemon.\n");
    return 0;
   }
--
1.5.0.g53756
```

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: [PATCH] s390/net/lcs: Convert to the kthread API
Posted by ebiederm on Thu, 19 Apr 2007 06:55:35 GMT
View Forum Message <> Reply to Message

From: Eric W. Biederman <ebiederm@xmission.com> - unquoted

Use kthread_run to start the lcs kernel threads not a
combination of kernel_thread and daemonize. This makes
the code slightly simpler and more maintainable.

Cc: Frank Pavlic <fpavlic@de.ibm.com>
Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
---
 drivers/s390/net/lcs.c | 8 ++++-----

```
 1 files changed, 3 insertions(+), 5 deletions(-)

diff --git a/drivers/s390/net/lcs.c b/drivers/s390/net/lcs.c
index 08a994f..0300d87 100644
--- a/drivers/s390/net/lcs.c
+++ b/drivers/s390/net/lcs.c
@@ -36,6 +36,7 @@
 #include <linux/in.h>
 #include <linux/igmp.h>
 #include <linux/delay.h>
+#include <linux/kthread.h>
 #include <net/arp.h>
 #include <net/ip.h>

@@ -1248,7 +1249,6 @@ lcs_register_mc_addresses(void *data)
  struct in_device *in4_dev;

  card = (struct lcs_card *) data;
- daemonize("regipm");

  if (!lcs_do_run_thread(card, LCS_SET_MC_THREAD))
   return 0;
@@ -1728,11 +1728,10 @@ lcs_start_kernel_thread(struct work_struct *work)
  struct lcs_card *card = container_of(work, struct lcs_card, kernel_thread_starter);
  LCS_DBF_TEXT(5, trace, "krnthrd");
  if (lcs_do_start_thread(card, LCS_RECOVERY_THREAD))
-  kernel_thread(lcs_recovery, (void *) card, SIGCHLD);
+  kthread_run(lcs_recovery, card, "lcs_recover");
 #ifdef CONFIG_IP_MULTICAST
  if (lcs_do_start_thread(card, LCS_SET_MC_THREAD))
-  kernel_thread(lcs_register_mc_addresses,
-    (void *) card, SIGCHLD);
+  kernel_run(lcs_register_mc_addresses, card, "regipm");
 #endif
 }

@@ -2232,7 +2231,6 @@ lcs_recovery(void *ptr)
        int rc;

  card = (struct lcs_card *) ptr;
- daemonize("lcs_recover");

  LCS_DBF_TEXT(4, trace, "recover1");
  if (!lcs_do_run_thread(card, LCS_RECOVERY_THREAD))
--
1.5.0.g53756
```

Subject: [PATCH] s390 qeth: Convert to use the kthread API
Posted by ebiederm on Thu, 19 Apr 2007 06:55:36 GMT
View Forum Message <> Reply to Message

From: Eric W. Biederman <ebiederm@xmission.com> - unquoted

This patch modifies the qeth_recover thread to be started
with kthread_run not a combination of kernel_thread and
daemonize. Resulting in slightly simpler and more maintainable
code.

Cc: Frank Pavlic <fpavlic@de.ibm.com>
Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
---
 drivers/s390/net/qeth_main.c | 4 ++--
 1 files changed, 2 insertions(+), 2 deletions(-)

diff --git a/drivers/s390/net/qeth_main.c b/drivers/s390/net/qeth_main.c
index ad7792d..8234846 100644
--- a/drivers/s390/net/qeth_main.c
+++ b/drivers/s390/net/qeth_main.c
@@ -50,6 +50,7 @@
 #include <linux/mii.h>
 #include <linux/rcupdate.h>
 #include <linux/ethtool.h>
+#include <linux/kthread.h>

 #include <net/arp.h>
 #include <net/ip.h>
@@ -957,7 +958,6 @@ qeth_recover(void *ptr)
  int rc = 0;

  card = (struct qeth_card *) ptr;
- daemonize("qeth_recover");
  QETH_DBF_TEXT(trace,2,"recover1");
  QETH_DBF_HEX(trace, 2, &card, sizeof(void *));
  if (!qeth_do_run_thread(card, QETH_RECOVER_THREAD))
@@ -1014,7 +1014,7 @@ qeth_start_kernel_thread(struct work_struct *work)
      card->write.state != CH_STATE_UP)
   return;
  if (qeth_do_start_thread(card, QETH_RECOVER_THREAD))
- kernel_thread(qeth_recover, (void *) card, SIGCHLD);
+ kthread_run(qeth_recover, card, "qeth_recover");

}

--
1.5.0.g53756

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: [PATCH] s390/scsi/zfcp_erp: Convert to use the kthread API
Posted by ebiederm on Thu, 19 Apr 2007 06:55:37 GMT
View Forum Message <> Reply to Message

From: Eric W. Biederman <ebiederm@xmission.com> - unquoted

Modify zfcperp%s to be started with kthread_run not
a combination of kernel_thread, daemonize and siginitsetinv
making the code slightly simpler and more maintainable.

Cc: Swen Schillig <swen@vnet.ibm.com>
Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
---
 drivers/s390/scsi/zfcp_erp.c | 13 ++++++-------
 1 files changed, 6 insertions(+), 7 deletions(-)

diff --git a/drivers/s390/scsi/zfcp_erp.c b/drivers/s390/scsi/zfcp_erp.c
index 66c0b09..f26536d 100644
--- a/drivers/s390/scsi/zfcp_erp.c
+++ b/drivers/s390/scsi/zfcp_erp.c
@@ -21,6 +21,7 @@

 #define ZFCP_LOG_AREA   ZFCP_LOG_AREA_ERP

+#include <linux/kthread.h>
 #include "zfcp_ext.h"

 static int zfcp_erp_adisc(struct zfcp_port *);
@@ -985,12 +986,13 @@ static void zfcp_erp_action_dismiss(struct zfcp_erp_action
*erp_action)
 int
 zfcp_erp_thread_setup(struct zfcp_adapter *adapter)
 {
- int retval = 0;
+ struct task_struct *task;

```
  atomic_clear_mask(ZFCP_STATUS_ADAPTER_ERP_THREAD_UP, &adapter->status);

- retval = kernel_thread(zfcp_erp_thread, adapter, SIGCHLD);
- if (retval < 0) {
+ task = kthread_run(zfcp_erp_thread, adapter,
+       "zfcperp%s", zfcp_get_busid_by_adapter(adapter));
+ if (IS_ERR(task)) {
    ZFCP_LOG_NORMAL("error: creation of erp thread failed for "
      "adapter %s\n",
      zfcp_get_busid_by_adapter(adapter));
@@ -1002,7 +1004,7 @@ zfcp_erp_thread_setup(struct zfcp_adapter *adapter)
    debug_text_event(adapter->erp_dbf, 5, "a_thset_ok");
  }

- return (retval < 0);
+ return IS_ERR(task);
 }

 /*
@@ -1054,9 +1056,6 @@ zfcp_erp_thread(void *data)
  struct zfcp_erp_action *erp_action;
  unsigned long flags;

- daemonize("zfcperp%s", zfcp_get_busid_by_adapter(adapter));
- /* Block all signals */
- siginitsetinv(&current->blocked, 0);
  atomic_set_mask(ZFCP_STATUS_ADAPTER_ERP_THREAD_UP, &adapter->status);
  debug_text_event(adapter->erp_dbf, 5, "a_th_run");
  wake_up(&adapter->erp_thread_wqh);
--
1.5.0.g53756
```

Subject: [PATCH] arm ecard: Conver to use the kthread API.
Posted by ebiederm on Thu, 19 Apr 2007 06:55:38 GMT
View Forum Message <> Reply to Message

From: Eric W. Biederman <ebiederm@xmission.com> - unquoted

This patch modifies the startup of kecardd to use
kthread_run not a kernel_thread combination of kernel_thread
and daemonize. Making the code slightly simpler and more
maintainable.

```
Cc: Russell King <rmk+kernel@arm.linux.org.uk>
Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
---
 arch/arm/kernel/ecard.c |   14 +++++++-------
 1 files changed, 7 insertions(+), 7 deletions(-)

diff --git a/arch/arm/kernel/ecard.c b/arch/arm/kernel/ecard.c
index f1c0fb9..6c15f5f 100644
--- a/arch/arm/kernel/ecard.c
+++ b/arch/arm/kernel/ecard.c
@@ -40,6 +40,7 @@
 #include <linux/device.h>
 #include <linux/init.h>
 #include <linux/mutex.h>
+#include <linux/kthread.h>

 #include <asm/dma.h>
 #include <asm/ecard.h>
@@ -263,8 +264,6 @@ static int ecard_init_mm(void)
 static int
 ecard_task(void * unused)
 {
- daemonize("kecardd");
-
   /*
    * Allocate a mm.  We're not a lazy-TLB kernel task since we need
    * to set page table entries where the user space would be.  Note
@@ -1058,13 +1057,14 @@ ecard_probe(int slot, card_type_t type)
  */
 static int __init ecard_init(void)
 {
- int slot, irqhw, ret;
+ struct task_struct *task;
+ int slot, irqhw;

- ret = kernel_thread(ecard_task, NULL, CLONE_KERNEL);
- if (ret < 0) {
+ task = kthread_run(ecard_task, NULL, "kecardd");
+ if (IS_ERR(task)) {
    printk(KERN_ERR "Ecard: unable to create kernel thread: %d\n",
-        ret);
-  return ret;
+         PTR_ERR(task));
+  return PTR_ERR(task);
   }

   printk("Probing expansion cards\n");
```

--
1.5.0.g53756

____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: [PATCH] ia64 sn xpc: Convert to use kthread API.
Posted by ebiederm on Thu, 19 Apr 2007 06:55:39 GMT
View Forum Message <> Reply to Message

From: Eric W. Biederman <ebiederm@xmission.com> - unquoted

This patch starts the xpc kernel threads using kthread_run
not a combination of kernel_thread and daemonize. Resuling
in slightly simpler and more maintainable code.

Cc: Jes Sorensen <jes@sgi.com>
Cc: Tony Luck <tony.luck@intel.com>
Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>

```
---
 arch/ia64/sn/kernel/xpc_main.c |   31 +++++++++++++------------------
 1 files changed, 13 insertions(+), 18 deletions(-)

diff --git a/arch/ia64/sn/kernel/xpc_main.c b/arch/ia64/sn/kernel/xpc_main.c
index e336e16..5b53642 100644
--- a/arch/ia64/sn/kernel/xpc_main.c
+++ b/arch/ia64/sn/kernel/xpc_main.c
@@ -56,6 +56,7 @@
 #include <linux/reboot.h>
 #include <linux/completion.h>
 #include <linux/kdebug.h>
+#include <linux/kthread.h>
 #include <asm/sn/intr.h>
 #include <asm/sn/sn_sal.h>
 #include <asm/uaccess.h>
@@ -253,8 +254,6 @@ xpc_hb_checker(void *ignore)

  /* this thread was marked active by xpc_hb_init() */

- daemonize(XPC_HB_CHECK_THREAD_NAME);
-
  set_cpus_allowed(current, cpumask_of_cpu(XPC_HB_CHECK_CPU));

  xpc_hb_check_timeout = jiffies + (xpc_hb_check_interval * HZ);
@@ -324,8 +323,6 @@ xpc_hb_checker(void *ignore)
```

```
 static int
 xpc_initiate_discovery(void *ignore)
 {
- daemonize(XPC_DISCOVERY_THREAD_NAME);
-
  xpc_discovery();

  dev_dbg(xpc_part, "discovery thread is exiting\n");
@@ -494,8 +491,6 @@ xpc_activating(void *__partid)

  dev_dbg(xpc_part, "bringing partition %d up\n", partid);

- daemonize("xpc%02d", partid);
-
  /*
   * This thread needs to run at a realtime priority to prevent a
   * significant performance degradation.
@@ -559,7 +554,7 @@ xpc_activate_partition(struct xpc_partition *part)
 {
  partid_t partid = XPC_PARTID(part);
  unsigned long irq_flags;
- pid_t pid;
+ struct task_struct *task;


  spin_lock_irqsave(&part->act_lock, irq_flags);
@@ -571,9 +566,10 @@ xpc_activate_partition(struct xpc_partition *part)

  spin_unlock_irqrestore(&part->act_lock, irq_flags);

- pid = kernel_thread(xpc_activating, (void *) ((u64) partid), 0);
+ task = kthread_run(xpc_activating, (void *) ((u64) partid),
+     "xpc%02d", partid);

- if (unlikely(pid <= 0)) {
+ if (unlikely(IS_ERR(task))) {
   spin_lock_irqsave(&part->act_lock, irq_flags);
   part->act_state = XPC_P_INACTIVE;
   XPC_SET_REASON(part, xpcCloneKThreadFailed, __LINE__);
@@ -724,8 +720,6 @@ xpc_daemonize_kthread(void *args)
  unsigned long irq_flags;


- daemonize("xpc%02dc%d", partid, ch_number);
-
  dev_dbg(xpc_chan, "kthread starting, partid=%d, channel=%d\n",
   partid, ch_number);
```

```
@@ -844,8 +838,9 @@ xpc_create_kthreads(struct xpc_channel *ch, int needed,
   (void) xpc_part_ref(part);
   xpc_msgqueue_ref(ch);

-  pid = kernel_thread(xpc_daemonize_kthread, (void *) args, 0);
-  if (pid < 0) {
+  task = kthread_run(xpc_daemonize_kthread, args,
+       "xpc%02dc%d", partid, ch_number);
+  if (IS_ERR(task)) {
    /* the fork failed */

    /*
@@ -1222,7 +1217,7 @@ xpc_init(void)
  int ret;
  partid_t partid;
  struct xpc_partition *part;
- pid_t pid;
+ struct task_struct *task;
  size_t buf_size;


@@ -1353,8 +1348,8 @@ xpc_init(void)
   * The real work-horse behind xpc.  This processes incoming
   * interrupts and monitors remote heartbeats.
   */
- pid = kernel_thread(xpc_hb_checker, NULL, 0);
- if (pid < 0) {
+ task = kthread_run(xpc_hb_checker, NULL, XPC_HB_CHECK_THREAD_NAME);
+ if (IS_ERR(task)) {
   dev_err(xpc_part, "failed while forking hb check thread\n");

   /* indicate to others that our reserved page is uninitialized */
@@ -1384,8 +1379,8 @@ xpc_init(void)
   * activate based on info provided by SAL. This new thread is short
   * lived and will exit once discovery is complete.
   */
- pid = kernel_thread(xpc_initiate_discovery, NULL, 0);
- if (pid < 0) {
+ task = kthread_run(xpc_initiate_discovery, NULL, XPC_DISCOVERY_THREAD_NAME);
+ if (IS_ERR(task)) {
   dev_err(xpc_part, "failed while forking discovery thread\n");

   /* mark this new thread as a non-starter */
--
1.5.0.g53756
```

_____
Containers mailing list

Subject: [PATCH] powerpc pseries eeh: Convert to kthread API
Posted by ebiederm on Thu, 19 Apr 2007 06:55:40 GMT
View Forum Message <> Reply to Message

From: Eric W. Biederman <ebiederm@xmission.com> - unquoted

This patch modifies the startup of eehd to use kthread_run
not a combination of kernel_thread and daemonize. Making
the code slightly simpler and more maintainable.

Cc: Paul Mackerras <paulus@samba.org>
Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
---
 arch/powerpc/platforms/pseries/eeh_event.c | 4 ++--
 1 files changed, 2 insertions(+), 2 deletions(-)

diff --git a/arch/powerpc/platforms/pseries/eeh_event.c
b/arch/powerpc/platforms/pseries/eeh_event.c
index 221dec8..fe7c2e0 100644
--- a/arch/powerpc/platforms/pseries/eeh_event.c
+++ b/arch/powerpc/platforms/pseries/eeh_event.c
@@ -23,6 +23,7 @@
 #include <linux/mutex.h>
 #include <linux/pci.h>
 #include <linux/workqueue.h>
+#include <linux/kthread.h>
 #include <asm/eeh_event.h>
 #include <asm/ppc-pci.h>

@@ -59,7 +60,6 @@ static int eeh_event_handler(void * dummy)
  struct eeh_event *event;
  struct pci_dn *pdn;

- daemonize ("eehd");
  set_current_state(TASK_INTERRUPTIBLE);

  spin_lock_irqsave(&eeh_eventlist_lock, flags);
@@ -105,7 +105,7 @@ static int eeh_event_handler(void * dummy)
  */
 static void eeh_thread_launcher(struct work_struct *dummy)
 {
- if (kernel_thread(eeh_event_handler, NULL, CLONE_KERNEL) < 0)
+ if (IS_ERR(kthread_run(eeh_event_handler, NULL, "eehd")))
   printk(KERN_ERR "Failed to start EEH daemon\n");

```
 }

--
1.5.0.g53756
```

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: [PATCH] powerpc pseries rtasd: Convert to kthread API.
Posted by ebiederm on Thu, 19 Apr 2007 06:55:41 GMT
View Forum Message <> Reply to Message

From: Eric W. Biederman <ebiederm@xmission.com> - unquoted

This patch modifies the startup of rtasd to use kthread_run instaed of
a combination of kernel_thread and daemonize. Making the code a little
simpler and more maintainble.

Cc: Paul Mackerras <paulus@samba.org>
Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
---
 arch/powerpc/platforms/pseries/rtasd.c | 5 ++---
 1 files changed, 2 insertions(+), 3 deletions(-)

diff --git a/arch/powerpc/platforms/pseries/rtasd.c b/arch/powerpc/platforms/pseries/rtasd.c
index 77d0937..919a374 100644
--- a/arch/powerpc/platforms/pseries/rtasd.c
+++ b/arch/powerpc/platforms/pseries/rtasd.c
@@ -20,6 +20,7 @@
 #include <linux/spinlock.h>
 #include <linux/cpu.h>
 #include <linux/delay.h>
+#include <linux/kthread.h>

 #include <asm/uaccess.h>
 #include <asm/io.h>
@@ -429,8 +430,6 @@ static int rtasd(void *unused)
  int event_scan = rtas_token("event-scan");
  int rc;

- daemonize("rtasd");
-
  if (event_scan == RTAS_UNKNOWN_SERVICE || get_eventscan_parms() == -1)
   goto error;

```
@@ -497,7 +496,7 @@ static int __init rtas_init(void)
  else
   printk(KERN_ERR "Failed to create error_log proc entry\n");

- if (kernel_thread(rtasd, NULL, CLONE_FS) < 0)
+ if (IS_ERR(kthread_run(rtasd, NULL, "rtasd")))
   printk(KERN_ERR "Failed to start RTAS daemon\n");

  return 0;
--
1.5.0.g53756
```

Subject: [PATCH] macintosh/therm_pm72.c: Convert to kthread API.
Posted by ebiederm on Thu, 19 Apr 2007 06:55:42 GMT
View Forum Message <> Reply to Message

From: Eric W. Biederman <ebiederm@xmission.com> - unquoted

This patch modifies startup of the kfand to use kthread_run
not a combination of kernel_thread and daemonize, making
the code a little simpler and more maintaintable.

Cc: Benjamin Herrenschmidt <benh@kernel.crashing.org>
Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
---
 drivers/macintosh/therm_pm72.c | 11 ++++++-----
 1 files changed, 6 insertions(+), 5 deletions(-)

```
diff --git a/drivers/macintosh/therm_pm72.c b/drivers/macintosh/therm_pm72.c
index b002a4b..7e9cbb7 100644
--- a/drivers/macintosh/therm_pm72.c
+++ b/drivers/macintosh/therm_pm72.c
@@ -121,6 +121,7 @@
 #include <linux/reboot.h>
 #include <linux/kmod.h>
 #include <linux/i2c.h>
+#include <linux/kthread.h>
 #include <asm/prom.h>
 #include <asm/machdep.h>
 #include <asm/io.h>
@@ -161,7 +162,7 @@ static struct slots_pid_state  slots_state;
 static int    state;
```

```
 static int    cpu_count;
 static int    cpu_pid_type;
-static pid_t    ctrl_task;
+static int    ctrl_task;
 static struct completion  ctrl_complete;
 static int    critical_state;
 static int    rackmac;
@@ -1779,8 +1780,6 @@ static int call_critical_overtemp(void)
  */
 static int main_control_loop(void *x)
 {
- daemonize("kfand");
-
  DBG("main_control_loop started\n");

  down(&driver_lock);
@@ -1859,7 +1858,6 @@ static int main_control_loop(void *x)
    machine_power_off();
   }

-  // FIXME: Deal with signals
   elapsed = jiffies - start;
   if (elapsed < HZ)
    schedule_timeout_interruptible(HZ - elapsed);
@@ -1954,9 +1952,12 @@ static int create_control_loops(void)
  */
 static void start_control_loops(void)
 {
+ struct task_struct *task;
  init_completion(&ctrl_complete);

- ctrl_task = kernel_thread(main_control_loop, NULL, SIGCHLD | CLONE_KERNEL);
+ task = kthread_run(main_control_loop, NULL, "kfand");
+ if (!IS_ERR(task))
+  ctrl_task = 1;
 }

 /*
--
1.5.0.g53756
```

Subject: [PATCH] macintosh/therm_windtunnel.c: Convert to kthread API.
Posted by ebiederm on Thu, 19 Apr 2007 06:55:43 GMT
View Forum Message <> Reply to Message

From: Eric W. Biederman <ebiederm@xmission.com> - unquoted

Start the g4fand using kthread_run not a combination
of kernel_thread and deamonize. This makes the code
a little simpler and more maintainable.

Cc: Benjamin Herrenschmidt <benh@kernel.crashing.org>
Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
---
drivers/macintosh/therm_windtunnel.c | 5 ++---
1 files changed, 2 insertions(+), 3 deletions(-)

```
diff --git a/drivers/macintosh/therm_windtunnel.c b/drivers/macintosh/therm_windtunnel.c
index a1d3a98..5d888e7 100644
--- a/drivers/macintosh/therm_windtunnel.c
+++ b/drivers/macintosh/therm_windtunnel.c
@@ -36,6 +36,7 @@
 #include <linux/i2c.h>
 #include <linux/slab.h>
 #include <linux/init.h>
+#include <linux/kthread.h>

 #include <asm/prom.h>
 #include <asm/machdep.h>
@@ -62,7 +63,6 @@ I2C_CLIENT_INSMOD;
 static struct {
  volatile int  running;
  struct completion completion;
- pid_t   poll_task;

  struct semaphore  lock;
  struct of_device *of_dev;
@@ -285,7 +285,6 @@ restore_regs( void )
 static int
 control_loop( void *dummy )
 {
- daemonize("g4fand");

  down( &x.lock );
  setup_hardware();
@@ -323,7 +322,7 @@ do_attach( struct i2c_adapter *adapter )
   if( x.thermostat && x.fan ) {
    x.running = 1;
    init_completion( &x.completion );
-   x.poll_task = kernel_thread( control_loop, NULL, SIGCHLD | CLONE_KERNEL );
```

```
+   kthread_run( control_loop, NULL, "g4fand");
  }
 }
 return ret;
--
1.5.0.g53756
```

Subject: [PATCH] macintosh/adb: Convert to the kthread API
Posted by ebiederm on Thu, 19 Apr 2007 06:55:44 GMT
View Forum Message <> Reply to Message

From: Eric W. Biederman <ebiederm@xmission.com> - unquoted

This patch modifies the startup of kadbprobe to use
kthread_run instead of scheduling a work event which
later calls kernel_thread and in the thread calls
daemonize and blocks signals. kthread_run is simpler
and more maintainable.

The variable pid_t adb_probe_task_pid is replaced by
a struct task_struct variable named adb_probe_task.
Which works equally well with for testing if the current
process is the adb_probe thread, does not get confused
in the presence of a pid namespace and is easier to
compare against current as it is the same type.

The result is code that is slightly simpler and easier
to maintain.

Cc: Benjamin Herrenschmidt <benh@kernel.crashing.org>
Cc: Paul Mackerras <paulus@samba.org>
Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
---
 drivers/macintosh/adb.c | 32 +++++++------------------------
 1 files changed, 7 insertions(+), 25 deletions(-)

diff --git a/drivers/macintosh/adb.c b/drivers/macintosh/adb.c
index adfea3c..09c5261 100644
--- a/drivers/macintosh/adb.c
+++ b/drivers/macintosh/adb.c
@@ -35,6 +35,7 @@
 #include <linux/spinlock.h>

```
 #include <linux/completion.h>
 #include <linux/device.h>
+#include <linux/kthread.h>

 #include <asm/uaccess.h>
 #include <asm/semaphore.h>
@@ -82,7 +83,7 @@ struct adb_driver *adb_controller;
 BLOCKING_NOTIFIER_HEAD(adb_client_list);
 static int adb_got_sleep;
 static int adb_inited;
-static pid_t adb_probe_task_pid;
+static struct task_struct *adb_probe_task;
 static DECLARE_MUTEX(adb_probe_mutex);
 static struct completion adb_probe_task_comp;
 static int sleepy_trackpad;
@@ -137,8 +138,7 @@ static void printADBreply(struct adb_request *req)

 static __inline__ void adb_wait_ms(unsigned int ms)
 {
- if (current->pid && adb_probe_task_pid &&
-     adb_probe_task_pid == current->pid)
+ if (adb_probe_task == current)
   msleep(ms);
  else
   mdelay(ms);
@@ -245,35 +245,19 @@ static int adb_scan_bus(void)
  * This kernel task handles ADB probing. It dies once probing is
  * completed.
  */
-static int
-adb_probe_task(void *x)
+static int adb_probe(void *x)
 {
- sigset_t blocked;
-
- strcpy(current->comm, "kadbprobe");
-
- sigfillset(&blocked);
- sigprocmask(SIG_BLOCK, &blocked, NULL);
- flush_signals(current);

  printk(KERN_INFO "adb: starting probe task...\n");
  do_adb_reset_bus();
  printk(KERN_INFO "adb: finished probe task...\n");

- adb_probe_task_pid = 0;
+ adb_probe_task = NULL;
  up(&adb_probe_mutex);
```

```
  return 0;
 }

-static void
-__adb_probe_task(struct work_struct *bullshit)
-{
- adb_probe_task_pid = kernel_thread(adb_probe_task, NULL, SIGCHLD | CLONE_KERNEL);
-}
-
-static DECLARE_WORK(adb_reset_work, __adb_probe_task);
-
 int
 adb_reset_bus(void)
 {
@@ -283,7 +267,7 @@ adb_reset_bus(void)
  }

  down(&adb_probe_mutex);
- schedule_work(&adb_reset_work);
+ adb_probe_task = kthread_run(adb_probe, NULL, "kadbprobe");
  return 0;
 }

@@ -469,9 +453,7 @@ adb_request(struct adb_request *req, void (*done)(struct adb_request *),
  /* Synchronous requests send from the probe thread cause it to
   * block. Beware that the "done" callback will be overriden !
   */
- if ((flags & ADBREQ_SYNC) &&
-     (current->pid && adb_probe_task_pid &&
-     adb_probe_task_pid == current->pid)) {
+ if ((flags & ADBREQ_SYNC) && (current == adb_probe_task)) {
   req->done = adb_probe_wakeup;
   rc = adb_controller->send_request(req, 0);
   if (rc || req->complete)
--
1.5.0.g53756
```

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: [PATCH] macintosh/mediabay: Convert to kthread API.
Posted by ebiederm on Thu, 19 Apr 2007 06:55:45 GMT
View Forum Message <> Reply to Message

From: Eric W. Biederman <ebiederm@xmission.com> - unquoted

This patch modifies the startup of the media_bay_task
to use kthread_run and not a combination of kernel_thread,
deamonize and sigfillset.

In addition since we now always want to ignore signals
the MB_IGNORE_SIGNALS define is removed along with the
test for signal_pending.

The result is slightly simpler code that is more
maintainable.

Cc: Benjamin Herrenschmidt <benh@kernel.crashing.org>
Cc: Paul Mackerras <paulus@samba.org>
Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
---
 drivers/macintosh/mediabay.c | 11 ++---------
 1 files changed, 2 insertions(+), 9 deletions(-)

diff --git a/drivers/macintosh/mediabay.c b/drivers/macintosh/mediabay.c
index c803d2b..90c853e 100644
--- a/drivers/macintosh/mediabay.c
+++ b/drivers/macintosh/mediabay.c
@@ -20,6 +20,7 @@
 #include <linux/stddef.h>
 #include <linux/init.h>
 #include <linux/ide.h>
+#include <linux/kthread.h>
 #include <asm/prom.h>
 #include <asm/pgtable.h>
 #include <asm/io.h>
@@ -35,7 +36,6 @@

 #define MB_DEBUG
-#define MB_IGNORE_SIGNALS

 #ifdef MB_DEBUG
 #define MBDBG(fmt, arg...) printk(KERN_INFO fmt , ## arg)
@@ -622,11 +622,6 @@ static int media_bay_task(void *x)
 {
  int i;

- strcpy(current->comm, "media-bay");
-#ifdef MB_IGNORE_SIGNALS
- sigfillset(&current->blocked);
-#endif

```
-
  for (;;) {
   for (i = 0; i < media_bay_count; ++i) {
    down(&media_bays[i].lock);
@@ -636,8 +631,6 @@ static int media_bay_task(void *x)
   }

   msleep_interruptible(MB_POLL_DELAY);
-  if (signal_pending(current))
-   return 0;
  }
 }

@@ -699,7 +692,7 @@ static int __devinit media_bay_attach(struct macio_dev *mdev, const
struct of_de

  /* Startup kernel thread */
  if (i == 0)
-  kernel_thread(media_bay_task, NULL, CLONE_KERNEL);
+  kthread_run(media_bay_task, NULL, "media-bay");

  return 0;

--
1.5.0.g53756
```

Subject: [PATCH] bluetooth bnep: Convert to kthread API.
Posted by ebiederm on Thu, 19 Apr 2007 06:55:46 GMT
View Forum Message <> Reply to Message

From: Eric W. Biederman <ebiederm@xmission.com> - unquoted

This patch starts kbenpd using kthread_run replacing
a combination of kernel_thread and daemonize. Making
the code a little simpler and more maintainable.

Cc: Marcel Holtmann <marcel@holtmann.org>
Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
---
 net/bluetooth/bnep/core.c | 8 +++++---
 1 files changed, 5 insertions(+), 3 deletions(-)

```
diff --git a/net/bluetooth/bnep/core.c b/net/bluetooth/bnep/core.c
index a9f1e88..de3caed 100644
--- a/net/bluetooth/bnep/core.c
+++ b/net/bluetooth/bnep/core.c
@@ -32,6 +32,7 @@
 #include <linux/module.h>

 #include <linux/kernel.h>
+#include <linux/kthread.h>
 #include <linux/sched.h>
 #include <linux/signal.h>
 #include <linux/init.h>
@@ -473,7 +474,6 @@ static int bnep_session(void *arg)

  BT_DBG("");

- daemonize("kbnepd %s", dev->name);
  set_user_nice(current, -15);

  init_waitqueue_entry(&wait, current);
@@ -539,6 +539,7 @@ static struct device *bnep_get_device(struct bnep_session *session)

 int bnep_add_connection(struct bnep_connadd_req *req, struct socket *sock)
 {
+ struct task_struct *task;
  struct net_device *dev;
  struct bnep_session *s, *ss;
  u8 dst[ETH_ALEN], src[ETH_ALEN];
@@ -598,9 +599,10 @@ int bnep_add_connection(struct bnep_connadd_req *req, struct socket
*sock)

  __bnep_link_session(s);

- err = kernel_thread(bnep_session, s, CLONE_KERNEL);
- if (err < 0) {
+ task = kthread_run(bnep_session, s, "kbnepd %s", dev->name);
+ if (IS_ERR(task)) {
   /* Session thread start failed, gotta cleanup. */
+  err = PTR_ERR(task);
   unregister_netdev(dev);
   __bnep_unlink_session(s);
   goto failed;
--
1.5.0.g53756
```

_____
Containers mailing list
Containers@lists.linux-foundation.org

Subject: [PATCH] bluetooth cmtp: Convert to use kthread API.
Posted by ebiederm on Thu, 19 Apr 2007 06:55:47 GMT
View Forum Message <> Reply to Message

From: Eric W. Biederman <ebiederm@xmission.com> - unquoted

This patch modifies the kcmptd_ctr_%d daemon using kthread_run
instead of a combination of kernel_thread and daemonize making
the code a little simpler and more maintainable.

Cc: Marcel Holtmann <marcel@holtmann.org>
Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
---
 net/bluetooth/cmtp/core.c | 8 +++++---
 1 files changed, 5 insertions(+), 3 deletions(-)

diff --git a/net/bluetooth/cmtp/core.c b/net/bluetooth/cmtp/core.c
index e1b9db9..993303f 100644
--- a/net/bluetooth/cmtp/core.c
+++ b/net/bluetooth/cmtp/core.c
@@ -35,6 +35,7 @@
 #include <linux/file.h>
 #include <linux/init.h>
 #include <linux/freezer.h>
+#include <linux/kthread.h>
 #include <net/sock.h>

 #include <linux/isdn/capilli.h>
@@ -286,7 +287,6 @@ static int cmtp_session(void *arg)

   BT_DBG("session %p", session);

- daemonize("kcmtpd_ctr_%d", session->num);
   set_user_nice(current, -15);

   init_waitqueue_entry(&wait, current);
@@ -329,6 +329,7 @@ static int cmtp_session(void *arg)
 int cmtp_add_connection(struct cmtp_connadd_req *req, struct socket *sock)
 {
   struct cmtp_session *session, *s;
+ struct task_struct *task;
   bdaddr_t src, dst;
   int i, err;

@@ -375,8 +376,9 @@ int cmtp_add_connection(struct cmtp_connadd_req *req, struct socket

```
*sock)

  __cmtp_link_session(session);

- err = kernel_thread(cmtp_session, session, CLONE_KERNEL);
- if (err < 0)
+ task = kthread_run(cmtp_session, session, "kcmtpd_ctr_%d", session->num);
+ err = PTR_ERR(task);
+ if (IS_ERR(task))
   goto unlink;

  if (!(session->flags & (1 << CMTP_LOOPBACK))) {
--
1.5.0.g53756
```

Subject: [PATCH] bluetooth hidp: Convert to kthread API.
Posted by ebiederm on Thu, 19 Apr 2007 06:55:48 GMT
View Forum Message <> Reply to Message

From: Eric W. Biederman <ebiederm@xmission.com> - unquoted

This patch starts up khidp using kthread_run instead
of kernel_thread and daemonize, resulting is slightly
simpler and more maintainable code.

Cc: Marcel Holtmann <marcel@holtmann.org>
Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
---
 net/bluetooth/hidp/core.c | 29 ++++++++++++++++++-----------
 1 files changed, 16 insertions(+), 13 deletions(-)

```
diff --git a/net/bluetooth/hidp/core.c b/net/bluetooth/hidp/core.c
index df2c471..1c9b202 100644
--- a/net/bluetooth/hidp/core.c
+++ b/net/bluetooth/hidp/core.c
@@ -36,6 +36,7 @@
 #include <linux/init.h>
 #include <linux/wait.h>
 #include <linux/freezer.h>
+#include <linux/kthread.h>
 #include <net/sock.h>
```

```
 #include <linux/input.h>
@@ -531,22 +532,11 @@ static int hidp_session(void *arg)
  struct sock *ctrl_sk = session->ctrl_sock->sk;
  struct sock *intr_sk = session->intr_sock->sk;
  struct sk_buff *skb;
- int vendor = 0x0000, product = 0x0000;
  wait_queue_t ctrl_wait, intr_wait;

  BT_DBG("session %p", session);

- if (session->input) {
-  vendor  = session->input->id.vendor;
-  product = session->input->id.product;
- }
-
- if (session->hid) {
-  vendor  = session->hid->vendor;
-  product = session->hid->product;
- }

- daemonize("khidpd_%04x%04x", vendor, product);
  set_user_nice(current, -15);

  init_waitqueue_entry(&ctrl_wait, current);
@@ -747,7 +737,9 @@ static inline void hidp_setup_hid(struct hidp_session *session, struct
hidp_conn

 int hidp_add_connection(struct hidp_connadd_req *req, struct socket *ctrl_sock, struct socket
*intr_sock)
 {
+ int vendor = 0x0000, product = 0x0000;
  struct hidp_session *session, *s;
+ struct task_struct *task;
  int err;

  BT_DBG("");
@@ -834,8 +826,19 @@ int hidp_add_connection(struct hidp_connadd_req *req, struct socket
*ctrl_sock,

  hidp_set_timer(session);

- err = kernel_thread(hidp_session, session, CLONE_KERNEL);
- if (err < 0)
+ if (session->input) {
+  vendor  = session->input->id.vendor;
+  product = session->input->id.product;
+ }
+
```

```
+ if (session->hid) {
+  vendor  = session->hid->vendor;
+  product = session->hid->product;
+ }
+ task = kthread_run(hidp_session, session,
+       "khidpd_%04x%04x", vendor, product);
+ err = PTR_ERR(task);
+ if (IS_ERR(task))
   goto unlink;

  if (session->input) {
--
1.5.0.g53756
```

Subject: [PATCH] bluetooth rfcomm: Convert to kthread API.
Posted by ebiederm on Thu, 19 Apr 2007 06:55:49 GMT
View Forum Message <> Reply to Message

From: Eric W. Biederman <ebiederm@xmission.com> - unquoted

This patch starts krfcommd using kthread_run instead of a combination
of kernel_thread and daemonize making the code slightly simpler
and more maintainable.

Cc: Marcel Holtmann <marcel@holtmann.org>
Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
---
 net/bluetooth/rfcomm/core.c | 4 ++--
 1 files changed, 2 insertions(+), 2 deletions(-)

```
diff --git a/net/bluetooth/rfcomm/core.c b/net/bluetooth/rfcomm/core.c
index 34f993a..baaad49 100644
--- a/net/bluetooth/rfcomm/core.c
+++ b/net/bluetooth/rfcomm/core.c
@@ -38,6 +38,7 @@
 #include <linux/net.h>
 #include <linux/mutex.h>
 #include <linux/freezer.h>
+#include <linux/kthread.h>

 #include <net/sock.h>
 #include <asm/uaccess.h>
```

```
@@ -1938,7 +1939,6 @@ static int rfcomm_run(void *unused)

  atomic_inc(&running);

- daemonize("krfcommd");
  set_user_nice(current, -10);

  BT_DBG("");
@@ -2058,7 +2058,7 @@ static int __init rfcomm_init(void)

  hci_register_cb(&rfcomm_cb);

- kernel_thread(rfcomm_run, NULL, CLONE_KERNEL);
+ kthread_run(rfcomm_run, NULL, "krfcommd");

  if (class_create_file(bt_class, &class_attr_rfcomm_dlc) < 0)
   BT_ERR("Failed to create RFCOMM info file");
--
1.5.0.g53756
```

Subject: [PATCH] fs/afs: Convert to kthread API.
Posted by ebiederm on Thu, 19 Apr 2007 06:55:50 GMT
View Forum Message <> Reply to Message

From: Eric W. Biederman <ebiederm@xmission.com> - unquoted

This patch modifies the startup of kafscmd, kafsasyncd, and kafstimod
to use kthread_run instead of a combination of kernel_thread and
daemonize making the code slightly simpler and more maintainable.

In addition since by default all signals are ignored when delivered
to a kernel thread the code to flush signals has been removed.

Cc: David Howells <dhowells@redhat.com>
Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
---
 fs/afs/cmservice.c | 10 +++++-----
 fs/afs/internal.h | 11 -----------
 fs/afs/kafsasyncd.c | 17 ++++++-----------
 fs/afs/kafstimod.c | 16 ++++++----------
 4 files changed, 17 insertions(+), 37 deletions(-)

```
diff --git a/fs/afs/cmservice.c b/fs/afs/cmservice.c
index 3d097fd..f7e2355 100644
--- a/fs/afs/cmservice.c
+++ b/fs/afs/cmservice.c
@@ -13,6 +13,7 @@
 #include <linux/init.h>
 #include <linux/sched.h>
 #include <linux/completion.h>
+#include <linux/kthread.h>
 #include "server.h"
 #include "cell.h"
 #include "transport.h"
@@ -120,8 +121,6 @@ static int kafscmd(void *arg)

  printk(KERN_INFO "kAFS: Started kafscmd %d\n", current->pid);

- daemonize("kafscmd");
-
  complete(&kafscmd_alive);

  /* loop around looking for things to attend to */
@@ -133,7 +132,6 @@ static int kafscmd(void *arg)
    for (;;) {
     set_current_state(TASK_INTERRUPTIBLE);
     if (!list_empty(&kafscmd_attention_list) ||
-        signal_pending(current) ||
         kafscmd_die)
      break;

@@ -297,8 +295,10 @@ int afscm_start(void)

  down_write(&afscm_sem);
  if (!afscm_usage) {
-  ret = kernel_thread(kafscmd, NULL, 0);
-  if (ret < 0)
+  struct task_struct *task;
+  task = kthread_run(kafscmd, NULL, "kafscmd");
+  ret = PTR_ERR(task);
+  if (IS_ERR(task))
    goto out;

   wait_for_completion(&kafscmd_alive);
diff --git a/fs/afs/internal.h b/fs/afs/internal.h
index 5151d5d..2d667b7 100644
--- a/fs/afs/internal.h
+++ b/fs/afs/internal.h
@@ -40,17 +40,6 @@
 #define _net(FMT, a...)  do { } while(0)
```

```
 #endif

-static inline void afs_discard_my_signals(void)
-{
- while (signal_pending(current)) {
-  siginfo_t sinfo;
-
-  spin_lock_irq(&current->sighand->siglock);
-  dequeue_signal(current,&current->blocked, &sinfo);
-  spin_unlock_irq(&current->sighand->siglock);
- }
-}
-
 /*
  * cell.c
  */
diff --git a/fs/afs/kafsasyncd.c b/fs/afs/kafsasyncd.c
index 615df24..ead025f 100644
--- a/fs/afs/kafsasyncd.c
+++ b/fs/afs/kafsasyncd.c
@@ -21,6 +21,7 @@
 #include <linux/sched.h>
 #include <linux/completion.h>
 #include <linux/freezer.h>
+#include <linux/kthread.h>
 #include "cell.h"
 #include "server.h"
 #include "volume.h"
@@ -56,15 +57,15 @@ static void kafsasyncd_null_call_error_func(struct rxrpc_call *call)
  */
 int afs_kafsasyncd_start(void)
 {
- int ret;
+ struct task_struct *task;

- ret = kernel_thread(kafsasyncd, NULL, 0);
- if (ret < 0)
-  return ret;
+ task = kthread_run(kafsasyncd, NULL, "kafsasyncd");
+ if (IS_ERR(task))
+  return PTR_ERR(task);

  wait_for_completion(&kafsasyncd_alive);

- return ret;
+ return 0;
 } /* end afs_kafsasyncd_start() */
```

```
 /*****************************************************************************/
@@ -95,8 +96,6 @@ static int kafsasyncd(void *arg)

  printk("kAFS: Started kafsasyncd %d\n", current->pid);

- daemonize("kafsasyncd");
-
  complete(&kafsasyncd_alive);

  /* loop around looking for things to attend to */
@@ -106,7 +105,6 @@ static int kafsasyncd(void *arg)

   for (;;) {
    if (!list_empty(&kafsasyncd_async_attnq) ||
-       signal_pending(current) ||
        kafsasyncd_die)
     break;

@@ -119,9 +117,6 @@ static int kafsasyncd(void *arg)

   try_to_freeze();

-  /* discard pending signals */
-  afs_discard_my_signals();
-
   die = kafsasyncd_die;

   /* deal with the next asynchronous operation requiring
diff --git a/fs/afs/kafstimod.c b/fs/afs/kafstimod.c
index 694344e..caeac88 100644
--- a/fs/afs/kafstimod.c
+++ b/fs/afs/kafstimod.c
@@ -14,6 +14,7 @@
 #include <linux/sched.h>
 #include <linux/completion.h>
 #include <linux/freezer.h>
+#include <linux/kthread.h>
 #include "cell.h"
 #include "volume.h"
 #include "kafstimod.h"
@@ -36,15 +37,15 @@ static int kafstimod(void *arg);
  */
 int afs_kafstimod_start(void)
 {
- int ret;
+ struct task_struct *task;

- ret = kernel_thread(kafstimod, NULL, 0);
```

```
- if (ret < 0)
-  return ret;
+ task = kthread_run(kafstimod, NULL, "kafstimod");
+ if (IS_ERR(task))
+  return PTR_ERR(task);

  wait_for_completion(&kafstimod_alive);

- return ret;
+ return 0;
 } /* end afs_kafstimod_start() */

 /*****************************************************************************/
@@ -72,8 +73,6 @@ static int kafstimod(void *arg)

  printk("kAFS: Started kafstimod %d\n", current->pid);

- daemonize("kafstimod");
-
  complete(&kafstimod_alive);

  /* loop around looking for things to attend to */
@@ -94,9 +93,6 @@ static int kafstimod(void *arg)

   try_to_freeze();

-  /* discard pending signals */
-  afs_discard_my_signals();
-
   /* work out the time to elapse before the next event */
   spin_lock(&kafstimod_lock);
   if (list_empty(&kafstimod_list)) {
--
1.5.0.g53756
```

Subject: [PATCH] net/rxrpc: Convert to kthread API.
Posted by ebiederm on Thu, 19 Apr 2007 06:55:51 GMT
View Forum Message <> Reply to Message

From: Eric W. Biederman <ebiederm@xmission.com> - unquoted

This patch modifies the startup of krxtimod, krxiod, and krxsecd

to use kthread_run instead of a combination of kernel_thread
and daemonize making the code slightly simpler and more maintainable.

In addition since by default all signals are ignored when delivered
to a kernel thread the code to flush signals has been removed.

Cc: David Howells <dhowells@redhat.com>
Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
---
 net/rxrpc/internal.h | 11 -----------
 net/rxrpc/krxiod.c | 16 ++++++++--------
 net/rxrpc/krxsecd.c | 16 ++++++++--------
 net/rxrpc/krxtimod.c | 15 ++++++---------
 4 files changed, 22 insertions(+), 36 deletions(-)

diff --git a/net/rxrpc/internal.h b/net/rxrpc/internal.h
index cc0c579..1dd69aa 100644
--- a/net/rxrpc/internal.h
+++ b/net/rxrpc/internal.h
@@ -49,17 +49,6 @@ __RXACCT_DECL(extern atomic_t rxrpc_message_count);
 #define _net(FMT, a...) do { if (rxrpc_knet)   knet (FMT , ##a); } while(0)
 #endif

-static inline void rxrpc_discard_my_signals(void)
-{
- while (signal_pending(current)) {
- siginfo_t sinfo;
-
- spin_lock_irq(&current->sighand->siglock);
- dequeue_signal(current, &current->blocked, &sinfo);
- spin_unlock_irq(&current->sighand->siglock);
- }
-}
-
 /*
  * call.c
  */
diff --git a/net/rxrpc/krxiod.c b/net/rxrpc/krxiod.c
index bbbcd6c..c590ccd 100644
--- a/net/rxrpc/krxiod.c
+++ b/net/rxrpc/krxiod.c
@@ -14,6 +14,7 @@
 #include <linux/spinlock.h>
 #include <linux/init.h>
 #include <linux/freezer.h>
+#include <linux/kthread.h>
 #include <rxrpc/krxiod.h>
 #include <rxrpc/transport.h>

```
 #include <rxrpc/peer.h>
@@ -43,8 +44,6 @@ static int rxrpc_krxiod(void *arg)

  printk("Started krxiod %d\n",current->pid);

- daemonize("krxiod");
-
  /* loop around waiting for work to do */
  do {
   /* wait for work or to be told to exit */
@@ -57,8 +56,7 @@ static int rxrpc_krxiod(void *arg)
    for (;;) {
     set_current_state(TASK_INTERRUPTIBLE);
     if (atomic_read(&rxrpc_krxiod_qcount) ||
-        rxrpc_krxiod_die ||
-        signal_pending(current))
+        rxrpc_krxiod_die)
      break;

     schedule();
@@ -141,9 +139,6 @@ static int rxrpc_krxiod(void *arg)

   try_to_freeze();

-  /* discard pending signals */
-  rxrpc_discard_my_signals();
-
  } while (!rxrpc_krxiod_die);

  /* and that's all */
@@ -157,7 +152,12 @@ static int rxrpc_krxiod(void *arg)
  */
 int __init rxrpc_krxiod_init(void)
 {
- return kernel_thread(rxrpc_krxiod, NULL, 0);
+ struct task_struct *task;
+ int ret = 0;
+ task = kthread_run(rxrpc_krxiod, NULL, "krxiod");
+ if (IS_ERR(task))
+  ret = PTR_ERR(task);
+ return ret;

 } /* end rxrpc_krxiod_init() */

diff --git a/net/rxrpc/krxsecd.c b/net/rxrpc/krxsecd.c
index 9a1e7f5..150cd39 100644
--- a/net/rxrpc/krxsecd.c
+++ b/net/rxrpc/krxsecd.c
```

```
@@ -19,6 +19,7 @@
 #include <linux/completion.h>
 #include <linux/spinlock.h>
 #include <linux/init.h>
+#include <linux/kthread.h>
 #include <rxrpc/krxsecd.h>
 #include <rxrpc/transport.h>
 #include <rxrpc/connection.h>
@@ -56,8 +57,6 @@ static int rxrpc_krxsecd(void *arg)

  printk("Started krxsecd %d\n", current->pid);

- daemonize("krxsecd");
-
  /* loop around waiting for work to do */
  do {
   /* wait for work or to be told to exit */
@@ -70,8 +69,7 @@ static int rxrpc_krxsecd(void *arg)
    for (;;) {
     set_current_state(TASK_INTERRUPTIBLE);
     if (atomic_read(&rxrpc_krxsecd_qcount) ||
-        rxrpc_krxsecd_die ||
-        signal_pending(current))
+        rxrpc_krxsecd_die)
      break;

     schedule();
@@ -110,9 +108,6 @@ static int rxrpc_krxsecd(void *arg)

   try_to_freeze();

-  /* discard pending signals */
-  rxrpc_discard_my_signals();
-
  } while (!die);

  /* and that's all */
@@ -126,7 +121,12 @@ static int rxrpc_krxsecd(void *arg)
  */
 int __init rxrpc_krxsecd_init(void)
 {
- return kernel_thread(rxrpc_krxsecd, NULL, 0);
+ struct task_struct *task;
+ int ret = 0;
+ task = kthread_run(rxrpc_krxsecd, NULL, "krxsecd");
+ if (IS_ERR(task))
+  ret = PTR_ERR(task);
+ return ret;
```

```
 } /* end rxrpc_krxsecd_init() */

diff --git a/net/rxrpc/krxtimod.c b/net/rxrpc/krxtimod.c
index 9a9b613..3b5f062 100644
--- a/net/rxrpc/krxtimod.c
+++ b/net/rxrpc/krxtimod.c
@@ -14,6 +14,7 @@
 #include <linux/sched.h>
 #include <linux/completion.h>
 #include <linux/freezer.h>
+#include <linux/kthread.h>
 #include <rxrpc/rxrpc.h>
 #include <rxrpc/krxtimod.h>
 #include <asm/errno.h>
@@ -35,11 +36,12 @@ static int krxtimod(void *arg);
  */
 int rxrpc_krxtimod_start(void)
 {
- int ret;
+ struct task_struct *task;
+ int ret = 0;

- ret = kernel_thread(krxtimod, NULL, 0);
- if (ret < 0)
-  return ret;
+ task = kthread_run(krxtimod, NULL, "krxtimod");
+ if (IS_ERR(task))
+  ret = PTR_ERR(task);

  wait_for_completion(&krxtimod_alive);

@@ -71,8 +73,6 @@ static int krxtimod(void *arg)

  printk("Started krxtimod %d\n", current->pid);

- daemonize("krxtimod");
-
  complete(&krxtimod_alive);

  /* loop around looking for things to attend to */
@@ -93,9 +93,6 @@ static int krxtimod(void *arg)

   try_to_freeze();

-  /* discard pending signals */
-  rxrpc_discard_my_signals();
-
```

```
   /* work out the time to elapse before the next event */
   spin_lock(&krxtimod_lock);
   if (list_empty(&krxtimod_list)) {
--
1.5.0.g53756
```

Subject: [PATCH] ipv4/ipvs: Convert to kthread API
Posted by ebiederm on Thu, 19 Apr 2007 06:55:52 GMT
View Forum Message <> Reply to Message

From: Eric W. Biederman <ebiederm@xmission.com> - unquoted

Modify startup of ipvs sync threads to use kthread_run
instead of a weird combination of calling kernel_thread
to start a fork_sync_thread whose hole purpose in life was
to call kernel_thread again starting the actually sync thread
which called daemonize.

To use kthread_run I had to move the name calcuation from
sync_thread into start_sync_thread resulting in a small
amount of code motion.

The result is simpler and more maintainable piece of code.

Cc: Wensong Zhang <wensong@linux-vs.org>
Cc: Julian Anastasov <ja@ssi.bg>
Cc: Simon Horman <horms@verge.net.au>
Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
---
 net/ipv4/ipvs/ip_vs_sync.c | 49 ++++++++++++----------------------------
 1 files changed, 12 insertions(+), 37 deletions(-)

```
diff --git a/net/ipv4/ipvs/ip_vs_sync.c b/net/ipv4/ipvs/ip_vs_sync.c
index 7ea2d98..c4be9dc 100644
--- a/net/ipv4/ipvs/ip_vs_sync.c
+++ b/net/ipv4/ipvs/ip_vs_sync.c
@@ -29,6 +29,7 @@
 #include <linux/in.h>
 #include <linux/igmp.h>                 /* for ip_mc_join_group */
 #include <linux/udp.h>
+#include <linux/kthread.h>
```

```
 #include <net/ip.h>
 #include <net/sock.h>
@@ -750,34 +751,23 @@ static int sync_thread(void *startup)
  DECLARE_WAITQUEUE(wait, current);
  mm_segment_t oldmm;
  int state;
- const char *name;

  /* increase the module use count */
  ip_vs_use_count_inc();

- if (ip_vs_sync_state & IP_VS_STATE_MASTER && !sync_master_pid) {
+ if (ip_vs_sync_state & IP_VS_STATE_MASTER && !sync_master_pid)
   state = IP_VS_STATE_MASTER;
-  name = "ipvs_syncmaster";
- } else if (ip_vs_sync_state & IP_VS_STATE_BACKUP && !sync_backup_pid) {
+ else if (ip_vs_sync_state & IP_VS_STATE_BACKUP && !sync_backup_pid)
   state = IP_VS_STATE_BACKUP;
-  name = "ipvs_syncbackup";
- } else {
+ else {
   IP_VS_BUG();
   ip_vs_use_count_dec();
   return -EINVAL;
  }

- daemonize(name);
-
  oldmm = get_fs();
  set_fs(KERNEL_DS);

- /* Block all signals */
- spin_lock_irq(&current->sighand->siglock);
- siginitsetinv(&current->blocked, 0);
- recalc_sigpending();
- spin_unlock_irq(&current->sighand->siglock);
-
  /* set the maximum length of sync message */
  set_sync_mesg_maxlen(state);

@@ -815,29 +805,11 @@ static int sync_thread(void *startup)
  return 0;
 }

-
-static int fork_sync_thread(void *startup)
-{
- pid_t pid;
```

```
-
- /* fork the sync thread here, then the parent process of the
-    sync thread is the init process after this thread exits. */
-  repeat:
- if ((pid = kernel_thread(sync_thread, startup, 0)) < 0) {
-  IP_VS_ERR("could not create sync_thread due to %d... "
-     "retrying.\n", pid);
-  msleep_interruptible(1000);
-  goto repeat;
- }
-
- return 0;
-}
-
-
 int start_sync_thread(int state, char *mcast_ifn, __u8 syncid)
 {
  DECLARE_COMPLETION_ONSTACK(startup);
- pid_t pid;
+ struct task_struct *task;
+ const char *name;

  if ((state == IP_VS_STATE_MASTER && sync_master_pid) ||
      (state == IP_VS_STATE_BACKUP && sync_backup_pid))
@@ -852,16 +824,19 @@ int start_sync_thread(int state, char *mcast_ifn, __u8 syncid)
   strlcpy(ip_vs_master_mcast_ifn, mcast_ifn,
    sizeof(ip_vs_master_mcast_ifn));
   ip_vs_master_syncid = syncid;
+  name = "ipvs_syncmaster";
  } else {
   strlcpy(ip_vs_backup_mcast_ifn, mcast_ifn,
    sizeof(ip_vs_backup_mcast_ifn));
   ip_vs_backup_syncid = syncid;
+  name = "ipvs_syncbackup";
  }

   repeat:
- if ((pid = kernel_thread(fork_sync_thread, &startup, 0)) < 0) {
-  IP_VS_ERR("could not create fork_sync_thread due to %d... "
-     "retrying.\n", pid);
+ task = kthread_run(sync_thread, &startup, name);
+ if (IS_ERR(task)) {
+  IP_VS_ERR("could not create sync_thread due to %ld... "
+     "retrying.\n", PTR_ERR(task));
   msleep_interruptible(1000);
   goto repeat;
  }
--
```

1.5.0.g53756

_____

Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: [PATCH] saa7134-tvaudio: Convert to kthread API.
Posted by ebiederm on Thu, 19 Apr 2007 06:55:53 GMT
View Forum Message <> Reply to Message

From: Eric W. Biederman <ebiederm@xmission.com> - unquoted

It is my goal to replace all kernel code that handles signals
from user space, calls kernel_thread or calls daemonize. All
of which the kthread_api makes unncessary. Handling signals
from user space is a maintenance problem becuase using a
kernel thread is an implementation detail and if user space
cares it does not allow us to change the implementation. Calling
daemonize is a problem because it has to undo a continually changing
set of state generated by user space, requiring the implemetation
to change continually. kernel_thread is a problem because it
returns a pid_t value. Numeric pids are inherently racy and
in the presence of a pid namespace they are no longer global
making them useless for general use in the kernel.

So this patch renames the pid member of struct saa7134_thread
started and changes it's type from pid_t to int. All it
has ever been used for is to detect if the kernel thread
is has been started so this works.

allow_signal(SIGTERM) and the calls to signal_pending have
been removed they are needed for the driver to operation.

The startup of tvaudio_thread and tvaudio_thread_dep have
been modified to use kthread_run instead of a combination
of kernel_thread and daemonize.

The result is code that is slightly simpler and more
maintainable.

Cc: Hartmut Hackmann <hartmut.hackmann@t-online.de>
Cc: Mauro Carvalho Chehab <mchehab@infradead.org>
Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
---
drivers/media/video/saa7134/saa7134-tvaudio.c | 27 ++++++++++++----------
drivers/media/video/saa7134/saa7134.h | 2 +-

```
 2 files changed, 14 insertions(+), 15 deletions(-)

diff --git a/drivers/media/video/saa7134/saa7134-tvaudio.c
b/drivers/media/video/saa7134/saa7134-tvaudio.c
index 7b56041..b636cb1 100644
--- a/drivers/media/video/saa7134/saa7134-tvaudio.c
+++ b/drivers/media/video/saa7134/saa7134-tvaudio.c
@@ -27,6 +27,7 @@
 #include <linux/kernel.h>
 #include <linux/slab.h>
 #include <linux/delay.h>
+#include <linux/kthread.h>
 #include <asm/div64.h>

 #include "saa7134-reg.h"
@@ -505,11 +506,9 @@ static int tvaudio_thread(void *data)
  unsigned int i, audio, nscan;
  int max1,max2,carrier,rx,mode,lastmode,default_carrier;

- daemonize("%s", dev->name);
- allow_signal(SIGTERM);
  for (;;) {
   tvaudio_sleep(dev,-1);
-  if (dev->thread.shutdown || signal_pending(current))
+  if (dev->thread.shutdown)
    goto done;

  restart:
@@ -618,7 +617,7 @@ static int tvaudio_thread(void *data)
   for (;;) {
    if (tvaudio_sleep(dev,5000))
     goto restart;
-   if (dev->thread.shutdown || signal_pending(current))
+   if (dev->thread.shutdown)
     break;
    if (UNSET == dev->thread.mode) {
     rx = tvaudio_getstereo(dev,&tvaudio[i]);
@@ -782,9 +781,6 @@ static int tvaudio_thread_ddep(void *data)
  struct saa7134_dev *dev = data;
  u32 value, norms, clock;

- daemonize("%s", dev->name);
- allow_signal(SIGTERM);
-
  clock = saa7134_boards[dev->board].audio_clock;
  if (UNSET != audio_clock_override)
   clock = audio_clock_override;
@@ -796,7 +792,7 @@ static int tvaudio_thread_ddep(void *data)
```

```
  for (;;) {
    tvaudio_sleep(dev,-1);
-   if (dev->thread.shutdown || signal_pending(current))
+   if (dev->thread.shutdown)
      goto done;

  restart:
@@ -986,14 +982,17 @@ int saa7134_tvaudio_init2(struct saa7134_dev *dev)
    break;
  }

- dev->thread.pid = -1;
+ dev->thread.started = 0;
  if (my_thread) {
+   struct task_struct *task;
    /* start tvaudio thread */
    init_waitqueue_head(&dev->thread.wq);
    init_completion(&dev->thread.exit);
-   dev->thread.pid = kernel_thread(my_thread,dev,0);
-   if (dev->thread.pid < 0)
-     printk(KERN_WARNING "%s: kernel_thread() failed\n",
+   task = kthread_run(my_thread, dev, "%s", dev->name);
+   if (!IS_ERR(task))
+     dev->thread.started = 1;
+   else
+     printk(KERN_WARNING "%s: kthread_create() failed\n",
            dev->name);
    saa7134_tvaudio_do_scan(dev);
  }
@@ -1005,7 +1004,7 @@ int saa7134_tvaudio_init2(struct saa7134_dev *dev)
 int saa7134_tvaudio_fini(struct saa7134_dev *dev)
 {
  /* shutdown tvaudio thread */
- if (dev->thread.pid >= 0) {
+ if (dev->thread.started) {
    dev->thread.shutdown = 1;
    wake_up_interruptible(&dev->thread.wq);
    wait_for_completion(&dev->thread.exit);
@@ -1020,7 +1019,7 @@ int saa7134_tvaudio_do_scan(struct saa7134_dev *dev)
    dprintk("sound IF not in use, skipping scan\n");
    dev->automute = 0;
    saa7134_tvaudio_setmute(dev);
- } else if (dev->thread.pid >= 0) {
+ } else if (dev->thread.started) {
    dev->thread.mode = UNSET;
    dev->thread.scan2++;
    wake_up_interruptible(&dev->thread.wq);
```

```
diff --git a/drivers/media/video/saa7134/saa7134.h b/drivers/media/video/saa7134/saa7134.h
index 62224cc..3a10ce7 100644
--- a/drivers/media/video/saa7134/saa7134.h
+++ b/drivers/media/video/saa7134/saa7134.h
@@ -324,7 +324,7 @@ struct saa7134_pgtable {

 /* tvaudio thread status */
 struct saa7134_thread {
- pid_t                    pid;
+ int      started;
  struct completion          exit;
  wait_queue_head_t          wq;
  unsigned int               shutdown;
--
1.5.0.g53756
```

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: [PATCH] nfs lockd reclaimer: Convert to kthread API
Posted by ebiederm on Thu, 19 Apr 2007 06:55:54 GMT
View Forum Message <> Reply to Message

From: Eric W. Biederman <ebiederm@xmission.com> - unquoted

Start the reclaimer thread using kthread_run instead
of a combination of kernel_thread and daemonize.
The small amount of signal handling code is also removed
as it makes no sense and is a maintenance problem to handle
signals in kernel threads.

Cc: Neil Brown <neilb@suse.de>
Cc: Trond Myklebust <trond.myklebust@fys.uio.no>
Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
---
 fs/lockd/clntlock.c | 8 ++------
 1 files changed, 2 insertions(+), 6 deletions(-)

```
diff --git a/fs/lockd/clntlock.c b/fs/lockd/clntlock.c
index f4d45d4..83591f6 100644
--- a/fs/lockd/clntlock.c
+++ b/fs/lockd/clntlock.c
@@ -9,6 +9,7 @@
 #include <linux/module.h>
 #include <linux/types.h>
```

```
 #include <linux/time.h>
+#include <linux/kthread.h>
 #include <linux/nfs_fs.h>
 #include <linux/sunrpc/clnt.h>
 #include <linux/sunrpc/svc.h>
@@ -153,7 +154,7 @@ nlmclnt_recovery(struct nlm_host *host)
  if (!host->h_reclaiming++) {
   nlm_get_host(host);
   __module_get(THIS_MODULE);
-  if (kernel_thread(reclaimer, host, CLONE_KERNEL) < 0)
+  if (IS_ERR(kthread_run(reclaimer, host, "%s-reclaim", host->h_name)))
    module_put(THIS_MODULE);
  }
 }
@@ -166,9 +167,6 @@ reclaimer(void *ptr)
  struct file_lock *fl, *next;
  u32 nsmstate;

- daemonize("%s-reclaim", host->h_name);
- allow_signal(SIGKILL);
-
  down_write(&host->h_rwsem);

  /* This one ensures that our parent doesn't terminate while the
@@ -193,8 +191,6 @@ restart:
   list_del_init(&fl->fl_u.nfs_fl.list);

   /* Why are we leaking memory here? --okir */
-  if (signalled())
-   continue;
   if (nlmclnt_reclaim(host, fl) != 0)
    continue;
   list_add_tail(&fl->fl_u.nfs_fl.list, &host->h_granted);
--
1.5.0.g53756
```

Subject: [PATCH] nfsv4 delegation: Convert to kthread API
Posted by ebiederm on Thu, 19 Apr 2007 06:55:55 GMT
View Forum Message <> Reply to Message

From: Eric W. Biederman <ebiederm@xmission.com> - unquoted

To start the nfsv4-delegreturn thread this patch uses
kthread_run instead of a combination of kernel_thread
and daemonize.

In addition allow_signal(SIGKILL) is removed from
the expire delegations thread.

Cc: Neil Brown <neilb@suse.de>
Cc: Trond Myklebust <trond.myklebust@fys.uio.no>
Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
---
 fs/nfs/delegation.c | 11 ++++++-------
 1 files changed, 4 insertions(+), 7 deletions(-)

diff --git a/fs/nfs/delegation.c b/fs/nfs/delegation.c
index 841c99a..7b9b88c 100644
--- a/fs/nfs/delegation.c
+++ b/fs/nfs/delegation.c
@@ -232,7 +232,6 @@ int nfs_do_expire_all_delegations(void *ptr)
  struct nfs_delegation *delegation;
  struct inode *inode;

- allow_signal(SIGKILL);
 restart:
  spin_lock(&clp->cl_lock);
  if (test_bit(NFS4CLNT_STATE_RECOVER, &clp->cl_state) != 0)
@@ -310,8 +309,6 @@ static int recall_thread(void *data)
  struct nfs_inode *nfsi = NFS_I(inode);
  struct nfs_delegation *delegation;

- daemonize("nfsv4-delegreturn");
-
  nfs_msync_inode(inode);
  down_read(&clp->cl_sem);
  down_write(&nfsi->rwsem);
@@ -350,18 +347,18 @@ int nfs_async_inode_return_delegation(struct inode *inode, const
nfs4_stateid *s
   .inode = inode,
   .stateid = stateid,
  };
- int status;
+ struct task_struct *task;

  init_completion(&data.started);
  __module_get(THIS_MODULE);
- status = kernel_thread(recall_thread, &data, CLONE_KERNEL);
- if (status < 0)
+ task = kthread_run(recall_thread, &data, "nfsv4-delegreturn");

```
+ if (IS_ERR(task))
   goto out_module_put;
  wait_for_completion(&data.started);
  return data.result;
 out_module_put:
  module_put(THIS_MODULE);
- return status;
+ return PTR_ERR(task);
 }

 /*
--
1.5.0.g53756
```

Subject: [PATCH] nfsd/nfs4state: Remove unnecessary daemonize call.
Posted by ebiederm on Thu, 19 Apr 2007 06:55:56 GMT
View Forum Message <> Reply to Message

From: Eric W. Biederman <ebiederm@xmission.com> - unquoted

Cc: Neil Brown <neilb@suse.de>
Cc: Trond Myklebust <trond.myklebust@fys.uio.no>
Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
---
 fs/nfsd/nfs4state.c | 2 --
 1 files changed, 0 insertions(+), 2 deletions(-)

```
diff --git a/fs/nfsd/nfs4state.c b/fs/nfsd/nfs4state.c
index 678f3be..3cc8ce4 100644
--- a/fs/nfsd/nfs4state.c
+++ b/fs/nfsd/nfs4state.c
@@ -1326,8 +1326,6 @@ do_recall(void *__dp)
 {
  struct nfs4_delegation *dp = __dp;

- daemonize("nfsv4-recall");
-
  nfsd4_cb_recall(dp);
  return 0;
 }
--
1.5.0.g53756
```

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

Subject: [PATCH] nfs4state reclaimer: Remove unnecessary allow_signal
Posted by ebiederm on Thu, 19 Apr 2007 06:55:57 GMT
View Forum Message <> Reply to Message

From: Eric W. Biederman <ebiederm@xmission.com> - unquoted

Cc: Neil Brown <neilb@suse.de>
Cc: Trond Myklebust <trond.myklebust@fys.uio.no>
Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
---
 fs/nfs/nfs4state.c | 2 --
 1 files changed, 0 insertions(+), 2 deletions(-)

diff --git a/fs/nfs/nfs4state.c b/fs/nfs/nfs4state.c
index 5fffbdf..d16393f 100644
--- a/fs/nfs/nfs4state.c
+++ b/fs/nfs/nfs4state.c
@@ -775,8 +775,6 @@ static int reclaimer(void *ptr)
  struct rpc_cred *cred;
  int status = 0;

- allow_signal(SIGKILL);
-
  /* Ensure exclusive access to NFSv4 state */
  lock_kernel();
  down_write(&clp->cl_sem);
--
1.5.0.g53756

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

Subject: [PATCH] smbfs: Remove unnecessary allow_signal
Posted by ebiederm on Thu, 19 Apr 2007 06:55:58 GMT
View Forum Message <> Reply to Message

From: Eric W. Biederman <ebiederm@xmission.com> - unquoted

Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
---
 fs/smbfs/smbiod.c | 2 --
 1 files changed, 0 insertions(+), 2 deletions(-)

diff --git a/fs/smbfs/smbiod.c b/fs/smbfs/smbiod.c
index 3e61b44..67176af 100644
--- a/fs/smbfs/smbiod.c
+++ b/fs/smbfs/smbiod.c
@@ -298,8 +298,6 @@ out:
  */
 static int smbiod(void *unused)
 {
- allow_signal(SIGKILL);
-
  VERBOSE("SMB Kernel thread starting (%d) ...\n", current->pid);

  for (;;) {
--
1.5.0.g53756

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: [PATCH] dvb_en_50221: Convert to kthread API
Posted by ebiederm on Thu, 19 Apr 2007 06:55:59 GMT
View Forum Message <> Reply to Message

From: Eric W. Biederman <ebiederm@xmission.com> - unquoted

This patch is a minimal transformation to use the kthread API
doing it's best to preserve the existing logic.

Instead of starting kdvb-ca by calling kernel_thread,
daemonize and sigfillset we kthread_run is used.

Instead of tracking the pid of the running thread we instead
simply keep a flag to indicate that the current thread is
running, as that is all the pid is really used for.

And finally the kill_proc sending signal 0 to the kernel thread to
ensure it is alive before we wait for it to shutdown is removed.
The kthread API does not provide the pid so we don't have that
information readily available and the test is just silly. If there

is no shutdown race the test is a useless confirmation of that the
thread is running. If there is a race the test doesn't fix it and
we should fix the race properly.

Cc: Andrew de Quincey <adq_dvb@lidskialf.net>
Cc: Mauro Carvalho Chehab <mchehab@infradead.org>
Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
---
 drivers/media/dvb/dvb-core/dvb_ca_en50221.c | 46 ++++++++++++----------------
 1 files changed, 18 insertions(+), 28 deletions(-)

diff --git a/drivers/media/dvb/dvb-core/dvb_ca_en50221.c
b/drivers/media/dvb/dvb-core/dvb_ca_en50221.c
index 2a03bf5..b28bc15 100644
--- a/drivers/media/dvb/dvb-core/dvb_ca_en50221.c
+++ b/drivers/media/dvb/dvb-core/dvb_ca_en50221.c
@@ -37,6 +37,7 @@
 #include <linux/delay.h>
 #include <linux/spinlock.h>
 #include <linux/sched.h>
+#include <linux/kthread.h>

 #include "dvb_ca_en50221.h"
 #include "dvb_ringbuffer.h"
@@ -139,8 +140,8 @@ struct dvb_ca_private {
  /* wait queues for read() and write() operations */
  wait_queue_head_t wait_queue;

- /* PID of the monitoring thread */
- pid_t thread_pid;
+ /* Flag indicating the monitoring thread is running */
+ int thread_running;

  /* Wait queue used when shutting thread down */
  wait_queue_head_t thread_queue;
@@ -982,7 +983,6 @@ static void dvb_ca_en50221_thread_update_delay(struct dvb_ca_private
*ca)
 static int dvb_ca_en50221_thread(void *data)
 {
  struct dvb_ca_private *ca = data;
- char name[15];
  int slot;
  int flags;
  int status;
@@ -991,14 +991,6 @@ static int dvb_ca_en50221_thread(void *data)

  dprintk("%s\n", __FUNCTION__);

```
- /* setup kernel thread */
- snprintf(name, sizeof(name), "kdvb-ca-%i:%i", ca->dvbdev->adapter->num, ca->dvbdev->id);
-
- lock_kernel();
- daemonize(name);
- sigfillset(&current->blocked);
- unlock_kernel();
-
  /* choose the correct initial delay */
  dvb_ca_en50221_thread_update_delay(ca);

@@ -1182,7 +1174,7 @@ static int dvb_ca_en50221_thread(void *data)
  }

  /* completed */
- ca->thread_pid = 0;
+ ca->thread_running = 0;
  mb();
  wake_up_interruptible(&ca->thread_queue);
  return 0;
@@ -1660,6 +1652,7 @@ static struct dvb_device dvbdev_ca = {
 int dvb_ca_en50221_init(struct dvb_adapter *dvb_adapter,
     struct dvb_ca_en50221 *pubca, int flags, int slot_count)
 {
+ struct task_struct *task;
  int ret;
  struct dvb_ca_private *ca = NULL;
  int i;
@@ -1682,7 +1675,7 @@ int dvb_ca_en50221_init(struct dvb_adapter *dvb_adapter,
   goto error;
  }
  init_waitqueue_head(&ca->wait_queue);
- ca->thread_pid = 0;
+ ca->thread_running = 0;
  init_waitqueue_head(&ca->thread_queue);
  ca->exit = 0;
  ca->open = 0;
@@ -1711,13 +1704,15 @@ int dvb_ca_en50221_init(struct dvb_adapter *dvb_adapter,

  /* create a kthread for monitoring this CA device */

- ret = kernel_thread(dvb_ca_en50221_thread, ca, 0);
-
- if (ret < 0) {
-  printk("dvb_ca_init: failed to start kernel_thread (%d)\n", ret);
+ task = kthread_run(dvb_ca_en50221_thread, ca,
+       "kdvb-ca-%i:%i",
+       ca->dvbdev->adapter->num, ca->dvbdev->id);
```

```
+ if (IS_ERR(task)) {
+  ret = PTR_ERR(task);
+  printk("dvb_ca_init: failed to start kthread (%d)\n", ret);
   goto error;
  }
- ca->thread_pid = ret;
+ ca->thread_running = 1;
  return 0;

 error:
@@ -1748,16 +1743,11 @@ void dvb_ca_en50221_release(struct dvb_ca_en50221 *pubca)
  dprintk("%s\n", __FUNCTION__);

  /* shutdown the thread if there was one */
- if (ca->thread_pid) {
-  if (kill_proc(ca->thread_pid, 0, 1) == -ESRCH) {
-   printk("dvb_ca_release adapter %d: thread PID %d already died\n",
-          ca->dvbdev->adapter->num, ca->thread_pid);
-  } else {
-   ca->exit = 1;
-   mb();
-   dvb_ca_en50221_thread_wakeup(ca);
-   wait_event_interruptible(ca->thread_queue, ca->thread_pid == 0);
-  }
+ if (ca->thread_running) {
+  ca->exit = 1;
+  mb();
+  dvb_ca_en50221_thread_wakeup(ca);
+  wait_event_interruptible(ca->thread_queue, ca->thread_running == 0);
  }

  for (i = 0; i < ca->slot_count; i++) {
--
1.5.0.g53756
```

Subject: [PATCH] md: Remove broken SIGKILL support
Posted by ebiederm on Thu, 19 Apr 2007 06:56:00 GMT
View Forum Message <> Reply to Message

From: Eric W. Biederman <ebiederm@xmission.com> - unquoted

Currently md_thread calls allow_signal so it can receive a

SIGKILL but then does nothing with it except flush the
sigkill so that it not can use an interruptible sleep.

This whole dance is silly so remove the unnecessary
and broken signal handling logic.

Cc: Neil Brown <neilb@suse.de>
Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
---
 drivers/md/md.c | 6 ------
 1 files changed, 0 insertions(+), 6 deletions(-)

diff --git a/drivers/md/md.c b/drivers/md/md.c
index 1299c23..dfd0cb9 100644
--- a/drivers/md/md.c
+++ b/drivers/md/md.c
@@ -4542,17 +4542,11 @@ static int md_thread(void * arg)
   */

  current->flags |= PF_NOFREEZE;
- allow_signal(SIGKILL);
  while (!kthread_should_stop()) {

   /* We need to wait INTERRUPTIBLE so that
    * we don't add to the load-average.
-   * That means we need to be sure no signals are
-   * pending
    */
- if (signal_pending(current))
-   flush_signals(current);
-
   wait_event_interruptible_timeout
    (thread->wqueue,
     test_bit(THREAD_WAKEUP, &thread->flags)
--
1.5.0.g53756

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: [PATCH] synchro_test: Convert to the kthread API.
Posted by ebiederm on Thu, 19 Apr 2007 06:56:01 GMT
View Forum Message <> Reply to Message

From: Eric W. Biederman <ebiederm@xmission.com> - unquoted

```
Cc: David Howells <dhowells@redhat.com>
Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
---
 kernel/synchro-test.c |   16 ++++++----------
 1 files changed, 6 insertions(+), 10 deletions(-)

diff --git a/kernel/synchro-test.c b/kernel/synchro-test.c
index a4747a6..b1d7fd6 100644
--- a/kernel/synchro-test.c
+++ b/kernel/synchro-test.c
@@ -30,6 +30,7 @@
 #include <linux/timer.h>
 #include <linux/completion.h>
 #include <linux/mutex.h>
+#include <linux/kthread.h>

 #define MAX_THREADS 64

@@ -224,7 +225,6 @@ static int mutexer(void *arg)
 {
  unsigned int N = (unsigned long) arg;

- daemonize("Mutex%u", N);
  set_user_nice(current, 19);

  while (atomic_read(&do_stuff)) {
@@ -246,7 +246,6 @@ static int semaphorer(void *arg)
 {
  unsigned int N = (unsigned long) arg;

- daemonize("Sem%u", N);
  set_user_nice(current, 19);

  while (atomic_read(&do_stuff)) {
@@ -268,7 +267,6 @@ static int reader(void *arg)
 {
  unsigned int N = (unsigned long) arg;

- daemonize("Read%u", N);
  set_user_nice(current, 19);

  while (atomic_read(&do_stuff)) {
@@ -292,7 +290,6 @@ static int writer(void *arg)
 {
  unsigned int N = (unsigned long) arg;

- daemonize("Write%u", N);
```

```
  set_user_nice(current, 19);

  while (atomic_read(&do_stuff)) {
@@ -316,7 +313,6 @@ static int downgrader(void *arg)
 {
  unsigned int N = (unsigned long) arg;

- daemonize("Down%u", N);
  set_user_nice(current, 19);

  while (atomic_read(&do_stuff)) {
@@ -433,27 +429,27 @@ static int __init do_tests(void)
  for (loop = 0; loop < MAX_THREADS; loop++) {
   if (loop < nummx) {
    init_completion(&mx_comp[loop]);
-   kernel_thread(mutexer, (void *) loop, 0);
+   kthread_run(mutexer, (void *) loop, "Mutex%u", loop);
   }

   if (loop < numsm) {
    init_completion(&sm_comp[loop]);
-   kernel_thread(semaphorer, (void *) loop, 0);
+   kthread_run(semaphorer, (void *) loop, "Sem%u", loop);
   }

   if (loop < numrd) {
    init_completion(&rd_comp[loop]);
-   kernel_thread(reader, (void *) loop, 0);
+   kthread_run(reader, (void *) loop, "Read%u", loop);
   }

   if (loop < numwr) {
    init_completion(&wr_comp[loop]);
-   kernel_thread(writer, (void *) loop, 0);
+   kthread_run(writer, (void *) loop, "Write%u", loop);
   }

   if (loop < numdg) {
    init_completion(&dg_comp[loop]);
-   kernel_thread(downgrader, (void *) loop, 0);
+   kthread_run(downgrader, (void *) loop, "Down%u", loop);
   }
  }

--
1.5.0.g53756
```

Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

Subject: [PATCH] synchro_test: Convert to the kthread API.
Posted by ebiederm on Thu, 19 Apr 2007 06:56:02 GMT
View Forum Message <> Reply to Message

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

Subject: [PATCH] i386 balance_irq: Convert to the kthread api.
Posted by ebiederm on Thu, 19 Apr 2007 07:58:31 GMT
View Forum Message <> Reply to Message

This patch just trivial converts from calling kernel_thread and daemonize
to just calling kthread_run.

Cc: Andi Kleen <ak@suse.de>
Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
---
 arch/i386/kernel/io_apic.c | 5 ++---
 1 files changed, 2 insertions(+), 3 deletions(-)

diff --git a/arch/i386/kernel/io_apic.c b/arch/i386/kernel/io_apic.c
index 24ac67c..84b412a 100644
--- a/arch/i386/kernel/io_apic.c
+++ b/arch/i386/kernel/io_apic.c
@@ -34,6 +34,7 @@
 #include <linux/msi.h>
 #include <linux/htirq.h>
 #include <linux/freezer.h>
+#include <linux/kthread.h>

 #include <asm/io.h>
 #include <asm/smp.h>
@@ -660,8 +661,6 @@ static int balanced_irq(void *unused)
  unsigned long prev_balance_time = jiffies;
  long time_remaining = balanced_irq_interval;

- daemonize("kirqd");
-
  /* push everything to CPU 0 to give us a starting point. */

```
  for (i = 0 ; i < NR_IRQS ; i++) {
   irq_desc[i].pending_mask = cpumask_of_cpu(0);
@@ -721,7 +720,7 @@ static int __init balanced_irq_init(void)
  }

  printk(KERN_INFO "Starting balanced_irq\n");
- if (kernel_thread(balanced_irq, NULL, CLONE_KERNEL) >= 0)
+ if (!IS_ERR(kthread_run(balanced_irq, NULL, "kirqd")))
   return 0;
  else
   printk(KERN_ERR "balanced_irq_init: failed to spawn balanced_irq");
--
1.5.0.g53756
```

Subject: [PATCH] i386 voyager: Convert the monitor thread to use the kthread API
Posted by ebiederm on Thu, 19 Apr 2007 07:58:32 GMT
View Forum Message <> Reply to Message

From: Eric W. Biederman <ebiederm@xmission.com>

This patch just trivially replaces kernel_thread and daemonize
with a single call to kthread_run.

CC: James Bottomley <James.Bottomley@HansenPartnership.com>
Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
---
 arch/i386/mach-voyager/voyager_thread.c | 5 ++---
 1 files changed, 2 insertions(+), 3 deletions(-)

```
diff --git a/arch/i386/mach-voyager/voyager_thread.c b/arch/i386/mach-voyager/voyager_thread.c
index ebfd913..ee23d9b 100644
--- a/arch/i386/mach-voyager/voyager_thread.c
+++ b/arch/i386/mach-voyager/voyager_thread.c
@@ -23,6 +23,7 @@
 #include <linux/kmod.h>
 #include <linux/completion.h>
 #include <linux/sched.h>
+#include <linux/kthread.h>
 #include <asm/desc.h>
 #include <asm/voyager.h>
 #include <asm/vic.h>
@@ -43,7 +44,7 @@ static __u8 set_timeout = 0;
```

```
 static int __init
 voyager_thread_start(void)
 {
- if(kernel_thread(thread, NULL, CLONE_KERNEL) < 0) {
+ if (IS_ERR(kthread_run(thread, NULL, "%s", THREAD_NAME))) {
    /* This is serious, but not fatal */
    printk(KERN_ERR "Voyager: Failed to create system monitor thread!!!\n");
    return 1;
@@ -122,8 +123,6 @@ thread(void *unused)

  kvoyagerd_running = 1;

- daemonize(THREAD_NAME);
-
  set_timeout = 0;

  init_timer(&wakeup_timer);
--
1.5.0.g53756
```


Subject: [PATCH] mtd_blkdevs: Convert to use the kthread API
Posted by ebiederm on Thu, 19 Apr 2007 07:58:33 GMT
View Forum Message <> Reply to Message

From: Eric W. Biederman <ebiederm@xmission.com>

thread_run is used intead of kernel_thread, daemonize, and mucking
around blocking signals directly.

CC: David Woodhouse <dwmw2@infradead.org>
Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
---
 drivers/mtd/mtd_blkdevs.c | 19 +++++----------------
 1 files changed, 5 insertions(+), 14 deletions(-)

```
diff --git a/drivers/mtd/mtd_blkdevs.c b/drivers/mtd/mtd_blkdevs.c
index db7397c..ed71d5e 100644
--- a/drivers/mtd/mtd_blkdevs.c
+++ b/drivers/mtd/mtd_blkdevs.c
@@ -21,6 +21,7 @@
 #include <linux/init.h>
 #include <linux/mutex.h>
```

```
 #include <linux/freezer.h>
+#include <linux/kthread.h>
 #include <asm/uaccess.h>

 static LIST_HEAD(blktrans_majors);
@@ -84,17 +85,6 @@ static int mtd_blktrans_thread(void *arg)
  /* we might get involved when memory gets low, so use PF_MEMALLOC */
  current->flags |= PF_MEMALLOC | PF_NOFREEZE;

- daemonize("%sd", tr->name);
-
- /* daemonize() doesn't do this for us since some kernel threads
-    actually want to deal with signals. We can't just call
-    exit_sighand() since that'll cause an oops when we finally
-    do exit. */
- spin_lock_irq(&current->sighand->siglock);
- sigfillset(&current->blocked);
- recalc_sigpending();
- spin_unlock_irq(&current->sighand->siglock);
-
  spin_lock_irq(rq->queue_lock);

  while (!tr->blkcore_priv->exiting) {
@@ -368,6 +358,7 @@ static struct mtd_notifier blktrans_notifier = {

 int register_mtd_blktrans(struct mtd_blktrans_ops *tr)
 {
+ struct task_struct *task;
  int ret, i;

  /* Register the notifier if/when the first device type is
@@ -406,13 +397,13 @@ int register_mtd_blktrans(struct mtd_blktrans_ops *tr)
  blk_queue_hardsect_size(tr->blkcore_priv->rq, tr->blksize);
  tr->blkshift = ffs(tr->blksize) - 1;

- ret = kernel_thread(mtd_blktrans_thread, tr, CLONE_KERNEL);
- if (ret < 0) {
+ task = kthread_run(mtd_blktrans_thread, tr, "%sd", tr->name);
+ if (IS_ERR(task)) {
   blk_cleanup_queue(tr->blkcore_priv->rq);
   unregister_blkdev(tr->major, tr->name);
   kfree(tr->blkcore_priv);
   mutex_unlock(&mtd_table_mutex);
-  return ret;
+  return PTR_ERR(task);
  }

  INIT_LIST_HEAD(&tr->devs);
```

--
1.5.0.g53756

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: [PATCH] cpci_hotplug: Convert to use the kthread API
Posted by ebiederm on Thu, 19 Apr 2007 07:58:34 GMT
View Forum Message <> Reply to Message

From: Eric W. Biederman <ebiederm@xmission.com>

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.

CC: Scott Murray <scottm@somanetworks.com>
Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
---
 drivers/pci/hotplug/cpci_hotplug_core.c | 22 +++++++----------------
 1 files changed, 7 insertions(+), 15 deletions(-)

diff --git a/drivers/pci/hotplug/cpci_hotplug_core.c b/drivers/pci/hotplug/cpci_hotplug_core.c
index 6845515..c620c7e 100644
--- a/drivers/pci/hotplug/cpci_hotplug_core.c
+++ b/drivers/pci/hotplug/cpci_hotplug_core.c
@@ -33,6 +33,7 @@
 #include <linux/init.h>
 #include <linux/interrupt.h>
 #include <linux/smp_lock.h>
+#include <linux/kthread.h>
 #include <asm/atomic.h>
 #include <linux/delay.h>
 #include "cpci_hotplug.h"
@@ -521,17 +522,13 @@ 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))
+  if (thread_finished)
    break;
   do {
    rc = check_slots();
@@ -562,12 +559,8 @@ poll_thread(void *data)
 {
  int rc;

- lock_kernel();
- daemonize("cpci_hp_polld");
- unlock_kernel();
-
  while (1) {
-  if (thread_finished || signal_pending(current))
+  if (thread_finished)
    break;
   do {
    rc = check_slots();
@@ -592,7 +585,7 @@ poll_thread(void *data)
 static int
 cpci_start_thread(void)
 {
- int pid;
+ struct task_struct *task;

  /* initialize our semaphores */
  init_MUTEX_LOCKED(&event_semaphore);
@@ -600,14 +593,13 @@ cpci_start_thread(void)
  thread_finished = 0;

  if (controller->irq)
-  pid = kernel_thread(event_thread, NULL, 0);
+  task = kthread_run(event_thread, NULL, "cpci_hp_eventd");
  else
-  pid = kernel_thread(poll_thread, NULL, 0);
- if (pid < 0) {
+  task = kthread_run(poll_thread, NULL, "cpci_hp_polld");
+ if (IS_ERR(task)) {
   err("Can't start up our thread");
   return -1;
  }
- dbg("Our thread pid = %d", pid);
```

```
 return 0;
}

--
1.5.0.g53756
```

Subject: [PATCH] ibmphp: Convert to use the kthreads API
Posted by ebiederm on Thu, 19 Apr 2007 07:58:35 GMT
View Forum Message <> Reply to Message

From: Eric W. Biederman <ebiederm@xmission.com>

kthread_run replaces kernel_thread and dameonize.

allow_signal is unnecessary and has been removed.
tid_poll was unused and has been removed.

Cc: Jyoti Shah <jshah@us.ibm.com>
Cc: Greg Kroah-Hartman <gregkh@suse.de>
Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
---
 drivers/pci/hotplug/ibmphp_hpc.c | 14 +++++---------
 1 files changed, 5 insertions(+), 9 deletions(-)

```
diff --git a/drivers/pci/hotplug/ibmphp_hpc.c b/drivers/pci/hotplug/ibmphp_hpc.c
index 46abaa8..27e12f1 100644
--- a/drivers/pci/hotplug/ibmphp_hpc.c
+++ b/drivers/pci/hotplug/ibmphp_hpc.c
@@ -34,6 +34,7 @@
 #include <linux/pci.h>
 #include <linux/init.h>
 #include <linux/mutex.h>
+#include <linux/kthread.h>

 #include "ibmphp.h"

@@ -101,7 +102,6 @@ static int to_debug = 0;
 // global variables
 //--------------------------------------------------------------------
 static int ibmphp_shutdown;
-static int tid_poll;
 static struct mutex sem_hpcaccess; // lock access to HPC
```

```
 static struct semaphore semOperations; // lock all operations and
      // access to data structures
@@ -137,7 +137,6 @@ void __init ibmphp_hpc_initvars (void)
  init_MUTEX_LOCKED (&sem_exit);
  to_debug = 0;
  ibmphp_shutdown = 0;
- tid_poll = 0;

  debug ("%s - Exit\n", __FUNCTION__);
 }
@@ -1060,12 +1059,8 @@ static int hpc_poll_thread (void *data)
 {
  debug ("%s - Entry\n", __FUNCTION__);

- daemonize("hpc_poll");
- allow_signal(SIGKILL);
-
  poll_hpc ();

- tid_poll = 0;
  debug ("%s - Exit\n", __FUNCTION__);
  return 0;
 }
@@ -1078,17 +1073,18 @@ static int hpc_poll_thread (void *data)
 *----------------------------------------------------------------------*/
 int __init ibmphp_hpc_start_poll_thread (void)
 {
+ struct task_struct *task;
  int rc = 0;

  debug ("%s - Entry\n", __FUNCTION__);

- tid_poll = kernel_thread (hpc_poll_thread, NULL, 0);
- if (tid_poll < 0) {
+ task = kthread_run(hpc_poll_thread, NULL, "hpc_poll");
+ if (IS_ERR(task)) {
   err ("%s - Error, thread not started\n", __FUNCTION__);
   rc = -1;
  }

- debug ("%s - Exit tid_poll[%d] rc[%d]\n", __FUNCTION__, tid_poll, rc);
+ debug ("%s - Exit rc[%d]\n", __FUNCTION__, rc);
  return rc;
 }

--
1.5.0.g53756
```

Subject: [PATCH] cpqphp: Convert to use the kthread API
Posted by ebiederm on Thu, 19 Apr 2007 07:58:36 GMT
View Forum Message <> Reply to Message

From: Eric W. Biederman <ebiederm@xmission.com>

This patch changes cpqphp to use kthread_run and not
kernel_thread and daemonize to startup and setup
the cpqphp thread.

Cc: Greg Kroah-Hartman <gregkh@suse.de>
Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
---
 drivers/pci/hotplug/cpqphp_ctrl.c | 12 ++++++--------
 1 files changed, 4 insertions(+), 8 deletions(-)

diff --git a/drivers/pci/hotplug/cpqphp_ctrl.c b/drivers/pci/hotplug/cpqphp_ctrl.c
index 79ff6b4..c2c06c4 100644
--- a/drivers/pci/hotplug/cpqphp_ctrl.c
+++ b/drivers/pci/hotplug/cpqphp_ctrl.c
@@ -37,6 +37,7 @@
 #include <linux/smp_lock.h>
 #include <linux/pci.h>
 #include <linux/pci_hotplug.h>
+#include <linux/kthread.h>
 #include "cpqphp.h"

 static u32 configure_new_device(struct controller* ctrl, struct pci_func *func,
@@ -1746,10 +1747,6 @@ static void pushbutton_helper_thread(unsigned long data)
 static int event_thread(void* data)
 {
  struct controller *ctrl;
- lock_kernel();
- daemonize("phpd_event");
-
- unlock_kernel();

  while (1) {
   dbg("!!!!event_thread sleeping\n");
@@ -1771,7 +1768,7 @@ static int event_thread(void* data)

 int cpqhp_event_start_thread(void)

```
 {
- int pid;
+ struct task_struct *task;

  /* initialize our semaphores */
  init_MUTEX(&delay_sem);
@@ -1779,12 +1776,11 @@ int cpqhp_event_start_thread(void)
  init_MUTEX_LOCKED(&event_exit);
  event_finished=0;

- pid = kernel_thread(event_thread, NULL, 0);
- if (pid < 0) {
+ task = kthread_run(event_thread, NULL, "phpd_event");
+ if (IS_ERR(task)) {
   err ("Can't start up our event thread\n");
   return -1;
  }
- dbg("Our event thread pid = %d\n", pid);
  return 0;
 }

--
1.5.0.g53756
```

Subject: [PATCH] pnpbios: Conert to use the kthread API.
Posted by ebiederm on Thu, 19 Apr 2007 07:58:37 GMT
View Forum Message <> Reply to Message

From: Eric W. Biederman <ebiederm@xmission.com>

This patches modifies the pnpbios kernel thread to start
with ktrhead_run not kernel_thread and deamonize. Doing
this makes the code a little simpler and easier to maintain.

Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
---
 drivers/pnp/pnpbios/core.c | 16 +++++++---------
 1 files changed, 7 insertions(+), 9 deletions(-)

diff --git a/drivers/pnp/pnpbios/core.c b/drivers/pnp/pnpbios/core.c
index c2ed53f..3a201b7 100644
--- a/drivers/pnp/pnpbios/core.c

```
+++ b/drivers/pnp/pnpbios/core.c
@@ -62,6 +62,7 @@
 #include <linux/delay.h>
 #include <linux/acpi.h>
 #include <linux/freezer.h>
+#include <linux/kthread.h>

 #include <asm/page.h>
 #include <asm/desc.h>
@@ -159,9 +160,7 @@ static int pnp_dock_thread(void * unused)
 {
  static struct pnp_docking_station_info now;
  int docked = -1, d = 0;
- daemonize("kpnpbiosd");
- allow_signal(SIGKILL);
- while(!unloading && !signal_pending(current))
+ while (!unloading)
  {
   int status;

@@ -170,11 +169,8 @@ static int pnp_dock_thread(void * unused)
    */
   msleep_interruptible(2000);

-  if(signal_pending(current)) {
-   if (try_to_freeze())
-    continue;
-   break;
-  }
+  if (try_to_freeze())
+   continue;

   status = pnp_bios_dock_station_info(&now);

@@ -582,6 +578,7 @@ subsys_initcall(pnpbios_init);

 static int __init pnpbios_thread_init(void)
 {
+ struct task_struct *task;
 #if defined(CONFIG_PPC_MERGE)
  if (check_legacy_ioport(PNPBIOS_BASE))
   return 0;
@@ -590,7 +587,8 @@ static int __init pnpbios_thread_init(void)
   return 0;
 #ifdef CONFIG_HOTPLUG
  init_completion(&unload_sem);
- if (kernel_thread(pnp_dock_thread, NULL, CLONE_KERNEL) > 0)
+ task = kthread_run(pnp_dock_thread, NULL, "kpnpbiosd");
```

```
+ if (!IS_ERR(task))
   unloading = 0;
 #endif
  return 0;
--
1.5.0.g53756
```

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: [PATCH] sas_scsi_host: Convert to use the kthread API
Posted by ebiederm on Thu, 19 Apr 2007 07:58:38 GMT
View Forum Message <> Reply to Message

From: Eric W. Biederman <ebiederm@xmission.com>

This patch modifies the sas scsi host thread startup
to use kthread_run not kernel_thread and deamonize.
kthread_run is slightly simpler and more maintainable.

Cc: Darrick J. Wong <djwong@us.ibm.com>
Cc: James Bottomley <James.Bottomley@SteelEye.com>
Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
---
 drivers/scsi/libsas/sas_scsi_host.c | 11 +++++++-----
 1 files changed, 6 insertions(+), 5 deletions(-)

```
diff --git a/drivers/scsi/libsas/sas_scsi_host.c b/drivers/scsi/libsas/sas_scsi_host.c
index 46ba3a7..7a38ac5 100644
--- a/drivers/scsi/libsas/sas_scsi_host.c
+++ b/drivers/scsi/libsas/sas_scsi_host.c
@@ -40,6 +40,7 @@
 #include <linux/blkdev.h>
 #include <linux/scatterlist.h>
 #include <linux/freezer.h>
+#include <linux/kthread.h>

 /* ---------- SCSI Host glue ---------- */

@@ -870,7 +871,6 @@ static int sas_queue_thread(void *_sas_ha)
  struct sas_ha_struct *sas_ha = _sas_ha;
  struct scsi_core *core = &sas_ha->core;

- daemonize("sas_queue_%d", core->shost->host_no);
  current->flags |= PF_NOFREEZE;
```

```
  complete(&queue_th_comp);
@@ -891,19 +891,20 @@ static int sas_queue_thread(void *_sas_ha)

 int sas_init_queue(struct sas_ha_struct *sas_ha)
 {
- int res;
  struct scsi_core *core = &sas_ha->core;
+ struct task_struct *task;

  spin_lock_init(&core->task_queue_lock);
  core->task_queue_size = 0;
  INIT_LIST_HEAD(&core->task_queue);
  init_MUTEX_LOCKED(&core->queue_thread_sema);

- res = kernel_thread(sas_queue_thread, sas_ha, 0);
- if (res >= 0)
+ task = kthread_run(sas_queue_thread, sas_ha,
+       "sas_queue_%d", core->shost->host_no);
+ if (!IS_ERR(task))
   wait_for_completion(&queue_th_comp);

- return res < 0 ? res : 0;
+ return IS_ERR(task) ? PTR_ERR(task) : 0;
 }

 void sas_shutdown_queue(struct sas_ha_struct *sas_ha)
--
1.5.0.g53756
```

Subject: [PATCH] sparc64/power.c: Convert to use the kthread API
Posted by ebiederm on Thu, 19 Apr 2007 07:58:39 GMT
View Forum Message <> Reply to Message

From: Eric W. Biederman <ebiederm@xmission.com>

This starts the sparc64 powerd using kthread_run
instead of kernel_thread and daemonize. Making the
code slightly simpler and more maintainable.

In addition the unnecessary flush_signals is removed.

```
Cc: David S. Miller <davem@davemloft.net>
Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
---
 arch/sparc64/kernel/power.c |    8 ++++----
 1 files changed, 4 insertions(+), 4 deletions(-)

diff --git a/arch/sparc64/kernel/power.c b/arch/sparc64/kernel/power.c
index 699b24b..03feb8b 100644
--- a/arch/sparc64/kernel/power.c
+++ b/arch/sparc64/kernel/power.c
@@ -13,6 +13,7 @@
 #include <linux/interrupt.h>
 #include <linux/pm.h>
 #include <linux/syscalls.h>
+#include <linux/kthread.h>

 #include <asm/system.h>
 #include <asm/auxio.h>
@@ -81,15 +82,12 @@ static int powerd(void *__unused)
  char *argv[] = { "/sbin/shutdown", "-h", "now", NULL };
  DECLARE_WAITQUEUE(wait, current);

- daemonize("powerd");
-
  add_wait_queue(&powerd_wait, &wait);
 again:
  for (;;) {
   set_task_state(current, TASK_INTERRUPTIBLE);
   if (button_pressed)
    break;
-  flush_signals(current);
   schedule();
  }
  __set_current_state(TASK_RUNNING);
@@ -128,7 +126,9 @@ static int __devinit power_probe(struct of_device *op, const struct
of_device_id
  poweroff_method = machine_halt;  /* able to use the standard halt */

  if (has_button_interrupt(irq, op->node)) {
-  if (kernel_thread(powerd, NULL, CLONE_FS) < 0) {
+  struct task_struct *task;
+  task = kthread_urn(powerd, NULL, "powerd");
+  if (IS_ERR(task)) {
    printk("Failed to start power daemon.\n");
    return 0;
   }
--
1.5.0.g53756
```

Subject: [PATCH] s390/net/lcs: Convert to the kthread API
Posted by ebiederm on Thu, 19 Apr 2007 07:58:40 GMT
View Forum Message <> Reply to Message

From: Eric W. Biederman <ebiederm@xmission.com>

Use kthread_run to start the lcs kernel threads not a
combination of kernel_thread and daemonize. This makes
the code slightly simpler and more maintainable.

Cc: Frank Pavlic <fpavlic@de.ibm.com>
Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
---
drivers/s390/net/lcs.c | 8 ++++-----
1 files changed, 3 insertions(+), 5 deletions(-)

```
diff --git a/drivers/s390/net/lcs.c b/drivers/s390/net/lcs.c
index 08a994f..0300d87 100644
--- a/drivers/s390/net/lcs.c
+++ b/drivers/s390/net/lcs.c
@@ -36,6 +36,7 @@
 #include <linux/in.h>
 #include <linux/igmp.h>
 #include <linux/delay.h>
+#include <linux/kthread.h>
 #include <net/arp.h>
 #include <net/ip.h>

@@ -1248,7 +1249,6 @@ lcs_register_mc_addresses(void *data)
  struct in_device *in4_dev;

  card = (struct lcs_card *) data;
- daemonize("regipm");

  if (!lcs_do_run_thread(card, LCS_SET_MC_THREAD))
   return 0;
@@ -1728,11 +1728,10 @@ lcs_start_kernel_thread(struct work_struct *work)
  struct lcs_card *card = container_of(work, struct lcs_card, kernel_thread_starter);
  LCS_DBF_TEXT(5, trace, "krnthrd");
  if (lcs_do_start_thread(card, LCS_RECOVERY_THREAD))
-  kernel_thread(lcs_recovery, (void *) card, SIGCHLD);
```

```
+  kthread_run(lcs_recovery, card, "lcs_recover");
 #ifdef CONFIG_IP_MULTICAST
  if (lcs_do_start_thread(card, LCS_SET_MC_THREAD))
-  kernel_thread(lcs_register_mc_addresses,
-    (void *) card, SIGCHLD);
+  kernel_run(lcs_register_mc_addresses, card, "regipm");
 #endif
 }

@@ -2232,7 +2231,6 @@ lcs_recovery(void *ptr)
        int rc;

  card = (struct lcs_card *) ptr;
- daemonize("lcs_recover");

  LCS_DBF_TEXT(4, trace, "recover1");
  if (!lcs_do_run_thread(card, LCS_RECOVERY_THREAD))
--
1.5.0.g53756
```

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: [PATCH] s390 qeth: Convert to use the kthread API
Posted by ebiederm on Thu, 19 Apr 2007 07:58:41 GMT
View Forum Message <> Reply to Message

From: Eric W. Biederman <ebiederm@xmission.com>

This patch modifies the qeth_recover thread to be started
with kthread_run not a combination of kernel_thread and
daemonize. Resulting in slightly simpler and more maintainable
code.

Cc: Frank Pavlic <fpavlic@de.ibm.com>
Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
---
 drivers/s390/net/qeth_main.c | 4 ++--
 1 files changed, 2 insertions(+), 2 deletions(-)

diff --git a/drivers/s390/net/qeth_main.c b/drivers/s390/net/qeth_main.c
index ad7792d..8234846 100644
--- a/drivers/s390/net/qeth_main.c
+++ b/drivers/s390/net/qeth_main.c
@@ -50,6 +50,7 @@

```
 #include <linux/mii.h>
 #include <linux/rcupdate.h>
 #include <linux/ethtool.h>
+#include <linux/kthread.h>

 #include <net/arp.h>
 #include <net/ip.h>
@@ -957,7 +958,6 @@ qeth_recover(void *ptr)
  int rc = 0;

  card = (struct qeth_card *) ptr;
- daemonize("qeth_recover");
  QETH_DBF_TEXT(trace,2,"recover1");
  QETH_DBF_HEX(trace, 2, &card, sizeof(void *));
  if (!qeth_do_run_thread(card, QETH_RECOVER_THREAD))
@@ -1014,7 +1014,7 @@ qeth_start_kernel_thread(struct work_struct *work)
      card->write.state != CH_STATE_UP)
   return;
  if (qeth_do_start_thread(card, QETH_RECOVER_THREAD))
-  kernel_thread(qeth_recover, (void *) card, SIGCHLD);
+  kthread_run(qeth_recover, card, "qeth_recover");
 }


--
1.5.0.g53756


_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers
```

Subject: [PATCH] s390/scsi/zfcp_erp: Convert to use the kthread API
Posted by ebiederm on Thu, 19 Apr 2007 07:58:42 GMT

View Forum Message <> Reply to Message

From: Eric W. Biederman <ebiederm@xmission.com>

Modify zfcperp%s to be started with kthread_run not
a combination of kernel_thread, daemonize and siginitsetinv
making the code slightly simpler and more maintainable.

Cc: Swen Schillig <swen@vnet.ibm.com>
Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
---
 drivers/s390/scsi/zfcp_erp.c | 13 ++++++-------
 1 files changed, 6 insertions(+), 7 deletions(-)

```
diff --git a/drivers/s390/scsi/zfcp_erp.c b/drivers/s390/scsi/zfcp_erp.c
index 66c0b09..f26536d 100644
--- a/drivers/s390/scsi/zfcp_erp.c
+++ b/drivers/s390/scsi/zfcp_erp.c
@@ -21,6 +21,7 @@

 #define ZFCP_LOG_AREA   ZFCP_LOG_AREA_ERP

+#include <linux/kthread.h>
 #include "zfcp_ext.h"

 static int zfcp_erp_adisc(struct zfcp_port *);
@@ -985,12 +986,13 @@ static void zfcp_erp_action_dismiss(struct zfcp_erp_action
*erp_action)
 int
 zfcp_erp_thread_setup(struct zfcp_adapter *adapter)
 {
- int retval = 0;
+ struct task_struct *task;

  atomic_clear_mask(ZFCP_STATUS_ADAPTER_ERP_THREAD_UP, &adapter->status);

- retval = kernel_thread(zfcp_erp_thread, adapter, SIGCHLD);
- if (retval < 0) {
+ task = kthread_run(zfcp_erp_thread, adapter,
+      "zfcperp%s", zfcp_get_busid_by_adapter(adapter));
+ if (IS_ERR(task)) {
    ZFCP_LOG_NORMAL("error: creation of erp thread failed for "
      "adapter %s\n",
      zfcp_get_busid_by_adapter(adapter));
@@ -1002,7 +1004,7 @@ zfcp_erp_thread_setup(struct zfcp_adapter *adapter)
   debug_text_event(adapter->erp_dbf, 5, "a_thset_ok");
  }

- return (retval < 0);
+ return IS_ERR(task);
 }

 /*
@@ -1054,9 +1056,6 @@ zfcp_erp_thread(void *data)
  struct zfcp_erp_action *erp_action;
  unsigned long flags;

- daemonize("zfcperp%s", zfcp_get_busid_by_adapter(adapter));
- /* Block all signals */
- siginitsetinv(&current->blocked, 0);
  atomic_set_mask(ZFCP_STATUS_ADAPTER_ERP_THREAD_UP, &adapter->status);
```

```
  debug_text_event(adapter->erp_dbf, 5, "a_th_run");
  wake_up(&adapter->erp_thread_wqh);
--
1.5.0.g53756
```

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: [PATCH] arm ecard: Conver to use the kthread API.
Posted by ebiederm on Thu, 19 Apr 2007 07:58:43 GMT
View Forum Message <> Reply to Message

From: Eric W. Biederman <ebiederm@xmission.com>

This patch modifies the startup of kecardd to use
kthread_run not a kernel_thread combination of kernel_thread
and daemonize. Making the code slightly simpler and more
maintainable.

Cc: Russell King <rmk+kernel@arm.linux.org.uk>
Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
---
 arch/arm/kernel/ecard.c | 14 +++++++-------
 1 files changed, 7 insertions(+), 7 deletions(-)

diff --git a/arch/arm/kernel/ecard.c b/arch/arm/kernel/ecard.c
index f1c0fb9..6c15f5f 100644
--- a/arch/arm/kernel/ecard.c
+++ b/arch/arm/kernel/ecard.c
@@ -40,6 +40,7 @@
 #include <linux/device.h>
 #include <linux/init.h>
 #include <linux/mutex.h>
+#include <linux/kthread.h>

 #include <asm/dma.h>
 #include <asm/ecard.h>
@@ -263,8 +264,6 @@ static int ecard_init_mm(void)
 static int
 ecard_task(void * unused)
 {
-	daemonize("kecardd");
-
 	/*
 	 * Allocate a mm. We're not a lazy-TLB kernel task since we need

```
   * to set page table entries where the user space would be.  Note
@@ -1058,13 +1057,14 @@ ecard_probe(int slot, card_type_t type)
  */
 static int __init ecard_init(void)
 {
- int slot, irqhw, ret;
+ struct task_struct *task;
+ int slot, irqhw;

- ret = kernel_thread(ecard_task, NULL, CLONE_KERNEL);
- if (ret < 0) {
+ task = kthread_run(ecard_task, NULL, "kecardd");
+ if (IS_ERR(task)) {
   printk(KERN_ERR "Ecard: unable to create kernel thread: %d\n",
-        ret);
-  return ret;
+        PTR_ERR(task));
+  return PTR_ERR(task);
  }

  printk("Probing expansion cards\n");
--
1.5.0.g53756
```

____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

Subject: [PATCH] ia64 sn xpc: Convert to use kthread API.
Posted by ebiederm on Thu, 19 Apr 2007 07:58:44 GMT
View Forum Message <> Reply to Message

From: Eric W. Biederman <ebiederm@xmission.com>

This patch starts the xpc kernel threads using kthread_run
not a combination of kernel_thread and daemonize. Resuling
in slightly simpler and more maintainable code.

Cc: Jes Sorensen <jes@sgi.com>
Cc: Tony Luck <tony.luck@intel.com>
Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>

```
 arch/ia64/sn/kernel/xpc_main.c |   31 +++++++++++++------------------
 1 files changed, 13 insertions(+), 18 deletions(-)

diff --git a/arch/ia64/sn/kernel/xpc_main.c b/arch/ia64/sn/kernel/xpc_main.c
```

```
index e336e16..5b53642 100644
--- a/arch/ia64/sn/kernel/xpc_main.c
+++ b/arch/ia64/sn/kernel/xpc_main.c
@@ -56,6 +56,7 @@
 #include <linux/reboot.h>
 #include <linux/completion.h>
 #include <linux/kdebug.h>
+#include <linux/kthread.h>
 #include <asm/sn/intr.h>
 #include <asm/sn/sn_sal.h>
 #include <asm/uaccess.h>
@@ -253,8 +254,6 @@ xpc_hb_checker(void *ignore)

  /* this thread was marked active by xpc_hb_init() */

- daemonize(XPC_HB_CHECK_THREAD_NAME);
-
  set_cpus_allowed(current, cpumask_of_cpu(XPC_HB_CHECK_CPU));

  xpc_hb_check_timeout = jiffies + (xpc_hb_check_interval * HZ);
@@ -324,8 +323,6 @@ xpc_hb_checker(void *ignore)
 static int
 xpc_initiate_discovery(void *ignore)
 {
- daemonize(XPC_DISCOVERY_THREAD_NAME);
-
  xpc_discovery();

  dev_dbg(xpc_part, "discovery thread is exiting\n");
@@ -494,8 +491,6 @@ xpc_activating(void *__partid)

  dev_dbg(xpc_part, "bringing partition %d up\n", partid);

- daemonize("xpc%02d", partid);
-
  /*
   * This thread needs to run at a realtime priority to prevent a
   * significant performance degradation.
@@ -559,7 +554,7 @@ xpc_activate_partition(struct xpc_partition *part)
 {
  partid_t partid = XPC_PARTID(part);
  unsigned long irq_flags;
- pid_t pid;
+ struct task_struct *task;


  spin_lock_irqsave(&part->act_lock, irq_flags);
@@ -571,9 +566,10 @@ xpc_activate_partition(struct xpc_partition *part)
```

```
  spin_unlock_irqrestore(&part->act_lock, irq_flags);

- pid = kernel_thread(xpc_activating, (void *) ((u64) partid), 0);
+ task = kthread_run(xpc_activating, (void *) ((u64) partid),
+     "xpc%02d", partid);

- if (unlikely(pid <= 0)) {
+ if (unlikely(IS_ERR(task))) {
    spin_lock_irqsave(&part->act_lock, irq_flags);
    part->act_state = XPC_P_INACTIVE;
    XPC_SET_REASON(part, xpcCloneKThreadFailed, __LINE__);
@@ -724,8 +720,6 @@ xpc_daemonize_kthread(void *args)
  unsigned long irq_flags;


- daemonize("xpc%02dc%d", partid, ch_number);
-
  dev_dbg(xpc_chan, "kthread starting, partid=%d, channel=%d\n",
    partid, ch_number);

@@ -844,8 +838,9 @@ xpc_create_kthreads(struct xpc_channel *ch, int needed,
    (void) xpc_part_ref(part);
    xpc_msgqueue_ref(ch);

-   pid = kernel_thread(xpc_daemonize_kthread, (void *) args, 0);
-   if (pid < 0) {
+   task = kthread_run(xpc_daemonize_kthread, args,
+        "xpc%02dc%d", partid, ch_number);
+   if (IS_ERR(task)) {
      /* the fork failed */

      /*
@@ -1222,7 +1217,7 @@ xpc_init(void)
  int ret;
  partid_t partid;
  struct xpc_partition *part;
- pid_t pid;
+ struct task_struct *task;
  size_t buf_size;


@@ -1353,8 +1348,8 @@ xpc_init(void)
    * The real work-horse behind xpc.  This processes incoming
    * interrupts and monitors remote heartbeats.
    */
- pid = kernel_thread(xpc_hb_checker, NULL, 0);
- if (pid < 0) {
```

```
+ task = kthread_run(xpc_hb_checker, NULL, XPC_HB_CHECK_THREAD_NAME);
+ if (IS_ERR(task)) {
   dev_err(xpc_part, "failed while forking hb check thread\n");

   /* indicate to others that our reserved page is uninitialized */
@@ -1384,8 +1379,8 @@ xpc_init(void)
   * activate based on info provided by SAL. This new thread is short
   * lived and will exit once discovery is complete.
   */
- pid = kernel_thread(xpc_initiate_discovery, NULL, 0);
- if (pid < 0) {
+ task = kthread_run(xpc_initiate_discovery, NULL, XPC_DISCOVERY_THREAD_NAME);
+ if (IS_ERR(task)) {
   dev_err(xpc_part, "failed while forking discovery thread\n");

   /* mark this new thread as a non-starter */
--
1.5.0.g53756
```

Subject: [PATCH] powerpc pseries eeh: Convert to kthread API
Posted by ebiederm on Thu, 19 Apr 2007 07:58:45 GMT
View Forum Message <> Reply to Message

From: Eric W. Biederman <ebiederm@xmission.com>

This patch modifies the startup of eehd to use kthread_run
not a combination of kernel_thread and daemonize. Making
the code slightly simpler and more maintainable.

Cc: Paul Mackerras <paulus@samba.org>
Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
---
 arch/powerpc/platforms/pseries/eeh_event.c | 4 ++--
 1 files changed, 2 insertions(+), 2 deletions(-)

diff --git a/arch/powerpc/platforms/pseries/eeh_event.c
b/arch/powerpc/platforms/pseries/eeh_event.c
index 221dec8..fe7c2e0 100644
--- a/arch/powerpc/platforms/pseries/eeh_event.c
+++ b/arch/powerpc/platforms/pseries/eeh_event.c
@@ -23,6 +23,7 @@
 #include <linux/mutex.h>

```
 #include <linux/pci.h>
 #include <linux/workqueue.h>
+#include <linux/kthread.h>
 #include <asm/eeh_event.h>
 #include <asm/ppc-pci.h>

@@ -59,7 +60,6 @@ static int eeh_event_handler(void * dummy)
  struct eeh_event *event;
  struct pci_dn *pdn;

- daemonize ("eehd");
  set_current_state(TASK_INTERRUPTIBLE);

  spin_lock_irqsave(&eeh_eventlist_lock, flags);
@@ -105,7 +105,7 @@ static int eeh_event_handler(void * dummy)
  */
 static void eeh_thread_launcher(struct work_struct *dummy)
 {
- if (kernel_thread(eeh_event_handler, NULL, CLONE_KERNEL) < 0)
+ if (IS_ERR(kthread_run(eeh_event_handler, NULL, "eehd")))
   printk(KERN_ERR "Failed to start EEH daemon\n");
 }

--
1.5.0.g53756
```

Subject: [PATCH] powerpc pseries rtasd: Convert to kthread API.
Posted by ebiederm on Thu, 19 Apr 2007 07:58:46 GMT
View Forum Message <> Reply to Message

From: Eric W. Biederman <ebiederm@xmission.com>

This patch modifies the startup of rtasd to use kthread_run instaed of
a combination of kernel_thread and daemonize. Making the code a little
simpler and more maintainble.

Cc: Paul Mackerras <paulus@samba.org>
Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
---
 arch/powerpc/platforms/pseries/rtasd.c | 5 ++---
 1 files changed, 2 insertions(+), 3 deletions(-)

```
diff --git a/arch/powerpc/platforms/pseries/rtasd.c b/arch/powerpc/platforms/pseries/rtasd.c
index 77d0937..919a374 100644
--- a/arch/powerpc/platforms/pseries/rtasd.c
+++ b/arch/powerpc/platforms/pseries/rtasd.c
@@ -20,6 +20,7 @@
 #include <linux/spinlock.h>
 #include <linux/cpu.h>
 #include <linux/delay.h>
+#include <linux/kthread.h>

 #include <asm/uaccess.h>
 #include <asm/io.h>
@@ -429,8 +430,6 @@ static int rtasd(void *unused)
  int event_scan = rtas_token("event-scan");
  int rc;

- daemonize("rtasd");
-
  if (event_scan == RTAS_UNKNOWN_SERVICE || get_eventscan_parms() == -1)
   goto error;

@@ -497,7 +496,7 @@ static int __init rtas_init(void)
  else
   printk(KERN_ERR "Failed to create error_log proc entry\n");

- if (kernel_thread(rtasd, NULL, CLONE_FS) < 0)
+ if (IS_ERR(kthread_run(rtasd, NULL, "rtasd")))
   printk(KERN_ERR "Failed to start RTAS daemon\n");

  return 0;
--
1.5.0.g53756
```

Subject: [PATCH] macintosh/therm_pm72.c: Convert to kthread API.
Posted by ebiederm on Thu, 19 Apr 2007 07:58:47 GMT
View Forum Message <> Reply to Message

From: Eric W. Biederman <ebiederm@xmission.com>

This patch modifies startup of the kfand to use kthread_run
not a combination of kernel_thread and daemonize, making
the code a little simpler and more maintaintable.

```
Cc: Benjamin Herrenschmidt <benh@kernel.crashing.org>
Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
---
 drivers/macintosh/therm_pm72.c |   11 +++++++-----
 1 files changed, 6 insertions(+), 5 deletions(-)

diff --git a/drivers/macintosh/therm_pm72.c b/drivers/macintosh/therm_pm72.c
index b002a4b..7e9cbb7 100644
--- a/drivers/macintosh/therm_pm72.c
+++ b/drivers/macintosh/therm_pm72.c
@@ -121,6 +121,7 @@
 #include <linux/reboot.h>
 #include <linux/kmod.h>
 #include <linux/i2c.h>
+#include <linux/kthread.h>
 #include <asm/prom.h>
 #include <asm/machdep.h>
 #include <asm/io.h>
@@ -161,7 +162,7 @@ static struct slots_pid_state  slots_state;
 static int    state;
 static int    cpu_count;
 static int    cpu_pid_type;
-static pid_t    ctrl_task;
+static int    ctrl_task;
 static struct completion  ctrl_complete;
 static int    critical_state;
 static int    rackmac;
@@ -1779,8 +1780,6 @@ static int call_critical_overtemp(void)
  */
 static int main_control_loop(void *x)
 {
- daemonize("kfand");
-
  DBG("main_control_loop started\n");

  down(&driver_lock);
@@ -1859,7 +1858,6 @@ static int main_control_loop(void *x)
    machine_power_off();
   }

-  // FIXME: Deal with signals
   elapsed = jiffies - start;
   if (elapsed < HZ)
    schedule_timeout_interruptible(HZ - elapsed);
@@ -1954,9 +1952,12 @@ static int create_control_loops(void)
 */
 static void start_control_loops(void)
```

```
 {
+ struct task_struct *task;
  init_completion(&ctrl_complete);

- ctrl_task = kernel_thread(main_control_loop, NULL, SIGCHLD | CLONE_KERNEL);
+ task = kthread_run(main_control_loop, NULL, "kfand");
+ if (!IS_ERR(task))
+  ctrl_task = 1;
 }

 /*
--
1.5.0.g53756
```

Subject: [PATCH] macintosh/therm_windtunnel.c: Convert to kthread API.
Posted by ebiederm on Thu, 19 Apr 2007 07:58:48 GMT
View Forum Message <> Reply to Message

From: Eric W. Biederman <ebiederm@xmission.com>

Start the g4fand using kthread_run not a combination
of kernel_thread and deamonize. This makes the code
a little simpler and more maintainable.

Cc: Benjamin Herrenschmidt <benh@kernel.crashing.org>
Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
---
 drivers/macintosh/therm_windtunnel.c | 5 ++---
 1 files changed, 2 insertions(+), 3 deletions(-)

```
diff --git a/drivers/macintosh/therm_windtunnel.c b/drivers/macintosh/therm_windtunnel.c
index a1d3a98..5d888e7 100644
--- a/drivers/macintosh/therm_windtunnel.c
+++ b/drivers/macintosh/therm_windtunnel.c
@@ -36,6 +36,7 @@
 #include <linux/i2c.h>
 #include <linux/slab.h>
 #include <linux/init.h>
+#include <linux/kthread.h>

 #include <asm/prom.h>
 #include <asm/machdep.h>
```

```
@@ -62,7 +63,6 @@ I2C_CLIENT_INSMOD;
 static struct {
  volatile int  running;
  struct completion completion;
- pid_t   poll_task;

  struct semaphore  lock;
  struct of_device *of_dev;
@@ -285,7 +285,6 @@ restore_regs( void )
 static int
 control_loop( void *dummy )
 {
- daemonize("g4fand");

  down( &x.lock );
  setup_hardware();
@@ -323,7 +322,7 @@ do_attach( struct i2c_adapter *adapter )
    if( x.thermostat && x.fan ) {
     x.running = 1;
     init_completion( &x.completion );
-    x.poll_task = kernel_thread( control_loop, NULL, SIGCHLD | CLONE_KERNEL );
+    kthread_run( control_loop, NULL, "g4fand");
    }
   }
  return ret;
--
1.5.0.g53756
```

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: [PATCH] macintosh/adb: Convert to the kthread API
Posted by ebiederm on Thu, 19 Apr 2007 07:58:49 GMT
View Forum Message <> Reply to Message

From: Eric W. Biederman <ebiederm@xmission.com>

This patch modifies the startup of kadbprobe to use
kthread_run instead of scheduling a work event which
later calls kernel_thread and in the thread calls
daemonize and blocks signals. kthread_run is simpler
and more maintainable.

The variable pid_t adb_probe_task_pid is replaced by
a struct task_struct variable named adb_probe_task.

Which works equally well with for testing if the current
process is the adb_probe thread, does not get confused
in the presence of a pid namespace and is easier to
compare against current as it is the same type.

The result is code that is slightly simpler and easier
to maintain.

Cc: Benjamin Herrenschmidt <benh@kernel.crashing.org>
Cc: Paul Mackerras <paulus@samba.org>
Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
---
 drivers/macintosh/adb.c | 32 +++++++------------------------
 1 files changed, 7 insertions(+), 25 deletions(-)

diff --git a/drivers/macintosh/adb.c b/drivers/macintosh/adb.c
index adfea3c..09c5261 100644
--- a/drivers/macintosh/adb.c
+++ b/drivers/macintosh/adb.c
@@ -35,6 +35,7 @@
 #include <linux/spinlock.h>
 #include <linux/completion.h>
 #include <linux/device.h>
+#include <linux/kthread.h>

 #include <asm/uaccess.h>
 #include <asm/semaphore.h>
@@ -82,7 +83,7 @@ struct adb_driver *adb_controller;
 BLOCKING_NOTIFIER_HEAD(adb_client_list);
 static int adb_got_sleep;
 static int adb_inited;
-static pid_t adb_probe_task_pid;
+static struct task_struct *adb_probe_task;
 static DECLARE_MUTEX(adb_probe_mutex);
 static struct completion adb_probe_task_comp;
 static int sleepy_trackpad;
@@ -137,8 +138,7 @@ static void printADBreply(struct adb_request *req)

 static __inline__ void adb_wait_ms(unsigned int ms)
 {
- if (current->pid && adb_probe_task_pid &&
-   adb_probe_task_pid == current->pid)
+ if (adb_probe_task == current)
   msleep(ms);
  else
   mdelay(ms);
@@ -245,35 +245,19 @@ static int adb_scan_bus(void)
  * This kernel task handles ADB probing. It dies once probing is

```
  * completed.
  */
-static int
-adb_probe_task(void *x)
+static int adb_probe(void *x)
 {
- sigset_t blocked;
-
- strcpy(current->comm, "kadbprobe");
-
- sigfillset(&blocked);
- sigprocmask(SIG_BLOCK, &blocked, NULL);
- flush_signals(current);

  printk(KERN_INFO "adb: starting probe task...\n");
  do_adb_reset_bus();
  printk(KERN_INFO "adb: finished probe task...\n");

- adb_probe_task_pid = 0;
+ adb_probe_task = NULL;
  up(&adb_probe_mutex);

  return 0;
 }

-static void
-__adb_probe_task(struct work_struct *bullshit)
-{
- adb_probe_task_pid = kernel_thread(adb_probe_task, NULL, SIGCHLD | CLONE_KERNEL);
-}
-
-static DECLARE_WORK(adb_reset_work, __adb_probe_task);
-
 int
 adb_reset_bus(void)
 {
@@ -283,7 +267,7 @@ adb_reset_bus(void)
  }

  down(&adb_probe_mutex);
- schedule_work(&adb_reset_work);
+ adb_probe_task = kthread_run(adb_probe, NULL, "kadbprobe");
  return 0;
 }

@@ -469,9 +453,7 @@ adb_request(struct adb_request *req, void (*done)(struct adb_request *),
  /* Synchronous requests send from the probe thread cause it to
   * block. Beware that the "done" callback will be overriden !
```

```
   */
- if ((flags & ADBREQ_SYNC) &&
-     (current->pid && adb_probe_task_pid &&
-     adb_probe_task_pid == current->pid)) {
+ if ((flags & ADBREQ_SYNC) && (current == adb_probe_task)) {
   req->done = adb_probe_wakeup;
   rc = adb_controller->send_request(req, 0);
   if (rc || req->complete)
--
1.5.0.g53756
```

Subject: [PATCH] macintosh/mediabay: Convert to kthread API.
Posted by ebiederm on Thu, 19 Apr 2007 07:58:50 GMT
View Forum Message <> Reply to Message

From: Eric W. Biederman <ebiederm@xmission.com>

This patch modifies the startup of the media_bay_task
to use kthread_run and not a combination of kernel_thread,
deamonize and sigfillset.

In addition since we now always want to ignore signals
the MB_IGNORE_SIGNALS define is removed along with the
test for signal_pending.

The result is slightly simpler code that is more
maintainable.

Cc: Benjamin Herrenschmidt <benh@kernel.crashing.org>
Cc: Paul Mackerras <paulus@samba.org>
Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
---
 drivers/macintosh/mediabay.c | 11 ++---------
 1 files changed, 2 insertions(+), 9 deletions(-)

diff --git a/drivers/macintosh/mediabay.c b/drivers/macintosh/mediabay.c
index c803d2b..90c853e 100644
--- a/drivers/macintosh/mediabay.c
+++ b/drivers/macintosh/mediabay.c
@@ -20,6 +20,7 @@
 #include <linux/stddef.h>
 #include <linux/init.h>

```
 #include <linux/ide.h>
+#include <linux/kthread.h>
 #include <asm/prom.h>
 #include <asm/pgtable.h>
 #include <asm/io.h>
@@ -35,7 +36,6 @@


 #define MB_DEBUG
-#define MB_IGNORE_SIGNALS

 #ifdef MB_DEBUG
 #define MBDBG(fmt, arg...) printk(KERN_INFO fmt , ## arg)
@@ -622,11 +622,6 @@ static int media_bay_task(void *x)
 {
  int i;

- strcpy(current->comm, "media-bay");
-#ifdef MB_IGNORE_SIGNALS
- sigfillset(&current->blocked);
-#endif
-
  for (;;) {
   for (i = 0; i < media_bay_count; ++i) {
    down(&media_bays[i].lock);
@@ -636,8 +631,6 @@ static int media_bay_task(void *x)
   }

   msleep_interruptible(MB_POLL_DELAY);
-  if (signal_pending(current))
-   return 0;
  }
 }

@@ -699,7 +692,7 @@ static int __devinit media_bay_attach(struct macio_dev *mdev, const
struct of_de

  /* Startup kernel thread */
  if (i == 0)
-  kernel_thread(media_bay_task, NULL, CLONE_KERNEL);
+  kthread_run(media_bay_task, NULL, "media-bay");

  return 0;

--
1.5.0.g53756
```

_____

Subject: [PATCH] bluetooth bnep: Convert to kthread API.
Posted by ebiederm on Thu, 19 Apr 2007 07:58:51 GMT
View Forum Message <> Reply to Message

From: Eric W. Biederman <ebiederm@xmission.com>

This patch starts kbenpd using kthread_run replacing
a combination of kernel_thread and daemonize. Making
the code a little simpler and more maintainable.

Cc: Marcel Holtmann <marcel@holtmann.org>
Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
---
net/bluetooth/bnep/core.c | 8 +++++---
1 files changed, 5 insertions(+), 3 deletions(-)

```
diff --git a/net/bluetooth/bnep/core.c b/net/bluetooth/bnep/core.c
index a9f1e88..de3caed 100644
--- a/net/bluetooth/bnep/core.c
+++ b/net/bluetooth/bnep/core.c
@@ -32,6 +32,7 @@
 #include <linux/module.h>

 #include <linux/kernel.h>
+#include <linux/kthread.h>
 #include <linux/sched.h>
 #include <linux/signal.h>
 #include <linux/init.h>
@@ -473,7 +474,6 @@ static int bnep_session(void *arg)

  BT_DBG("");

- daemonize("kbnepd %s", dev->name);
  set_user_nice(current, -15);

  init_waitqueue_entry(&wait, current);
@@ -539,6 +539,7 @@ static struct device *bnep_get_device(struct bnep_session *session)

 int bnep_add_connection(struct bnep_connadd_req *req, struct socket *sock)
 {
+ struct task_struct *task;
  struct net_device *dev;
  struct bnep_session *s, *ss;
```

```
  u8 dst[ETH_ALEN], src[ETH_ALEN];
@@ -598,9 +599,10 @@ int bnep_add_connection(struct bnep_connadd_req *req, struct socket
*sock)

  __bnep_link_session(s);

- err = kernel_thread(bnep_session, s, CLONE_KERNEL);
- if (err < 0) {
+ task = kthread_run(bnep_session, s, "kbnepd %s", dev->name);
+ if (IS_ERR(task)) {
   /* Session thread start failed, gotta cleanup. */
+  err = PTR_ERR(task);
   unregister_netdev(dev);
   __bnep_unlink_session(s);
   goto failed;
--
1.5.0.g53756
```

Subject: [PATCH] bluetooth cmtp: Convert to use kthread API.
Posted by ebiederm on Thu, 19 Apr 2007 07:58:52 GMT
View Forum Message <> Reply to Message

From: Eric W. Biederman <ebiederm@xmission.com>

This patch modifies the kcmptd_ctr_%d daemon using kthread_run
instead of a combination of kernel_thread and daemonize making
the code a little simpler and more maintainable.

Cc: Marcel Holtmann <marcel@holtmann.org>
Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
---
 net/bluetooth/cmtp/core.c | 8 +++++---
 1 files changed, 5 insertions(+), 3 deletions(-)

diff --git a/net/bluetooth/cmtp/core.c b/net/bluetooth/cmtp/core.c
index e1b9db9..993303f 100644
--- a/net/bluetooth/cmtp/core.c
+++ b/net/bluetooth/cmtp/core.c
@@ -35,6 +35,7 @@
 #include <linux/file.h>
 #include <linux/init.h>
 #include <linux/freezer.h>

```
+#include <linux/kthread.h>
 #include <net/sock.h>

 #include <linux/isdn/capilli.h>
@@ -286,7 +287,6 @@ static int cmtp_session(void *arg)

  BT_DBG("session %p", session);

- daemonize("kcmtpd_ctr_%d", session->num);
  set_user_nice(current, -15);

  init_waitqueue_entry(&wait, current);
@@ -329,6 +329,7 @@ static int cmtp_session(void *arg)
 int cmtp_add_connection(struct cmtp_connadd_req *req, struct socket *sock)
 {
  struct cmtp_session *session, *s;
+ struct task_struct *task;
  bdaddr_t src, dst;
  int i, err;

@@ -375,8 +376,9 @@ int cmtp_add_connection(struct cmtp_connadd_req *req, struct socket
*sock)

  __cmtp_link_session(session);

- err = kernel_thread(cmtp_session, session, CLONE_KERNEL);
- if (err < 0)
+ task = kthread_run(cmtp_session, session, "kcmtpd_ctr_%d", session->num);
+ err = PTR_ERR(task);
+ if (IS_ERR(task))
   goto unlink;

  if (!(session->flags & (1 << CMTP_LOOPBACK))) {
--
1.5.0.g53756
```


Subject: [PATCH] bluetooth hidp: Convert to kthread API.
Posted by ebiederm on Thu, 19 Apr 2007 07:58:53 GMT
View Forum Message <> Reply to Message

From: Eric W. Biederman <ebiederm@xmission.com>

This patch starts up khidp using kthread_run instead
of kernel_thread and daemonize, resulting is slightly
simpler and more maintainable code.

Cc: Marcel Holtmann <marcel@holtmann.org>
Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
---
 net/bluetooth/hidp/core.c | 29 +++++++++++++++++++----------
 1 files changed, 16 insertions(+), 13 deletions(-)

diff --git a/net/bluetooth/hidp/core.c b/net/bluetooth/hidp/core.c
index df2c471..1c9b202 100644
--- a/net/bluetooth/hidp/core.c
+++ b/net/bluetooth/hidp/core.c
@@ -36,6 +36,7 @@
 #include <linux/init.h>
 #include <linux/wait.h>
 #include <linux/freezer.h>
+#include <linux/kthread.h>
 #include <net/sock.h>

 #include <linux/input.h>
@@ -531,22 +532,11 @@ static int hidp_session(void *arg)
  struct sock *ctrl_sk = session->ctrl_sock->sk;
  struct sock *intr_sk = session->intr_sock->sk;
  struct sk_buff *skb;
- int vendor = 0x0000, product = 0x0000;
  wait_queue_t ctrl_wait, intr_wait;

  BT_DBG("session %p", session);

- if (session->input) {
- vendor = session->input->id.vendor;
- product = session->input->id.product;
- }
-
- if (session->hid) {
- vendor = session->hid->vendor;
- product = session->hid->product;
- }

- daemonize("khidpd_%04x%04x", vendor, product);
  set_user_nice(current, -15);

  init_waitqueue_entry(&ctrl_wait, current);
@@ -747,7 +737,9 @@ static inline void hidp_setup_hid(struct hidp_session *session, struct
hidp_conn

```
 int hidp_add_connection(struct hidp_connadd_req *req, struct socket *ctrl_sock, struct socket
*intr_sock)
 {
+ int vendor = 0x0000, product = 0x0000;
  struct hidp_session *session, *s;
+ struct task_struct *task;
  int err;

  BT_DBG("");
@@ -834,8 +826,19 @@ int hidp_add_connection(struct hidp_connadd_req *req, struct socket
*ctrl_sock,

  hidp_set_timer(session);

- err = kernel_thread(hidp_session, session, CLONE_KERNEL);
- if (err < 0)
+ if (session->input) {
+  vendor  = session->input->id.vendor;
+  product = session->input->id.product;
+ }
+
+ if (session->hid) {
+  vendor  = session->hid->vendor;
+  product = session->hid->product;
+ }
+ task = kthread_run(hidp_session, session,
+       "khidpd_%04x%04x", vendor, product);
+ err = PTR_ERR(task);
+ if (IS_ERR(task))
   goto unlink;

  if (session->input) {
--
1.5.0.g53756
```

Subject: [PATCH] bluetooth rfcomm: Convert to kthread API.
Posted by ebiederm on Thu, 19 Apr 2007 07:58:54 GMT
View Forum Message <> Reply to Message

From: Eric W. Biederman <ebiederm@xmission.com>

This patch starts krfcommd using kthread_run instead of a combination

of kernel_thread and daemonize making the code slightly simpler
and more maintainable.

Cc: Marcel Holtmann <marcel@holtmann.org>
Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
---
 net/bluetooth/rfcomm/core.c | 4 ++--
 1 files changed, 2 insertions(+), 2 deletions(-)

diff --git a/net/bluetooth/rfcomm/core.c b/net/bluetooth/rfcomm/core.c
index 34f993a..baaad49 100644
--- a/net/bluetooth/rfcomm/core.c
+++ b/net/bluetooth/rfcomm/core.c
@@ -38,6 +38,7 @@
 #include <linux/net.h>
 #include <linux/mutex.h>
 #include <linux/freezer.h>
+#include <linux/kthread.h>

 #include <net/sock.h>
 #include <asm/uaccess.h>
@@ -1938,7 +1939,6 @@ static int rfcomm_run(void *unused)

  atomic_inc(&running);

- daemonize("krfcommd");
  set_user_nice(current, -10);

  BT_DBG("");
@@ -2058,7 +2058,7 @@ static int __init rfcomm_init(void)

  hci_register_cb(&rfcomm_cb);

- kernel_thread(rfcomm_run, NULL, CLONE_KERNEL);
+ kthread_run(rfcomm_run, NULL, "krfcommd");

  if (class_create_file(bt_class, &class_attr_rfcomm_dlc) < 0)
   BT_ERR("Failed to create RFCOMM info file");
--
1.5.0.g53756

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

Subject: [PATCH] fs/afs: Convert to kthread API.
Posted by ebiederm on Thu, 19 Apr 2007 07:58:55 GMT
View Forum Message <> Reply to Message

From: Eric W. Biederman <ebiederm@xmission.com>

This patch modifies the startup of kafscmd, kafsasyncd, and kafstimod
to use kthread_run instead of a combination of kernel_thread and
daemonize making the code slightly simpler and more maintainable.

In addition since by default all signals are ignored when delivered
to a kernel thread the code to flush signals has been removed.

Cc: David Howells <dhowells@redhat.com>
Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
---
 fs/afs/cmservice.c | 10 +++++-----
 fs/afs/internal.h | 11 -----------
 fs/afs/kafsasyncd.c | 17 ++++++-----------
 fs/afs/kafstimod.c | 16 ++++++----------
 4 files changed, 17 insertions(+), 37 deletions(-)

diff --git a/fs/afs/cmservice.c b/fs/afs/cmservice.c
index 3d097fd..f7e2355 100644
--- a/fs/afs/cmservice.c
+++ b/fs/afs/cmservice.c
@@ -13,6 +13,7 @@
 #include <linux/init.h>
 #include <linux/sched.h>
 #include <linux/completion.h>
+#include <linux/kthread.h>
 #include "server.h"
 #include "cell.h"
 #include "transport.h"
@@ -120,8 +121,6 @@ static int kafscmd(void *arg)

  printk(KERN_INFO "kAFS: Started kafscmd %d\n", current->pid);

- daemonize("kafscmd");
-
  complete(&kafscmd_alive);

  /* loop around looking for things to attend to */
@@ -133,7 +132,6 @@ static int kafscmd(void *arg)
    for (;;) {
     set_current_state(TASK_INTERRUPTIBLE);
     if (!list_empty(&kafscmd_attention_list) ||
-        signal_pending(current) ||
         kafscmd_die)

```
      break;

@@ -297,8 +295,10 @@ int afscm_start(void)

  down_write(&afscm_sem);
  if (!afscm_usage) {
-  ret = kernel_thread(kafscmd, NULL, 0);
-  if (ret < 0)
+  struct task_struct *task;
+  task = kthread_run(kafscmd, NULL, "kafscmd");
+  ret = PTR_ERR(task);
+  if (IS_ERR(task))
     goto out;

   wait_for_completion(&kafscmd_alive);
diff --git a/fs/afs/internal.h b/fs/afs/internal.h
index 5151d5d..2d667b7 100644
--- a/fs/afs/internal.h
+++ b/fs/afs/internal.h
@@ -40,17 +40,6 @@
 #define _net(FMT, a...)  do { } while(0)
 #endif

-static inline void afs_discard_my_signals(void)
-{
- while (signal_pending(current)) {
-  siginfo_t sinfo;
-
-  spin_lock_irq(&current->sighand->siglock);
-  dequeue_signal(current,&current->blocked, &sinfo);
-  spin_unlock_irq(&current->sighand->siglock);
- }
-}
-
 /*
  * cell.c
  */
diff --git a/fs/afs/kafsasyncd.c b/fs/afs/kafsasyncd.c
index 615df24..ead025f 100644
--- a/fs/afs/kafsasyncd.c
+++ b/fs/afs/kafsasyncd.c
@@ -21,6 +21,7 @@
 #include <linux/sched.h>
 #include <linux/completion.h>
 #include <linux/freezer.h>
+#include <linux/kthread.h>
 #include "cell.h"
 #include "server.h"
```

```
 #include "volume.h"
@@ -56,15 +57,15 @@ static void kafsasyncd_null_call_error_func(struct rxrpc_call *call)
  */
 int afs_kafsasyncd_start(void)
 {
- int ret;
+ struct task_struct *task;

- ret = kernel_thread(kafsasyncd, NULL, 0);
- if (ret < 0)
-  return ret;
+ task = kthread_run(kafsasyncd, NULL, "kafsasyncd");
+ if (IS_ERR(task))
+  return PTR_ERR(task);

  wait_for_completion(&kafsasyncd_alive);

- return ret;
+ return 0;
 } /* end afs_kafsasyncd_start() */

 /*****************************************************************************/
@@ -95,8 +96,6 @@ static int kafsasyncd(void *arg)

  printk("kAFS: Started kafsasyncd %d\n", current->pid);

- daemonize("kafsasyncd");
-
  complete(&kafsasyncd_alive);

  /* loop around looking for things to attend to */
@@ -106,7 +105,6 @@ static int kafsasyncd(void *arg)
   for (;;) {
    if (!list_empty(&kafsasyncd_async_attnq) ||
-       signal_pending(current) ||
        kafsasyncd_die)
     break;

@@ -119,9 +117,6 @@ static int kafsasyncd(void *arg)

   try_to_freeze();

-  /* discard pending signals */
-  afs_discard_my_signals();
-
   die = kafsasyncd_die;
```

```
   /* deal with the next asynchronous operation requiring
diff --git a/fs/afs/kafstimod.c b/fs/afs/kafstimod.c
index 694344e..caeac88 100644
--- a/fs/afs/kafstimod.c
+++ b/fs/afs/kafstimod.c
@@ -14,6 +14,7 @@
 #include <linux/sched.h>
 #include <linux/completion.h>
 #include <linux/freezer.h>
+#include <linux/kthread.h>
 #include "cell.h"
 #include "volume.h"
 #include "kafstimod.h"
@@ -36,15 +37,15 @@ static int kafstimod(void *arg);
  */
 int afs_kafstimod_start(void)
 {
- int ret;
+ struct task_struct *task;

- ret = kernel_thread(kafstimod, NULL, 0);
- if (ret < 0)
-  return ret;
+ task = kthread_run(kafstimod, NULL, "kafstimod");
+ if (IS_ERR(task))
+  return PTR_ERR(task);

  wait_for_completion(&kafstimod_alive);

- return ret;
+ return 0;
 } /* end afs_kafstimod_start() */

 /*****************************************************************************/
@@ -72,8 +73,6 @@ static int kafstimod(void *arg)

  printk("kAFS: Started kafstimod %d\n", current->pid);

- daemonize("kafstimod");
-
  complete(&kafstimod_alive);

  /* loop around looking for things to attend to */
@@ -94,9 +93,6 @@ static int kafstimod(void *arg)

   try_to_freeze();

-  /* discard pending signals */
```

```
-  afs_discard_my_signals();
-
   /* work out the time to elapse before the next event */
   spin_lock(&kafstimod_lock);
   if (list_empty(&kafstimod_list)) {
--
1.5.0.g53756
```

Subject: [PATCH] net/rxrpc: Convert to kthread API.
Posted by ebiederm on Thu, 19 Apr 2007 07:58:56 GMT
View Forum Message <> Reply to Message

From: Eric W. Biederman <ebiederm@xmission.com>

This patch modifies the startup of krxtimod, krxiod, and krxsecd
to use kthread_run instead of a combination of kernel_thread
and daemonize making the code slightly simpler and more maintainable.

In addition since by default all signals are ignored when delivered
to a kernel thread the code to flush signals has been removed.

Cc: David Howells <dhowells@redhat.com>
Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
---
 net/rxrpc/internal.h | 11 -----------
 net/rxrpc/krxiod.c | 16 ++++++++--------
 net/rxrpc/krxsecd.c | 16 ++++++++--------
 net/rxrpc/krxtimod.c | 15 ++++++---------
 4 files changed, 22 insertions(+), 36 deletions(-)

diff --git a/net/rxrpc/internal.h b/net/rxrpc/internal.h
index cc0c579..1dd69aa 100644
--- a/net/rxrpc/internal.h
+++ b/net/rxrpc/internal.h
@@ -49,17 +49,6 @@ __RXACCT_DECL(extern atomic_t rxrpc_message_count);
 #define _net(FMT, a...) do { if (rxrpc_knet)   knet (FMT , ##a); } while(0)
 #endif

-static inline void rxrpc_discard_my_signals(void)
-{
- while (signal_pending(current)) {
- siginfo_t sinfo;

```
-
-  spin_lock_irq(&current->sighand->siglock);
-  dequeue_signal(current, &current->blocked, &sinfo);
-  spin_unlock_irq(&current->sighand->siglock);
- }
-}
-
 /*
  * call.c
  */
diff --git a/net/rxrpc/krxiod.c b/net/rxrpc/krxiod.c
index bbbcd6c..c590ccd 100644
--- a/net/rxrpc/krxiod.c
+++ b/net/rxrpc/krxiod.c
@@ -14,6 +14,7 @@
 #include <linux/spinlock.h>
 #include <linux/init.h>
 #include <linux/freezer.h>
+#include <linux/kthread.h>
 #include <rxrpc/krxiod.h>
 #include <rxrpc/transport.h>
 #include <rxrpc/peer.h>
@@ -43,8 +44,6 @@ static int rxrpc_krxiod(void *arg)

  printk("Started krxiod %d\n",current->pid);

- daemonize("krxiod");
-
  /* loop around waiting for work to do */
  do {
   /* wait for work or to be told to exit */
@@ -57,8 +56,7 @@ static int rxrpc_krxiod(void *arg)
    for (;;) {
     set_current_state(TASK_INTERRUPTIBLE);
     if (atomic_read(&rxrpc_krxiod_qcount) ||
-        rxrpc_krxiod_die ||
-        signal_pending(current))
+        rxrpc_krxiod_die)
      break;

     schedule();
@@ -141,9 +139,6 @@ static int rxrpc_krxiod(void *arg)

   try_to_freeze();

-  /* discard pending signals */
-  rxrpc_discard_my_signals();
-
```

```
  } while (!rxrpc_krxiod_die);

  /* and that's all */
@@ -157,7 +152,12 @@ static int rxrpc_krxiod(void *arg)
  */
 int __init rxrpc_krxiod_init(void)
 {
- return kernel_thread(rxrpc_krxiod, NULL, 0);
+ struct task_struct *task;
+ int ret = 0;
+ task = kthread_run(rxrpc_krxiod, NULL, "krxiod");
+ if (IS_ERR(task))
+  ret = PTR_ERR(task);
+ return ret;

 } /* end rxrpc_krxiod_init() */

diff --git a/net/rxrpc/krxsecd.c b/net/rxrpc/krxsecd.c
index 9a1e7f5..150cd39 100644
--- a/net/rxrpc/krxsecd.c
+++ b/net/rxrpc/krxsecd.c
@@ -19,6 +19,7 @@
 #include <linux/completion.h>
 #include <linux/spinlock.h>
 #include <linux/init.h>
+#include <linux/kthread.h>
 #include <rxrpc/krxsecd.h>
 #include <rxrpc/transport.h>
 #include <rxrpc/connection.h>
@@ -56,8 +57,6 @@ static int rxrpc_krxsecd(void *arg)

  printk("Started krxsecd %d\n", current->pid);

- daemonize("krxsecd");
-
  /* loop around waiting for work to do */
  do {
   /* wait for work or to be told to exit */
@@ -70,8 +69,7 @@ static int rxrpc_krxsecd(void *arg)
    for (;;) {
     set_current_state(TASK_INTERRUPTIBLE);
     if (atomic_read(&rxrpc_krxsecd_qcount) ||
-        rxrpc_krxsecd_die ||
-        signal_pending(current))
+        rxrpc_krxsecd_die)
      break;

     schedule();
```

```
@@ -110,9 +108,6 @@ static int rxrpc_krxsecd(void *arg)

   try_to_freeze();

-  /* discard pending signals */
-  rxrpc_discard_my_signals();
-
  } while (!die);

  /* and that's all */
@@ -126,7 +121,12 @@ static int rxrpc_krxsecd(void *arg)
  */
 int __init rxrpc_krxsecd_init(void)
 {
- return kernel_thread(rxrpc_krxsecd, NULL, 0);
+ struct task_struct *task;
+ int ret = 0;
+ task = kthread_run(rxrpc_krxsecd, NULL, "krxsecd");
+ if (IS_ERR(task))
+  ret = PTR_ERR(task);
+ return ret;

 } /* end rxrpc_krxsecd_init() */

diff --git a/net/rxrpc/krxtimod.c b/net/rxrpc/krxtimod.c
index 9a9b613..3b5f062 100644
--- a/net/rxrpc/krxtimod.c
+++ b/net/rxrpc/krxtimod.c
@@ -14,6 +14,7 @@
 #include <linux/sched.h>
 #include <linux/completion.h>
 #include <linux/freezer.h>
+#include <linux/kthread.h>
 #include <rxrpc/rxrpc.h>
 #include <rxrpc/krxtimod.h>
 #include <asm/errno.h>
@@ -35,11 +36,12 @@ static int krxtimod(void *arg);
  */
 int rxrpc_krxtimod_start(void)
 {
- int ret;
+ struct task_struct *task;
+ int ret = 0;

- ret = kernel_thread(krxtimod, NULL, 0);
- if (ret < 0)
-  return ret;
+ task = kthread_run(krxtimod, NULL, "krxtimod");
```

```
+ if (IS_ERR(task))
+  ret = PTR_ERR(task);

  wait_for_completion(&krxtimod_alive);

@@ -71,8 +73,6 @@ static int krxtimod(void *arg)

  printk("Started krxtimod %d\n", current->pid);

- daemonize("krxtimod");
-
  complete(&krxtimod_alive);

  /* loop around looking for things to attend to */
@@ -93,9 +93,6 @@ static int krxtimod(void *arg)

   try_to_freeze();

-  /* discard pending signals */
-  rxrpc_discard_my_signals();
-
   /* work out the time to elapse before the next event */
   spin_lock(&krxtimod_lock);
   if (list_empty(&krxtimod_list)) {
--
1.5.0.g53756
```

Subject: [PATCH] ipv4/ipvs: Convert to kthread API
Posted by ebiederm on Thu, 19 Apr 2007 07:58:57 GMT
View Forum Message <> Reply to Message

From: Eric W. Biederman <ebiederm@xmission.com>

Modify startup of ipvs sync threads to use kthread_run
instead of a weird combination of calling kernel_thread
to start a fork_sync_thread whose hole purpose in life was
to call kernel_thread again starting the actually sync thread
which called daemonize.

To use kthread_run I had to move the name calcuation from
sync_thread into start_sync_thread resulting in a small
amount of code motion.

The result is simpler and more maintainable piece of code.

Cc: Wensong Zhang <wensong@linux-vs.org>
Cc: Julian Anastasov <ja@ssi.bg>
Cc: Simon Horman <horms@verge.net.au>
Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
---
 net/ipv4/ipvs/ip_vs_sync.c | 49 ++++++++++++----------------------------
 1 files changed, 12 insertions(+), 37 deletions(-)

```
diff --git a/net/ipv4/ipvs/ip_vs_sync.c b/net/ipv4/ipvs/ip_vs_sync.c
index 7ea2d98..c4be9dc 100644
--- a/net/ipv4/ipvs/ip_vs_sync.c
+++ b/net/ipv4/ipvs/ip_vs_sync.c
@@ -29,6 +29,7 @@
 #include <linux/in.h>
 #include <linux/igmp.h>                 /* for ip_mc_join_group */
 #include <linux/udp.h>
+#include <linux/kthread.h>

 #include <net/ip.h>
 #include <net/sock.h>
@@ -750,34 +751,23 @@ static int sync_thread(void *startup)
  DECLARE_WAITQUEUE(wait, current);
  mm_segment_t oldmm;
  int state;
- const char *name;

  /* increase the module use count */
  ip_vs_use_count_inc();

- if (ip_vs_sync_state & IP_VS_STATE_MASTER && !sync_master_pid) {
+ if (ip_vs_sync_state & IP_VS_STATE_MASTER && !sync_master_pid)
   state = IP_VS_STATE_MASTER;
-  name = "ipvs_syncmaster";
- } else if (ip_vs_sync_state & IP_VS_STATE_BACKUP && !sync_backup_pid) {
+ else if (ip_vs_sync_state & IP_VS_STATE_BACKUP && !sync_backup_pid)
   state = IP_VS_STATE_BACKUP;
-  name = "ipvs_syncbackup";
- } else {
+ else {
   IP_VS_BUG();
   ip_vs_use_count_dec();
   return -EINVAL;
  }

- daemonize(name);
```

```
-
  oldmm = get_fs();
  set_fs(KERNEL_DS);

- /* Block all signals */
- spin_lock_irq(&current->sighand->siglock);
- siginitsetinv(&current->blocked, 0);
- recalc_sigpending();
- spin_unlock_irq(&current->sighand->siglock);
-
  /* set the maximum length of sync message */
  set_sync_mesg_maxlen(state);

@@ -815,29 +805,11 @@ static int sync_thread(void *startup)
  return 0;
 }

-
-static int fork_sync_thread(void *startup)
-{
- pid_t pid;
-
- /* fork the sync thread here, then the parent process of the
-    sync thread is the init process after this thread exits. */
-  repeat:
- if ((pid = kernel_thread(sync_thread, startup, 0)) < 0) {
-  IP_VS_ERR("could not create sync_thread due to %d... "
-     "retrying.\n", pid);
-  msleep_interruptible(1000);
-  goto repeat;
- }
-
- return 0;
-}
-
-
 int start_sync_thread(int state, char *mcast_ifn, __u8 syncid)
 {
  DECLARE_COMPLETION_ONSTACK(startup);
- pid_t pid;
+ struct task_struct *task;
+ const char *name;

  if ((state == IP_VS_STATE_MASTER && sync_master_pid) ||
      (state == IP_VS_STATE_BACKUP && sync_backup_pid))
@@ -852,16 +824,19 @@ int start_sync_thread(int state, char *mcast_ifn, __u8 syncid)
   strlcpy(ip_vs_master_mcast_ifn, mcast_ifn,
    sizeof(ip_vs_master_mcast_ifn));
```

```
   ip_vs_master_syncid = syncid;
+  name = "ipvs_syncmaster";
  } else {
   strlcpy(ip_vs_backup_mcast_ifn, mcast_ifn,
    sizeof(ip_vs_backup_mcast_ifn));
   ip_vs_backup_syncid = syncid;
+  name = "ipvs_syncbackup";
  }

   repeat:
- if ((pid = kernel_thread(fork_sync_thread, &startup, 0)) < 0) {
-  IP_VS_ERR("could not create fork_sync_thread due to %d... "
-     "retrying.\n", pid);
+ task = kthread_run(sync_thread, &startup, name);
+ if (IS_ERR(task)) {
+  IP_VS_ERR("could not create sync_thread due to %ld... "
+     "retrying.\n", PTR_ERR(task));
   msleep_interruptible(1000);
   goto repeat;
  }
--
1.5.0.g53756
```

Subject: [PATCH] saa7134-tvaudio: Convert to kthread API.
Posted by ebiederm on Thu, 19 Apr 2007 07:58:58 GMT
View Forum Message <> Reply to Message

From: Eric W. Biederman <ebiederm@xmission.com>

It is my goal to replace all kernel code that handles signals
from user space, calls kernel_thread or calls daemonize. All
of which the kthread_api makes unncessary. Handling signals
from user space is a maintenance problem becuase using a
kernel thread is an implementation detail and if user space
cares it does not allow us to change the implementation. Calling
daemonize is a problem because it has to undo a continually changing
set of state generated by user space, requiring the implemetation
to change continually. kernel_thread is a problem because it
returns a pid_t value. Numeric pids are inherently racy and
in the presence of a pid namespace they are no longer global
making them useless for general use in the kernel.

So this patch renames the pid member of struct saa7134_thread
started and changes it's type from pid_t to int. All it
has ever been used for is to detect if the kernel thread
is has been started so this works.

allow_signal(SIGTERM) and the calls to signal_pending have
been removed they are needed for the driver to operation.

The startup of tvaudio_thread and tvaudio_thread_dep have
been modified to use kthread_run instead of a combination
of kernel_thread and daemonize.

The result is code that is slightly simpler and more
maintainable.

Cc: Hartmut Hackmann <hartmut.hackmann@t-online.de>
Cc: Mauro Carvalho Chehab <mchehab@infradead.org>
Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
---
 drivers/media/video/saa7134/saa7134-tvaudio.c | 27 ++++++++++++++----------
 drivers/media/video/saa7134/saa7134.h | 2 +-
 2 files changed, 14 insertions(+), 15 deletions(-)

diff --git a/drivers/media/video/saa7134/saa7134-tvaudio.c
b/drivers/media/video/saa7134/saa7134-tvaudio.c
index 7b56041..b636cb1 100644
--- a/drivers/media/video/saa7134/saa7134-tvaudio.c
+++ b/drivers/media/video/saa7134/saa7134-tvaudio.c
@@ -27,6 +27,7 @@
 #include <linux/kernel.h>
 #include <linux/slab.h>
 #include <linux/delay.h>
+#include <linux/kthread.h>
 #include <asm/div64.h>

 #include "saa7134-reg.h"
@@ -505,11 +506,9 @@ static int tvaudio_thread(void *data)
   unsigned int i, audio, nscan;
   int max1,max2,carrier,rx,mode,lastmode,default_carrier;

- daemonize("%s", dev->name);
- allow_signal(SIGTERM);
   for (;;) {
    tvaudio_sleep(dev,-1);
- if (dev->thread.shutdown || signal_pending(current))
+ if (dev->thread.shutdown)
     goto done;

```
  restart:
@@ -618,7 +617,7 @@ static int tvaudio_thread(void *data)
    for (;;) {
     if (tvaudio_sleep(dev,5000))
      goto restart;
-    if (dev->thread.shutdown || signal_pending(current))
+    if (dev->thread.shutdown)
      break;
     if (UNSET == dev->thread.mode) {
      rx = tvaudio_getstereo(dev,&tvaudio[i]);
@@ -782,9 +781,6 @@ static int tvaudio_thread_ddep(void *data)
  struct saa7134_dev *dev = data;
  u32 value, norms, clock;

- daemonize("%s", dev->name);
- allow_signal(SIGTERM);
-
  clock = saa7134_boards[dev->board].audio_clock;
  if (UNSET != audio_clock_override)
   clock = audio_clock_override;
@@ -796,7 +792,7 @@ static int tvaudio_thread_ddep(void *data)

  for (;;) {
   tvaudio_sleep(dev,-1);
-  if (dev->thread.shutdown || signal_pending(current))
+  if (dev->thread.shutdown)
    goto done;

  restart:
@@ -986,14 +982,17 @@ int saa7134_tvaudio_init2(struct saa7134_dev *dev)
   break;
  }

- dev->thread.pid = -1;
+ dev->thread.started = 0;
  if (my_thread) {
+  struct task_struct *task;
   /* start tvaudio thread */
   init_waitqueue_head(&dev->thread.wq);
   init_completion(&dev->thread.exit);
-  dev->thread.pid = kernel_thread(my_thread,dev,0);
-  if (dev->thread.pid < 0)
-   printk(KERN_WARNING "%s: kernel_thread() failed\n",
+  task = kthread_run(my_thread, dev, "%s", dev->name);
+  if (!IS_ERR(task))
+   dev->thread.started = 1;
+  else
+   printk(KERN_WARNING "%s: kthread_create() failed\n",
```

```
          dev->name);
   saa7134_tvaudio_do_scan(dev);
  }
@@ -1005,7 +1004,7 @@ int saa7134_tvaudio_init2(struct saa7134_dev *dev)
 int saa7134_tvaudio_fini(struct saa7134_dev *dev)
 {
  /* shutdown tvaudio thread */
- if (dev->thread.pid >= 0) {
+ if (dev->thread.started) {
   dev->thread.shutdown = 1;
   wake_up_interruptible(&dev->thread.wq);
   wait_for_completion(&dev->thread.exit);
@@ -1020,7 +1019,7 @@ int saa7134_tvaudio_do_scan(struct saa7134_dev *dev)
   dprintk("sound IF not in use, skipping scan\n");
   dev->automute = 0;
   saa7134_tvaudio_setmute(dev);
- } else if (dev->thread.pid >= 0) {
+ } else if (dev->thread.started) {
   dev->thread.mode = UNSET;
   dev->thread.scan2++;
   wake_up_interruptible(&dev->thread.wq);
diff --git a/drivers/media/video/saa7134/saa7134.h b/drivers/media/video/saa7134/saa7134.h
index 62224cc..3a10ce7 100644
--- a/drivers/media/video/saa7134/saa7134.h
+++ b/drivers/media/video/saa7134/saa7134.h
@@ -324,7 +324,7 @@ struct saa7134_pgtable {

 /* tvaudio thread status */
 struct saa7134_thread {
- pid_t                   pid;
+ int      started;
  struct completion       exit;
  wait_queue_head_t       wq;
  unsigned int            shutdown;
--
1.5.0.g53756
```


Subject: [PATCH] nfs lockd reclaimer: Convert to kthread API
Posted by ebiederm on Thu, 19 Apr 2007 07:58:59 GMT
View Forum Message <> Reply to Message

From: Eric W. Biederman <ebiederm@xmission.com>

Start the reclaimer thread using kthread_run instead
of a combination of kernel_thread and daemonize.
The small amount of signal handling code is also removed
as it makes no sense and is a maintenance problem to handle
signals in kernel threads.

Cc: Neil Brown <neilb@suse.de>
Cc: Trond Myklebust <trond.myklebust@fys.uio.no>
Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
---
 fs/lockd/clntlock.c | 8 ++------
 1 files changed, 2 insertions(+), 6 deletions(-)

diff --git a/fs/lockd/clntlock.c b/fs/lockd/clntlock.c
index f4d45d4..83591f6 100644
--- a/fs/lockd/clntlock.c
+++ b/fs/lockd/clntlock.c
@@ -9,6 +9,7 @@
 #include <linux/module.h>
 #include <linux/types.h>
 #include <linux/time.h>
+#include <linux/kthread.h>
 #include <linux/nfs_fs.h>
 #include <linux/sunrpc/clnt.h>
 #include <linux/sunrpc/svc.h>
@@ -153,7 +154,7 @@ nlmclnt_recovery(struct nlm_host *host)
  if (!host->h_reclaiming++) {
   nlm_get_host(host);
   __module_get(THIS_MODULE);
- if (kernel_thread(reclaimer, host, CLONE_KERNEL) < 0)
+ if (IS_ERR(kthread_run(reclaimer, host, "%s-reclaim", host->h_name)))
    module_put(THIS_MODULE);
  }
 }
@@ -166,9 +167,6 @@ reclaimer(void *ptr)
  struct file_lock *fl, *next;
  u32 nsmstate;

- daemonize("%s-reclaim", host->h_name);
- allow_signal(SIGKILL);
-
  down_write(&host->h_rwsem);

  /* This one ensures that our parent doesn't terminate while the
@@ -193,8 +191,6 @@ restart:
   list_del_init(&fl->fl_u.nfs_fl.list);

```
   /* Why are we leaking memory here? --okir */
-  if (signalled())
-   continue;
   if (nlmclnt_reclaim(host, fl) != 0)
    continue;
   list_add_tail(&fl->fl_u.nfs_fl.list, &host->h_granted);
--
1.5.0.g53756
```

Subject: [PATCH] nfsv4 delegation: Convert to kthread API
Posted by ebiederm on Thu, 19 Apr 2007 07:59:00 GMT
View Forum Message <> Reply to Message

From: Eric W. Biederman <ebiederm@xmission.com>

To start the nfsv4-delegreturn thread this patch uses
kthread_run instead of a combination of kernel_thread
and daemonize.

In addition allow_signal(SIGKILL) is removed from
the expire delegations thread.

Cc: Neil Brown <neilb@suse.de>
Cc: Trond Myklebust <trond.myklebust@fys.uio.no>
Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
---
 fs/nfs/delegation.c | 11 ++++++-------
 1 files changed, 4 insertions(+), 7 deletions(-)

```
diff --git a/fs/nfs/delegation.c b/fs/nfs/delegation.c
index 841c99a..7b9b88c 100644
--- a/fs/nfs/delegation.c
+++ b/fs/nfs/delegation.c
@@ -232,7 +232,6 @@ int nfs_do_expire_all_delegations(void *ptr)
  struct nfs_delegation *delegation;
  struct inode *inode;

- allow_signal(SIGKILL);
 restart:
  spin_lock(&clp->cl_lock);
  if (test_bit(NFS4CLNT_STATE_RECOVER, &clp->cl_state) != 0)
@@ -310,8 +309,6 @@ static int recall_thread(void *data)
```

```
  struct nfs_inode *nfsi = NFS_I(inode);
  struct nfs_delegation *delegation;

- daemonize("nfsv4-delegreturn");
-
  nfs_msync_inode(inode);
  down_read(&clp->cl_sem);
  down_write(&nfsi->rwsem);
@@ -350,18 +347,18 @@ int nfs_async_inode_return_delegation(struct inode *inode, const
nfs4_stateid *s
   .inode = inode,
   .stateid = stateid,
  };
- int status;
+ struct task_struct *task;

  init_completion(&data.started);
  __module_get(THIS_MODULE);
- status = kernel_thread(recall_thread, &data, CLONE_KERNEL);
- if (status < 0)
+ task = kthread_run(recall_thread, &data, "nfsv4-delegreturn");
+ if (IS_ERR(task))
   goto out_module_put;
  wait_for_completion(&data.started);
  return data.result;
 out_module_put:
  module_put(THIS_MODULE);
- return status;
+ return PTR_ERR(task);
 }

 /*
--
1.5.0.g53756
```

Subject: [PATCH] nfsd/nfs4state: Remove unnecessary daemonize call.
Posted by ebiederm on Thu, 19 Apr 2007 07:59:01 GMT
View Forum Message <> Reply to Message

From: Eric W. Biederman <ebiederm@xmission.com>

Cc: Neil Brown <neilb@suse.de>

Cc: Trond Myklebust <trond.myklebust@fys.uio.no>
Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
---
 fs/nfsd/nfs4state.c | 2 --
 1 files changed, 0 insertions(+), 2 deletions(-)

diff --git a/fs/nfsd/nfs4state.c b/fs/nfsd/nfs4state.c
index 678f3be..3cc8ce4 100644
--- a/fs/nfsd/nfs4state.c
+++ b/fs/nfsd/nfs4state.c
@@ -1326,8 +1326,6 @@ do_recall(void *__dp)
 {
  struct nfs4_delegation *dp = __dp;

- daemonize("nfsv4-recall");
-
  nfsd4_cb_recall(dp);
  return 0;
 }
--
1.5.0.g53756

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: [PATCH] nfs4state reclaimer: Remove unnecessary allow_signal
Posted by ebiederm on Thu, 19 Apr 2007 07:59:02 GMT
View Forum Message <> Reply to Message

From: Eric W. Biederman <ebiederm@xmission.com>

Cc: Neil Brown <neilb@suse.de>
Cc: Trond Myklebust <trond.myklebust@fys.uio.no>
Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
---
 fs/nfs/nfs4state.c | 2 --
 1 files changed, 0 insertions(+), 2 deletions(-)

diff --git a/fs/nfs/nfs4state.c b/fs/nfs/nfs4state.c
index 5fffbdf..d16393f 100644
--- a/fs/nfs/nfs4state.c
+++ b/fs/nfs/nfs4state.c
@@ -775,8 +775,6 @@ static int reclaimer(void *ptr)
  struct rpc_cred *cred;
  int status = 0;

```
- allow_signal(SIGKILL);
-
  /* Ensure exclusive access to NFSv4 state */
  lock_kernel();
  down_write(&clp->cl_sem);
--
1.5.0.g53756
```


Subject: [PATCH] smbfs: Remove unnecessary allow_signal
Posted by ebiederm on Thu, 19 Apr 2007 07:59:03 GMT
View Forum Message <> Reply to Message

From: Eric W. Biederman <ebiederm@xmission.com>

Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>

```
---
 fs/smbfs/smbiod.c |    2 --
 1 files changed, 0 insertions(+), 2 deletions(-)

diff --git a/fs/smbfs/smbiod.c b/fs/smbfs/smbiod.c
index 3e61b44..67176af 100644
--- a/fs/smbfs/smbiod.c
+++ b/fs/smbfs/smbiod.c
@@ -298,8 +298,6 @@ out:
  */
 static int smbiod(void *unused)
 {
- allow_signal(SIGKILL);
-
  VERBOSE("SMB Kernel thread starting (%d) ...\n", current->pid);

  for (;;) {
--
1.5.0.g53756
```

Subject: [PATCH] dvb_en_50221: Convert to kthread API
Posted by ebiederm on Thu, 19 Apr 2007 07:59:04 GMT
View Forum Message <> Reply to Message

From: Eric W. Biederman <ebiederm@xmission.com>

This patch is a minimal transformation to use the kthread API
doing it's best to preserve the existing logic.

Instead of starting kdvb-ca by calling kernel_thread,
daemonize and sigfillset we kthread_run is used.

Instead of tracking the pid of the running thread we instead
simply keep a flag to indicate that the current thread is
running, as that is all the pid is really used for.

And finally the kill_proc sending signal 0 to the kernel thread to
ensure it is alive before we wait for it to shutdown is removed.
The kthread API does not provide the pid so we don't have that
information readily available and the test is just silly. If there
is no shutdown race the test is a useless confirmation of that the
thread is running. If there is a race the test doesn't fix it and
we should fix the race properly.

Cc: Andrew de Quincey <adq_dvb@lidskialf.net>
Cc: Mauro Carvalho Chehab <mchehab@infradead.org>
Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
---
 drivers/media/dvb/dvb-core/dvb_ca_en50221.c | 46 ++++++++++++----------------
 1 files changed, 18 insertions(+), 28 deletions(-)

diff --git a/drivers/media/dvb/dvb-core/dvb_ca_en50221.c
b/drivers/media/dvb/dvb-core/dvb_ca_en50221.c
index 2a03bf5..b28bc15 100644
--- a/drivers/media/dvb/dvb-core/dvb_ca_en50221.c
+++ b/drivers/media/dvb/dvb-core/dvb_ca_en50221.c
@@ -37,6 +37,7 @@
 #include <linux/delay.h>
 #include <linux/spinlock.h>
 #include <linux/sched.h>
+#include <linux/kthread.h>

 #include "dvb_ca_en50221.h"
 #include "dvb_ringbuffer.h"
@@ -139,8 +140,8 @@ struct dvb_ca_private {
  /* wait queues for read() and write() operations */
  wait_queue_head_t wait_queue;

- /* PID of the monitoring thread */

```
- pid_t thread_pid;
+ /* Flag indicating the monitoring thread is running */
+ int thread_running;

  /* Wait queue used when shutting thread down */
  wait_queue_head_t thread_queue;
@@ -982,7 +983,6 @@ static void dvb_ca_en50221_thread_update_delay(struct dvb_ca_private
*ca)
 static int dvb_ca_en50221_thread(void *data)
 {
  struct dvb_ca_private *ca = data;
- char name[15];
  int slot;
  int flags;
  int status;
@@ -991,14 +991,6 @@ static int dvb_ca_en50221_thread(void *data)

  dprintk("%s\n", __FUNCTION__);

- /* setup kernel thread */
- snprintf(name, sizeof(name), "kdvb-ca-%i:%i", ca->dvbdev->adapter->num, ca->dvbdev->id);
-
- lock_kernel();
- daemonize(name);
- sigfillset(&current->blocked);
- unlock_kernel();
-
  /* choose the correct initial delay */
  dvb_ca_en50221_thread_update_delay(ca);

@@ -1182,7 +1174,7 @@ static int dvb_ca_en50221_thread(void *data)
  }

  /* completed */
- ca->thread_pid = 0;
+ ca->thread_running = 0;
  mb();
  wake_up_interruptible(&ca->thread_queue);
  return 0;
@@ -1660,6 +1652,7 @@ static struct dvb_device dvbdev_ca = {
 int dvb_ca_en50221_init(struct dvb_adapter *dvb_adapter,
    struct dvb_ca_en50221 *pubca, int flags, int slot_count)
 {
+ struct task_struct *task;
  int ret;
  struct dvb_ca_private *ca = NULL;
  int i;
@@ -1682,7 +1675,7 @@ int dvb_ca_en50221_init(struct dvb_adapter *dvb_adapter,
```

```
   goto error;
  }
  init_waitqueue_head(&ca->wait_queue);
- ca->thread_pid = 0;
+ ca->thread_running = 0;
  init_waitqueue_head(&ca->thread_queue);
  ca->exit = 0;
  ca->open = 0;
@@ -1711,13 +1704,15 @@ int dvb_ca_en50221_init(struct dvb_adapter *dvb_adapter,

  /* create a kthread for monitoring this CA device */

- ret = kernel_thread(dvb_ca_en50221_thread, ca, 0);
-
- if (ret < 0) {
-  printk("dvb_ca_init: failed to start kernel_thread (%d)\n", ret);
+ task = kthread_run(dvb_ca_en50221_thread, ca,
+       "kdvb-ca-%i:%i",
+       ca->dvbdev->adapter->num, ca->dvbdev->id);
+ if (IS_ERR(task)) {
+  ret = PTR_ERR(task);
+  printk("dvb_ca_init: failed to start kthread (%d)\n", ret);
   goto error;
  }
- ca->thread_pid = ret;
+ ca->thread_running = 1;
  return 0;

 error:
@@ -1748,16 +1743,11 @@ void dvb_ca_en50221_release(struct dvb_ca_en50221 *pubca)
  dprintk("%s\n", __FUNCTION__);

  /* shutdown the thread if there was one */
- if (ca->thread_pid) {
-  if (kill_proc(ca->thread_pid, 0, 1) == -ESRCH) {
-   printk("dvb_ca_release adapter %d: thread PID %d already died\n",
-          ca->dvbdev->adapter->num, ca->thread_pid);
-  } else {
-   ca->exit = 1;
-   mb();
-   dvb_ca_en50221_thread_wakeup(ca);
-   wait_event_interruptible(ca->thread_queue, ca->thread_pid == 0);
-  }
+ if (ca->thread_running) {
+  ca->exit = 1;
+  mb();
+  dvb_ca_en50221_thread_wakeup(ca);
+  wait_event_interruptible(ca->thread_queue, ca->thread_running == 0);
```

```
 }

 for (i = 0; i < ca->slot_count; i++) {
--
1.5.0.g53756
```

Subject: [PATCH] md: Remove broken SIGKILL support
Posted by ebiederm on Thu, 19 Apr 2007 07:59:05 GMT
View Forum Message <> Reply to Message

From: Eric W. Biederman <ebiederm@xmission.com>

Currently md_thread calls allow_signal so it can receive a
SIGKILL but then does nothing with it except flush the
sigkill so that it not can use an interruptible sleep.

This whole dance is silly so remove the unnecessary
and broken signal handling logic.

Cc: Neil Brown <neilb@suse.de>
Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
---
 drivers/md/md.c | 6 ------
 1 files changed, 0 insertions(+), 6 deletions(-)

```
diff --git a/drivers/md/md.c b/drivers/md/md.c
index 1299c23..dfd0cb9 100644
--- a/drivers/md/md.c
+++ b/drivers/md/md.c
@@ -4542,17 +4542,11 @@ static int md_thread(void * arg)
   */

  current->flags |= PF_NOFREEZE;
- allow_signal(SIGKILL);
  while (!kthread_should_stop()) {

   /* We need to wait INTERRUPTIBLE so that
    * we don't add to the load-average.
-   * That means we need to be sure no signals are
-   * pending
    */
-  if (signal_pending(current))
```

```
-   flush_signals(current);
-
   wait_event_interruptible_timeout
    (thread->wqueue,
     test_bit(THREAD_WAKEUP, &thread->flags)
--
1.5.0.g53756
```

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: [PATCH] synchro_test: Convert to the kthread API.
Posted by ebiederm on Thu, 19 Apr 2007 07:59:06 GMT
View Forum Message <> Reply to Message

From: Eric W. Biederman <ebiederm@xmission.com>

Cc: David Howells <dhowells@redhat.com>
Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
---
 kernel/synchro-test.c | 16 ++++++----------
 1 files changed, 6 insertions(+), 10 deletions(-)

```
diff --git a/kernel/synchro-test.c b/kernel/synchro-test.c
index a4747a6..b1d7fd6 100644
--- a/kernel/synchro-test.c
+++ b/kernel/synchro-test.c
@@ -30,6 +30,7 @@
 #include <linux/timer.h>
 #include <linux/completion.h>
 #include <linux/mutex.h>
+#include <linux/kthread.h>

 #define MAX_THREADS 64

@@ -224,7 +225,6 @@ static int mutexer(void *arg)
 {
  unsigned int N = (unsigned long) arg;

- daemonize("Mutex%u", N);
  set_user_nice(current, 19);

  while (atomic_read(&do_stuff)) {
@@ -246,7 +246,6 @@ static int semaphorer(void *arg)
 {
```

```
  unsigned int N = (unsigned long) arg;

- daemonize("Sem%u", N);
  set_user_nice(current, 19);

  while (atomic_read(&do_stuff)) {
@@ -268,7 +267,6 @@ static int reader(void *arg)
 {
  unsigned int N = (unsigned long) arg;

- daemonize("Read%u", N);
  set_user_nice(current, 19);

  while (atomic_read(&do_stuff)) {
@@ -292,7 +290,6 @@ static int writer(void *arg)
 {
  unsigned int N = (unsigned long) arg;

- daemonize("Write%u", N);
  set_user_nice(current, 19);

  while (atomic_read(&do_stuff)) {
@@ -316,7 +313,6 @@ static int downgrader(void *arg)
 {
  unsigned int N = (unsigned long) arg;

- daemonize("Down%u", N);
  set_user_nice(current, 19);

  while (atomic_read(&do_stuff)) {
@@ -433,27 +429,27 @@ static int __init do_tests(void)
  for (loop = 0; loop < MAX_THREADS; loop++) {
   if (loop < nummx) {
    init_completion(&mx_comp[loop]);
-   kernel_thread(mutexer, (void *) loop, 0);
+   kthread_run(mutexer, (void *) loop, "Mutex%u", loop);
   }

   if (loop < numsm) {
    init_completion(&sm_comp[loop]);
-   kernel_thread(semaphorer, (void *) loop, 0);
+   kthread_run(semaphorer, (void *) loop, "Sem%u", loop);
   }

   if (loop < numrd) {
    init_completion(&rd_comp[loop]);
-   kernel_thread(reader, (void *) loop, 0);
+   kthread_run(reader, (void *) loop, "Read%u", loop);
```

```
  }

  if (loop < numwr) {
   init_completion(&wr_comp[loop]);
-  kernel_thread(writer, (void *) loop, 0);
+   kthread_run(writer, (void *) loop, "Write%u", loop);
  }

  if (loop < numdg) {
   init_completion(&dg_comp[loop]);
-  kernel_thread(downgrader, (void *) loop, 0);
+   kthread_run(downgrader, (void *) loop, "Down%u", loop);
  }
 }

--
1.5.0.g53756
```

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: [PATCH] synchro_test: Convert to the kthread API.
Posted by ebiederm on Thu, 19 Apr 2007 07:59:07 GMT
View Forum Message <> Reply to Message

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: [PATCH] s390/net/lcs: Convert to the kthread API
Posted by Frank Pavlic on Thu, 19 Apr 2007 08:19:55 GMT
View Forum Message <> Reply to Message

ACK for both patches,
thank you Eric for the patches, will add them to my patchset
for Jeff.

Frank

On Thu, Apr 19, 2007 at 01:58:40AM -0600, Eric W. Biederman wrote:
> From: Eric W. Biederman <ebiederm@xmission.com>
>

> Use kthread_run to start the lcs kernel threads not a
> combination of kernel_thread and daemonize. This makes
> the code slightly simpler and more maintainable.
>
> Cc: Frank Pavlic <fpavlic@de.ibm.com>
> Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
> ---
> drivers/s390/net/lcs.c | 8 ++++-----
> 1 files changed, 3 insertions(+), 5 deletions(-)
>
> diff --git a/drivers/s390/net/lcs.c b/drivers/s390/net/lcs.c
> index 08a994f..0300d87 100644
> --- a/drivers/s390/net/lcs.c
> +++ b/drivers/s390/net/lcs.c
> @@ -36,6 +36,7 @@
> #include <linux/in.h>
> #include <linux/igmp.h>
> #include <linux/delay.h>
> +#include <linux/kthread.h>
> #include <net/arp.h>
> #include <net/ip.h>
>
> @@ -1248,7 +1249,6 @@ lcs_register_mc_addresses(void *data)
>   struct in_device *in4_dev;
>
>   card = (struct lcs_card *) data;
> - daemonize("regipm");
>
>   if (!lcs_do_run_thread(card, LCS_SET_MC_THREAD))
>   return 0;
> @@ -1728,11 +1728,10 @@ lcs_start_kernel_thread(struct work_struct *work)
>   struct lcs_card *card = container_of(work, struct lcs_card, kernel_thread_starter);
>   LCS_DBF_TEXT(5, trace, "krnthrd");
>   if (lcs_do_start_thread(card, LCS_RECOVERY_THREAD))
> - kernel_thread(lcs_recovery, (void *) card, SIGCHLD);
> + kthread_run(lcs_recovery, card, "lcs_recover");
> #ifdef CONFIG_IP_MULTICAST
>   if (lcs_do_start_thread(card, LCS_SET_MC_THREAD))
> - kernel_thread(lcs_register_mc_addresses,
> -   (void *) card, SIGCHLD);
> + kernel_run(lcs_register_mc_addresses, card, "regipm");
> #endif
> }
>
> @@ -2232,7 +2231,6 @@ lcs_recovery(void *ptr)
>   int rc;
>
>   card = (struct lcs_card *) ptr;

> - daemonize("lcs_recover");
>
>   LCS_DBF_TEXT(4, trace, "recover1");
>   if (!lcs_do_run_thread(card, LCS_RECOVERY_THREAD))
> --
> 1.5.0.g53756

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: [PATCH] ipv4/ipvs: Convert to kthread API
Posted by Simon Horman on Thu, 19 Apr 2007 09:04:36 GMT
View Forum Message <> Reply to Message

On Thu, Apr 19, 2007 at 01:58:57AM -0600, Eric W. Biederman wrote:
> From: Eric W. Biederman <ebiederm@xmission.com>
>
> Modify startup of ipvs sync threads to use kthread_run
> instead of a weird combination of calling kernel_thread
> to start a fork_sync_thread whose hole purpose in life was
> to call kernel_thread again starting the actually sync thread
> which called daemonize.
>
> To use kthread_run I had to move the name calcuation from
> sync_thread into start_sync_thread resulting in a small
> amount of code motion.
>
> The result is simpler and more maintainable piece of code.

Thanks Eric, I'll review this and get back to you shortly.

--
Horms
  H: http://www.vergenet.net/~horms/
  W: http://www.valinux.co.jp/en/

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: [PATCH] fs/afs: Convert to kthread API.
Posted by David Howells on Thu, 19 Apr 2007 09:32:07 GMT
View Forum Message <> Reply to Message

Eric W. Biederman <ebiederm@xmission.com> wrote:

> This patch modifies the startup of kafscmd, kafsasyncd, and kafstimod
> to use kthread_run instead of a combination of kernel_thread and
> daemonize making the code slightly simpler and more maintainable.

Please drop this patch for the moment as I have my own patches to convert them
to keventd-type threads, in addition to implementing a host of other changes.

David

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: [PATCH] net/rxrpc: Convert to kthread API.
Posted by David Howells on Thu, 19 Apr 2007 09:32:38 GMT
View Forum Message <> Reply to Message

Eric W. Biederman <ebiederm@xmission.com> wrote:

> This patch modifies the startup of krxtimod, krxiod, and krxsecd
> to use kthread_run instead of a combination of kernel_thread
> and daemonize making the code slightly simpler and more maintainable.

Again, please drop in favour of my RxRPC patches.

David

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: [PATCH] net/rxrpc: Convert to kthread API.
Posted by ebiederm on Thu, 19 Apr 2007 13:05:32 GMT
View Forum Message <> Reply to Message

David Howells <dhowells@redhat.com> writes:

> Eric W. Biederman <ebiederm@xmission.com> wrote:
>
>> This patch modifies the startup of krxtimod, krxiod, and krxsecd
>> to use kthread_run instead of a combination of kernel_thread
>> and daemonize making the code slightly simpler and more maintainable.
>

> Again, please drop in favour of my RxRPC patches.

What is the ETA on your patches?

Eric

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Getting the new RxRPC patches upstream
Posted by David Howells on Thu, 19 Apr 2007 14:18:23 GMT
View Forum Message <> Reply to Message

Eric W. Biederman <ebiederm@xmission.com> wrote:

> What is the ETA on your patches?

That depends on Dave Miller now, I think. I'm assuming they need to go through the network GIT tree to get to Linus. Certainly Andrew Morton seems to think so.

David

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: [PATCH] nfs lockd reclaimer: Convert to kthread API
Posted by Trond Myklebust on Thu, 19 Apr 2007 16:21:50 GMT
View Forum Message <> Reply to Message

On Thu, 2007-04-19 at 01:58 -0600, Eric W. Biederman wrote:
> From: Eric W. Biederman <ebiederm@xmission.com>
>
> Start the reclaimer thread using kthread_run instead
> of a combination of kernel_thread and daemonize.
> The small amount of signal handling code is also removed
> as it makes no sense and is a maintenance problem to handle
> signals in kernel threads.

Vetoed. Removing stuff just because it doesn't make sense to you is not acceptable.

Signal handling in reclaimer threads is there in order to allow

administrators to deal with the case where the server never comes up
again.

Trond

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: [PATCH] nfsv4 delegation: Convert to kthread API
Posted by Trond Myklebust on Thu, 19 Apr 2007 16:22:24 GMT
View Forum Message <> Reply to Message

On Thu, 2007-04-19 at 01:59 -0600, Eric W. Biederman wrote:
> From: Eric W. Biederman <ebiederm@xmission.com>
>
> To start the nfsv4-delegreturn thread this patch uses
> kthread_run instead of a combination of kernel_thread
> and daemonize.
>
> In addition allow_signal(SIGKILL) is removed from
> the expire delegations thread.

Again vetoed, for the same reason.

Trond

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: [PATCH] nfs4state reclaimer: Remove unnecessary allow_signal
Posted by Trond Myklebust on Thu, 19 Apr 2007 16:26:42 GMT
View Forum Message <> Reply to Message

On Thu, 2007-04-19 at 01:59 -0600, Eric W. Biederman wrote:
> From: Eric W. Biederman <ebiederm@xmission.com>
>
> Cc: Neil Brown <neilb@suse.de>
> Cc: Trond Myklebust <trond.myklebust@fys.uio.no>
> Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
> ---
> fs/nfs/nfs4state.c | 2 --

> 1 files changed, 0 insertions(+), 2 deletions(-)
>
> diff --git a/fs/nfs/nfs4state.c b/fs/nfs/nfs4state.c
> index 5fffbdf..d16393f 100644
> --- a/fs/nfs/nfs4state.c
> +++ b/fs/nfs/nfs4state.c
> @@ -775,8 +775,6 @@ static int reclaimer(void *ptr)
>   struct rpc_cred *cred;
>   int status = 0;
>
> - allow_signal(SIGKILL);
> -
>   /* Ensure exclusive access to NFSv4 state */
>   lock_kernel();
>   down_write(&clp->cl_sem);

Ditto...

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: [PATCH] mtd_blkdevs: Convert to use the kthread API
Posted by Christoph Hellwig on Thu, 19 Apr 2007 16:47:46 GMT
View Forum Message <> Reply to Message

On Thu, Apr 19, 2007 at 12:55:28AM -0600, Eric W. Biederman wrote:
> From: Eric W. Biederman <ebiederm@xmission.com> - unquoted
>
> thread_run is used intead of kernel_thread, daemonize, and mucking
> around blocking signals directly.

Please don't do incomplete transitions like that. We don't really
want people to use kthread_run, but not the kthread stopping
mechanisms, because people will simply forget about that bit and
we'll never get rid of the enormous amount of, erm creativity, in
handling kernel thread stopping.

This is just the first patch in your series where the thread is mutable,
but it equally applies to all following patches where this is the case
aswell.

_____
Containers mailing list
Containers@lists.linux-foundation.org

Subject: Re: [PATCH] mtd_blkdevs: Convert to use the kthread API
Posted by ebiederm on Thu, 19 Apr 2007 19:13:22 GMT
View Forum Message <> Reply to Message

Christoph Hellwig <hch@infradead.org> writes:

> On Thu, Apr 19, 2007 at 12:55:28AM -0600, Eric W. Biederman wrote:
>> From: Eric W. Biederman <ebiederm@xmission.com> - unquoted
>>
>> thread_run is used intead of kernel_thread, daemonize, and mucking
>> around blocking signals directly.
>
> Please don't do incomplete transitions like that. We don't really
> want people to use kthread_run, but not the kthread stopping
> mechanisms, because people will simply forget about that bit and
> we'll never get rid of the enormous amount of, erm creativity, in
> handling kernel thread stopping.
>
> This is just the first patch in your series where the thread is mutable,
> but it equally applies to all following patches where this is the case
> aswell.

I don't really care about the creativity. Although it would
be nice if it wasn't there. I deliberately left it in so I would be
certain my patches were correct.

I care about killing the maintenance and forward development roadblocks
that are kernel_thread and daemonize. And the user interface problem
that is handling signals in kernel threads.

Eric

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

Subject: Re: [PATCH] nfs lockd reclaimer: Convert to kthread API
Posted by ebiederm on Thu, 19 Apr 2007 19:20:29 GMT
View Forum Message <> Reply to Message

Trond Myklebust <trond.myklebust@fys.uio.no> writes:

> On Thu, 2007-04-19 at 01:58 -0600, Eric W. Biederman wrote:
>> From: Eric W. Biederman <ebiederm@xmission.com>
>>
>> Start the reclaimer thread using kthread_run instead
>> of a combination of kernel_thread and daemonize.
>> The small amount of signal handling code is also removed
>> as it makes no sense and is a maintenance problem to handle
>> signals in kernel threads.
>
> Vetoed. Removing stuff just because it doesn't make sense to you is not
> acceptable.
>
> Signal handling in reclaimer threads is there in order to allow
> administrators to deal with the case where the server never comes up
> again.

Doesn't unmount handle that?

Regardless kernel threads should be an implementation detail
not a part of the user interface. If kernel threads are part
of the user interface it makes them very hard to change.

So it isn't that it doesn't make sense to me it is that it looks
fundamentally broken and like a maintenance nightmare.

I would rather kill kernel threads then try and simulate them
when the kernel implementation has changed and kernel threads
are not visible.

If I could be convinced that signal handling in kernel threads
is not something that will impede code modifications and refactoring
I would have less of a problem, and might not care.

With pid namespaces all kernel threads will disappear so how do
we cope with the problem when the sysadmin can not see the kernel
threads?

Eric

Subject: Re: [PATCH] nfs lockd reclaimer: Convert to kthread API

On Thu, 2007-04-19 at 13:20 -0600, Eric W. Biederman wrote:
> Trond Myklebust <trond.myklebust@fys.uio.no> writes:
>
> > On Thu, 2007-04-19 at 01:58 -0600, Eric W. Biederman wrote:
> >> From: Eric W. Biederman <ebiederm@xmission.com>
> >>
> >> Start the reclaimer thread using kthread_run instead
> >> of a combination of kernel_thread and daemonize.
> >> The small amount of signal handling code is also removed
> >> as it makes no sense and is a maintenance problem to handle
> >> signals in kernel threads.
> >
> > Vetoed. Removing stuff just because it doesn't make sense to you is not
> > acceptable.
> >
> > Signal handling in reclaimer threads is there in order to allow
> > administrators to deal with the case where the server never comes up
> > again.
>
> Doesn't unmount handle that?

On a pinned filesystem?

> Regardless kernel threads should be an implementation detail
> not a part of the user interface. If kernel threads are part
> of the user interface it makes them very hard to change.
>
> So it isn't that it doesn't make sense to me it is that it looks
> fundamentally broken and like a maintenance nightmare.
>
> I would rather kill kernel threads then try and simulate them
> when the kernel implementation has changed and kernel threads
> are not visible.
>
> If I could be convinced that signal handling in kernel threads
> is not something that will impede code modifications and refactoring
> I would have less of a problem, and might not care.

Tough. You're the one proposing to change existing code.

> With pid namespaces all kernel threads will disappear so how do
> we cope with the problem when the sysadmin can not see the kernel
> threads?

Then you have a usability problem. How does the sysadmin reboot the
system if there is no way to shut down the processes that are hanging on

an unresponsive filesystem?

Trond

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: [PATCH] nfs lockd reclaimer: Convert to kthread API
Posted by Dave Hansen on Thu, 19 Apr 2007 21:25:02 GMT
View Forum Message <> Reply to Message

On Thu, 2007-04-19 at 17:19 -0400, Trond Myklebust wrote:
> > With pid namespaces all kernel threads will disappear so how do
> > we cope with the problem when the sysadmin can not see the kernel
> > threads?

Do they actually always disappear, or do we keep them in the
init_pid_namespace?

-- Dave

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: [PATCH] nfs lockd reclaimer: Convert to kthread API
Posted by akpm on Thu, 19 Apr 2007 21:40:59 GMT
View Forum Message <> Reply to Message

On Thu, 19 Apr 2007 17:19:24 -0400
Trond Myklebust <trond.myklebust@fys.uio.no> wrote:

> > Regardless kernel threads should be an implementation detail
> > not a part of the user interface. If kernel threads are part
> > of the user interface it makes them very hard to change.
> >
> > So it isn't that it doesn't make sense to me it is that it looks
> > fundamentally broken and like a maintenance nightmare.
> >
> > I would rather kill kernel threads then try and simulate them
> > when the kernel implementation has changed and kernel threads
> > are not visible.

> >
> > If I could be convinced that signal handling in kernel threads
> > is not something that will impede code modifications and refactoring
> > I would have less of a problem, and might not care.
>
> Tough. You're the one proposing to change existing code.

Using signals to communicate with kernel threads is fairly unpleasant, IMO.
We have much simpler, faster and more idiomatic ways of communicating
between threads in-kernel and there are better ways in which userspace can
communicate with the kernel - system calls, for example...

So I think generally any move which gets us away from using signals in
kernel threads is moving in a good direction.

> > With pid namespaces all kernel threads will disappear so how do
> > we cope with the problem when the sysadmin can not see the kernel
> > threads?
>
> Then you have a usability problem. How does the sysadmin reboot the
> system if there is no way to shut down the processes that are hanging on
> an unresponsive filesystem?

Where's the hang? A user process is stuck on h_rwsem?

If so, would it be appropriate to convert the user process to use
down_foo_interruptible(), so that the operator can just kill the user
process as expected, rather than having to futz around killing kernel
threads?

Subject: Re: [PATCH] nfs lockd reclaimer: Convert to kthread API
Posted by Trond Myklebust on Thu, 19 Apr 2007 22:04:44 GMT
View Forum Message <> Reply to Message

On Thu, 2007-04-19 at 14:40 -0700, Andrew Morton wrote:
> Using signals to communicate with kernel threads is fairly unpleasant, IMO.
> We have much simpler, faster and more idiomatic ways of communicating
> between threads in-kernel and there are better ways in which userspace can
> communicate with the kernel - system calls, for example...
>
> So I think generally any move which gets us away from using signals in
> kernel threads is moving in a good direction.

I have yet to see a proposal which did. Eric's patch was eliminating
signals in kernel threads that used them without proposing any
replacement mechanism or showing that he had plans to do so. That is a
good reason for a veto.

> > > With pid namespaces all kernel threads will disappear so how do
> > > we cope with the problem when the sysadmin can not see the kernel
> > > threads?
> >
> > Then you have a usability problem. How does the sysadmin reboot the
> > system if there is no way to shut down the processes that are hanging on
> > an unresponsive filesystem?
>
> Where's the hang? A user process is stuck on h_rwsem?
>
> If so, would it be appropriate to convert the user process to use
> down_foo_interruptible(), so that the operator can just kill the user
> process as expected, rather than having to futz around killing kernel
> threads?

If an NFS server reboots, then the locks held by user processes on the
client need to be re-established by when it comes up again. Otherwise,
the processes that thought they were holding locks will suddenly fail.
This recovery job is currently the done by a kernel thread.

The question is then what to do if the server crashes again while the
kernel thread is re-establishing the locks. Particularly if it never
comes back again.
Currently, the administrator can intervene by killing anything that has
open files on that volume and kill the recovery kernel thread.
You'll also note that lockd_down(), nfsd_down() etc all use signals to
inform lockd(), nfsd() etc that they should be shutting down. Since the
reclaimer thread is started by the lockd() thread using CLONE_SIGHAND,
this means that we also automatically kill any lingering recovery
threads whenever we shutdown lockd().

These mechanisms need to be replaced _before_ we start shooting down
sigallow() etc in the kernel.

  Trond

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

Subject: Re: [PATCH] mtd_blkdevs: Convert to use the kthread API
Posted by akpm on Thu, 19 Apr 2007 22:26:39 GMT
View Forum Message <> Reply to Message

On Thu, 19 Apr 2007 13:13:22 -0600
ebiederm@xmission.com (Eric W. Biederman) wrote:

> Christoph Hellwig <hch@infradead.org> writes:
>
> > On Thu, Apr 19, 2007 at 12:55:28AM -0600, Eric W. Biederman wrote:
> >> From: Eric W. Biederman <ebiederm@xmission.com> - unquoted
> >>
> >> thread_run is used intead of kernel_thread, daemonize, and mucking
> >> around blocking signals directly.
> >
> > Please don't do incomplete transitions like that. We don't really
> > want people to use kthread_run, but not the kthread stopping
> > mechanisms, because people will simply forget about that bit and
> > we'll never get rid of the enormous amount of, erm creativity, in
> > handling kernel thread stopping.
> >
> > This is just the first patch in your series where the thread is mutable,
> > but it equally applies to all following patches where this is the case
> > aswell.
>
> I don't really care about the creativity. Although it would
> be nice if it wasn't there. I deliberately left it in so I would be
> certain my patches were correct.
>
> I care about killing the maintenance and forward development roadblocks
> that are kernel_thread and daemonize. And the user interface problem
> that is handling signals in kernel threads.
>

Yes, I think that is a practical position, if not an ideal one.

MTD (to pick one example) does need to be decruftified: remove
r->blkcore_priv->exiting, probably ->blkcore_priv->thread_dead, switch
deregister_mtd_blktrans() to use kthread_stop(). But it's a bit much to
expect Eric to make that conversion, and to suitably test it. All he can
do is to make a best-effort and hope that someone else tests it, which
isn't very reliable.

This partial patch at least gets us some of the way there, and serves as a
gentle reminder to dwmwyouknowwho to finish cleaning this stuff up.

I'd be more concerned about a part-conversion in a subsystem which has no
identifiable maintainer, because in that case the chances are that we'll
just forget about it an the conversion would never be completed.

And of course, these are not simply cleanup patches: we actually need to get
the kernel threads out of the daemonize() and signalling game to complete
the virtualisation work.

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: [PATCH] dvb_en_50221: Convert to kthread API
Posted by akpm on Thu, 19 Apr 2007 22:34:13 GMT
View Forum Message <> Reply to Message

On Thu, 19 Apr 2007 01:59:04 -0600
"Eric W. Biederman" <ebiederm@xmission.com> wrote:

> This patch is a minimal transformation to use the kthread API
> doing it's best to preserve the existing logic.
>
> Instead of starting kdvb-ca by calling kernel_thread,
> daemonize and sigfillset we kthread_run is used.
>
> Instead of tracking the pid of the running thread we instead
> simply keep a flag to indicate that the current thread is
> running, as that is all the pid is really used for.
>
> And finally the kill_proc sending signal 0 to the kernel thread to
> ensure it is alive before we wait for it to shutdown is removed.
> The kthread API does not provide the pid so we don't have that
> information readily available and the test is just silly. If there
> is no shutdown race the test is a useless confirmation of that the
> thread is running. If there is a race the test doesn't fix it and
> we should fix the race properly.

urgh, yes, this is just sad. We should convert this driver fully to
the kthread API - it will end up much better.

I'll queue this up as a -mm-only thing as a gentle reminder that
we should do it properly.

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: [PATCH] smbfs: Remove unnecessary allow_signal
Posted by akpm on Thu, 19 Apr 2007 22:47:42 GMT
View Forum Message <> Reply to Message

On Thu, 19 Apr 2007 01:59:03 -0600
"Eric W. Biederman" <ebiederm@xmission.com> wrote:

> From: Eric W. Biederman <ebiederm@xmission.com>
>
> Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
> ---
> fs/smbfs/smbiod.c | 2 --
> 1 files changed, 0 insertions(+), 2 deletions(-)
>
> diff --git a/fs/smbfs/smbiod.c b/fs/smbfs/smbiod.c
> index 3e61b44..67176af 100644
> --- a/fs/smbfs/smbiod.c
> +++ b/fs/smbfs/smbiod.c
> @@ -298,8 +298,6 @@ out:
>   */
> static int smbiod(void *unused)
> {
> - allow_signal(SIGKILL);
> -
>   VERBOSE("SMB Kernel thread starting (%d) ...\n", current->pid);
>

Why is it unnecessary? afaict we can presently terminate smbiod
with a SIGKILL, and this change will alter (ie: break) that behaviour?

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

Subject: Re: [PATCH] saa7134-tvaudio: Convert to kthread API.
Posted by akpm on Thu, 19 Apr 2007 22:52:45 GMT
View Forum Message <> Reply to Message

On Thu, 19 Apr 2007 01:58:58 -0600
"Eric W. Biederman" <ebiederm@xmission.com> wrote:

> It is my goal to replace all kernel code that handles signals
> from user space, calls kernel_thread or calls daemonize. All
> of which the kthread_api makes unncessary. Handling signals
> from user space is a maintenance problem becuase using a
> kernel thread is an implementation detail and if user space
> cares it does not allow us to change the implementation. Calling

> daemonize is a problem because it has to undo a continually changing
> set of state generated by user space, requiring the implemetation
> to change continually. kernel_thread is a problem because it
> returns a pid_t value. Numeric pids are inherently racy and
> in the presence of a pid namespace they are no longer global
> making them useless for general use in the kernel.
>
> So this patch renames the pid member of struct saa7134_thread
> started and changes it's type from pid_t to int. All it
> has ever been used for is to detect if the kernel thread
> is has been started so this works.
>
> allow_signal(SIGTERM) and the calls to signal_pending have
> been removed they are needed for the driver to operation.
>
> The startup of tvaudio_thread and tvaudio_thread_dep have
> been modified to use kthread_run instead of a combination
> of kernel_thread and daemonize.
>
> The result is code that is slightly simpler and more
> maintainable.

This one also really wants to be converted to full use of the
API. ie: use kthread_stop(), kthread_should_stop(), remove all
the hand-woven equivalent stuff we have in there.

I'll tag this as an -mm-only thing as well, in the hope that someone
who can test the changes will be able to find time to address
all this.

Subject: Re: [PATCH] ipv4/ipvs: Convert to kthread API
Posted by akpm on Thu, 19 Apr 2007 22:59:44 GMT
View Forum Message <> Reply to Message

On Thu, 19 Apr 2007 18:04:36 +0900
Simon Horman <horms@verge.net.au> wrote:

> On Thu, Apr 19, 2007 at 01:58:57AM -0600, Eric W. Biederman wrote:
> > From: Eric W. Biederman <ebiederm@xmission.com>
> >
> > Modify startup of ipvs sync threads to use kthread_run
> > instead of a weird combination of calling kernel_thread
> > to start a fork_sync_thread whose hole purpose in life was

> > to call kernel_thread again starting the actually sync thread
> > which called daemonize.
> >
> > To use kthread_run I had to move the name calcuation from
> > sync_thread into start_sync_thread resulting in a small
> > amount of code motion.
> >
> > The result is simpler and more maintainable piece of code.
>
> Thanks Eric, I'll review this and get back to you shortly.
>

There still seems to be quite a lot of complexity in this driver's
thread handling which could be removed if we did a full conversion
to the kthread API.

It all looks.... surprisingly complex in there.

Subject: Re: [PATCH] net/rxrpc: Convert to kthread API.
Posted by akpm on Thu, 19 Apr 2007 23:05:30 GMT
View Forum Message <> Reply to Message

On Thu, 19 Apr 2007 10:32:38 +0100
David Howells <dhowells@redhat.com> wrote:

> Eric W. Biederman <ebiederm@xmission.com> wrote:
>
> > This patch modifies the startup of krxtimod, krxiod, and krxsecd
> > to use kthread_run instead of a combination of kernel_thread
> > and daemonize making the code slightly simpler and more maintainable.
>
> Again, please drop in favour of my RxRPC patches.
>

Do those patches convert all this code over to full use of the kthread
API? Because it seems that a conversion would be straightforward, and
is needed.

Subject: Re: [PATCH] bluetooth rfcomm: Convert to kthread API.
Posted by akpm on Thu, 19 Apr 2007 23:12:53 GMT
View Forum Message <> Reply to Message

On Thu, 19 Apr 2007 01:58:54 -0600
"Eric W. Biederman" <ebiederm@xmission.com> wrote:

> From: Eric W. Biederman <ebiederm@xmission.com>
>
> This patch starts krfcommd using kthread_run instead of a combination
> of kernel_thread and daemonize making the code slightly simpler
> and more maintainable.

gargh, the more I look at these things, the more I agree with Christoph.

> Cc: Marcel Holtmann <marcel@holtmann.org>
> Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
> ---
> net/bluetooth/rfcomm/core.c | 4 ++--
> 1 files changed, 2 insertions(+), 2 deletions(-)
>
> diff --git a/net/bluetooth/rfcomm/core.c b/net/bluetooth/rfcomm/core.c
> index 34f993a..baaad49 100644
> --- a/net/bluetooth/rfcomm/core.c
> +++ b/net/bluetooth/rfcomm/core.c
> @@ -38,6 +38,7 @@
> #include <linux/net.h>
> #include <linux/mutex.h>
> #include <linux/freezer.h>
> +#include <linux/kthread.h>
>
> #include <net/sock.h>
> #include <asm/uaccess.h>
> @@ -1938,7 +1939,6 @@ static int rfcomm_run(void *unused)
>
>   atomic_inc(&running);
>
> - daemonize("krfcommd");
>   set_user_nice(current, -10);
>
>   BT_DBG("");
> @@ -2058,7 +2058,7 @@ static int __init rfcomm_init(void)
>
>   hci_register_cb(&rfcomm_cb);
>
> - kernel_thread(rfcomm_run, NULL, CLONE_KERNEL);
> + kthread_run(rfcomm_run, NULL, "krfcommd");
>
>   if (class_create_file(bt_class, &class_attr_rfcomm_dlc) < 0)

>   BT_ERR("Failed to create RFCOMM info file");

We should remove the file-wide `terminate' and `running' and switch the
thread management over to kthread_run(), kthread_stop() and
kthread_should_stop().

btw, this:

```
static void rfcomm_worker(void)
{
 BT_DBG("");

 while (!atomic_read(&terminate)) {
  try_to_freeze();

  if (!test_bit(RFCOMM_SCHED_WAKEUP, &rfcomm_event)) {
   /* No pending events. Let's sleep.
    * Incoming connections and data will wake us up. */
   set_current_state(TASK_INTERRUPTIBLE);
   schedule();
  }

  /* Process stuff */
  clear_bit(RFCOMM_SCHED_WAKEUP, &rfcomm_event);
  rfcomm_process_sessions();
 }
 set_current_state(TASK_RUNNING);
 return;
}
```

appears to have the classic sleep/wakeup bug: if the wakeup happens after
we tested RFCOMM_SCHED_WAKEUP we will miss it.

Easy fix:

From: Andrew Morton <akpm@linux-foundation.org>

Signed-off-by: Andrew Morton <akpm@linux-foundation.org>
---

 net/bluetooth/rfcomm/core.c | 4 ++--
 1 files changed, 2 insertions(+), 2 deletions(-)

diff -puN net/bluetooth/rfcomm/core.c~rfcomm_worker-fix-wakeup-race
net/bluetooth/rfcomm/core.c
--- a/net/bluetooth/rfcomm/core.c~rfcomm_worker-fix-wakeup-race
+++ a/net/bluetooth/rfcomm/core.c
@@ -1855,18 +1855,18 @@ static void rfcomm_worker(void)

```
  while (!atomic_read(&terminate)) {
   try_to_freeze();

+  set_current_state(TASK_INTERRUPTIBLE);
   if (!test_bit(RFCOMM_SCHED_WAKEUP, &rfcomm_event)) {
    /* No pending events. Let's sleep.
     * Incoming connections and data will wake us up. */
-   set_current_state(TASK_INTERRUPTIBLE);
    schedule();
   }
+  set_current_state(TASK_RUNNING);

   /* Process stuff */
   clear_bit(RFCOMM_SCHED_WAKEUP, &rfcomm_event);
   rfcomm_process_sessions();
  }
- set_current_state(TASK_RUNNING);
  return;
 }


_
```

(I think it's safer and saner to always run rfcomm_process_sessions() while
in state TASK_RUNNING, not maybe-in-state-TASK_INTERRUPTIBLE)

Subject: Re: [PATCH] bluetooth hidp: Convert to kthread API.
Posted by akpm on Thu, 19 Apr 2007 23:20:37 GMT
View Forum Message <> Reply to Message

On Thu, 19 Apr 2007 01:58:53 -0600
"Eric W. Biederman" <ebiederm@xmission.com> wrote:

> This patch starts up khidp using kthread_run instead
> of kernel_thread and daemonize, resulting is slightly
> simpler and more maintainable code.

argh, they're all like this :(

It's a shame your changelogs didn't fully spell out the reasons for
this conversion. Right now, the maintainers probably think that these
are nice-to-have cleanups, not must-have-to-make-virtualisation-work-right

fixes.

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: [PATCH] bluetooth bnep: Convert to kthread API.
Posted by akpm on Thu, 19 Apr 2007 23:24:59 GMT
View Forum Message <> Reply to Message

On Thu, 19 Apr 2007 01:58:51 -0600
"Eric W. Biederman" <ebiederm@xmission.com> wrote:

> From: Eric W. Biederman <ebiederm@xmission.com>
>
> This patch starts kbenpd using kthread_run replacing
> a combination of kernel_thread and daemonize. Making
> the code a little simpler and more maintainable.
>
>

 while (!atomic_read(&s->killed)) {

ho hum.

> + task = kthread_run(bnep_session, s, "kbnepd %s", dev->name);

It's unusual to have a kernel thread which has a space in its name. That
could trip up infufficient-defensive userspace tools.

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: [PATCH] macintosh/mediabay: Convert to kthread API.
Posted by akpm on Thu, 19 Apr 2007 23:30:01 GMT
View Forum Message <> Reply to Message

On Thu, 19 Apr 2007 01:58:50 -0600
"Eric W. Biederman" <ebiederm@xmission.com> wrote:

> This patch modifies the startup of the media_bay_task
> to use kthread_run and not a combination of kernel_thread,
> deamonize and sigfillset.

>
> In addition since we now always want to ignore signals
> the MB_IGNORE_SIGNALS define is removed along with the
> test for signal_pending.
>
> The result is slightly simpler code that is more
> maintainable.

Looks OK - there's no way of stopping the kernel thread anyway.

It appears that nobody has tried to use this driver at the same time as
software-suspend. At least, not successfully. A strategic try_to_freeze()
should fix it.

This will become (a little) more serious when cpu hotplug is switched to
use the process freezer, and perhaps it breaks kprobes already.

Subject: Re: [PATCH] macintosh/therm_windtunnel.c: Convert to kthread API.
Posted by akpm on Thu, 19 Apr 2007 23:37:40 GMT
View Forum Message <> Reply to Message

On Thu, 19 Apr 2007 01:58:48 -0600
"Eric W. Biederman" <ebiederm@xmission.com> wrote:

> Start the g4fand using kthread_run not a combination
> of kernel_thread and deamonize. This makes the code
> a little simpler and more maintainable.

I had a bit of trouble reviewing this one because I was laughing so hard at
the attempted coding-style in that driver. Oh well.

I continue creeping into Christoph's camp - there's quite a bit of
open-coded gunk which would go away if we were to teach this driver about
kthread_should_stop() and kthread_stop(), and the conversion looks awfully
easy to do. It's a shame to stop here.

Oh well, I guess at least this is some forward progress.

Subject: Re: [PATCH] powerpc pseries eeh: Convert to kthread API
Posted by akpm on Thu, 19 Apr 2007 23:47:23 GMT
View Forum Message <> Reply to Message

On Thu, 19 Apr 2007 01:58:45 -0600
"Eric W. Biederman" <ebiederm@xmission.com> wrote:

> This patch modifies the startup of eehd to use kthread_run
> not a combination of kernel_thread and daemonize. Making
> the code slightly simpler and more maintainable.
>

You're making me look at a lot of things which I'd prefer not to have
looked at.

> arch/powerpc/platforms/pseries/eeh_event.c | 4 ++--

This one kicks off a kernel thread in response to each "PCI error event",
and that kernel thread hangs about for one hour then exits.

One wonders what happens if we get 1,000,000 of these events per
second.

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

Subject: Re: [PATCH] ia64 sn xpc: Convert to use kthread API.
Posted by akpm on Thu, 19 Apr 2007 23:51:03 GMT
View Forum Message <> Reply to Message

On Thu, 19 Apr 2007 01:58:44 -0600
"Eric W. Biederman" <ebiederm@xmission.com> wrote:

>
> This patch starts the xpc kernel threads using kthread_run
> not a combination of kernel_thread and daemonize. Resuling
> in slightly simpler and more maintainable code.
>
> Cc: Jes Sorensen <jes@sgi.com>
> Cc: Tony Luck <tony.luck@intel.com>
> Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
> ---
> arch/ia64/sn/kernel/xpc_main.c | 31 ++++++++++++++++++----------

Another driver which should be fully converted to the kthread API:

kthread_stop() and kthread_should_stop().

And according to my logs, this driver was added to the tree more than
a year _after_ the kthread interface was made available.

This isn't good.

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: [PATCH] sparc64/power.c: Convert to use the kthread API
Posted by akpm on Fri, 20 Apr 2007 00:30:21 GMT
View Forum Message <> Reply to Message

On Thu, 19 Apr 2007 01:58:39 -0600
"Eric W. Biederman" <ebiederm@xmission.com> wrote:

> From: Eric W. Biederman <ebiederm@xmission.com>
>
> This starts the sparc64 powerd using kthread_run
> instead of kernel_thread and daemonize. Making the
> code slightly simpler and more maintainable.
>
> In addition the unnecessary flush_signals is removed.

Looks OK. This code could perhaps be switched to call_usermodehelper().

> + task = kthread_urn(powerd, NULL, "powerd");

I'll fix that up before Dave notices ;)

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: [PATCH] sas_scsi_host: Convert to use the kthread API
Posted by akpm on Fri, 20 Apr 2007 00:37:53 GMT
View Forum Message <> Reply to Message

On Thu, 19 Apr 2007 01:58:38 -0600
"Eric W. Biederman" <ebiederm@xmission.com> wrote:

> From: Eric W. Biederman <ebiederm@xmission.com>
>

> This patch modifies the sas scsi host thread startup
> to use kthread_run not kernel_thread and deamonize.
> kthread_run is slightly simpler and more maintainable.
>

Again, I'll rename this to "partially convert...". This driver should be
using kthread_should_stop() and kthread_stop() rather than the
apparently-unnecessary ->queue_thread_kill thing.

This driver was merged two and a half years after the kthread API was
available. Our coding-vs-reviewing effort is out of balance.

> ---
> drivers/scsi/libsas/sas_scsi_host.c | 11 +++++++-----
> 1 files changed, 6 insertions(+), 5 deletions(-)
>
> diff --git a/drivers/scsi/libsas/sas_scsi_host.c b/drivers/scsi/libsas/sas_scsi_host.c
> index 46ba3a7..7a38ac5 100644
> --- a/drivers/scsi/libsas/sas_scsi_host.c
> +++ b/drivers/scsi/libsas/sas_scsi_host.c
> @@ -40,6 +40,7 @@
> #include <linux/blkdev.h>
> #include <linux/scatterlist.h>
> #include <linux/freezer.h>
> +#include <linux/kthread.h>
>
> /* ---------- SCSI Host glue ---------- */
>
> @@ -870,7 +871,6 @@ static int sas_queue_thread(void *_sas_ha)
>   struct sas_ha_struct *sas_ha = _sas_ha;
>   struct scsi_core *core = &sas_ha->core;
>
> - daemonize("sas_queue_%d", core->shost->host_no);
>   current->flags |= PF_NOFREEZE;
>
>   complete(&queue_th_comp);
> @@ -891,19 +891,20 @@ static int sas_queue_thread(void *_sas_ha)
>
> int sas_init_queue(struct sas_ha_struct *sas_ha)
> {
> - int res;
>   struct scsi_core *core = &sas_ha->core;
> + struct task_struct *task;
>
>   spin_lock_init(&core->task_queue_lock);
>   core->task_queue_size = 0;
>   INIT_LIST_HEAD(&core->task_queue);

> init_MUTEX_LOCKED(&core->queue_thread_sema);
>
> - res = kernel_thread(sas_queue_thread, sas_ha, 0);
> - if (res >= 0)
> + task = kthread_run(sas_queue_thread, sas_ha,
> + "sas_queue_%d", core->shost->host_no);
> + if (!IS_ERR(task))
> wait_for_completion(&queue_th_comp);
>
> - return res < 0 ? res : 0;
> + return IS_ERR(task) ? PTR_ERR(task) : 0;

Does that wait_for_completion(&queue_th_comp) actually do anything useful?

If so, what is serialising access to the single queue_th_comp?

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: [PATCH] cpqphp: Convert to use the kthread API
Posted by akpm on Fri, 20 Apr 2007 01:54:26 GMT
View Forum Message <> Reply to Message

On Thu, 19 Apr 2007 01:58:36 -0600 "Eric W. Biederman" <ebiederm@xmission.com> wrote:

> This patch changes cpqphp to use kthread_run and not
> kernel_thread and daemonize to startup and setup
> the cpqphp thread.

ok.. I'll rename this to "partially convert" and shall add a note
to the changelog,

This is another driver which will look a lot nicer when it has been
converted to kthread_should_stop() and kthread_stop()

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: [PATCH] ia64 sn xpc: Convert to use kthread API.
Posted by Jes Sorensen on Fri, 20 Apr 2007 06:23:39 GMT
View Forum Message <> Reply to Message

Andrew Morton wrote:

> Another driver which should be fully converted to the kthread API:
> kthread_stop() and kthread_should_stop().
>
> And according to my logs, this driver was added to the tree more than
> a year _after_ the kthread interface was made available.
>
> This isn't good.

Andrew,

Per my previous response, I'd prefer to have either Russ or Robin ack
the patch doesn't break before it's pushed to Linus.

I don't know much about the xpmem and I am not comfortable testing it.

Cheers,
Jes

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: [PATCH] dvb_en_50221: Convert to kthread API
Posted by Christoph Hellwig on Fri, 20 Apr 2007 06:37:14 GMT
View Forum Message <> Reply to Message

On Thu, Apr 19, 2007 at 03:34:13PM -0700, Andrew Morton wrote:
> On Thu, 19 Apr 2007 01:59:04 -0600
> "Eric W. Biederman" <ebiederm@xmission.com> wrote:
>
> > This patch is a minimal transformation to use the kthread API
> > doing it's best to preserve the existing logic.
> >
> > Instead of starting kdvb-ca by calling kernel_thread,
> > daemonize and sigfillset we kthread_run is used.
> >
> > Instead of tracking the pid of the running thread we instead
> > simply keep a flag to indicate that the current thread is
> > running, as that is all the pid is really used for.
> >
> > And finally the kill_proc sending signal 0 to the kernel thread to
> > ensure it is alive before we wait for it to shutdown is removed.
> > The kthread API does not provide the pid so we don't have that
> > information readily available and the test is just silly. If there
> > is no shutdown race the test is a useless confirmation of that the
> > thread is running. If there is a race the test doesn't fix it and
> > we should fix the race properly.

>
> urgh, yes, this is just sad. We should convert this driver fully to
> the kthread API - it will end up much better.
>
> I'll queue this up as a -mm-only thing as a gentle reminder that
> we should do it properly.

Here's an attempted update to the full kthread API + wake_up_process:

```
Index: linux-2.6/drivers/media/dvb/dvb-core/dvb_ca_en50221.c
===================================================================
--- linux-2.6.orig/drivers/media/dvb/dvb-core/dvb_ca_en50221.c 2007-04-20 07:25:07.000000000
+0200
+++ linux-2.6/drivers/media/dvb/dvb-core/dvb_ca_en50221.c 2007-04-20 07:35:54.000000000
+0200
@@ -37,6 +37,7 @@
 #include <linux/delay.h>
 #include <linux/spinlock.h>
 #include <linux/sched.h>
+#include <linux/kthread.h>

 #include "dvb_ca_en50221.h"
 #include "dvb_ringbuffer.h"
@@ -140,13 +141,7 @@ struct dvb_ca_private {
  wait_queue_head_t wait_queue;

  /* PID of the monitoring thread */
- pid_t thread_pid;
-
- /* Wait queue used when shutting thread down */
- wait_queue_head_t thread_queue;
-
- /* Flag indicating when thread should exit */
- unsigned int exit:1;
+ struct task_struct *thread;

  /* Flag indicating if the CA device is open */
  unsigned int open:1;
@@ -902,28 +897,10 @@ static void dvb_ca_en50221_thread_wakeup

  ca->wakeup = 1;
  mb();
- wake_up_interruptible(&ca->thread_queue);
+ wake_up_process(ca->thread);
 }

 /**
```

```
- * Used by the CA thread to determine if an early wakeup is necessary
- *
- * @param ca CA instance.
- */
-static int dvb_ca_en50221_thread_should_wakeup(struct dvb_ca_private *ca)
-{
- if (ca->wakeup) {
-  ca->wakeup = 0;
-  return 1;
- }
- if (ca->exit)
-  return 1;
-
- return 0;
-}
-
-
-/**
  * Update the delay used by the thread.
  *
  * @param ca CA instance.
@@ -982,7 +959,6 @@ static void dvb_ca_en50221_thread_update
 static int dvb_ca_en50221_thread(void *data)
 {
  struct dvb_ca_private *ca = data;
- char name[15];
  int slot;
  int flags;
  int status;
@@ -991,28 +967,17 @@ static int dvb_ca_en50221_thread(void *d

  dprintk("%s\n", __FUNCTION__);

- /* setup kernel thread */
- snprintf(name, sizeof(name), "kdvb-ca-%i:%i", ca->dvbdev->adapter->num, ca->dvbdev->id);
-
- lock_kernel();
- daemonize(name);
- sigfillset(&current->blocked);
- unlock_kernel();
-
  /* choose the correct initial delay */
  dvb_ca_en50221_thread_update_delay(ca);

  /* main loop */
- while (!ca->exit) {
+ while (!kthread_should_stop()) {
   /* sleep for a bit */
```

```
-  if (!ca->wakeup) {
-   flags = wait_event_interruptible_timeout(ca->thread_queue,
-         dvb_ca_en50221_thread_should_wakeup(ca),
-         ca->delay);
-   if ((flags == -ERESTARTSYS) || ca->exit) {
-    /* got signal or quitting */
-    break;
-   }
+  while (!ca->wakeup) {
+   set_current_state(TASK_INTERRUPTIBLE);
+   schedule_timeout(ca->delay);
+   if (kthread_should_stop())
+    return 0;
   }
   ca->wakeup = 0;

@@ -1181,10 +1146,6 @@ static int dvb_ca_en50221_thread(void *d
   }
  }

- /* completed */
- ca->thread_pid = 0;
- mb();
- wake_up_interruptible(&ca->thread_queue);
  return 0;
 }

@@ -1682,9 +1643,6 @@ int dvb_ca_en50221_init(struct dvb_adapt
   goto error;
  }
  init_waitqueue_head(&ca->wait_queue);
- ca->thread_pid = 0;
- init_waitqueue_head(&ca->thread_queue);
- ca->exit = 0;
  ca->open = 0;
  ca->wakeup = 0;
  ca->next_read_slot = 0;
@@ -1710,14 +1668,14 @@ int dvb_ca_en50221_init(struct dvb_adapt
  mb();

  /* create a kthread for monitoring this CA device */
-
- ret = kernel_thread(dvb_ca_en50221_thread, ca, 0);
-
- if (ret < 0) {
-  printk("dvb_ca_init: failed to start kernel_thread (%d)\n", ret);
+ ca->thread = kthread_run(dvb_ca_en50221_thread, ca, "kdvb-ca-%i:%i",
+      ca->dvbdev->adapter->num, ca->dvbdev->id);
```

```
+ if (IS_ERR(ca->thread)) {
+  ret = PTR_ERR(ca->thread);
+  printk("dvb_ca_init: failed to start kernel_thread (%d)\n",
+   ret);
   goto error;
  }
- ca->thread_pid = ret;
  return 0;

 error:
@@ -1748,17 +1706,7 @@ void dvb_ca_en50221_release(struct dvb_c
  dprintk("%s\n", __FUNCTION__);

  /* shutdown the thread if there was one */
- if (ca->thread_pid) {
-  if (kill_proc(ca->thread_pid, 0, 1) == -ESRCH) {
-   printk("dvb_ca_release adapter %d: thread PID %d already died\n",
-          ca->dvbdev->adapter->num, ca->thread_pid);
-  } else {
-   ca->exit = 1;
-   mb();
-   dvb_ca_en50221_thread_wakeup(ca);
-   wait_event_interruptible(ca->thread_queue, ca->thread_pid == 0);
-  }
- }
+ kthread_stop(ca->thread);

  for (i = 0; i < ca->slot_count; i++) {
   dvb_ca_en50221_slot_shutdown(ca, i);
```

Subject: Re: [PATCH] dvb_en_50221: Convert to kthread API
Posted by akpm on Fri, 20 Apr 2007 06:48:09 GMT
View Forum Message <> Reply to Message

On Fri, 20 Apr 2007 07:37:14 +0100 Christoph Hellwig <hch@infradead.org> wrote:

> > urgh, yes, this is just sad. We should convert this driver fully to
> > the kthread API - it will end up much better.
> >
> > I'll queue this up as a -mm-only thing as a gentle reminder that
> > we should do it properly.
>
> Here's an attempted update to the full kthread API + wake_up_process:

drivers/media/dvb/dvb-core/dvb_ca_en50221.c | 84 +++----------
1 file changed, 16 insertions(+), 68 deletions(-)

tasty!

Subject: Re: [PATCH] net/rxrpc: Convert to kthread API.
Posted by David Howells on Fri, 20 Apr 2007 07:47:18 GMT
View Forum Message <> Reply to Message

Andrew Morton <akpm@linux-foundation.org> wrote:

> Do those patches convert all this code over to full use of the kthread
> API? Because it seems that a conversion would be straightforward, and
> is needed.

No. They delete all that code entirely and use workqueues instead. So, I suppose merging Eric's patches first should be a simple matter of just deleting his revised code instead.

David

Subject: Re: [PATCH] macintosh/mediabay: Convert to kthread API.
Posted by Benjamin Herrenschmid on Fri, 20 Apr 2007 08:51:58 GMT
View Forum Message <> Reply to Message

> Looks OK - there's no way of stopping the kernel thread anyway.
>
> It appears that nobody has tried to use this driver at the same time as
> software-suspend. At least, not successfully. A strategic try_to_freeze()
> should fix it.
>
> This will become (a little) more serious when cpu hotplug is switched to
> use the process freezer, and perhaps it breaks kprobes already.

I'll dig a box with that hardware and do some tests, but it looks nice.

Thanks Eric !

There should be no problem with cpu hotplug, the only machines using the media bay driver are old Apple laptops with only one CPU and no HW threads.

Ben.

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: [PATCH] macintosh/therm_windtunnel.c: Convert to kthread API.
Posted by Benjamin Herrenschmid on Fri, 20 Apr 2007 08:53:00 GMT
View Forum Message <> Reply to Message

On Thu, 2007-04-19 at 16:37 -0700, Andrew Morton wrote:
> On Thu, 19 Apr 2007 01:58:48 -0600
> "Eric W. Biederman" <ebiederm@xmission.com> wrote:
>
> > > Start the g4fand using kthread_run not a combination
> > > of kernel_thread and deamonize. This makes the code
> > > a little simpler and more maintainable.
>
> I had a bit of trouble reviewing this one because I was laughing so hard at
> the attempted coding-style in that driver. Oh well.

Heh

> I continue creeping into Christoph's camp - there's quite a bit of
> open-coded gunk which would go away if we were to teach this driver about
> kthread_should_stop() and kthread_stop(), and the conversion looks awfully
> easy to do. It's a shame to stop here.
>
> Oh well, I guess at least this is some forward progress.

My main problem with touching that driver is that I don't have the hardware to test. I'll try to find a user to play the ginea pig.

Ben.

_____
Containers mailing list
Containers@lists.linux-foundation.org

Subject: Re: Getting the new RxRPC patches upstream
Posted by davem on Fri, 20 Apr 2007 08:58:38 GMT
View Forum Message <> Reply to Message

From: David Howells <dhowells@redhat.com>
Date: Fri, 20 Apr 2007 09:02:07 +0100

> David Miller <davem@davemloft.net> wrote:
>
> > I applied already the patches I thought were appropriate,
> > you had some crypto layer changes that you need to work
> > out with Herbert Xu before the rest can be applied.
>
> Should the rest of it go via Andrew's tree then?

Now that Herbert cleared up the crypto layer issues
the only problem left is that there are generic changes
in there which are not strictly networking but which
your subsequent networking changes depend upon.

This is a mess, and makes merging your work into the
net-2.6.22 tree more difficult.

Is it possible for your changes to be purely networking
and not need those changes outside of the networking?

I guess one of them was just a symbol export which I
could add to the net-2.6.22 tree, but weren't there some
more involved non-networking bits in there?

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

Subject: Re: [PATCH] dvb_en_50221: Convert to kthread API
Posted by Cedric Le Goater on Fri, 20 Apr 2007 09:37:10 GMT
View Forum Message <> Reply to Message

Andrew Morton wrote:
> On Fri, 20 Apr 2007 07:37:14 +0100 Christoph Hellwig <hch@infradead.org> wrote:
>
>>> urgh, yes, this is just sad. We should convert this driver fully to
>>> the kthread API - it will end up much better.

>>>
>>> I'll queue this up as a -mm-only thing as a gentle reminder that
>>> we should do it properly.
>> Here's an attempted update to the full kthread API + wake_up_process:
>
> drivers/media/dvb/dvb-core/dvb_ca_en50221.c | 84 +++----------------
> 1 file changed, 16 insertions(+), 68 deletions(-)
>
> tasty!

Indeed !

I have sent a similar patch a few months ago :

 http://lkml.org/lkml/2007/1/24/178

with a less aggressive diffstat though :)

Andrew (de Quincey) just drop mine, if you haven't already done. Christoph's is
more recent and looks better.

Thanks,

C.

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: Getting the new RxRPC patches upstream
Posted by David Howells on Fri, 20 Apr 2007 10:41:46 GMT
View Forum Message <> Reply to Message

David Miller <davem@davemloft.net> wrote:

> Now that Herbert cleared up the crypto layer issues
> the only problem left is that there are generic changes
> in there which are not strictly networking but which
> your subsequent networking changes depend upon.
>
> This is a mess, and makes merging your work into the
> net-2.6.22 tree more difficult.

There are only two non-net patches that AF_RXRPC depends on:

 (1) The key facility changes. That's all my code anyway, and shouldn't be a
     problem to merge unless someone else has put some changes in there that I

don't know about.

(2) try_to_cancel_delayed_work(). I suppose I could use cancel_delayed_work() instead, but that's less efficient as it waits for the timer completion function to finish.

And one that AFS depends on:

(3) Cache the key in nameidata. I still don't have Al's agreement on this, but it's purely caching, so I could drop that patch for the moment and excise the stuff that uses it from my AFS patches if that would help.

Do you class the AFS patches as "networking changes"?

Do you want me to consolidate my patches to make things simpler for you?

Do you want me to rebase my patches onto net-2.6.22?

I have the following patches, in order, available now, though I haven't yet released the last few (they can all be downloaded from my RH people pages):

move-skb-generic.diff (you've got this)
timers.diff
keys.diff
af_rxrpc.diff
afs-cleanup.diff
af_rxrpc-kernel.diff
af_rxrpc-afs.diff
af_rxrpc-delete-old.diff
af_rxrpc-own-workqueues.diff
af_rxrpc-fixes.diff
afs-callback-wq.diff
afs-vlocation.diff
afs-multimount.diff
afs-rxrpc-key.diff
afs-nameidata-key.diff
afs-security.diff
afs-doc.diff
netlink-support-MSG_TRUNC.diff (you've got this)
afs-get-capabilities.diff
afs-initcallbackstate3.diff
afs-dir-write-support.diff

David

Subject: Re: [PATCH] saa7134-tvaudio: Convert to kthread API.
Posted by Cedric Le Goater on Fri, 20 Apr 2007 12:48:35 GMT
View Forum Message <> Reply to Message

Eric W. Biederman wrote:
> From: Eric W. Biederman <ebiederm@xmission.com> - unquoted
>
> It is my goal to replace all kernel code that handles signals
> from user space, calls kernel_thread or calls daemonize. All
> of which the kthread_api makes unncessary. Handling signals
> from user space is a maintenance problem becuase using a
> kernel thread is an implementation detail and if user space
> cares it does not allow us to change the implementation. Calling
> daemonize is a problem because it has to undo a continually changing
> set of state generated by user space, requiring the implemetation
> to change continually. kernel_thread is a problem because it
> returns a pid_t value. Numeric pids are inherently racy and
> in the presence of a pid namespace they are no longer global
> making them useless for general use in the kernel.
>
> So this patch renames the pid member of struct saa7134_thread
> started and changes it's type from pid_t to int. All it
> has ever been used for is to detect if the kernel thread
> is has been started so this works.
>
> allow_signal(SIGTERM) and the calls to signal_pending have
> been removed they are needed for the driver to operation.
>
> The startup of tvaudio_thread and tvaudio_thread_dep have
> been modified to use kthread_run instead of a combination
> of kernel_thread and daemonize.
>
> The result is code that is slightly simpler and more
> maintainable.

Here's a refreshed attempt using kthread_should_stop().
Unfortunately, not tested bc we don't have the hardware.

cheers,

C.

From: Sukadev Bhattiprolu <sukadev@us.ibm.com>

Replace kernel_thread() with kthread_run() since kernel_thread()
is deprecated in drivers/modules. Also remove signalling code
as it is not needed in the driver.

Signed-off-by: Sukadev Bhattiprolu <sukadev@us.ibm.com>

```
Signed-off-by: Cedric Le Goater <clg@fr.ibm.com>
Cc: Mauro Carvalho Chehab <mchehab@infradead.org>
Cc: Containers@lists.osdl.org
Cc: video4linux-list@redhat.com
Cc: v4l-dvb-maintainer@linuxtv.org

---
 drivers/media/video/saa7134/saa7134-tvaudio.c |   45 ++++++++++++++-----------
 drivers/media/video/saa7134/saa7134.h         |    4 --
 2 files changed, 24 insertions(+), 25 deletions(-)

Index: 2.6.21-rc6-mm1/drivers/media/video/saa7134/saa7134.h
===================================================================
--- 2.6.21-rc6-mm1.orig/drivers/media/video/saa7134/saa7134.h
+++ 2.6.21-rc6-mm1/drivers/media/video/saa7134/saa7134.h
@@ -324,10 +324,8 @@ struct saa7134_pgtable {

 /* tvaudio thread status */
 struct saa7134_thread {
- pid_t                   pid;
- struct completion       exit;
+ struct task_struct *    task;
  wait_queue_head_t       wq;
- unsigned int            shutdown;
  unsigned int            scan1;
  unsigned int            scan2;
  unsigned int            mode;
Index: 2.6.21-rc6-mm1/drivers/media/video/saa7134/saa7134-tvaudio.c
===================================================================
--- 2.6.21-rc6-mm1.orig/drivers/media/video/saa7134/saa7134-tvaudio.c
+++ 2.6.21-rc6-mm1/drivers/media/video/saa7134/saa7134-tvaudio.c
@@ -27,6 +27,7 @@
 #include <linux/kernel.h>
 #include <linux/slab.h>
 #include <linux/delay.h>
+#include <linux/kthread.h>
 #include <asm/div64.h>

 #include "saa7134-reg.h"
@@ -344,16 +345,22 @@ static int tvaudio_sleep(struct saa7134_
  DECLARE_WAITQUEUE(wait, current);

  add_wait_queue(&dev->thread.wq, &wait);
- if (dev->thread.scan1 == dev->thread.scan2 && !dev->thread.shutdown) {
+
+ set_current_state(TASK_INTERRUPTIBLE);
+
+ if (dev->thread.scan1 == dev->thread.scan2 && !kthread_should_stop()) {
```

```
   if (timeout < 0) {
-   set_current_state(TASK_INTERRUPTIBLE);
    schedule();
   } else {
    schedule_timeout_interruptible
       (msecs_to_jiffies(timeout));
   }
  }
+
+ set_current_state(TASK_RUNNING);
+
  remove_wait_queue(&dev->thread.wq, &wait);
+
  return dev->thread.scan1 != dev->thread.scan2;
 }

@@ -505,11 +512,9 @@ static int tvaudio_thread(void *data)
  unsigned int i, audio, nscan;
  int max1,max2,carrier,rx,mode,lastmode,default_carrier;

- daemonize("%s", dev->name);
- allow_signal(SIGTERM);
  for (;;) {
   tvaudio_sleep(dev,-1);
-  if (dev->thread.shutdown || signal_pending(current))
+  if (kthread_should_stop())
    goto done;

  restart:
@@ -618,7 +623,7 @@ static int tvaudio_thread(void *data)
   for (;;) {
    if (tvaudio_sleep(dev,5000))
     goto restart;
-   if (dev->thread.shutdown || signal_pending(current))
+   if (kthread_should_stop())
     break;
    if (UNSET == dev->thread.mode) {
     rx = tvaudio_getstereo(dev,&tvaudio[i]);
@@ -634,7 +639,6 @@ static int tvaudio_thread(void *data)
  }

  done:
- complete_and_exit(&dev->thread.exit, 0);
  return 0;
 }

@@ -782,9 +786,6 @@ static int tvaudio_thread_ddep(void *dat
  struct saa7134_dev *dev = data;
```

```
  u32 value, norms, clock;

- daemonize("%s", dev->name);
- allow_signal(SIGTERM);
-
  clock = saa7134_boards[dev->board].audio_clock;
  if (UNSET != audio_clock_override)
   clock = audio_clock_override;
@@ -796,7 +797,7 @@ static int tvaudio_thread_ddep(void *dat

  for (;;) {
   tvaudio_sleep(dev,-1);
-  if (dev->thread.shutdown || signal_pending(current))
+  if (kthread_should_stop())
    goto done;

  restart:
@@ -876,7 +877,6 @@ static int tvaudio_thread_ddep(void *dat
  }

  done:
- complete_and_exit(&dev->thread.exit, 0);
  return 0;
 }

@@ -986,15 +986,16 @@ int saa7134_tvaudio_init2(struct saa7134
   break;
  }

- dev->thread.pid = -1;
+ dev->thread.task = NULL;
  if (my_thread) {
   /* start tvaudio thread */
   init_waitqueue_head(&dev->thread.wq);
-  init_completion(&dev->thread.exit);
-  dev->thread.pid = kernel_thread(my_thread,dev,0);
-  if (dev->thread.pid < 0)
-   printk(KERN_WARNING "%s: kernel_thread() failed\n",
+  dev->thread.task = kthread_run(my_thread, dev, dev->name);
+  if (IS_ERR(dev->thread.task)) {
+   printk(KERN_WARNING "%s: failed to create kthread\n",
          dev->name);
+   dev->thread.task = NULL;
+  }
   saa7134_tvaudio_do_scan(dev);
  }

@@ -1005,10 +1006,10 @@ int saa7134_tvaudio_init2(struct saa7134
```

```
 int saa7134_tvaudio_fini(struct saa7134_dev *dev)
 {
  /* shutdown tvaudio thread */
- if (dev->thread.pid >= 0) {
-  dev->thread.shutdown = 1;
-  wake_up_interruptible(&dev->thread.wq);
-  wait_for_completion(&dev->thread.exit);
+ if (dev->thread.task) {
+  /* kthread_stop() wakes up the thread */
+  kthread_stop(dev->thread.task);
+  dev->thread.task = NULL;
  }
  saa_andorb(SAA7134_ANALOG_IO_SELECT, 0x07, 0x00); /* LINE1 */
  return 0;
@@ -1020,7 +1021,7 @@ int saa7134_tvaudio_do_scan(struct saa71
   dprintk("sound IF not in use, skipping scan\n");
   dev->automute = 0;
   saa7134_tvaudio_setmute(dev);
- } else if (dev->thread.pid >= 0) {
+ } else if (dev->thread.task) {
   dev->thread.mode = UNSET;
   dev->thread.scan2++;
   wake_up_interruptible(&dev->thread.wq);
```

Subject: Re: [PATCH] saa7134-tvaudio: Convert to kthread API.
Posted by Christoph Hellwig on Fri, 20 Apr 2007 13:05:01 GMT
View Forum Message <> Reply to Message

On Fri, Apr 20, 2007 at 02:48:35PM +0200, Cedric Le Goater wrote:
> Eric W. Biederman wrote:
> > From: Eric W. Biederman <ebiederm@xmission.com> - unquoted
> >
> > It is my goal to replace all kernel code that handles signals
> > from user space, calls kernel_thread or calls daemonize. All
> > of which the kthread_api makes unncessary. Handling signals
> > from user space is a maintenance problem becuase using a
> > kernel thread is an implementation detail and if user space
> > cares it does not allow us to change the implementation. Calling
> > daemonize is a problem because it has to undo a continually changing
> > set of state generated by user space, requiring the implemetation
> > to change continually. kernel_thread is a problem because it
> > returns a pid_t value. Numeric pids are inherently racy and
> > in the presence of a pid namespace they are no longer global

> > making them useless for general use in the kernel.
> >
> > So this patch renames the pid member of struct saa7134_thread
> > started and changes it's type from pid_t to int. All it
> > has ever been used for is to detect if the kernel thread
> > is has been started so this works.
> >
> > allow_signal(SIGTERM) and the calls to signal_pending have
> > been removed they are needed for the driver to operation.
> >
> > The startup of tvaudio_thread and tvaudio_thread_dep have
> > been modified to use kthread_run instead of a combination
> > of kernel_thread and daemonize.
> >
> > The result is code that is slightly simpler and more
> > maintainable.
>
> Here's a refreshed attempt using kthread_should_stop().
> Unfortunately, not tested bc we don't have the hardware.

I have a patch for this one flying around somewhere aswell.
It's trivial to kill the waitqueue and just use wake_up_process,
otherwise it looks pretty similar.

Subject: Re: [PATCH] ia64 sn xpc: Convert to use kthread API.
Posted by Robin Holt on Fri, 20 Apr 2007 14:21:40 GMT
View Forum Message <> Reply to Message

On Fri, Apr 20, 2007 at 08:23:39AM +0200, Jes Sorensen wrote:
> Andrew Morton wrote:
> >Another driver which should be fully converted to the kthread API:
> >kthread_stop() and kthread_should_stop().
> >
> >And according to my logs, this driver was added to the tree more than
> >a year _after_ the kthread interface was made available.
> >
> >This isn't good.
>
> Andrew,
>
> Per my previous response, I'd prefer to have either Russ or Robin ack
> the patch doesn't break before it's pushed to Linus.

>
> I don't know much about the xpmem and I am not comfortable testing it.

I think this was originally coded with daemonize to avoid issues with
reaping children. Dean Nelson can correct me if I am wrong. I assume
this patch is going in as part of the set which will make these threads
clear themselves from the children list and if that is the case, I can
see no issues.

Thanks,
Robin

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: Getting the new RxRPC patches upstream
Posted by akpm on Fri, 20 Apr 2007 18:38:05 GMT
View Forum Message <> Reply to Message

On Fri, 20 Apr 2007 11:41:46 +0100
David Howells <dhowells@redhat.com> wrote:

> There are only two non-net patches that AF_RXRPC depends on:
>
> (1) The key facility changes. That's all my code anyway, and shouldn't be a
>   problem to merge unless someone else has put some changes in there that I
>   don't know about.
>
> (2) try_to_cancel_delayed_work(). I suppose I could use
>   cancel_delayed_work() instead, but that's less efficient as it waits for
>   the timer completion function to finish.

There are significant workqueue changes in -mm and I plan to send them
in for 2.6.22. I doubt if there's anything in there which directly
affects cancel_delayed_work(), but making changes of this nature against
2.6.21 might lead to grief.

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: Getting the new RxRPC patches upstream
Posted by Oleg Nesterov on Fri, 20 Apr 2007 21:28:35 GMT
View Forum Message <> Reply to Message

On 04/20, Andrew Morton wrote:
>
> On Fri, 20 Apr 2007 11:41:46 +0100
> David Howells <dhowells@redhat.com> wrote:
>
> > There are only two non-net patches that AF_RXRPC depends on:
> >
> > (1) The key facility changes. That's all my code anyway, and shouldn't be a
> >   problem to merge unless someone else has put some changes in there that I
> >   don't know about.
> >
> > (2) try_to_cancel_delayed_work(). I suppose I could use
> >   cancel_delayed_work() instead, but that's less efficient as it waits for
> >   the timer completion function to finish.
>
> There are significant workqueue changes in -mm and I plan to send them
> in for 2.6.22. I doubt if there's anything in there which directly
> affects cancel_delayed_work(), but making changes of this nature against
> 2.6.21 might lead to grief.

I think it is better to use cancel_delayed_work(), but change it to use
del_timer(). I belive cancel_delayed_work() doesn't need del_timer_sync().

We only care when del_timer() returns true. In that case, if the timer
function still runs (possible for single-threaded wqs), it has already
passed __queue_work().

Oleg.

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: [PATCH] nfs lockd reclaimer: Convert to kthread API
Posted by ebiederm on Sat, 21 Apr 2007 19:04:16 GMT
View Forum Message <> Reply to Message

Dave Hansen <hansendc@us.ibm.com> writes:

> On Thu, 2007-04-19 at 17:19 -0400, Trond Myklebust wrote:
>> > With pid namespaces all kernel threads will disappear so how do
>> > we cope with the problem when the sysadmin can not see the kernel
>> > threads?
>
> Do they actually always disappear, or do we keep them in the
> init_pid_namespace?

In the init pid namespace but not in any of it's children.

Eric

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: [PATCH] nfs lockd reclaimer: Convert to kthread API
Posted by ebiederm on Sat, 21 Apr 2007 19:47:39 GMT
View Forum Message <> Reply to Message

Trond Myklebust <trond.myklebust@fys.uio.no> writes:

> On Thu, 2007-04-19 at 14:40 -0700, Andrew Morton wrote:
>> Using signals to communicate with kernel threads is fairly unpleasant, IMO.
>> We have much simpler, faster and more idiomatic ways of communicating
>> between threads in-kernel and there are better ways in which userspace can
>> communicate with the kernel - system calls, for example...
>>
>> So I think generally any move which gets us away from using signals in
>> kernel threads is moving in a good direction.
>
> I have yet to see a proposal which did. Eric's patch was eliminating
> signals in kernel threads that used them without proposing any
> replacement mechanism or showing that he had plans to do so. That is a
> good reason for a veto.

Possibly I just hadn't looked close enough. The signals looked like
a redundant mechanism.

>> > > With pid namespaces all kernel threads will disappear so how do
>> > > we cope with the problem when the sysadmin can not see the kernel
>> > > threads?
>> >
>> > Then you have a usability problem. How does the sysadmin reboot the
>> > system if there is no way to shut down the processes that are hanging on
>> > an unresponsive filesystem?
>>
>> Where's the hang? A user process is stuck on h_rwsem?
>>
>> If so, would it be appropriate to convert the user process to use
>> down_foo_interruptible(), so that the operator can just kill the user
>> process as expected, rather than having to futz around killing kernel
>> threads?
>

> If an NFS server reboots, then the locks held by user processes on the
> client need to be re-established by when it comes up again. Otherwise,
> the processes that thought they were holding locks will suddenly fail.
> This recovery job is currently the done by a kernel thread.
>
> The question is then what to do if the server crashes again while the
> kernel thread is re-establishing the locks. Particularly if it never
> comes back again.
> Currently, the administrator can intervene by killing anything that has
> open files on that volume and kill the recovery kernel thread.
> You'll also note that lockd_down(), nfsd_down() etc all use signals to
> inform lockd(), nfsd() etc that they should be shutting down. Since the
> reclaimer thread is started by the lockd() thread using CLONE_SIGHAND,
> this means that we also automatically kill any lingering recovery
> threads whenever we shutdown lockd().

Maybe I'm missing something but I think you are referring to the semantics
of do_group_exit in the presence of CLONE_THREAD. All sharing a
sighand should do is cause the sharing of the signal handler. Causing
allow_signal and disallow_signal to act on a group of threads instead
of a single thread. I don't recall clone_sighand having any
other effects.

> These mechanisms need to be replaced _before_ we start shooting down
> sigallow() etc in the kernel.

Reasonable if these mechanisms are not redundant.

Thinking it through because everything having to do with nfs mounting and
unmounting is behind the privileged mount operation this is not going to
become an issue until we start allowing unprivileged nfs mounts. Because
we cannot delegate control of nfs mount and unmount operations until then.

Since signals do not pose a immediate barrier to forward progress like
daemonize and kernel_thread we can leave things as is until we can
sort this out.

Eric

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: [PATCH] ia64 sn xpc: Convert to use kthread API.
Posted by ebiederm on Sat, 21 Apr 2007 19:53:20 GMT
View Forum Message <> Reply to Message

Robin Holt <holt@sgi.com> writes:

> I think this was originally coded with daemonize to avoid issues with
> reaping children. Dean Nelson can correct me if I am wrong. I assume
> this patch is going in as part of the set which will make these threads
> clear themselves from the children list and if that is the case, I can
> see no issues.

One of my earlier patches guarantees that kthreadd will have pid == 2.

daemonize actually explicitly reparents to init so using daemonize and
kernel_thread provides no help at all with respect to scaling. It in
fact guarantees you will be on init's list of child processes.

The work to enhance wait is a little tricky and it conflicts with the
utrace patches, which makes it hard to pursue at the moment.

I'm actually sorting out kthread stop so I can complete the pid namespace.
But since all kthreads are children of kthreadd this helps in a small
way with the scaling issue.

Eric

Subject: Re: [PATCH] cpci_hotplug: Convert to use the kthread API
Posted by Christoph Hellwig on Sun, 22 Apr 2007 12:05:35 GMT
View Forum Message <> Reply to Message

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.

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

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);
  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);
 }

 int

_____

Subject: Re: [PATCH] ibmphp: Convert to use the kthreads API
Posted by Christoph Hellwig on Sun, 22 Apr 2007 12:09:50 GMT
View Forum Message <> Reply to Message

On Thu, Apr 19, 2007 at 12:55:30AM -0600, Eric W. Biederman wrote:
> From: Eric W. Biederman <ebiederm@xmission.com> - unquoted
>
> kthread_run replaces kernel_thread and dameonize.
>
> allow_signal is unnecessary and has been removed.
> tid_poll was unused and has been removed.

Thread handling in this driver is quite interesting. Greg has his name
in there, so we can blame everything on him ;-)

Below is my take at cleaning at least the thread-related bits up:

 - full switch to kthread infrastructure
 - switch semOperations to a mutex, and give it a proper name
 - remove the useless hpc_poll_thread wrapper
 - remove ibmphp_hpc_initvars - everything left can easily be
   initialized statically

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

Index: linux-2.6/drivers/pci/hotplug/ibmphp.h
===
--- linux-2.6.orig/drivers/pci/hotplug/ibmphp.h 2007-04-22 12:44:04.000000000 +0200
+++ linux-2.6/drivers/pci/hotplug/ibmphp.h 2007-04-22 12:44:07.000000000 +0200
@@ -392,7 +392,6 @@ extern int ibmphp_add_pfmem_from_mem (st
 extern struct bus_node *ibmphp_find_res_bus (u8);

```
 extern void ibmphp_print_test (void); /* for debugging purposes */

-extern void ibmphp_hpc_initvars (void);
 extern int ibmphp_hpc_readslot (struct slot *, u8, u8 *);
 extern int ibmphp_hpc_writeslot (struct slot *, u8);
 extern void ibmphp_lock_operations (void);
Index: linux-2.6/drivers/pci/hotplug/ibmphp_core.c
===================================================================
--- linux-2.6.orig/drivers/pci/hotplug/ibmphp_core.c 2007-04-22 12:44:11.000000000 +0200
+++ linux-2.6/drivers/pci/hotplug/ibmphp_core.c 2007-04-22 12:44:16.000000000 +0200
@@ -1368,8 +1368,6 @@ static int __init ibmphp_init(void)

  ibmphp_debug = debug;

- ibmphp_hpc_initvars();
-
  for (i = 0; i < 16; i++)
   irqs[i] = 0;

Index: linux-2.6/drivers/pci/hotplug/ibmphp_hpc.c
===================================================================
--- linux-2.6.orig/drivers/pci/hotplug/ibmphp_hpc.c 2007-04-22 12:31:23.000000000 +0200
+++ linux-2.6/drivers/pci/hotplug/ibmphp_hpc.c 2007-04-22 12:45:51.000000000 +0200
@@ -35,6 +35,7 @@
 #include <linux/smp_lock.h>
 #include <linux/init.h>
 #include <linux/mutex.h>
+#include <linux/kthread.h>

 #include "ibmphp.h"

@@ -101,12 +102,10 @@ static int to_debug = 0;
 //--------------------------------------------------------------------
 // global variables
 //--------------------------------------------------------------------
-static int ibmphp_shutdown;
-static int tid_poll;
-static struct mutex sem_hpcaccess; // lock access to HPC
-static struct semaphore semOperations; // lock all operations and
+static struct task_struct *ibmphp_poll_thread;
+static DEFINE_MUTEX(sem_hpcaccess); // lock access to HPC
+static DEFINE_MUTEX(ibmphp_op_sem); // lock all operations and
      // access to data structures
-static struct semaphore sem_exit; // make sure polling thread goes away
 //--------------------------------------------------------------------
 // local function prototypes
 //--------------------------------------------------------------------
@@ -116,33 +115,11 @@ static u8 hpc_writecmdtoindex (u8, u8);
```

```
 static u8 hpc_readcmdtoindex (u8, u8);
 static void get_hpc_access (void);
 static void free_hpc_access (void);
-static void poll_hpc (void);
 static int process_changeinstatus (struct slot *, struct slot *);
 static int process_changeinlatch (u8, u8, struct controller *);
-static int hpc_poll_thread (void *);
 static int hpc_wait_ctlr_notworking (int, struct controller *, void __iomem *, u8 *);
 //------------------------------------------------------------------------

-
-/*------------------------------------------------------------------
-* Name:    ibmphp_hpc_initvars
-*
-* Action:  initialize semaphores and variables
-*------------------------------------------------------------------*/
-void __init ibmphp_hpc_initvars (void)
-{
- debug ("%s - Entry\n", __FUNCTION__);
-
- mutex_init(&sem_hpcaccess);
- init_MUTEX (&semOperations);
- init_MUTEX_LOCKED (&sem_exit);
- to_debug = 0;
- ibmphp_shutdown = 0;
- tid_poll = 0;
-
- debug ("%s - Exit\n", __FUNCTION__);
-}
-
 /*------------------------------------------------------------------
 * Name:    i2c_ctrl_read
 *
@@ -798,7 +775,7 @@ void free_hpc_access (void)
 *------------------------------------------------------------------*/
 void ibmphp_lock_operations (void)
 {
- down (&semOperations);
+ mutex_lock(&ibmphp_op_sem);
  to_debug = 1;
 }

@@ -808,7 +785,7 @@ void ibmphp_lock_operations (void)
 void ibmphp_unlock_operations (void)
 {
  debug ("%s - Entry\n", __FUNCTION__);
- up (&semOperations);
+ mutex_unlock(&ibmphp_op_sem);
```

```
  to_debug = 0;
  debug ("%s - Exit\n", __FUNCTION__);
 }
@@ -819,7 +796,7 @@ void ibmphp_unlock_operations (void)
 #define POLL_LATCH_REGISTER 0
 #define POLL_SLOTS  1
 #define POLL_SLEEP  2
-static void poll_hpc (void)
+static int poll_hpc(void *data)
 {
  struct slot myslot;
  struct slot *pslot = NULL;
@@ -833,12 +810,9 @@ static void poll_hpc (void)

  debug ("%s - Entry\n", __FUNCTION__);

- while (!ibmphp_shutdown) {
-  if (ibmphp_shutdown)
-   break;
-
+ while (!kthread_should_stop()) {
   /* try to get the lock to do some kind of hardware access */
-  down (&semOperations);
+  mutex_lock(&ibmphp_op_sem);

   switch (poll_state) {
   case POLL_LATCH_REGISTER:
@@ -893,14 +867,13 @@ static void poll_hpc (void)
    break;
   case POLL_SLEEP:
    /* don't sleep with a lock on the hardware */
-   up (&semOperations);
+   mutex_unlock(&ibmphp_op_sem);
    msleep(POLL_INTERVAL_SEC * 1000);

-   if (ibmphp_shutdown)
+   if (kthread_should_stop())
     break;

-   down (&semOperations);
-
+   mutex_lock(&ibmphp_op_sem);
    if (poll_count >= POLL_LATCH_CNT) {
     poll_count = 0;
     poll_state = POLL_SLOTS;
@@ -909,12 +882,13 @@ static void poll_hpc (void)
    break;
   }
```

```
   /* give up the hardware semaphore */
-  up (&semOperations);
+  mutex_unlock(&ibmphp_op_sem);
   /* sleep for a short time just for good measure */
   msleep(100);
  }
- up (&sem_exit);
  debug ("%s - Exit\n", __FUNCTION__);
+
+ return 0;
 }


@@ -1050,47 +1024,19 @@ static int process_changeinlatch (u8 old
 }

 /*---------------------------------------------------------------------
-* Name:    hpc_poll_thread
-*
-* Action:  polling
-*
-* Return   0
-* Value:
-*---------------------------------------------------------------------*/
-static int hpc_poll_thread (void *data)
-{
- debug ("%s - Entry\n", __FUNCTION__);
-
- daemonize("hpc_poll");
- allow_signal(SIGKILL);
-
- poll_hpc ();
-
- tid_poll = 0;
- debug ("%s - Exit\n", __FUNCTION__);
- return 0;
-}
-
-
-/*---------------------------------------------------------------------
 * Name:    ibmphp_hpc_start_poll_thread
 *
 * Action:  start polling thread
 *---------------------------------------------------------------------*/
 int __init ibmphp_hpc_start_poll_thread (void)
 {
- int rc = 0;
-
```

```
- debug ("%s - Entry\n", __FUNCTION__);
-
- tid_poll = kernel_thread (hpc_poll_thread, NULL, 0);
- if (tid_poll < 0) {
+ ibmphp_poll_thread = kthread_run(poll_hpc, NULL, "hpc_poll");
+ if (IS_ERR(ibmphp_poll_thread)) {
   err ("%s - Error, thread not started\n", __FUNCTION__);
-  rc = -1;
+  return PTR_ERR(ibmphp_poll_thread);
  }

- debug ("%s - Exit tid_poll[%d] rc[%d]\n", __FUNCTION__, tid_poll, rc);
- return rc;
+ return 0;
 }

 /*---------------------------------------------------------------------
@@ -1100,28 +1046,11 @@ int __init ibmphp_hpc_start_poll_thread
 *---------------------------------------------------------------------*/
 void __exit ibmphp_hpc_stop_poll_thread (void)
 {
- debug ("%s - Entry\n", __FUNCTION__);
+ kthread_stop(ibmphp_poll_thread);

- ibmphp_shutdown = 1;
- debug ("before locking operations \n");
  ibmphp_lock_operations ();
- debug ("after locking operations \n");
-
- // wait for poll thread to exit
- debug ("before sem_exit down \n");
- down (&sem_exit);
- debug ("after sem_exit down \n");
-
- // cleanup
- debug ("before free_hpc_access \n");
  free_hpc_access ();
- debug ("after free_hpc_access \n");
  ibmphp_unlock_operations ();
- debug ("after unlock operations \n");
- up (&sem_exit);
- debug ("after sem exit up\n");
-
- debug ("%s - Exit\n", __FUNCTION__);
 }

 /*---------------------------------------------------------------------
```

Subject: Re: [PATCH] cpqphp: Convert to use the kthread API
Posted by Christoph Hellwig on Sun, 22 Apr 2007 12:12:08 GMT
View Forum Message <> Reply to Message

On Thu, Apr 19, 2007 at 12:55:31AM -0600, Eric W. Biederman wrote:
> From: Eric W. Biederman <ebiederm@xmission.com> - unquoted
>
> This patch changes cpqphp to use kthread_run and not
> kernel_thread and daemonize to startup and setup
> the cpqphp thread.

Thread handling in this driver (and actually everything else) seems
to be written by a crackmonkey.

Here's my take at fixing everything slightly related to thread handling
up:

 - full switch to kthread infrastructure
 - remove unused semaphore as mutex and waitqueue in long_delay -
   in fact that whole function should just go away as the user would
   be a lot more happy with just msleep_interruptible.
 - use wake_up_process for waking the thread

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

Index: linux-2.6/drivers/pci/hotplug/cpqphp_ctrl.c
===
--- linux-2.6.orig/drivers/pci/hotplug/cpqphp_ctrl.c 2007-04-22 12:46:33.000000000 +0200
+++ linux-2.6/drivers/pci/hotplug/cpqphp_ctrl.c 2007-04-22 12:53:58.000000000 +0200
@@ -37,6 +37,7 @@
 #include <linux/smp_lock.h>
 #include <linux/pci.h>
 #include <linux/pci_hotplug.h>
+#include <linux/kthread.h>
 #include "cpqphp.h"

 static u32 configure_new_device(struct controller* ctrl, struct pci_func *func,
@@ -45,34 +46,20 @@ static int configure_new_function(struct
    u8 behind_bridge, struct resource_lists *resources);
 static void interrupt_event_handler(struct controller *ctrl);

-static struct semaphore event_semaphore; /* mutex for process loop (up if something to process)

```
*/
-static struct semaphore event_exit;  /* guard ensure thread has exited before calling it quits */
-static int event_finished;
-static unsigned long pushbutton_pending; /* = 0 */

-/* things needed for the long_delay function */
-static struct semaphore  delay_sem;
-static wait_queue_head_t delay_wait;
+static struct task_struct *cpqhp_event_thread;
+static unsigned long pushbutton_pending; /* = 0 */

 /* delay is in jiffies to wait for */
 static void long_delay(int delay)
 {
- DECLARE_WAITQUEUE(wait, current);
-
- /* only allow 1 customer into the delay queue at once
-  * yes this makes some people wait even longer, but who really cares?
-  * this is for _huge_ delays to make the hardware happy as the
-  * signals bounce around
+ /*
+  * XXX(hch): if someone is bored please convert all callers
+  * to call msleep_interruptible directly.  They really want
+  * to specify timeouts in natural units and spend a lot of
+  * effort converting them to jiffies..
   */
- down (&delay_sem);
-
- init_waitqueue_head(&delay_wait);
-
- add_wait_queue(&delay_wait, &wait);
  msleep_interruptible(jiffies_to_msecs(delay));
- remove_wait_queue(&delay_wait, &wait);
-
- up(&delay_sem);
 }


@@ -955,8 +942,8 @@ irqreturn_t cpqhp_ctrl_intr(int IRQ, voi
  }

  if (schedule_flag) {
-  up(&event_semaphore);
-  dbg("Signal event_semaphore\n");
+  wake_up_process(cpqhp_event_thread);
+  dbg("Waking even thread");
  }
  return IRQ_HANDLED;
```

```
 }
@@ -1738,7 +1725,7 @@ static u32 remove_board(struct pci_func
 static void pushbutton_helper_thread(unsigned long data)
 {
  pushbutton_pending = data;
- up(&event_semaphore);
+ wake_up_process(cpqhp_event_thread);
 }


@@ -1746,16 +1733,14 @@ static void pushbutton_helper_thread(uns
 static int event_thread(void* data)
 {
  struct controller *ctrl;
- lock_kernel();
- daemonize("phpd_event");
-
- unlock_kernel();

  while (1) {
   dbg("!!!!event_thread sleeping\n");
-  down_interruptible (&event_semaphore);
-  dbg("event_thread woken finished = %d\n", event_finished);
-  if (event_finished) break;
+  set_current_state(TASK_INTERRUPTIBLE);
+  schedule();
+
+  if (kthread_should_stop())
+   break;
   /* Do stuff here */
   if (pushbutton_pending)
    cpqhp_pushbutton_thread(pushbutton_pending);
@@ -1764,38 +1749,24 @@ static int event_thread(void* data)
    interrupt_event_handler(ctrl);
  }
  dbg("event_thread signals exit\n");
- up(&event_exit);
  return 0;
 }

-
 int cpqhp_event_start_thread(void)
 {
- int pid;
-
- /* initialize our semaphores */
- init_MUTEX(&delay_sem);
- init_MUTEX_LOCKED(&event_semaphore);
```

```
- init_MUTEX_LOCKED(&event_exit);
- event_finished=0;
-
- pid = kernel_thread(event_thread, NULL, 0);
- if (pid < 0) {
+ cpqhp_event_thread = kthread_run(event_thread, NULL, "phpd_event");
+ if (IS_ERR(cpqhp_event_thread)) {
   err ("Can't start up our event thread\n");
-  return -1;
+  return PTR_ERR(cpqhp_event_thread);
  }
- dbg("Our event thread pid = %d\n", pid);
+
  return 0;
 }


 void cpqhp_event_stop_thread(void)
 {
- event_finished = 1;
- dbg("event_thread finish command given\n");
- up(&event_semaphore);
- dbg("wait for event_thread to exit\n");
- down(&event_exit);
+ kthread_stop(cpqhp_event_thread);
 }
```

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: Remaining straight forward kthread API conversions...
Posted by Christoph Hellwig on Sun, 22 Apr 2007 12:15:24 GMT
View Forum Message <> Reply to Message

Looks like you were missing at least the pcie hotplug driver. Another
one of the horrible thread abuses in drivers/pci/hotpug.

 - full conversion to kthread infrastructure
 - use wake_up_process to wake the thread up

Like most pci hotplug drivers it still uses very race non-atomic variable
assignment to communicated with the thread, but that's something the
maintainers should look into.

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

Index: linux-2.6/drivers/pci/hotplug/pciehp_ctrl.c
===
--- linux-2.6.orig/drivers/pci/hotplug/pciehp_ctrl.c 2007-04-22 11:36:58.000000000 +0200
+++ linux-2.6/drivers/pci/hotplug/pciehp_ctrl.c 2007-04-22 11:42:56.000000000 +0200
@@ -32,14 +32,13 @@
 #include <linux/types.h>
 #include <linux/smp_lock.h>
 #include <linux/pci.h>
+#include <linux/kthread.h>
 #include "../pci.h"
 #include "pciehp.h"

 static void interrupt_event_handler(struct controller *ctrl);

-static struct semaphore event_semaphore; /* mutex for process loop (up if something to process)
*/
-static struct semaphore event_exit; /* guard ensure thread has exited before calling it quits */
-static int event_finished;
+static struct task_struct *pciehpd_event_thread;
 static unsigned long pushbutton_pending; /* = 0 */
 static unsigned long surprise_rm_pending; /* = 0 */

@@ -93,8 +92,9 @@ u8 pciehp_handle_attention_button(u8 hp_
   info("Button ignore on Slot(%s)\n", slot_name(p_slot));
  }

+ /* signal event thread that new event is posted */
  if (rc)
- up(&event_semaphore); /* signal event thread that new event is posted */
+ wake_up_process(pciehpd_event_thread);

  return 0;

@@ -135,8 +135,9 @@ u8 pciehp_handle_switch_change(u8 hp_slo
   taskInfo->event_type = INT_SWITCH_CLOSE;
  }

+ /* signal event thread that new event is posted */
  if (rc)
- up(&event_semaphore); /* signal event thread that new event is posted */
+ wake_up_process(pciehpd_event_thread);

  return rc;
 }
@@ -178,8 +179,9 @@ u8 pciehp_handle_presence_change(u8 hp_s

```
   taskInfo->event_type = INT_PRESENCE_OFF;
  }

+ /* signal event thread that new event is posted */
  if (rc)
-  up(&event_semaphore); /* signal event thread that new event is posted */
+   wake_up_process(pciehpd_event_thread);

  return rc;
 }
@@ -217,8 +219,10 @@ u8 pciehp_handle_power_fault(u8 hp_slot,
   taskInfo->event_type = INT_POWER_FAULT;
   info("power fault bit %x set\n", hp_slot);
  }
+
+ /* signal event thread that new event is posted */
  if (rc)
-  up(&event_semaphore); /* signal event thread that new event is posted */
+   wake_up_process(pciehpd_event_thread);

  return rc;
 }
@@ -362,7 +366,7 @@ static void pushbutton_helper_thread(uns
 {
  pushbutton_pending = data;

- up(&event_semaphore);
+ wake_up_process(pciehpd_event_thread);
 }

 /**
@@ -452,19 +456,14 @@ static void pciehp_surprise_rm_thread(un


 /* this is the main worker thread */
-static int event_thread(void* data)
+static int event_thread(void *data)
 {
  struct controller *ctrl;
- lock_kernel();
- daemonize("pciehpd_event");
-
- unlock_kernel();

  while (1) {
-  dbg("!!!!event_thread sleeping\n");
-  down_interruptible (&event_semaphore);
-  dbg("event_thread woken finished = %d\n", event_finished);
```

```
-  if (event_finished || signal_pending(current))
+  set_current_state(TASK_INTERRUPTIBLE);
+  schedule();
+  if (kthread_should_stop())
     break;
   /* Do stuff here */
   if (pushbutton_pending)
@@ -476,24 +475,15 @@ static int event_thread(void* data)
     interrupt_event_handler(ctrl);
  }
  dbg("event_thread signals exit\n");
- up(&event_exit);
  return 0;
 }

 int pciehp_event_start_thread(void)
 {
- int pid;
-
- /* initialize our semaphores */
- init_MUTEX_LOCKED(&event_exit);
- event_finished=0;
-
- init_MUTEX_LOCKED(&event_semaphore);
- pid = kernel_thread(event_thread, NULL, 0);
-
- if (pid < 0) {
+ pciehpd_event_thread = kthread_run(event_thread, NULL, "pciehpd_event");
+ if (IS_ERR(pciehpd_event_thread)) {
   err ("Can't start up our event thread\n");
-  return -1;
+  return PTR_ERR(pciehpd_event_thread);
  }
  return 0;
 }
@@ -501,9 +491,7 @@ int pciehp_event_start_thread(void)

 void pciehp_event_stop_thread(void)
 {
- event_finished = 1;
- up(&event_semaphore);
- down(&event_exit);
+ kthread_stop(pciehpd_event_thread);
 }


@@ -624,7 +612,7 @@ static void interrupt_event_handler(stru
     dbg("Surprise Removal\n");
```

```
       if (p_slot) {
        surprise_rm_pending = (unsigned long) p_slot;
-       up(&event_semaphore);
+        wake_up_process(pciehpd_event_thread);
        update_slot_info(p_slot);
       }
      }
```

Subject: Re: [PATCH] mtd_blkdevs: Convert to use the kthread API
Posted by Christoph Hellwig on Sun, 22 Apr 2007 12:24:53 GMT
View Forum Message <> Reply to Message

On Thu, Apr 19, 2007 at 12:55:28AM -0600, Eric W. Biederman wrote:
> From: Eric W. Biederman <ebiederm@xmission.com> - unquoted
>
> thread_run is used intead of kernel_thread, daemonize, and mucking
> around blocking signals directly.

This is the full conversion I sent to Dave in April 2006, but never got
any feedback to:

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

```
Index: linux-2.6/drivers/mtd/mtd_blkdevs.c
===================================================================
--- linux-2.6.orig/drivers/mtd/mtd_blkdevs.c 2007-01-29 10:03:52.000000000 +0100
+++ linux-2.6/drivers/mtd/mtd_blkdevs.c 2007-04-22 13:22:03.000000000 +0200
@@ -20,6 +20,7 @@
 #include <linux/hdreg.h>
 #include <linux/init.h>
 #include <linux/mutex.h>
+#include <linux/kthread.h>
 #include <asm/uaccess.h>

 static LIST_HEAD(blktrans_majors);
@@ -28,9 +29,7 @@ extern struct mutex mtd_table_mutex;
 extern struct mtd_info *mtd_table[];

 struct mtd_blkcore_priv {
- struct completion thread_dead;
- int exiting;
- wait_queue_head_t thread_wq;
```

```
+ struct task_struct *thread;
  struct request_queue *rq;
  spinlock_t queue_lock;
 };
@@ -83,38 +82,19 @@ static int mtd_blktrans_thread(void *arg
  /* we might get involved when memory gets low, so use PF_MEMALLOC */
  current->flags |= PF_MEMALLOC | PF_NOFREEZE;

- daemonize("%sd", tr->name);
-
- /* daemonize() doesn't do this for us since some kernel threads
-    actually want to deal with signals. We can't just call
-    exit_sighand() since that'll cause an oops when we finally
-    do exit. */
- spin_lock_irq(&current->sighand->siglock);
- sigfillset(&current->blocked);
- recalc_sigpending();
- spin_unlock_irq(&current->sighand->siglock);
-
  spin_lock_irq(rq->queue_lock);
-
- while (!tr->blkcore_priv->exiting) {
+ while (!kthread_should_stop()) {
   struct request *req;
   struct mtd_blktrans_dev *dev;
   int res = 0;
-  DECLARE_WAITQUEUE(wait, current);

   req = elv_next_request(rq);

   if (!req) {
-   add_wait_queue(&tr->blkcore_priv->thread_wq, &wait);
-   set_current_state(TASK_INTERRUPTIBLE);
-
    spin_unlock_irq(rq->queue_lock);
-
+   set_current_state(TASK_INTERRUPTIBLE);
    schedule();
-   remove_wait_queue(&tr->blkcore_priv->thread_wq, &wait);
-
    spin_lock_irq(rq->queue_lock);
-
    continue;
   }

@@ -133,13 +113,13 @@ static int mtd_blktrans_thread(void *arg
  }
  spin_unlock_irq(rq->queue_lock);
```

```
- complete_and_exit(&tr->blkcore_priv->thread_dead, 0);
+ return 0;
 }

 static void mtd_blktrans_request(struct request_queue *rq)
 {
  struct mtd_blktrans_ops *tr = rq->queuedata;
- wake_up(&tr->blkcore_priv->thread_wq);
+ wake_up_process(tr->blkcore_priv->thread);
 }


@@ -388,8 +368,6 @@ int register_mtd_blktrans(struct mtd_blk
   return ret;
  }
  spin_lock_init(&tr->blkcore_priv->queue_lock);
- init_completion(&tr->blkcore_priv->thread_dead);
- init_waitqueue_head(&tr->blkcore_priv->thread_wq);

  tr->blkcore_priv->rq = blk_init_queue(mtd_blktrans_request, &tr->blkcore_priv->queue_lock);
  if (!tr->blkcore_priv->rq) {
@@ -403,13 +381,14 @@ int register_mtd_blktrans(struct mtd_blk
  blk_queue_hardsect_size(tr->blkcore_priv->rq, tr->blksize);
  tr->blkshift = ffs(tr->blksize) - 1;

- ret = kernel_thread(mtd_blktrans_thread, tr, CLONE_KERNEL);
- if (ret < 0) {
+ tr->blkcore_priv->thread = kthread_run(mtd_blktrans_thread, tr,
+   "%sd", tr->name);
+ if (IS_ERR(tr->blkcore_priv->thread)) {
   blk_cleanup_queue(tr->blkcore_priv->rq);
   unregister_blkdev(tr->major, tr->name);
   kfree(tr->blkcore_priv);
   mutex_unlock(&mtd_table_mutex);
-  return ret;
+  return PTR_ERR(tr->blkcore_priv->thread);
  }

  INIT_LIST_HEAD(&tr->devs);
@@ -432,9 +411,7 @@ int deregister_mtd_blktrans(struct mtd_b
  mutex_lock(&mtd_table_mutex);

  /* Clean up the kernel thread */
- tr->blkcore_priv->exiting = 1;
- wake_up(&tr->blkcore_priv->thread_wq);
- wait_for_completion(&tr->blkcore_priv->thread_dead);
+ kthread_stop(tr->blkcore_priv->thread);
```

```
 /* Remove it from the list of active majors */
 list_del(&tr->list);
```


Subject: Re: [PATCH] powerpc pseries eeh: Convert to kthread API
Posted by Christoph Hellwig on Sun, 22 Apr 2007 12:31:55 GMT
View Forum Message <> Reply to Message

On Thu, Apr 19, 2007 at 01:58:45AM -0600, Eric W. Biederman wrote:
> From: Eric W. Biederman <ebiederm@xmission.com>
>
> This patch modifies the startup of eehd to use kthread_run
> not a combination of kernel_thread and daemonize. Making
> the code slightly simpler and more maintainable.

This one has the same scheme as the various s390 drivers where a thread
is spawned using a workqueue on demand. I think we should not blindly
convert it but think a litte more about it.

The first question is obviously, is this really something we want?
spawning kernel thread on demand without reaping them properly seems
quite dangerous.

The second question is whether this is the right implementation.
kthread_create already works by using a workqueue to create the thread
and then waits for it. If we really want to support creating threads
asynchronously on demand we should have a proper API in kthread.c for
this instead of spreading workqueues.

Subject: Re: [PATCH] powerpc pseries rtasd: Convert to kthread API.
Posted by Christoph Hellwig on Sun, 22 Apr 2007 12:34:56 GMT
View Forum Message <> Reply to Message

On Thu, Apr 19, 2007 at 01:58:46AM -0600, Eric W. Biederman wrote:
> From: Eric W. Biederman <ebiederm@xmission.com>
>

> This patch modifies the startup of rtasd to use kthread_run instaed of
> a combination of kernel_thread and daemonize. Making the code a little
> simpler and more maintainble.

Looks okay, but I have some questions about the original code.

Why does the driver only check if it really needs to run in the
thread and calls vmalloc from it? If we did all these in the
initialization function we could actually properly unwind.

Subject: Re: [PATCH] mtd_blkdevs: Convert to use the kthread API
Posted by David Woodhouse on Sun, 22 Apr 2007 13:23:00 GMT
View Forum Message <> Reply to Message

On Sun, 2007-04-22 at 13:24 +0100, Christoph Hellwig wrote:
> This is the full conversion I sent to Dave in April 2006, but never
> got any feedback to:

Sorry about that; I need prodding sometimes. I'll provide some now...

Can you show me why the thread won't now miss a wakeup if it goes to
sleep just as a new request is added to its queue?

Having already applied Eric's patch, this is the delta to yours...

```
diff --git a/drivers/mtd/mtd_blkdevs.c b/drivers/mtd/mtd_blkdevs.c
index 1aa018a..d065dba 100644
--- a/drivers/mtd/mtd_blkdevs.c
+++ b/drivers/mtd/mtd_blkdevs.c
@@ -29,9 +29,7 @@ extern struct mutex mtd_table_mutex;
 extern struct mtd_info *mtd_table[];

 struct mtd_blkcore_priv {
- struct completion thread_dead;
- int exiting;
- wait_queue_head_t thread_wq;
+ struct task_struct *thread;
   struct request_queue *rq;
   spinlock_t queue_lock;
 };
@@ -85,26 +83,18 @@ static int mtd_blktrans_thread(void *arg)
   current->flags |= PF_MEMALLOC | PF_NOFREEZE;
```

```
  spin_lock_irq(rq->queue_lock);
-
- while (!tr->blkcore_priv->exiting) {
+ while (!kthread_should_stop()) {
    struct request *req;
    struct mtd_blktrans_dev *dev;
    int res = 0;
-   DECLARE_WAITQUEUE(wait, current);

    req = elv_next_request(rq);

    if (!req) {
-    add_wait_queue(&tr->blkcore_priv->thread_wq, &wait);
-    set_current_state(TASK_INTERRUPTIBLE);
-
     spin_unlock_irq(rq->queue_lock);
-
+    set_current_state(TASK_INTERRUPTIBLE);
     schedule();
-    remove_wait_queue(&tr->blkcore_priv->thread_wq, &wait);
-
     spin_lock_irq(rq->queue_lock);
-
     continue;
    }

@@ -123,13 +113,13 @@ static int mtd_blktrans_thread(void *arg)
   }
   spin_unlock_irq(rq->queue_lock);

- complete_and_exit(&tr->blkcore_priv->thread_dead, 0);
+ return 0;
 }

 static void mtd_blktrans_request(struct request_queue *rq)
 {
   struct mtd_blktrans_ops *tr = rq->queuedata;
- wake_up(&tr->blkcore_priv->thread_wq);
+ wake_up_process(tr->blkcore_priv->thread);
 }


@@ -355,7 +345,6 @@ static struct mtd_notifier blktrans_notifier = {

 int register_mtd_blktrans(struct mtd_blktrans_ops *tr)
 {
- struct task_struct *task;
```

```
  int ret, i;

  /* Register the notifier if/when the first device type is
@@ -379,8 +368,6 @@ int register_mtd_blktrans(struct mtd_blktrans_ops *tr)
   return ret;
  }
  spin_lock_init(&tr->blkcore_priv->queue_lock);
- init_completion(&tr->blkcore_priv->thread_dead);
- init_waitqueue_head(&tr->blkcore_priv->thread_wq);

  tr->blkcore_priv->rq = blk_init_queue(mtd_blktrans_request, &tr->blkcore_priv->queue_lock);
  if (!tr->blkcore_priv->rq) {
@@ -394,13 +381,14 @@ int register_mtd_blktrans(struct mtd_blktrans_ops *tr)
  blk_queue_hardsect_size(tr->blkcore_priv->rq, tr->blksize);
  tr->blkshift = ffs(tr->blksize) - 1;

- task = kthread_run(mtd_blktrans_thread, tr, "%sd", tr->name);
- if (IS_ERR(task)) {
+ tr->blkcore_priv->thread = kthread_run(mtd_blktrans_thread, tr,
+   "%sd", tr->name);
+ if (IS_ERR(tr->blkcore_priv->thread)) {
   blk_cleanup_queue(tr->blkcore_priv->rq);
   unregister_blkdev(tr->major, tr->name);
   kfree(tr->blkcore_priv);
   mutex_unlock(&mtd_table_mutex);
-  return PTR_ERR(task);
+  return PTR_ERR(tr->blkcore_priv->thread);
  }

  INIT_LIST_HEAD(&tr->devs);
@@ -423,9 +411,7 @@ int deregister_mtd_blktrans(struct mtd_blktrans_ops *tr)
  mutex_lock(&mtd_table_mutex);

  /* Clean up the kernel thread */
- tr->blkcore_priv->exiting = 1;
- wake_up(&tr->blkcore_priv->thread_wq);
- wait_for_completion(&tr->blkcore_priv->thread_dead);
+ kthread_stop(tr->blkcore_priv->thread);

  /* Remove it from the list of active majors */
  list_del(&tr->list);
```

--
dwmw2

_____
Containers mailing list

Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

Subject: Re: [PATCH] macintosh/therm_pm72.c: Convert to kthread API.
Posted by Christoph Hellwig on Sun, 22 Apr 2007 19:16:47 GMT
View Forum Message <> Reply to Message

On Thu, Apr 19, 2007 at 01:58:47AM -0600, Eric W. Biederman wrote:
> From: Eric W. Biederman <ebiederm@xmission.com>
>
> This patch modifies startup of the kfand to use kthread_run
> not a combination of kernel_thread and daemonize, making
> the code a little simpler and more maintaintable.

Why is this driver using a thread at all? It's only doing a bunch
of rather short-lived things in the thread.

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

Subject: Re: [PATCH] mtd_blkdevs: Convert to use the kthread API
Posted by Christoph Hellwig on Sun, 22 Apr 2007 19:26:46 GMT
View Forum Message <> Reply to Message

On Sun, Apr 22, 2007 at 02:23:00PM +0100, David Woodhouse wrote:
> On Sun, 2007-04-22 at 13:24 +0100, Christoph Hellwig wrote:
> > This is the full conversion I sent to Dave in April 2006, but never
> > got any feedback to:
>
> Sorry about that; I need prodding sometimes. I'll provide some now...
>
> Can you show me why the thread won't now miss a wakeup if it goes to
> sleep just as a new request is added to its queue?

Exactly the same thing that happened before. If you look at
wake_up_process it's just a tiny wrapper around try_to_wake_up.

And wake_up expands to __wake_up expaneds to __wake_up_common
which just walks the list of threads attached to the waitqueue
and then calls curr->func, which expands to try_to_wake_up.

So when your thread still is in running state nothing changes.
If your thread is not in running state it'll get woken by both

variants.

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: [PATCH] i386 voyager: Convert the monitor thread to use the kthread API
Posted by Christoph Hellwig on Sun, 22 Apr 2007 19:30:43 GMT
View Forum Message <> Reply to Message

On Thu, Apr 19, 2007 at 12:55:27AM -0600, Eric W. Biederman wrote:
> From: Eric W. Biederman <ebiederm@xmission.com> - unquoted
>
> This patch just trivially replaces kernel_thread and daemonize
> with a single call to kthread_run.

Here's a better patch that does the full kthread conversion +
switch to wake_up_process. Only compile tested of course due to
lack of voyager hardware.

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

Index: linux-2.6/arch/i386/mach-voyager/voyager_cat.c
===
--- linux-2.6.orig/arch/i386/mach-voyager/voyager_cat.c 2007-04-22 15:19:28.000000000 +0200
+++ linux-2.6/arch/i386/mach-voyager/voyager_cat.c 2007-04-22 15:27:03.000000000 +0200
@@ -1111,7 +1111,7 @@ voyager_cat_do_common_interrupt(void)
     printk(KERN_ERR "Voyager front panel switch turned off\n");
     voyager_status.switch_off = 1;
     voyager_status.request_from_kernel = 1;
-    up(&kvoyagerd_sem);
+   wake_up_process(voyager_thread);
    }
    /* Tell the hardware we're taking care of the
     * shutdown, otherwise it will power the box off
@@ -1157,7 +1157,7 @@ voyager_cat_do_common_interrupt(void)
    outb(VOYAGER_CAT_END, CAT_CMD);
    voyager_status.power_fail = 1;
    voyager_status.request_from_kernel = 1;
-   up(&kvoyagerd_sem);
+   wake_up_process(voyager_thread);
   }

```
Index: linux-2.6/arch/i386/mach-voyager/voyager_thread.c
===================================================================
--- linux-2.6.orig/arch/i386/mach-voyager/voyager_thread.c 2007-04-22 15:15:24.000000000
+0200
+++ linux-2.6/arch/i386/mach-voyager/voyager_thread.c 2007-04-22 15:25:51.000000000 +0200
@@ -24,33 +24,16 @@
 #include <linux/kmod.h>
 #include <linux/completion.h>
 #include <linux/sched.h>
+#include <linux/kthread.h>
 #include <asm/desc.h>
 #include <asm/voyager.h>
 #include <asm/vic.h>
 #include <asm/mtrr.h>
 #include <asm/msr.h>

-#define THREAD_NAME "kvoyagerd"

-/* external variables */
-int kvoyagerd_running = 0;
-DECLARE_MUTEX_LOCKED(kvoyagerd_sem);
-
-static int thread(void *);
-
-static __u8 set_timeout = 0;
-
-/* Start the machine monitor thread.  Return 1 if OK, 0 if fail */
-static int __init
-voyager_thread_start(void)
-{
- if(kernel_thread(thread, NULL, CLONE_KERNEL) < 0) {
-  /* This is serious, but not fatal */
-  printk(KERN_ERR "Voyager: Failed to create system monitor thread!!!\n");
-  return 1;
- }
- return 0;
-}
+struct task_struct *voyager_thread;
+static __u8 set_timeout;

 static int
 execute(const char *string)
@@ -110,31 +93,15 @@ check_continuing_condition(void)
  }
 }

-static void
-wakeup(unsigned long unused)
```

```
-{
- up(&kvoyagerd_sem);
-}
-
 static int
 thread(void *unused)
 {
- struct timer_list wakeup_timer;
-
- kvoyagerd_running = 1;
-
- daemonize(THREAD_NAME);
-
- set_timeout = 0;
-
- init_timer(&wakeup_timer);
-
- sigfillset(&current->blocked);
-
  printk(KERN_NOTICE "Voyager starting monitor thread\n");

- for(;;) {
-  down_interruptible(&kvoyagerd_sem);
+ for (;;) {
+  set_current_state(TASK_INTERRUPTIBLE);
+  schedule_timeout(set_timeout ? HZ : MAX_SCHEDULE_TIMEOUT);
+
   VDEBUG(("Voyager Daemon awoken\n"));
   if(voyager_status.request_from_kernel == 0) {
    /* probably awoken from timeout */
@@ -143,20 +110,26 @@ thread(void *unused)
    check_from_kernel();
    voyager_status.request_from_kernel = 0;
   }
-  if(set_timeout) {
-   del_timer(&wakeup_timer);
-   wakeup_timer.expires = HZ + jiffies;
-   wakeup_timer.function = wakeup;
-   add_timer(&wakeup_timer);
-  }
  }
 }

+static int __init
+voyager_thread_start(void)
+{
+ voyager_thread = kthread_run(thread, NULL, "kvoyagerd");
+ if (IS_ERR(voyager_thread)) {
```

```
+  printk(KERN_ERR "Voyager: Failed to create system monitor thread.\n");
+  return PTR_ERR(voyager_thread);
+ }
+ return 0;
+}
+
+
 static void __exit
 voyager_thread_stop(void)
 {
- /* FIXME: do nothing at the moment */
+ kthread_stop(voyager_thread);
 }

 module_init(voyager_thread_start);
-//module_exit(voyager_thread_stop);
+module_exit(voyager_thread_stop);
Index: linux-2.6/include/asm-i386/voyager.h
===================================================================
--- linux-2.6.orig/include/asm-i386/voyager.h 2007-04-22 15:18:39.000000000 +0200
+++ linux-2.6/include/asm-i386/voyager.h 2007-04-22 15:24:13.000000000 +0200
@@ -487,15 +487,11 @@ extern struct voyager_qic_cpi *voyager_q
 extern struct voyager_SUS *voyager_SUS;

 /* variables exported always */
+extern struct task_struct *voyager_thread;
 extern int voyager_level;
-extern int kvoyagerd_running;
-extern struct semaphore kvoyagerd_sem;
 extern struct voyager_status voyager_status;

-
-
 /* functions exported by the voyager and voyager_smp modules */
-
 extern int voyager_cat_readb(__u8 module, __u8 asic, int reg);
 extern void voyager_cat_init(void);
 extern void voyager_detect(struct voyager_bios_info *);
```

_____

Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: [PATCH] sas_scsi_host: Convert to use the kthread API
Posted by Christoph Hellwig on Sun, 22 Apr 2007 19:38:43 GMT
View Forum Message <> Reply to Message

On Thu, Apr 19, 2007 at 05:37:53PM -0700, Andrew Morton wrote:
> On Thu, 19 Apr 2007 01:58:38 -0600
> "Eric W. Biederman" <ebiederm@xmission.com> wrote:
>
> > From: Eric W. Biederman <ebiederm@xmission.com>
> >
> > This patch modifies the sas scsi host thread startup
> > to use kthread_run not kernel_thread and deamonize.
> > kthread_run is slightly simpler and more maintainable.
> >
>
> Again, I'll rename this to "partially convert...". This driver should be
> using kthread_should_stop() and kthread_stop() rather than the
> apparently-unnecessary ->queue_thread_kill thing.
>
> This driver was merged two and a half years after the kthread API was
> available. Our coding-vs-reviewing effort is out of balance.

Here's a full conversion.

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

Index: linux-2.6/drivers/scsi/libsas/sas_scsi_host.c
===
--- linux-2.6.orig/drivers/scsi/libsas/sas_scsi_host.c 2007-04-22 20:30:39.000000000 +0200
+++ linux-2.6/drivers/scsi/libsas/sas_scsi_host.c 2007-04-22 20:36:51.000000000 +0200
@@ -23,6 +23,8 @@
  *
  */

+#include <linux/kthread.h>
+
 #include "sas_internal.h"

 #include <scsi/scsi_host.h>
@@ -184,7 +186,7 @@ static int sas_queue_up(struct sas_task
  list_add_tail(&task->list, &core->task_queue);
  core->task_queue_size += 1;
  spin_unlock_irqrestore(&core->task_queue_lock, flags);
- up(&core->queue_thread_sema);
+ wake_up_process(core->queue_thread);

  return 0;
 }
@@ -819,7 +821,7 @@ static void sas_queue(struct sas_ha_stru
  struct sas_internal *i = to_sas_internal(core->shost->transportt);

```
  spin_lock_irqsave(&core->task_queue_lock, flags);
- while (!core->queue_thread_kill &&
+ while (!kthread_should_stop() &&
         !list_empty(&core->task_queue)) {

   can_queue = sas_ha->lldd_queue_size - core->task_queue_size;
@@ -858,8 +860,6 @@ static void sas_queue(struct sas_ha_stru
  spin_unlock_irqrestore(&core->task_queue_lock, flags);
 }

-static DECLARE_COMPLETION(queue_th_comp);
-
 /**
  * sas_queue_thread -- The Task Collector thread
  * @_sas_ha: pointer to struct sas_ha
@@ -867,40 +867,33 @@ static DECLARE_COMPLETION(queue_th_comp)
 static int sas_queue_thread(void *_sas_ha)
 {
  struct sas_ha_struct *sas_ha = _sas_ha;
- struct scsi_core *core = &sas_ha->core;

- daemonize("sas_queue_%d", core->shost->host_no);
  current->flags |= PF_NOFREEZE;

- complete(&queue_th_comp);
-
  while (1) {
-  down_interruptible(&core->queue_thread_sema);
+  set_current_state(TASK_INTERRUPTIBLE);
+  schedule();
   sas_queue(sas_ha);
-  if (core->queue_thread_kill)
+  if (kthread_should_stop())
    break;
  }

- complete(&queue_th_comp);
-
  return 0;
 }

 int sas_init_queue(struct sas_ha_struct *sas_ha)
 {
- int res;
  struct scsi_core *core = &sas_ha->core;

  spin_lock_init(&core->task_queue_lock);
  core->task_queue_size = 0;
```

```
  INIT_LIST_HEAD(&core->task_queue);
- init_MUTEX_LOCKED(&core->queue_thread_sema);

- res = kernel_thread(sas_queue_thread, sas_ha, 0);
- if (res >= 0)
-  wait_for_completion(&queue_th_comp);
-
- return res < 0 ? res : 0;
+ core->queue_thread = kthread_run(sas_queue_thread, sas_ha,
+       "sas_queue_%d", core->shost->host_no);
+ if (IS_ERR(core->queue_thread))
+  return PTR_ERR(core->queue_thread);
+ return 0;
 }

 void sas_shutdown_queue(struct sas_ha_struct *sas_ha)
@@ -909,10 +902,7 @@ void sas_shutdown_queue(struct sas_ha_st
  struct scsi_core *core = &sas_ha->core;
  struct sas_task *task, *n;

- init_completion(&queue_th_comp);
- core->queue_thread_kill = 1;
- up(&core->queue_thread_sema);
- wait_for_completion(&queue_th_comp);
+ kthread_stop(core->queue_thread);

  if (!list_empty(&core->task_queue))
   SAS_DPRINTK("HA: %llx: scsi core task queue is NOT empty!?\n",
Index: linux-2.6/include/scsi/libsas.h
===================================================================
--- linux-2.6.orig/include/scsi/libsas.h 2007-04-22 20:32:41.000000000 +0200
+++ linux-2.6/include/scsi/libsas.h 2007-04-22 20:32:59.000000000 +0200
@@ -314,8 +314,7 @@ struct scsi_core {
  struct list_head  task_queue;
  int               task_queue_size;

- struct semaphore  queue_thread_sema;
- int               queue_thread_kill;
+ struct task_struct *queue_thread;
 };

 struct sas_ha_event {
```

Subject: Re: [PATCH] mtd_blkdevs: Convert to use the kthread API
Posted by Christoph Hellwig on Sun, 22 Apr 2007 19:40:57 GMT
View Forum Message <> Reply to Message

On Sun, Apr 22, 2007 at 01:24:53PM +0100, Christoph Hellwig wrote:
> On Thu, Apr 19, 2007 at 12:55:28AM -0600, Eric W. Biederman wrote:
> > From: Eric W. Biederman <ebiederm@xmission.com> - unquoted
> >
> > thread_run is used intead of kernel_thread, daemonize, and mucking
> > around blocking signals directly.
>
> This is the full conversion I sent to Dave in April 2006, but never got
> any feedback to:

Here's a slightly updated version that corrects the set_current_state
placement as discussed with Dave on irc:

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

```
Index: linux-2.6/drivers/mtd/mtd_blkdevs.c
===================================================================
--- linux-2.6.orig/drivers/mtd/mtd_blkdevs.c 2007-01-29 10:03:52.000000000 +0100
+++ linux-2.6/drivers/mtd/mtd_blkdevs.c 2007-04-22 20:39:20.000000000 +0200
@@ -20,6 +20,7 @@
 #include <linux/hdreg.h>
 #include <linux/init.h>
 #include <linux/mutex.h>
+#include <linux/kthread.h>
 #include <asm/uaccess.h>

 static LIST_HEAD(blktrans_majors);
@@ -28,9 +29,7 @@ extern struct mutex mtd_table_mutex;
 extern struct mtd_info *mtd_table[];

 struct mtd_blkcore_priv {
- struct completion thread_dead;
- int exiting;
- wait_queue_head_t thread_wq;
+ struct task_struct *thread;
  struct request_queue *rq;
  spinlock_t queue_lock;
 };
@@ -83,38 +82,19 @@ static int mtd_blktrans_thread(void *arg
  /* we might get involved when memory gets low, so use PF_MEMALLOC */
  current->flags |= PF_MEMALLOC | PF_NOFREEZE;

- daemonize("%sd", tr->name);
-
```

```
- /* daemonize() doesn't do this for us since some kernel threads
-    actually want to deal with signals. We can't just call
-    exit_sighand() since that'll cause an oops when we finally
-    do exit. */
- spin_lock_irq(&current->sighand->siglock);
- sigfillset(&current->blocked);
- recalc_sigpending();
- spin_unlock_irq(&current->sighand->siglock);
-
  spin_lock_irq(rq->queue_lock);
-
- while (!tr->blkcore_priv->exiting) {
+ while (!kthread_should_stop()) {
   struct request *req;
   struct mtd_blktrans_dev *dev;
   int res = 0;
-  DECLARE_WAITQUEUE(wait, current);

   req = elv_next_request(rq);

   if (!req) {
-   add_wait_queue(&tr->blkcore_priv->thread_wq, &wait);
    set_current_state(TASK_INTERRUPTIBLE);
-
    spin_unlock_irq(rq->queue_lock);
-
    schedule();
-   remove_wait_queue(&tr->blkcore_priv->thread_wq, &wait);
-
    spin_lock_irq(rq->queue_lock);
-
    continue;
   }

@@ -133,13 +113,13 @@ static int mtd_blktrans_thread(void *arg
  }
  spin_unlock_irq(rq->queue_lock);

- complete_and_exit(&tr->blkcore_priv->thread_dead, 0);
+ return 0;
 }

 static void mtd_blktrans_request(struct request_queue *rq)
 {
  struct mtd_blktrans_ops *tr = rq->queuedata;
- wake_up(&tr->blkcore_priv->thread_wq);
+ wake_up_process(tr->blkcore_priv->thread);
 }
```

```
@@ -388,8 +368,6 @@ int register_mtd_blktrans(struct mtd_blk
   return ret;
  }
  spin_lock_init(&tr->blkcore_priv->queue_lock);
- init_completion(&tr->blkcore_priv->thread_dead);
- init_waitqueue_head(&tr->blkcore_priv->thread_wq);

  tr->blkcore_priv->rq = blk_init_queue(mtd_blktrans_request, &tr->blkcore_priv->queue_lock);
  if (!tr->blkcore_priv->rq) {
@@ -403,13 +381,14 @@ int register_mtd_blktrans(struct mtd_blk
  blk_queue_hardsect_size(tr->blkcore_priv->rq, tr->blksize);
  tr->blkshift = ffs(tr->blksize) - 1;

- ret = kernel_thread(mtd_blktrans_thread, tr, CLONE_KERNEL);
- if (ret < 0) {
+ tr->blkcore_priv->thread = kthread_run(mtd_blktrans_thread, tr,
+    "%sd", tr->name);
+ if (IS_ERR(tr->blkcore_priv->thread)) {
   blk_cleanup_queue(tr->blkcore_priv->rq);
   unregister_blkdev(tr->major, tr->name);
   kfree(tr->blkcore_priv);
   mutex_unlock(&mtd_table_mutex);
-  return ret;
+  return PTR_ERR(tr->blkcore_priv->thread);
  }

  INIT_LIST_HEAD(&tr->devs);
@@ -432,9 +411,7 @@ int deregister_mtd_blktrans(struct mtd_b
  mutex_lock(&mtd_table_mutex);

  /* Clean up the kernel thread */
- tr->blkcore_priv->exiting = 1;
- wake_up(&tr->blkcore_priv->thread_wq);
- wait_for_completion(&tr->blkcore_priv->thread_dead);
+ kthread_stop(tr->blkcore_priv->thread);

  /* Remove it from the list of active majors */
  list_del(&tr->list);
```

Subject: Re: [PATCH] bluetooth bnep: Convert to kthread API.

Posted by Christoph Hellwig on Sun, 22 Apr 2007 19:44:55 GMT
View Forum Message <> Reply to Message

On Thu, Apr 19, 2007 at 04:24:59PM -0700, Andrew Morton wrote:
> On Thu, 19 Apr 2007 01:58:51 -0600
> "Eric W. Biederman" <ebiederm@xmission.com> wrote:
>
> > From: Eric W. Biederman <ebiederm@xmission.com>
> >
> > This patch starts kbenpd using kthread_run replacing
> > a combination of kernel_thread and daemonize. Making
> > the code a little simpler and more maintainable.
> >
> >
>
> while (!atomic_read(&s->killed)) {
>
> ho hum.

Note that this also stands against a full kthread conversion.
Marcel put my old patches for a full kthread conversion in, but
they didn't deal properly with some of the premaure exit cases,
and causes OOPSes.

I don't remember what the problems where, but the case of a thread
terminating earlier and possibly asynchronously is one of the
cases we'll probably have to add to the kthread infrastructure
before all uses of kernel_thread in drivers can be converted.

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

Subject: Re: [PATCH] ipv4/ipvs: Convert to kthread API
Posted by Christoph Hellwig on Sun, 22 Apr 2007 19:50:01 GMT
View Forum Message <> Reply to Message

On Thu, Apr 19, 2007 at 03:59:44PM -0700, Andrew Morton wrote:
> There still seems to be quite a lot of complexity in this driver's
> thread handling which could be removed if we did a full conversion
> to the kthread API.
>
> It all looks.... surprisingly complex in there.

It is. There quite a few interesting oddities in this code:

- creation of a forker thread. This is superflous when using the kthread infrastructure as a thread created by kthread_create always comes from our dedicated forker thread.
- the infinite retry on failure looks very bogus, the system doesn't recover very well if you try to fork forever in a loop :)
- a lot of very overlapping state variables. My reading of the code suggests that both a 'master' and 'backup' thread can run at the same time. I think the code would benefit a lot from totally separating these codepathes.
- start_sync_thread and stop_sync_thread are called with unchecked user supplied arguments and bug if they don't match the expected values. While all this is under capable(CAP_NET_ADMIN) it still sounds like something to fix.
- and the usual removal of semaphores and completions for startup/shutdown would benefit the code a lot, as for most thread users.

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: [PATCH] bluetooth rfcomm: Convert to kthread API.
Posted by Christoph Hellwig on Sun, 22 Apr 2007 20:14:27 GMT
View Forum Message <> Reply to Message

On Thu, Apr 19, 2007 at 04:12:53PM -0700, Andrew Morton wrote:
> On Thu, 19 Apr 2007 01:58:54 -0600
> "Eric W. Biederman" <ebiederm@xmission.com> wrote:
>
> > From: Eric W. Biederman <ebiederm@xmission.com>
> >
> > This patch starts krfcommd using kthread_run instead of a combination
> > of kernel_thread and daemonize making the code slightly simpler
> > and more maintainable.
>
> gargh, the more I look at these things, the more I agree with Christoph.

Hehe. Here's a patch to do the full kthread conversion for rfcomm, it doesn't have the asynchrnous termination issues the other bluetooth drivers have. Also handle init failures in rfcomm while we're at it.

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

Index: linux-2.6/net/bluetooth/rfcomm/core.c
==========

```
--- linux-2.6.orig/net/bluetooth/rfcomm/core.c 2007-04-22 21:01:31.000000000 +0200
+++ linux-2.6/net/bluetooth/rfcomm/core.c 2007-04-22 21:12:30.000000000 +0200
@@ -37,6 +37,7 @@
 #include <linux/device.h>
 #include <linux/net.h>
 #include <linux/mutex.h>
+#include <linux/kthread.h>

 #include <net/sock.h>
 #include <asm/uaccess.h>
@@ -67,7 +68,6 @@ static DEFINE_MUTEX(rfcomm_mutex);
 static unsigned long rfcomm_event;

 static LIST_HEAD(session_list);
-static atomic_t terminate, running;

 static int rfcomm_send_frame(struct rfcomm_session *s, u8 *data, int len);
 static int rfcomm_send_sabm(struct rfcomm_session *s, u8 dlci);
@@ -1846,26 +1846,6 @@ static inline void rfcomm_process_sessio
  rfcomm_unlock();
 }

-static void rfcomm_worker(void)
-{
- BT_DBG("");
-
- while (!atomic_read(&terminate)) {
-  if (!test_bit(RFCOMM_SCHED_WAKEUP, &rfcomm_event)) {
-   /* No pending events. Let's sleep.
-    * Incoming connections and data will wake us up. */
-   set_current_state(TASK_INTERRUPTIBLE);
-   schedule();
-  }
-
-  /* Process stuff */
-  clear_bit(RFCOMM_SCHED_WAKEUP, &rfcomm_event);
-  rfcomm_process_sessions();
- }
- set_current_state(TASK_RUNNING);
- return;
-}
-
 static int rfcomm_add_listener(bdaddr_t *ba)
 {
  struct sockaddr_l2 addr;
@@ -1931,23 +1911,27 @@ static void rfcomm_kill_listener(void)

 static int rfcomm_run(void *unused)
```

```
 {
- rfcomm_thread = current;
-
- atomic_inc(&running);
-
- daemonize("krfcommd");
  set_user_nice(current, -10);
  current->flags |= PF_NOFREEZE;

  BT_DBG("");

  rfcomm_add_listener(BDADDR_ANY);
+ while (!kthread_should_stop()) {
+  if (!test_bit(RFCOMM_SCHED_WAKEUP, &rfcomm_event)) {
+   /* No pending events. Let's sleep.
+    * Incoming connections and data will wake us up. */
+   set_current_state(TASK_INTERRUPTIBLE);
+   schedule();
+  }

- rfcomm_worker();
-
+  /* Process stuff */
+  clear_bit(RFCOMM_SCHED_WAKEUP, &rfcomm_event);
+  rfcomm_process_sessions();
+ }
+ set_current_state(TASK_RUNNING);
  rfcomm_kill_listener();

- atomic_dec(&running);
  return 0;
 }

@@ -2052,24 +2036,52 @@ static CLASS_ATTR(rfcomm_dlc, S_IRUGO, r
 /* ---- Initialization ---- */
 static int __init rfcomm_init(void)
 {
+ int err;
+
  l2cap_load();

- hci_register_cb(&rfcomm_cb);
+ err = hci_register_cb(&rfcomm_cb);
+ if (err)
+  goto out;

- kernel_thread(rfcomm_run, NULL, CLONE_KERNEL);
+ rfcomm_thread = kthread_run(rfcomm_run, NULL, "krfcommd");
```

```
+ if (IS_ERR(rfcomm_thread)) {
+  err = PTR_ERR(rfcomm_thread);
+  goto out_unregister_hci;
+ }

- if (class_create_file(bt_class, &class_attr_rfcomm_dlc) < 0)
+ err = class_create_file(bt_class, &class_attr_rfcomm_dlc);
+ if (err < 0) {
   BT_ERR("Failed to create RFCOMM info file");
+  goto out_kthread_stop;
+ }

- rfcomm_init_sockets();
+ err = rfcomm_init_sockets();
+ if (err)
+  goto out_remove_sysfs_files;

 #ifdef CONFIG_BT_RFCOMM_TTY
- rfcomm_init_ttys();
+ err = rfcomm_init_ttys();
+ if (err)
+  goto out_cleanup_sockets;
 #endif

  BT_INFO("RFCOMM ver %s", VERSION);

  return 0;
+
+#ifdef CONFIG_BT_RFCOMM_TTY
+ out_cleanup_sockets:
+ rfcomm_cleanup_sockets();
+#endif
+ out_remove_sysfs_files:
+ class_remove_file(bt_class, &class_attr_rfcomm_dlc);
+ out_unregister_hci:
+ hci_unregister_cb(&rfcomm_cb);
+ out_kthread_stop:
+ kthread_stop(rfcomm_thread);
+ out:
+ return err;
 }

 static void __exit rfcomm_exit(void)
@@ -2077,15 +2089,7 @@ static void __exit rfcomm_exit(void)
  class_remove_file(bt_class, &class_attr_rfcomm_dlc);

  hci_unregister_cb(&rfcomm_cb);
-
```

```
- /* Terminate working thread.
-  * ie. Set terminate flag and wake it up */
- atomic_inc(&terminate);
- rfcomm_schedule(RFCOMM_SCHED_STATE);
-
- /* Wait until thread is running */
- while (atomic_read(&running))
-  schedule();
+ kthread_stop(rfcomm_thread);

 #ifdef CONFIG_BT_RFCOMM_TTY
  rfcomm_cleanup_ttys();
```

_____

Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: [PATCH] s390/scsi/zfcp_erp: Convert to use the kthread API
Posted by Christoph Hellwig on Sun, 22 Apr 2007 20:17:09 GMT
View Forum Message <> Reply to Message

On Thu, Apr 19, 2007 at 01:58:42AM -0600, Eric W. Biederman wrote:
> From: Eric W. Biederman <ebiederm@xmission.com>
>
> Modify zfcperp%s to be started with kthread_run not
> a combination of kernel_thread, daemonize and siginitsetinv
> making the code slightly simpler and more maintainable.

This driver would also benefit from a full kthread conversion.
Unfortunately it has a strange dual-use semaphore (->erp_ready_sem)
that hinders a straight conversion. Maybe the maintainer can take
a look whether there's a nice way to get rid of that one?

_____

Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: [PATCH] arm ecard: Conver to use the kthread API.
Posted by Christoph Hellwig on Sun, 22 Apr 2007 20:18:24 GMT
View Forum Message <> Reply to Message

On Thu, Apr 19, 2007 at 01:58:43AM -0600, Eric W. Biederman wrote:
> From: Eric W. Biederman <ebiederm@xmission.com>
>

> This patch modifies the startup of kecardd to use
> kthread_run not a kernel_thread combination of kernel_thread
> and daemonize. Making the code slightly simpler and more
> maintainable.

Looks good. Given that this is non-modular and there's no
exit function there is no need for further action.

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: [PATCH] ia64 sn xpc: Convert to use kthread API.
Posted by Christoph Hellwig on Sun, 22 Apr 2007 20:36:47 GMT
View Forum Message <> Reply to Message

On Thu, Apr 19, 2007 at 01:58:44AM -0600, Eric W. Biederman wrote:
> From: Eric W. Biederman <ebiederm@xmission.com>
>
> This patch starts the xpc kernel threads using kthread_run
> not a combination of kernel_thread and daemonize. Resuling
> in slightly simpler and more maintainable code.

This driver is a really twisted maze. It has a lot of threads,
some of them running through the whole lifetime of the driver,
some short-lived and some in a sort of a pool.

The patch below fixes up the long-lived thread as well as fixing
gazillions of leaks in the init routine by switching to proper
goto-based unwinding.

Note that thread pools are something we have in a few places,
and might be worth handling in the core kthread infrastructure,
as tearing down pools will get a bit complicated using the
kthread APIs.

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

Index: linux-2.6/arch/ia64/sn/kernel/xpc_main.c
===
--- linux-2.6.orig/arch/ia64/sn/kernel/xpc_main.c 2007-04-22 21:19:22.000000000 +0200
+++ linux-2.6/arch/ia64/sn/kernel/xpc_main.c 2007-04-22 21:33:54.000000000 +0200
@@ -55,6 +55,7 @@
 #include <linux/delay.h>
 #include <linux/reboot.h>

```
 #include <linux/completion.h>
+#include <linux/kthread.h>
 #include <asm/sn/intr.h>
 #include <asm/sn/sn_sal.h>
 #include <asm/kdebug.h>
@@ -159,16 +160,14 @@ static struct ctl_table_header *xpc_sysc
 int xpc_disengage_request_timedout;

 /* #of IRQs received */
-static atomic_t xpc_act_IRQ_rcvd;
+static atomic_t xpc_act_IRQ_rcvd = ATOMIC_INIT(0);

 /* IRQ handler notifies this wait queue on receipt of an IRQ */
 static DECLARE_WAIT_QUEUE_HEAD(xpc_act_IRQ_wq);

+static struct task_struct *xpc_hb_checker_thread;
 static unsigned long xpc_hb_check_timeout;

-/* notification that the xpc_hb_checker thread has exited */
-static DECLARE_COMPLETION(xpc_hb_checker_exited);
-
 /* notification that the xpc_discovery thread has exited */
 static DECLARE_COMPLETION(xpc_discovery_exited);

@@ -250,17 +249,10 @@ xpc_hb_checker(void *ignore)
  int new_IRQ_count;
  int force_IRQ=0;

-
  /* this thread was marked active by xpc_hb_init() */
-
- daemonize(XPC_HB_CHECK_THREAD_NAME);
-
- set_cpus_allowed(current, cpumask_of_cpu(XPC_HB_CHECK_CPU));
-
  xpc_hb_check_timeout = jiffies + (xpc_hb_check_interval * HZ);

- while (!(volatile int) xpc_exiting) {
-
+ while (!kthread_should_stop()) {
   dev_dbg(xpc_part, "woke up with %d ticks rem; %d IRQs have "
    "been received\n",
    (int) (xpc_hb_check_timeout - jiffies),
@@ -304,14 +296,10 @@ xpc_hb_checker(void *ignore)
   (void) wait_event_interruptible(xpc_act_IRQ_wq,
        (last_IRQ_count < atomic_read(&xpc_act_IRQ_rcvd) ||
      jiffies >= xpc_hb_check_timeout ||
-      (volatile int) xpc_exiting));
```

```
+     kthread_should_stop()));
  }

  dev_dbg(xpc_part, "heartbeat checker is exiting\n");
-
-
- /* mark this thread as having exited */
- complete(&xpc_hb_checker_exited);
  return 0;
 }

@@ -966,9 +954,7 @@ xpc_do_exit(enum xpc_retval reason)
  /* wait for the discovery thread to exit */
  wait_for_completion(&xpc_discovery_exited);

- /* wait for the heartbeat checker thread to exit */
- wait_for_completion(&xpc_hb_checker_exited);
-
+ kthread_stop(xpc_hb_checker_thread);

  /* sleep for a 1/3 of a second or so */
  (void) msleep_interruptible(300);
@@ -1219,29 +1205,29 @@ xpc_system_die(struct notifier_block *nb
 int __init
 xpc_init(void)
 {
- int ret;
+ int ret = -ENODEV;
  partid_t partid;
  struct xpc_partition *part;
  pid_t pid;
  size_t buf_size;

+ if (!ia64_platform_is("sn2"))
+  goto out;

- if (!ia64_platform_is("sn2")) {
-  return -ENODEV;
- }
-
-
+ ret = -ENOMEM;
  buf_size = max(XPC_RP_VARS_SIZE,
     XPC_RP_HEADER_SIZE + XP_NASID_MASK_BYTES);
  xpc_remote_copy_buffer = xpc_kmalloc_cacheline_aligned(buf_size,
         GFP_KERNEL, &xpc_remote_copy_buffer_base);
- if (xpc_remote_copy_buffer == NULL)
-  return -ENOMEM;
```

```
+ if (!xpc_remote_copy_buffer)
+  goto out;

  snprintf(xpc_part->bus_id, BUS_ID_SIZE, "part");
  snprintf(xpc_chan->bus_id, BUS_ID_SIZE, "chan");

  xpc_sysctl = register_sysctl_table(xpc_sys_dir);
+ if (!xpc_sysctl)
+  goto out_free_remote_buffer;

  /*
   * The first few fields of each entry of xpc_partitions[] need to
@@ -1278,12 +1264,6 @@ xpc_init(void)
  xpc_allow_IPI_ops();

  /*
-  * Interrupts being processed will increment this atomic variable and
-  * awaken the heartbeat thread which will process the interrupts.
-  */
- atomic_set(&xpc_act_IRQ_rcvd, 0);
-
- /*
   * This is safe to do before the xpc_hb_checker thread has started
   * because the handler releases a wait queue.  If an interrupt is
   * received before the thread is waiting, it will not go to sleep,
@@ -1294,15 +1274,7 @@ xpc_init(void)
  if (ret != 0) {
   dev_err(xpc_part, "can't register ACTIVATE IRQ handler, "
    "errno=%d\n", -ret);
-
-  xpc_restrict_IPI_ops();
-
-  if (xpc_sysctl) {
-   unregister_sysctl_table(xpc_sysctl);
-  }
-
-  kfree(xpc_remote_copy_buffer_base);
-  return -EBUSY;
+  goto out_restrict_IPI_ops;
  }

  /*
@@ -1313,29 +1285,23 @@ xpc_init(void)
  xpc_rsvd_page = xpc_rsvd_page_init();
  if (xpc_rsvd_page == NULL) {
   dev_err(xpc_part, "could not setup our reserved page\n");
-
-  free_irq(SGI_XPC_ACTIVATE, NULL);
```

```
-  xpc_restrict_IPI_ops();
-
-  if (xpc_sysctl) {
-   unregister_sysctl_table(xpc_sysctl);
-  }
-
-  kfree(xpc_remote_copy_buffer_base);
-  return -EBUSY;
+  ret = -ENOMEM;
+  goto out_free_irq;
  }


  /* add ourselves to the reboot_notifier_list */
  ret = register_reboot_notifier(&xpc_reboot_notifier);
  if (ret != 0) {
-  dev_warn(xpc_part, "can't register reboot notifier\n");
+  dev_err(xpc_part, "can't register reboot notifier\n");
+  goto out_free_rsvd_page;
  }

  /* add ourselves to the die_notifier list (i.e., ia64die_chain) */
  ret = register_die_notifier(&xpc_die_notifier);
  if (ret != 0) {
-  dev_warn(xpc_part, "can't register die notifier\n");
+  dev_err(xpc_part, "can't register die notifier\n");
+  goto out_unregister_reboot_notifier;
  }


@@ -1353,31 +1319,16 @@ xpc_init(void)
   * The real work-horse behind xpc.  This processes incoming
   * interrupts and monitors remote heartbeats.
   */
- pid = kernel_thread(xpc_hb_checker, NULL, 0);
- if (pid < 0) {
+ xpc_hb_checker_thread = kthread_create(xpc_hb_checker, NULL,
+       XPC_HB_CHECK_THREAD_NAME);
+ if (IS_ERR(xpc_hb_checker_thread)) {
   dev_err(xpc_part, "failed while forking hb check thread\n");
-
-  /* indicate to others that our reserved page is uninitialized */
-  xpc_rsvd_page->vars_pa = 0;
-
-  /* take ourselves off of the reboot_notifier_list */
-  (void) unregister_reboot_notifier(&xpc_reboot_notifier);
-
-  /* take ourselves off of the die_notifier list */
```

```
-  (void) unregister_die_notifier(&xpc_die_notifier);
-
-  del_timer_sync(&xpc_hb_timer);
-  free_irq(SGI_XPC_ACTIVATE, NULL);
-  xpc_restrict_IPI_ops();
-
-  if (xpc_sysctl) {
-   unregister_sysctl_table(xpc_sysctl);
-  }
-
-  kfree(xpc_remote_copy_buffer_base);
-  return -EBUSY;
+  ret = PTR_ERR(xpc_hb_checker_thread);
+  goto out_del_hb_timer;
  }

+ kthread_bind(xpc_hb_checker_thread, XPC_HB_CHECK_CPU);
+ wake_up_process(xpc_hb_checker_thread);

  /*
   * Startup a thread that will attempt to discover other partitions to
@@ -1403,6 +1354,29 @@ xpc_init(void)
      xpc_initiate_partid_to_nasids);

  return 0;
+
+ if (ret != 0) {
+  dev_err(xpc_part, "can't register reboot notifier\n");
+  goto out_free_rsvd_page;
+ }
+
+ out_del_hb_timer:
+ unregister_die_notifier(&xpc_die_notifier);
+ out_unregister_reboot_notifier:
+ unregister_reboot_notifier(&xpc_reboot_notifier);
+ out_free_rsvd_page:
+ /* indicate to others that our reserved page is uninitialized */
+ xpc_rsvd_page->vars_pa = 0;
+ /* XXX(hch): xpc_rsvd_page gets leaked */
+ out_free_irq:
+ free_irq(SGI_XPC_ACTIVATE, NULL);
+ out_restrict_IPI_ops:
+ xpc_restrict_IPI_ops();
+ unregister_sysctl_table(xpc_sysctl);
+ out_free_remote_buffer:
+ kfree(xpc_remote_copy_buffer_base);
+ out:
+ return -EBUSY;
```

}
module_init(xpc_init);

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

Subject: Re: [PATCH] sas_scsi_host: Convert to use the kthread API
Posted by James Bottomley on Sun, 22 Apr 2007 21:37:48 GMT
View Forum Message <> Reply to Message

On Sun, 2007-04-22 at 20:38 +0100, Christoph Hellwig wrote:
> On Thu, Apr 19, 2007 at 05:37:53PM -0700, Andrew Morton wrote:
> > On Thu, 19 Apr 2007 01:58:38 -0600
> > "Eric W. Biederman" <ebiederm@xmission.com> wrote:
> >
> > > From: Eric W. Biederman <ebiederm@xmission.com>
> > >
> > > This patch modifies the sas scsi host thread startup
> > > to use kthread_run not kernel_thread and deamonize.
> > > kthread_run is slightly simpler and more maintainable.
> > >
> >
> > Again, I'll rename this to "partially convert...". This driver should be
> > using kthread_should_stop() and kthread_stop() rather than the
> > apparently-unnecessary ->queue_thread_kill thing.
> >
> > This driver was merged two and a half years after the kthread API was
> > available. Our coding-vs-reviewing effort is out of balance.
>
> Here's a full conversion.

Changelog and cc to linux-scsi, and I think it can go in ... not that it
matters; nothing ever activates this code inside libsas anyway ...

James

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

Subject: Re: [PATCH] sas_scsi_host: Convert to use the kthread API

Posted by ebiederm on Sun, 22 Apr 2007 21:48:05 GMT
View Forum Message <> Reply to Message

James Bottomley <James.Bottomley@SteelEye.com> writes:

> Changelog and cc to linux-scsi, and I think it can go in ... not that it
> matters; nothing ever activates this code inside libsas anyway ...

Should we just remove the relevant code then?

Eric

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

Subject: Re: [PATCH] macintosh/therm_pm72.c: Convert to kthread API.
Posted by Paul Mackerras on Sun, 22 Apr 2007 22:46:28 GMT
View Forum Message <> Reply to Message

Christoph Hellwig writes:

> Why is this driver using a thread at all? It's only doing a bunch
> of rather short-lived things in the thread.

It's doing i2c reads and writes, which block, and are actually quite
slow.

Paul.

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

Subject: [PATCH] kthread: Spontaneous exit support
Posted by ebiederm on Mon, 23 Apr 2007 03:12:55 GMT
View Forum Message <> Reply to Message

This patch implements the kthread helper functions kthread_start
and kthread_end which make it simple to support a kernel thread
that may decided to exit on it's own before we request it to.
It is still assumed that eventually we will get around to requesting
that the kernel thread stop.

Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>

```
 include/linux/kthread.h |   23 ++++++++++++++++++++++
 kernel/kthread.c        |   18 ++++++++++++++++++
 2 files changed, 41 insertions(+), 0 deletions(-)

diff --git a/include/linux/kthread.h b/include/linux/kthread.h
index a8ea31d..4f1eff1 100644
--- a/include/linux/kthread.h
+++ b/include/linux/kthread.h
@@ -28,6 +28,29 @@ struct task_struct *kthread_create(int (*threadfn)(void *data),

 void kthread_bind(struct task_struct *k, unsigned int cpu);
 int kthread_stop(struct task_struct *k);
+/**
+ * kthread_start - create and wake a thread.
+ * @threadfn: the function to run until kthread_should_stop().
+ * @data: data ptr for @threadfn.
+ * @namefmt: printf-style name for the thread.
+ *
+ * Description: Convenient wrapper for kthread_create() followed by
+ * get_task_struct() and wake_up_process. kthread_start should be paired
+ * with kthread_end() so we don't leak task structs.
+ *
+ * Returns the kthread or ERR_PTR(-ENOMEM).
+ */
+#define kthread_start(threadfn, data, namefmt, ...)      \
+({           \
+ struct task_struct *__k         \
+  = kthread_create(threadfn, data, namefmt, ## __VA_ARGS__); \
+ if (!IS_ERR(__k)) {         \
+  get_task_struct(__k);        \
+  wake_up_process(__k);        \
+ }           \
+ __k;           \
+})
+int kthread_end(struct task_struct *k);

 static inline int __kthread_should_stop(struct task_struct *tsk)
 {
diff --git a/kernel/kthread.c b/kernel/kthread.c
index 9b3c19f..d6d63c6 100644
--- a/kernel/kthread.c
+++ b/kernel/kthread.c
@@ -179,6 +179,24 @@ int kthread_stop(struct task_struct *tsk)
 }
 EXPORT_SYMBOL(kthread_stop);

+/**
+ * kthread_end - signal a kthread and wait for it to exit.
```

```
+ * @task: The kthread to end.
+ *
+ * Description: Convenient wrapper for kthread_stop() followed by
+ * put_task_struct().  Returns the kthread exit code.
+ *
+ * kthread_start()/kthread_end() can handle kthread that spontaneously exit
+ * before the kthread is requested to terminate.
+ */
+int kthread_end(struct task_struct *task)
+{
+ int ret;
+ ret = kthread_stop(task);
+ put_task_struct(task);
+ return ret;
+}
+EXPORT_SYMBOL(kthread_end);

 static __init void kthreadd_setup(void)
 {
--
1.5.0.g53756
```

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: Getting the new RxRPC patches upstream
Posted by David Howells on Mon, 23 Apr 2007 08:32:56 GMT
View Forum Message <> Reply to Message

> We only care when del_timer() returns true. In that case, if the timer
> function still runs (possible for single-threaded wqs), it has already
> passed __queue_work().

Why do you assume that?

David

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: [PATCH] kthread: Spontaneous exit support
Posted by Christoph Hellwig on Mon, 23 Apr 2007 11:25:37 GMT

On Sun, Apr 22, 2007 at 09:12:55PM -0600, Eric W. Biederman wrote:
>
> This patch implements the kthread helper functions kthread_start
> and kthread_end which make it simple to support a kernel thread
> that may decided to exit on it's own before we request it to.
> It is still assumed that eventually we will get around to requesting
> that the kernel thread stop.

I don't think having to parallel APIs is a good idea, people will
get utterly confused which one to use. Better always grab a reference
in kthread_create and drop it in kthread_stop. For normal thread
no change in behaviour and only slightly more code in the slowpath.

Of course it will need an audit for half-assed kthread conversion
first to avoid task_struct reference count leaks.

In addition to that kthrad_end implementation look wrong. When
the kthread has exited prematurely no one will call complete
on kthread_stop_info.done before it's been setup. Interestingly
the comment there indicates someone thought about threads exiting
early, but it became defunkt during all the rewrites of the
kthread code.

```
> +/**
> + * kthread_start - create and wake a thread.
> + * @threadfn: the function to run until kthread_should_stop().
> + * @data: data ptr for @threadfn.
> + * @namefmt: printf-style name for the thread.
> + *
> + * Description: Convenient wrapper for kthread_create() followed by
> + * get_task_struct() and wake_up_process. kthread_start should be paired
> + * with kthread_end() so we don't leak task structs.
> + *
> + * Returns the kthread or ERR_PTR(-ENOMEM).
> + */
> +#define kthread_start(threadfn, data, namefmt, ...)      \
> +({           \
> + struct task_struct *__k         \
> +  = kthread_create(threadfn, data, namefmt, ## __VA_ARGS__); \
> + if (!IS_ERR(__k)) {         \
> +  get_task_struct(__k);        \
> +  wake_up_process(__k);        \
> + }          \
> + __k;          \
> +})
> +int kthread_end(struct task_struct *k);
```

>
> static inline int __kthread_should_stop(struct task_struct *tsk)
> {
> diff --git a/kernel/kthread.c b/kernel/kthread.c
> index 9b3c19f..d6d63c6 100644
> --- a/kernel/kthread.c
> +++ b/kernel/kthread.c
> @@ -179,6 +179,24 @@ int kthread_stop(struct task_struct *tsk)
> }
> EXPORT_SYMBOL(kthread_stop);
>
> +/**
> + * kthread_end - signal a kthread and wait for it to exit.
> + * @task: The kthread to end.
> + *
> + * Description: Convenient wrapper for kthread_stop() followed by
> + * put_task_struct(). Returns the kthread exit code.
> + *
> + * kthread_start()/kthread_end() can handle kthread that spontaneously exit
> + * before the kthread is requested to terminate.
> + */
> +int kthread_end(struct task_struct *task)
> +{
> + int ret;
> + ret = kthread_stop(task);
> + put_task_struct(task);
> + return ret;
> +}
> +EXPORT_SYMBOL(kthread_end);

_____

Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: [PATCH] cpci_hotplug: Convert to use the kthread API
Posted by Scott Murray on Mon, 23 Apr 2007 16:19:40 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 :)

Heh, I guess I'll take that as a compliment. :)

> 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.
[snip]

I'm out of the office today, but I'll give it a spin on a test setup
tomorrow.

Thanks,

Scott

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

Subject: Re: [PATCH] kthread: Spontaneous exit support
Posted by Oleg Nesterov on Mon, 23 Apr 2007 16:58:10 GMT
View Forum Message <> Reply to Message

On 04/23, Christoph Hellwig wrote:
>
> On Sun, Apr 22, 2007 at 09:12:55PM -0600, Eric W. Biederman wrote:
> >
> > This patch implements the kthread helper functions kthread_start
> > and kthread_end which make it simple to support a kernel thread
> > that may decided to exit on it's own before we request it to.
> > It is still assumed that eventually we will get around to requesting
> > that the kernel thread stop.
>
> I don't think having to parallel APIs is a good idea, people will
> get utterly confused which one to use. Better always grab a reference
> in kthread_create and drop it in kthread_stop. For normal thread

> no change in behaviour and only slightly more code in the slowpath.
>
> Of course it will need an audit for half-assed kthread conversion
> first to avoid task_struct reference count leaks.

In that case it is better to grab a reference in kthread(). This also close the race when a new thread is woken (freezer) and exits before kthread_create() does get_task_struct().

> In addition to that kthrad_end implementation look wrong. When
> the kthread has exited prematurely no one will call complete
> on kthread_stop_info.done before it's been setup.

This is not true anymore, see another patch from Eric

kthread-enhance-kthread_stop-to-abort-interruptible-sleeps.patch

Oleg.

Subject: Re: [PATCH] ia64 sn xpc: Convert to use kthread API.
Posted by Jes Sorensen on Mon, 23 Apr 2007 17:11:49 GMT
View Forum Message <> Reply to Message

Christoph Hellwig wrote:
> On Thu, Apr 19, 2007 at 01:58:44AM -0600, Eric W. Biederman wrote:
>> From: Eric W. Biederman <ebiederm@xmission.com>
>>
>> This patch starts the xpc kernel threads using kthread_run
>> not a combination of kernel_thread and daemonize. Resuling
>> in slightly simpler and more maintainable code.
>
> This driver is a really twisted maze. It has a lot of threads,
> some of them running through the whole lifetime of the driver,
> some short-lived and some in a sort of a pool.
>
> The patch below fixes up the long-lived thread as well as fixing
> gazillions of leaks in the init routine by switching to proper
> goto-based unwinding.
>
> Note that thread pools are something we have in a few places,
> and might be worth handling in the core kthread infrastructure,
> as tearing down pools will get a bit complicated using the

> kthread APIs.

Like with the previous patch from Eric, I'm CC'ing the correct people
for this patch (forwarded it in a seperate email). CC'ing irrelevant
lists such as containers@ and not linux-ia64@ makes it somewhat
difficult to get proper reviews of these things.

Russ/Dean/Robin - could one of you provide some feedback to this one
please.

Thanks,
Jes

Subject: Re: Getting the new RxRPC patches upstream
Posted by Oleg Nesterov on Mon, 23 Apr 2007 17:11:57 GMT
View Forum Message <> Reply to Message

On 04/23, David Howells wrote:
>
> > We only care when del_timer() returns true. In that case, if the timer
> > function still runs (possible for single-threaded wqs), it has already
> > passed __queue_work().
>
> Why do you assume that?

If del_timer() returns true, the timer was pending. This means it was started
by work->func() (note that __run_timers() clears timer_pending() before calling
timer->function). This in turn means that delayed_work_timer_fn() has already
called __queue_work(dwork), otherwise work->func() has no chance to run.

When del_timer() returns true and delayed_work_timer_fn() doesn't run we are
safe, this doesn't differ from del_timer_sync().

Oleg.

Subject: Re: [PATCH] ia64 sn xpc: Convert to use kthread API.

Posted by ebiederm on Mon, 23 Apr 2007 17:36:00 GMT
View Forum Message <> Reply to Message

Jes Sorensen <jes@sgi.com> writes:

>
> Like with the previous patch from Eric, I'm CC'ing the correct people
> for this patch (forwarded it in a seperate email). CC'ing irrelevant
> lists such as containers@ and not linux-ia64@ makes it somewhat
> difficult to get proper reviews of these things.

containers is actually relevant because everything not being converted
to a kthread API is actually show stopper issue for the development of
the pid namespace because of the usage of pids by kernel_thread, and
the apparent impossibility to fix daemonize to sort this out.

Now I do agree linux-ia64 is also relevant.

> Russ/Dean/Robin - could one of you provide some feedback to this one
> please.

Eric

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

Subject: Re: [PATCH] kthread: Spontaneous exit support
Posted by ebiederm on Mon, 23 Apr 2007 17:45:51 GMT
View Forum Message <> Reply to Message

Oleg Nesterov <oleg@tv-sign.ru> writes:

> On 04/23, Christoph Hellwig wrote:
>>
>> On Sun, Apr 22, 2007 at 09:12:55PM -0600, Eric W. Biederman wrote:
>> >
>> > This patch implements the kthread helper functions kthread_start
>> > and kthread_end which make it simple to support a kernel thread
>> > that may decided to exit on it's own before we request it to.
>> > It is still assumed that eventually we will get around to requesting
>> > that the kernel thread stop.
>>
>> I don't think having to parallel APIs is a good idea, people will
>> get utterly confused which one to use. Better always grab a reference
>> in kthread_create and drop it in kthread_stop. For normal thread
>> no change in behaviour and only slightly more code in the slowpath.

>>
>> Of course it will need an audit for half-assed kthread conversion
>> first to avoid task_struct reference count leaks.
>
> In that case it is better to grab a reference in kthread(). This also
> close the race when a new thread is woken (freezer) and exits before
> kthread_create() does get_task_struct().
>
>> In addition to that kthrad_end implementation look wrong. When
>> the kthread has exited prematurely no one will call complete
>> on kthread_stop_info.done before it's been setup.
>
> This is not true anymore, see another patch from Eric
>
> kthread-enhance-kthread_stop-to-abort-interruptible-sleeps.patch

Ok. Thinking about it I agree with Christoph that parallel API's can
be a problem.

However we do still need to support kernel threads where kthread_stop will
never be called. There appear to be a few legitimate cases where
someone wants to fire off a thread and have it do some work but don't
care at all for stopping it before it is done.

So I propose we add a kthread_orphan as a basic primitive to decrement the
count on the task_struct if we want a kthread to simply exit after it
has done some work.

And as a helper function we can have a kthread_run_orphan.

I think having a kthread_orphan will document what we are doing better
and make it easy to find kernel threads that don't use kthread_stop.

The pain is that this requires an audit of all kernel kthread creators
so that we call kthread_orphan on the right ones, or else we will have
a task_struct leak. At least that isn't a fatal condition.

Eric

Subject: Re: [PATCH] kthread: Spontaneous exit support
Posted by Christoph Hellwig on Mon, 23 Apr 2007 18:09:18 GMT
View Forum Message <> Reply to Message

On Mon, Apr 23, 2007 at 11:45:51AM -0600, Eric W. Biederman wrote:
> Ok. Thinking about it I agree with Christoph that parallel API's can
> be a problem.
>
> However we do still need to support kernel threads where kthread_stop will
> never be called. There appear to be a few legitimate cases where
> someone wants to fire off a thread and have it do some work but don't
> care at all for stopping it before it is done.

There's two cases where it's valid that kthread_stop is not called:

 a) the user is always builtin and the thread runs until the kernel halts.
    examples: voyager, arm ecard
 b) the thread is normally started/stopped, e.g. at module_init/module_exit
    but there is some reason why it could terminate earlier.
    examples: the various bluetooth threads, nfs-related threads that
    can be killed using signals
 c) we have some kind of asynchronous helper thread.
    examples: various s390 drivers, usbatm, therm_pm72
 d) a driver has threadpools were we need to start/stop threads on demand.
    examples: nfsd, xpc

case a)
 is trivial, we can just ignore the refcounting issue.

case b)
 is what refcounting the task struct and proper handling in
 kthread_stop will deal with.

case c)
 should get a new kthread_create_async api which starts a thread
 without blocking, so we can get rid of the workqueues in the
 s390 drivers. it should probably also be safe to be called from
 irq context. What makes this a bit complicated is the need to
 make sure no more thread is running in case the caller terminates
 (shutdown of the structure it's associated with or module removal)

case d)
 should be deal with with a kthread_pool api

Subject: Re: [PATCH] kthread: Spontaneous exit support
Posted by Oleg Nesterov on Mon, 23 Apr 2007 18:20:14 GMT
View Forum Message <> Reply to Message

On 04/23, Eric W. Biederman wrote:
>
> So I propose we add a kthread_orphan as a basic primitive to decrement the
> count on the task_struct if we want a kthread to simply exit after it
> has done some work.
>
> And as a helper function we can have a kthread_run_orphan.

Speaking about helpers, could we also add kthread_start(), which should be
used instead of direct wake_up_process() ? Not that it is terribly important,
but still.

Note that "kthread_create() pins the task_struct" allows us to cleanup the code.
Look at this ugly "wait_to_die:" label in migration_thread(). Is is because
migration_thread() can't exit until CPU_DEAD reaps it. Other reasons were already
solved by kthread-enhance-kthread_stop-to-abort-interruptible-sleeps.patch

Oleg.

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

Subject: Re: [PATCH] ia64 sn xpc: Convert to use kthread API.
Posted by Russ Anderson on Mon, 23 Apr 2007 19:03:29 GMT
View Forum Message <> Reply to Message

Jes Sorensen wrote:
>
> Russ/Dean/Robin - could one of you provide some feedback to this one
> please.

Dean's on vacation for a couple days and will test it when he gets back.

--
Russ Anderson, OS RAS/Partitioning Project Lead
SGI - Silicon Graphics Inc rja@sgi.com

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

Subject: Re: [PATCH] powerpc pseries eeh: Convert to kthread API
Posted by linas on Mon, 23 Apr 2007 20:50:20 GMT
View Forum Message <> Reply to Message

On Sun, Apr 22, 2007 at 01:31:55PM +0100, Christoph Hellwig wrote:
> On Thu, Apr 19, 2007 at 01:58:45AM -0600, Eric W. Biederman wrote:
> > From: Eric W. Biederman <ebiederm@xmission.com>
> >
> > This patch modifies the startup of eehd to use kthread_run
> > not a combination of kernel_thread and daemonize. Making
> > the code slightly simpler and more maintainable.
>
> This one has the same scheme as the various s390 drivers where a thread
> is spawned using a workqueue on demand. I think we should not blindly
> convert it but think a litte more about it.
>
> The first question is obviously, is this really something we want?
> spawning kernel thread on demand without reaping them properly seems
> quite dangerous.

I'm not quite sure what the intent of this patch really is, being
at most a somewhat passing and casual user of kernel threads.

Some background may be useful: (this in reply to some comments from
Andrew Morton)

EEH events are supposed to be very rare, as they correspond to
hardware failures, typically PCI bus parity errors, but also
things like wild DMA's. The code that generates these will limit
them to no more than 6 per hour per pci device. Any more than that,
and the PCI device is permanently disabled (the sysadmin would
need to do something to recover).

The only reason for using threads here is to get the error recovery
out of an interrupt context (where errors may be detected), and then,
an hour later, decrement a counter (which is how we limit these to
6 per hour). Thread reaping is "trivial", the thread just exits
after an hour.

Since these are events rare, I've no particular concern about
performance or resource consumption. The current code seems
to work just fine. :-)

--linas

Subject: Re: [PATCH] powerpc pseries eeh: Convert to kthread API
Posted by Benjamin Herrenschmid on Tue, 24 Apr 2007 01:38:53 GMT
View Forum Message <> Reply to Message

> The only reason for using threads here is to get the error recovery
> out of an interrupt context (where errors may be detected), and then,
> an hour later, decrement a counter (which is how we limit these to
> 6 per hour). Thread reaping is "trivial", the thread just exits
> after an hour.

In addition, it should be a thread and not done from within keventd
because :

 - It can take a long time (well, relatively but still too long for a
work queue)

 - The driver callbacks might need to use keventd or do flush_workqueue
to synchronize with their own workqueues when doing an internal
recovery.

> Since these are events rare, I've no particular concern about
> performance or resource consumption. The current code seems
> to work just fine. :-)

I think moving to kthread's is cleaner (just a wrapper around kernel
threads that simplify dealing with reaping them out mostly) and I agree
with Christoph that it would be nice to be able to "fire off" kthreads
from interrupt context.. in many cases, we abuse work queues for things
that should really done from kthreads instead (basically anything that
takes more than a couple hundred microsecs or so).

Ben.

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

Subject: Re: [PATCH] powerpc pseries eeh: Convert to kthread API
Posted by ebiederm on Tue, 24 Apr 2007 02:08:42 GMT
View Forum Message <> Reply to Message

Benjamin Herrenschmidt <benh@kernel.crashing.org> writes:

>> The only reason for using threads here is to get the error recovery
>> out of an interrupt context (where errors may be detected), and then,

>> an hour later, decrement a counter (which is how we limit these to
>> 6 per hour). Thread reaping is "trivial", the thread just exits
>> after an hour.
>
> In addition, it should be a thread and not done from within keventd
> because :
>
> - It can take a long time (well, relatively but still too long for a
> work queue)
>
> - The driver callbacks might need to use keventd or do flush_workqueue
> to synchronize with their own workqueues when doing an internal
> recovery.
>
>> Since these are events rare, I've no particular concern about
>> performance or resource consumption. The current code seems
>> to work just fine. :-)
>
> I think moving to kthread's is cleaner (just a wrapper around kernel
> threads that simplify dealing with reaping them out mostly) and I agree
> with Christoph that it would be nice to be able to "fire off" kthreads
> from interrupt context.. in many cases, we abuse work queues for things
> that should really done from kthreads instead (basically anything that
> takes more than a couple hundred microsecs or so).

On that note does anyone have a problem is we manage the irq spawning
safe kthreads the same way that we manage the work queue entries.

i.e. by a structure allocated by the caller?

Eric

Subject: Re: [PATCH] powerpc pseries eeh: Convert to kthread API
Posted by Benjamin Herrenschmid on Tue, 24 Apr 2007 02:42:24 GMT
View Forum Message <> Reply to Message

On Mon, 2007-04-23 at 20:08 -0600, Eric W. Biederman wrote:
> Benjamin Herrenschmidt <benh@kernel.crashing.org> writes:
>
> >> The only reason for using threads here is to get the error recovery
> >> out of an interrupt context (where errors may be detected), and then,
> >> an hour later, decrement a counter (which is how we limit these to
> >> 6 per hour). Thread reaping is "trivial", the thread just exits

> >> after an hour.
> >
> > In addition, it should be a thread and not done from within keventd
> > because :
> >
> > - It can take a long time (well, relatively but still too long for a
> > work queue)
> >
> > - The driver callbacks might need to use keventd or do flush_workqueue
> > to synchronize with their own workqueues when doing an internal
> > recovery.
> >
> >> Since these are events rare, I've no particular concern about
> >> performance or resource consumption. The current code seems
> >> to work just fine. :-)
> >
> > I think moving to kthread's is cleaner (just a wrapper around kernel
> > threads that simplify dealing with reaping them out mostly) and I agree
> > with Christoph that it would be nice to be able to "fire off" kthreads
> > from interrupt context.. in many cases, we abuse work queues for things
> > that should really done from kthreads instead (basically anything that
> > takes more than a couple hundred microsecs or so).
>
> On that note does anyone have a problem is we manage the irq spawning
> safe kthreads the same way that we manage the work queue entries.
>
> i.e. by a structure allocated by the caller?

Not sure... I can see places where I might want to spawn an arbitrary
number of these without having to preallocate structures... and if I
allocate on the fly, then I need a way to free that structure when the
kthread is reaped which I don't think we have currently, do we ? (In
fact, I could use that for other things too now that I'm thinking of
it ... I might have a go at providing optional kthread destructors).

Ben.

Subject: Re: [PATCH] powerpc pseries eeh: Convert to kthread API
Posted by ebiederm on Tue, 24 Apr 2007 03:20:37 GMT
View Forum Message <> Reply to Message

Benjamin Herrenschmidt <benh@kernel.crashing.org> writes:

> Not sure... I can see places where I might want to spawn an arbitrary
> number of these without having to preallocate structures... and if I
> allocate on the fly, then I need a way to free that structure when the
> kthread is reaped which I don't think we have currently, do we ? (In
> fact, I could use that for other things too now that I'm thinking of
> it ... I might have a go at providing optional kthread destructors).

Well the basic problem is that for any piece of code that can be modular
we need a way to ensure all threads it has running are shutdown when we
remove the module.

Which means a fire and forget model however simple is unfortunately
the wrong thing.

Now we might be able to wrap this in some kind of manager construct,
so you don't have to manage each thread individually, but we still
have the problem of ensuring all of the threads exit when we terminate
the module.

Further in general it doesn't make sense to grab a module reference
and call that sufficient because we would like to request that the
module exits.

Eric

Subject: Re: [PATCH] powerpc pseries eeh: Convert to kthread API
Posted by Paul Mackerras on Tue, 24 Apr 2007 04:34:54 GMT
View Forum Message <> Reply to Message

Eric W. Biederman writes:

> Well the basic problem is that for any piece of code that can be modular
> we need a way to ensure all threads it has running are shutdown when we
> remove the module.

The EEH code can't be modular, and wouldn't make any sense to be
modular, since it's part of the infrastructure for accessing PCI
devices.

Paul.

Subject: Re: [PATCH] powerpc pseries eeh: Convert to kthread API
Posted by ebiederm on Tue, 24 Apr 2007 04:51:50 GMT
View Forum Message <> Reply to Message

Paul Mackerras <paulus@samba.org> writes:

> Eric W. Biederman writes:
>
>> Well the basic problem is that for any piece of code that can be modular
>> we need a way to ensure all threads it has running are shutdown when we
>> remove the module.
>
> The EEH code can't be modular, and wouldn't make any sense to be
> modular, since it's part of the infrastructure for accessing PCI
> devices.

Agreed. However most kthread users are modular and make sense to
be so we need to design to handle modular users.

I don't think the idiom of go fire off a thread to handle something
is specific to non-modular users.

Eric

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

Subject: Re: [PATCH] powerpc pseries eeh: Convert to kthread API
Posted by Benjamin Herrenschmid on Tue, 24 Apr 2007 05:00:42 GMT
View Forum Message <> Reply to Message

> Further in general it doesn't make sense to grab a module reference
> and call that sufficient because we would like to request that the
> module exits.

Which is, btw, I think a total misdesign of our module stuff, but heh, I
remember that lead to some flamewars back then...

Like anything else, modules should have separated the entrypoints for

- Initiating a removal request
- Releasing the module

The former is use did "rmmod", can unregister things from subsystems,
etc... (and can file if the driver decides to refuse removal requests
when it's busy doing things or whatever policy that module wants to
implement).

The later is called when all references to the modules have been
dropped, it's a bit like the kref "release" (and could be implemented as
one).

If we had done that (simple) thing back then, module refcounting would
have been much less of a problem... I remember some reasons why that was
veto'ed but I didn't and still don't agree.

Ben.

Subject: Re: [PATCH] powerpc pseries eeh: Convert to kthread API
Posted by ebiederm on Tue, 24 Apr 2007 05:43:55 GMT
View Forum Message <> Reply to Message

Benjamin Herrenschmidt <benh@kernel.crashing.org> writes:

>> Further in general it doesn't make sense to grab a module reference
>> and call that sufficient because we would like to request that the
>> module exits.
>
> Which is, btw, I think a total misdesign of our module stuff, but heh, I
> remember that lead to some flamewars back then...
>
> Like anything else, modules should have separated the entrypoints for
>
> - Initiating a removal request
> - Releasing the module
>
> The former is use did "rmmod", can unregister things from subsystems,
> etc... (and can file if the driver decides to refuse removal requests
> when it's busy doing things or whatever policy that module wants to
> implement).
>

> The later is called when all references to the modules have been
> dropped, it's a bit like the kref "release" (and could be implemented as
> one).
>
> If we had done that (simple) thing back then, module refcounting would
> have been much less of a problem... I remember some reasons why that was
> veto'ed but I didn't and still don't agree.

The basic point is because a thread can terminate sooner if we have an
explicit request to stop, we need that in the design.

Because we need to find the threads to request that they stop we need to
have some way to track them.

Since we need to have some way to track them having an explicit data
structure that the callers manage seems to make sense.

Eric

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: [PATCH] powerpc pseries eeh: Convert to kthread API
Posted by Paul Mackerras on Tue, 24 Apr 2007 05:55:06 GMT
View Forum Message <> Reply to Message

Christoph Hellwig writes:

> The first question is obviously, is this really something we want?
> spawning kernel thread on demand without reaping them properly seems
> quite dangerous.

What specifically has to be done to reap a kernel thread? Are you
concerned about the number of threads, or about having zombies hanging
around?

Paul.

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: [PATCH] powerpc pseries eeh: Convert to kthread API
Posted by Benjamin Herrenschmid on Tue, 24 Apr 2007 05:58:22 GMT

> Since we need to have some way to track them having an explicit data
> structure that the callers manage seems to make sense.

Oh sure, I wasn't arguing against that at all...

It might be handy to have a release() callback (optional) that gets
called after the kthread stops/exits, once we know the data structure
isn't going to be used anymore (if practical to implement, depends on
your approach).

Ben.

Subject: Re: [PATCH] powerpc pseries eeh: Convert to kthread API
Posted by Cornelia Huck on Tue, 24 Apr 2007 07:46:48 GMT

On Tue, 24 Apr 2007 15:00:42 +1000,
Benjamin Herrenschmidt <benh@kernel.crashing.org> wrote:

> Like anything else, modules should have separated the entrypoints for
>
> - Initiating a removal request
> - Releasing the module
>
> The former is use did "rmmod", can unregister things from subsystems,
> etc... (and can file if the driver decides to refuse removal requests
> when it's busy doing things or whatever policy that module wants to
> implement).
>
> The later is called when all references to the modules have been
> dropped, it's a bit like the kref "release" (and could be implemented as
> one).

That sounds quite similar to the problems we have with kobject
refcounting vs. module unloading. The patchset I posted at
http://marc.info/?l=linux-kernel&m=117679014404994&w=2 exposes the
refcount of the kobject embedded in the module. Maybe the kthread code
could use that reference as well?

Subject: Re: [PATCH] powerpc pseries eeh: Convert to kthread API
Posted by Christoph Hellwig on Tue, 24 Apr 2007 08:37:37 GMT
View Forum Message <> Reply to Message

On Tue, Apr 24, 2007 at 03:55:06PM +1000, Paul Mackerras wrote:
> Christoph Hellwig writes:
>
> > The first question is obviously, is this really something we want?
> > spawning kernel thread on demand without reaping them properly seems
> > quite dangerous.
>
> What specifically has to be done to reap a kernel thread? Are you
> concerned about the number of threads, or about having zombies hanging
> around?

I'm mostly concerned about number of threads and possible leakage of
threads. Linas already explained it's not a problem in this case,
so it's covered.

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

Subject: Re: [PATCH] kthread: Spontaneous exit support
Posted by Jan Engelhardt on Tue, 24 Apr 2007 13:08:57 GMT
View Forum Message <> Reply to Message

On Apr 23 2007 12:25, Christoph Hellwig wrote:
>On Sun, Apr 22, 2007 at 09:12:55PM -0600, Eric W. Biederman wrote:
>>
>> This patch implements the kthread helper functions kthread_start
>> and kthread_end which make it simple to support a kernel thread
>> that may decided to exit on it's own before we request it to.
>> It is still assumed that eventually we will get around to requesting
>> that the kernel thread stop.
>
>I don't think having to parallel APIs is a good idea, people will
>get utterly confused which one to use. Better always grab a reference
>in kthread_create and drop it in kthread_stop. For normal thread
>no change in behaviour and only slightly more code in the slowpath.

I *am* already confused... a driver of mine does:

```
static __init int thkd_init(void)
{
 touch_task = kthread_run(touch_thread, Device, "thkd");
 ...
}
```

and

```
static __exit void thkd_exit(void)
{
 kthread_stop(touch_task);
 /* I bet something is missing */
}
```

now what good would kthread_run do me?

Jan
--

Subject: Re: [PATCH] kthread: Spontaneous exit support
Posted by Christoph Hellwig on Tue, 24 Apr 2007 13:34:50 GMT
View Forum Message <> Reply to Message

On Tue, Apr 24, 2007 at 03:08:57PM +0200, Jan Engelhardt wrote:
> >I don't think having to parallel APIs is a good idea, people will
> >get utterly confused which one to use. Better always grab a reference
> >in kthread_create and drop it in kthread_stop. For normal thread
> >no change in behaviour and only slightly more code in the slowpath.
>
> I *am* already confused... a driver of mine does:
>
> static __init int thkd_init(void)
> {
> touch_task = kthread_run(touch_thread, Device, "thkd");
> ...
> }
>
> and
>

> static __exit void thkd_exit(void)
> {
> kthread_stop(touch_task);
> /* I bet something is missing */
> }
>
> now what good would kthread_run do me?

Please read the kerneldoc documentation for kthread_create

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: Getting the new RxRPC patches upstream
Posted by David Howells on Tue, 24 Apr 2007 13:37:04 GMT
View Forum Message <> Reply to Message

Oleg Nesterov <oleg@tv-sign.ru> wrote:

> > > We only care when del_timer() returns true. In that case, if the timer
> > > function still runs (possible for single-threaded wqs), it has already
> > > passed __queue_work().
> >
> > Why do you assume that?

Sorry, I should have been more clear. I meant the assumption that we only
care about a true return from del_timer().

> If del_timer() returns true, the timer was pending. This means it was
> started by work->func() (note that __run_timers() clears timer_pending()
> before calling timer->function). This in turn means that
> delayed_work_timer_fn() has already called __queue_work(dwork), otherwise
> work->func() has no chance to run.

But if del_timer() returns 0, then there may be a problem. We can't tell the
difference between the following two cases:

 (1) The timer hadn't been started.

 (2) The timer had been started, has expired and is no longer pending, but
     another CPU is running its handler routine.

try_to_del_timer_sync() _does_, however, distinguish between these cases: the
first is the 0 return, the second is the -1 return, and the case where it
dequeued the timer is the 1 return.

BTW, can a timer handler be preempted? I assume not... But it can be delayed
by interrupt processing.

David

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: Getting the new RxRPC patches upstream
Posted by Oleg Nesterov on Tue, 24 Apr 2007 14:22:44 GMT
View Forum Message <> Reply to Message

On 04/24, David Howells wrote:
>
> Oleg Nesterov <oleg@tv-sign.ru> wrote:
>
> > > > We only care when del_timer() returns true. In that case, if the timer
> > > > function still runs (possible for single-threaded wqs), it has already
> > > > passed __queue_work().
> > >
> > > Why do you assume that?
>
> Sorry, I should have been more clear. I meant the assumption that we only
> care about a true return from del_timer().
>
> > If del_timer() returns true, the timer was pending. This means it was
> > started by work->func() (note that __run_timers() clears timer_pending()
> > before calling timer->function). This in turn means that
> > delayed_work_timer_fn() has already called __queue_work(dwork), otherwise
> > work->func() has no chance to run.
>
> But if del_timer() returns 0, then there may be a problem. We can't tell the
> difference between the following two cases:
>
> (1) The timer hadn't been started.
>
> (2) The timer had been started, has expired and is no longer pending, but
> another CPU is running its handler routine.
>
> try_to_del_timer_sync() _does_, however, distinguish between these cases: the
> first is the 0 return, the second is the -1 return, and the case where it
> dequeued the timer is the 1 return.

Of course, del_timer() and del_timer_sync() are different. What I meant the
latter buys nothing for cancel_delayed_work() (which in fact could be named
try_to_cancel_delayed_work()).

Let's look at (2). cancel_delayed_work() (on top of del_timer()) returns 0,
and this is correct, we failed to cancel the timer, and we don't know whether
work->func() finished, or not.

The current code uses del_timer_sync(). It will also return 0. However, it will
spin waiting for timer->function() to complete. So we are just wasting CPU.

I guess I misunderstood you. Perhaps, you propose a new helper which use
try_to_del_timer_sync(), yes? Unless I missed something, this doesn't help.
Because the return value == -1 should be treated as 0. We failed to stop
the timer, and we can't free dwork.

IOW, currently we should do:

```
 if (!cancel_delayed_work(dwork))
  cancel_work_sync(dwork));
```

The same if we use del_timer(). If we use try_to_del_timer_sync(),

```
 if (cancel_delayed_work(dwork) <= 0)
  cancel_work_sync(dwork));
```

(of course, dwork shouldn't re-arm itself).

Could you clarify if I misunderstood you again?

> BTW, can a timer handler be preempted? I assume not... But it can be delayed
> by interrupt processing.

No, it can't be preempted, it runs in softirq context.

Oleg.

Subject: Re: Getting the new RxRPC patches upstream
Posted by David Howells on Tue, 24 Apr 2007 15:51:07 GMT
View Forum Message <> Reply to Message

Oleg Nesterov <oleg@tv-sign.ru> wrote:

> Let's look at (2). cancel_delayed_work() (on top of del_timer()) returns 0,
> and this is correct, we failed to cancel the timer, and we don't know whether

> work->func() finished, or not.

Yes.

> The current code uses del_timer_sync(). It will also return 0. However, it
> will spin waiting for timer->function() to complete. So we are just wasting
> CPU.

That's my objection to using cancel_delayed_work() as it stands, although in most cases it's a relatively minor waste of time. However, if the timer expiry routine gets interrupted then it may not be so minor... So, yes, I'm in full agreement with you there.

> I guess I misunderstood you. Perhaps, you propose a new helper which use
> try_to_del_timer_sync(), yes? Unless I missed something, this doesn't help.
> Because the return value == -1 should be treated as 0. We failed to stop
> the timer, and we can't free dwork.

Consider how I'm using try_to_cancel_delayed_work(): I use this when I want to queue a delayed work item with a particular timeout (usually for immediate processing), but the work item may already be pending.

If try_to_cancel_delayed_work() returns 0 or 1 (not pending or pending but dequeued) then I can go ahead and just schedule the work item (I'll be holding a lock to prevent anyone else from interfering).

However, if try_to_cancel_delayed_work() returns -1 then there's no usually no point attempting to schedule the work item because I know the timer expiry handler is doing that or going to do that.

The code looks like this in pretty much all cases:

```
 if (try_to_cancel_delayed_work(&afs_server_reaper) >= 0)
  schedule_delayed_work(&afs_server_reaper, 0);
```

And so could well be packaged into a convenience routine and placed in workqueue.[ch]. However, this would still concern Dave Miller as my patches would still be altering non-net stuff or depending on non-net patches he doesn't have in his GIT tree.

Using cancel_delayer_work() instead would be acceptable, functionally, as that just waits till the -1 return case no longer holds true, and so always returns 0 or 1.

In RxRPC, this is only used to cancel a pair global delayed work items in the rmmod path, and so the inefficiency of cancel_delayed_work() is something I

can live with, though it's something I'd want to reduce in the longer term.

In AFS, this is not only used in object destruction paths, but is also used to
cancel the callback timer and initiate synchronisation processing with
immediate effect.

David

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: Getting the new RxRPC patches upstream
Posted by Oleg Nesterov on Tue, 24 Apr 2007 16:40:01 GMT
View Forum Message <> Reply to Message

On 04/24, David Howells wrote:
>
> Oleg Nesterov <oleg@tv-sign.ru> wrote:
>
> > The current code uses del_timer_sync(). It will also return 0. However, it
> > will spin waiting for timer->function() to complete. So we are just wasting
> > CPU.
>
> That's my objection to using cancel_delayed_work() as it stands, although in
> most cases it's a relatively minor waste of time. However, if the timer
> expiry routine gets interrupted then it may not be so minor... So, yes, I'm
> in full agreement with you there.

Great. I'll send the s/del_timer_sync/del_timer/ patch.

> > I guess I misunderstood you. Perhaps, you propose a new helper which use
> > try_to_del_timer_sync(), yes? Unless I missed something, this doesn't help.
> > Because the return value == -1 should be treated as 0. We failed to stop
> > the timer, and we can't free dwork.
>
> Consider how I'm using try_to_cancel_delayed_work(): I use this when I want to
> queue a delayed work item with a particular timeout (usually for immediate
> processing), but the work item may already be pending.
>
> If try_to_cancel_delayed_work() returns 0 or 1 (not pending or pending but
> dequeued) then I can go ahead and just schedule the work item (I'll be holding
> a lock to prevent anyone else from interfering).
>
> However, if try_to_cancel_delayed_work() returns -1 then there's no usually no
> point attempting to schedule the work item because I know the timer expiry
> handler is doing that or going to do that.

>
>
> The code looks like this in pretty much all cases:
>
> if (try_to_cancel_delayed_work(&afs_server_reaper) >= 0)
>   schedule_delayed_work(&afs_server_reaper, 0);

Aha, now I see what you mean. However. Why the code above is better then

```
 cancel_delayed_work(&afs_server_reaper);
 schedule_delayed_work(&afs_server_reaper, 0);
```

? (I assume we already changed cancel_delayed_work() to use del_timer).

If delayed_work_timer_fn() is not running - both variants (let's denote them
as 1 and 2) do the same.

Now suppose that delayed_work_timer_fn() is running.

 1: lock_timer_base(), return -1, skip schedule_delayed_work().

 2: check timer_pending(), return 0, call schedule_delayed_work(),
    return immediately because test_and_set_bit(WORK_STRUCT_PENDING)
    fails.

So I still don't think try_to_del_timer_sync() can help in this particular
case.

To some extent, try_to_cancel_delayed_work is

```
 int try_to_cancel_delayed_work(dwork)
 {
  ret = cancel_delayed_work(dwork);
  if (!ret && work_pending(&dwork->work))
   ret = -1;
  return ret;
 }
```

iow, work_pending() looks like a more "precise" indication that work->func()
is going to run soon.

Oleg.

Subject: Re: Getting the new RxRPC patches upstream
Posted by David Howells on Tue, 24 Apr 2007 16:58:27 GMT
View Forum Message <> Reply to Message

Oleg Nesterov <oleg@tv-sign.ru> wrote:

> > > The current code uses del_timer_sync(). It will also return 0. However,
> > > it will spin waiting for timer->function() to complete. So we are just
> > > wasting CPU.
> >
> > That's my objection to using cancel_delayed_work() as it stands, although in
> > most cases it's a relatively minor waste of time. However, if the timer
> > expiry routine gets interrupted then it may not be so minor... So, yes, I'm
> > in full agreement with you there.
>
> Great. I'll send the s/del_timer_sync/del_timer/ patch.

I didn't say I necessarily agreed that this was a good idea. I just meant that
I agree that it will waste CPU. You must still audit all uses of
cancel_delayed_work().

> Aha, now I see what you mean. However. Why the code above is better then
>
> cancel_delayed_work(&afs_server_reaper);
> schedule_delayed_work(&afs_server_reaper, 0);
>
> ? (I assume we already changed cancel_delayed_work() to use del_timer).

Because calling schedule_delayed_work() is a waste of CPU if the timer expiry
handler is currently running at this time as *that* is going to also schedule
the delayed work item.

> If delayed_work_timer_fn() is not running - both variants (let's denote them
> as 1 and 2) do the same.

Yes, but that's not the point.

> Now suppose that delayed_work_timer_fn() is running.
>
> 1: lock_timer_base(), return -1, skip schedule_delayed_work().
>
> 2: check timer_pending(), return 0, call schedule_delayed_work(),
> return immediately because test_and_set_bit(WORK_STRUCT_PENDING)
> fails.

I don't see what you're illustrating here. Are these meant to be two steps in
a single process? Or are they two alternate steps?

> So I still don't think try_to_del_timer_sync() can help in this particular

> case.

It permits us to avoid the test_and_set_bit() under some circumstances.

> To some extent, try_to_cancel_delayed_work is
>
> int try_to_cancel_delayed_work(dwork)
> {
>   ret = cancel_delayed_work(dwork);
>   if (!ret && work_pending(&dwork->work))
>   ret = -1;
>   return ret;
> }
>
> iow, work_pending() looks like a more "precise" indication that work->func()
> is going to run soon.

Ah, but the timer routine may try to set the work item pending flag *after* the work_pending() check you have here. Furthermore, it would be better to avoid the work_pending() check entirely because that check involves interacting with atomic ops done on other CPUs. try_to_del_timer_sync() returning -1 tells us without a shadow of a doubt that the work item is either scheduled now or will be scheduled very shortly, thus allowing us to avoid having to do it ourself.

David

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: [PATCH] powerpc pseries eeh: Convert to kthread API
Posted by linas on Tue, 24 Apr 2007 17:24:48 GMT
View Forum Message <> Reply to Message

On Tue, Apr 24, 2007 at 11:38:53AM +1000, Benjamin Herrenschmidt wrote:
> > The only reason for using threads here is to get the error recovery
> > out of an interrupt context (where errors may be detected), and then,
> > an hour later, decrement a counter (which is how we limit these to
> > 6 per hour). Thread reaping is "trivial", the thread just exits
> > after an hour.
>
> In addition, it should be a thread and not done from within keventd
> because :
>
> - It can take a long time (well, relatively but still too long for a
> work queue)

Uhh, 15 or 20 seconds even. That's a long time by any kernel standard.

> - The driver callbacks might need to use keventd or do flush_workqueue
> to synchronize with their own workqueues when doing an internal
> recovery.
>
> > Since these are events rare, I've no particular concern about
> > performance or resource consumption. The current code seems
> > to work just fine. :-)
>
> I think moving to kthread's is cleaner (just a wrapper around kernel
> threads that simplify dealing with reaping them out mostly) and I agree
> with Christoph that it would be nice to be able to "fire off" kthreads
> from interrupt context.. in many cases, we abuse work queues for things
> that should really done from kthreads instead (basically anything that
> takes more than a couple hundred microsecs or so).

It would be nice to have threads that can be "fired off" from an
interrupt context. That would simplify the EEH code slightly
(removing a few dozen lines of code that do this bounce).

I presume that various device drivers might find this useful as well.

--linas

Subject: Re: Getting the new RxRPC patches upstream
Posted by Oleg Nesterov on Tue, 24 Apr 2007 17:33:29 GMT
View Forum Message <> Reply to Message

On 04/24, David Howells wrote:
>
> Oleg Nesterov <oleg@tv-sign.ru> wrote:
>
> > Great. I'll send the s/del_timer_sync/del_timer/ patch.
>
> I didn't say I necessarily agreed that this was a good idea. I just meant that
> I agree that it will waste CPU. You must still audit all uses of
> cancel_delayed_work().

Sure, I'll grep for cancel_delayed_work(). But unless I missed something,
this change should be completely transparent for all users. Otherwise, it
is buggy.

> > Aha, now I see what you mean. However. Why the code above is better then
> >
> > cancel_delayed_work(&afs_server_reaper);
> > schedule_delayed_work(&afs_server_reaper, 0);
> >
> > ? (I assume we already changed cancel_delayed_work() to use del_timer).
>
> Because calling schedule_delayed_work() is a waste of CPU if the timer expiry
> handler is currently running at this time as *that* is going to also schedule
> the delayed work item.

Yes. But otoh, try_to_del_timer_sync() is a waste of CPU compared to del_timer(),
when the timer is not pending.

> > 1: lock_timer_base(), return -1, skip schedule_delayed_work().
> >
> > 2: check timer_pending(), return 0, call schedule_delayed_work(),
> >   return immediately because test_and_set_bit(WORK_STRUCT_PENDING)
> >   fails.
>
> I don't see what you're illustrating here. Are these meant to be two steps in
> a single process? Or are they two alternate steps?

two alternate steps.

1 means
```
 if (try_to_cancel_delayed_work())
  schedule_delayed_work();
```

2 means
```
 cancel_delayed_work();
 schedule_delayed_work();
```

> > So I still don't think try_to_del_timer_sync() can help in this particular
> > case.
>
> It permits us to avoid the test_and_set_bit() under some circumstances.

Yes. But lock_timer_base() is more costly.

> > To some extent, try_to_cancel_delayed_work is
> >
> > int try_to_cancel_delayed_work(dwork)
> > {
> >   ret = cancel_delayed_work(dwork);
> >   if (!ret && work_pending(&dwork->work))
> >   ret = -1;

> >   return ret;
> > }
> >
> > iow, work_pending() looks like a more "precise" indication that work->func()
> > is going to run soon.
>
> Ah, but the timer routine may try to set the work item pending flag *after* the
> work_pending() check you have here.

No, delayed_work_timer_fn() doesn't set the _PENDING flag.

>   Furthermore, it would be better to avoid
> the work_pending() check entirely because that check involves interacting with
> atomic ops done on other CPUs.

Sure, the implementation of try_to_cancel_delayed_work() above is just for
illustration. I don't think we need try_to_cancel_delayed_work() at all.

>   try_to_del_timer_sync() returning -1 tells us
> without a shadow of a doubt that the work item is either scheduled now or will
> be scheduled very shortly, thus allowing us to avoid having to do it ourself.

First, this is very unlikely event, delayed_work_timer_fn() is very fast unless
interrupted.

_PENDING flag won't be cleared until this work is executed by run_workqueue().
In generak, work_pending() after del_timer() is imho better way to avoid the
unneeded schedule_delayed_work().

But again, I can't undertand the win for that particular case.

Oleg.

Subject: Re: [PATCH] powerpc pseries eeh: Convert to kthread API
Posted by linas on Tue, 24 Apr 2007 17:35:42 GMT
View Forum Message <> Reply to Message

On Thu, Apr 19, 2007 at 01:58:45AM -0600, Eric W. Biederman wrote:
> From: Eric W. Biederman <ebiederm@xmission.com>
>
> This patch modifies the startup of eehd to use kthread_run
> not a combination of kernel_thread and daemonize. Making

> the code slightly simpler and more maintainable.

For the patch that touched arch/powerpc/platforms/pseries/eeh_event.c,
I ran a variety of tests, and couldn't see/find/evoke any adverse
effects, so ..

Acked-by: Linas Vepstas <linas@austin.ibm.com>

> The second question is whether this is the right implementation.
> kthread_create already works by using a workqueue to create the thread
> and then waits for it. If we really want to support creating threads
> asynchronously on demand we should have a proper API in kthread.c for
> this instead of spreading workqueues.

Yes, exactly; all I really want is to start a thread from an
interrupt context, and pass a structure to it. This is pretty much
all that arch/powerpc/platforms/pseries/eeh_event.c is trying to do,
and little else.

--linas

Subject: Re: Getting the new RxRPC patches upstream
Posted by David Howells on Tue, 24 Apr 2007 18:22:50 GMT
View Forum Message <> Reply to Message

Oleg Nesterov <oleg@tv-sign.ru> wrote:

> Sure, I'll grep for cancel_delayed_work(). But unless I missed something,
> this change should be completely transparent for all users. Otherwise, it
> is buggy.

I guess you will have to make sure that cancel_delayed_work() is always
followed by a flush of the workqueue, otherwise you might get this situation:

```
 CPU 0    CPU 1
 ============================== ======================
    <timer expires>
cancel_delayed_work(x) == 0 -->delayed_work_timer_fn(x)
kfree(x);   -->do_IRQ()
y = kmalloc(); // reuses x
    <--do_IRQ()
    __queue_work(x)
```

--- OOPS ---

That's my main concern. If you are certain that can't happen, then fair enough.

Note that although you can call cancel_delayed_work() from within a work item handler, you can't then follow it up with a flush as it's very likely to deadlock.

> > Because calling schedule_delayed_work() is a waste of CPU if the timer
> > expiry handler is currently running at this time as *that* is going to
> > also schedule the delayed work item.
>
> Yes. But otoh, try_to_del_timer_sync() is a waste of CPU compared to
> del_timer(), when the timer is not pending.

I suppose that's true. As previously stated, my main objection to del_timer() is the fact that it doesn't tell you if the timer expiry function is still running.

Can you show me a patch illustrating exactly how you want to change cancel_delayed_work()? I can't remember whether you've done so already, but if you have, I can't find it. Is it basically this?:

```
 static inline int cancel_delayed_work(struct delayed_work *work)
 {
  int ret;

- ret = del_timer_sync(&work->timer);
+ ret = del_timer(&work->timer);
  if (ret)
   work_release(&work->work);
  return ret;
 }
```

I was thinking this situation might be a problem:

```
 CPU 0    CPU 1
 ============================== ======================
     <timer expires>
 cancel_delayed_work(x) == 0 -->delayed_work_timer_fn(x)
 schedule_delayed_work(x,0) -->do_IRQ()
 <keventd scheduled>
 x->work()
     <--do_IRQ()
     __queue_work(x)
```

But it won't, will it?

> > Ah, but the timer routine may try to set the work item pending flag
> > *after* the work_pending() check you have here.
>
> No, delayed_work_timer_fn() doesn't set the _PENDING flag.

Good point. I don't think that's a problem because cancel_delayed_work() won't clear the pending flag if it didn't remove a timer.

> First, this is very unlikely event, delayed_work_timer_fn() is very fast
> unless interrupted.

Yeah, I guess so.

Okay, you've convinced me, I think - provided you consider the case I outlinded at the top of this email.

If you give me a patch to alter cancel_delayed_work(), I'll substitute it for mine and use that that instead. Dave Miller will just have to live with that patch being there:-)

David

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: Getting the new RxRPC patches upstream
Posted by Oleg Nesterov on Tue, 24 Apr 2007 19:34:04 GMT
View Forum Message <> Reply to Message

On 04/24, David Howells wrote:
>
> Oleg Nesterov <oleg@tv-sign.ru> wrote:
>
> > Sure, I'll grep for cancel_delayed_work(). But unless I missed something,
> > this change should be completely transparent for all users. Otherwise, it
> > is buggy.
>
> I guess you will have to make sure that cancel_delayed_work() is always
> followed by a flush of the workqueue, otherwise you might get this situation:
>
> CPU 0 CPU 1
> ============================== ======================
> <timer expires>
> cancel_delayed_work(x) == 0 -->delayed_work_timer_fn(x)

> kfree(x);   -->do_IRQ()
> y = kmalloc(); // reuses x
>   <--do_IRQ()
>   __queue_work(x)
> --- OOPS ---
>
> That's my main concern. If you are certain that can't happen, then fair
> enough.

Yes sure. Note that this is documented:

 /*
  * Kill off a pending schedule_delayed_work(). Note that the work callback
  * function may still be running on return from cancel_delayed_work(). Run
  * flush_workqueue() or cancel_work_sync() to wait on it.
  */

This comment is not very precise though. If the work doesn't re-arm itself,
we need cancel_work_sync() only if cancel_delayed_work() returns 0.

So there is no difference with the proposed change. Except, return value == 0
means:

 currently (del_timer_sync): callback may still be running or scheduled

 with del_timer: may still be running, or scheduled, or will be scheduled
 right now.

However, this is the same from the caller POV.

> Can you show me a patch illustrating exactly how you want to change
> cancel_delayed_work()? I can't remember whether you've done so already, but
> if you have, I can't find it. Is it basically this?:
>
> static inline int cancel_delayed_work(struct delayed_work *work)
> {
>   int ret;
>
> - ret = del_timer_sync(&work->timer);
> + ret = del_timer(&work->timer);
>   if (ret)
>   work_release(&work->work);
>   return ret;
> }

Yes, exactly. The patch is trivial, but I need some time to write the
understandable changelog...

> I was thinking this situation might be a problem:
>
> CPU 0   CPU 1
> ============================== ======================
>   <timer expires>
> cancel_delayed_work(x) == 0 -->delayed_work_timer_fn(x)
> schedule_delayed_work(x,0) -->do_IRQ()
> <keventd scheduled>
> x->work()
>   <--do_IRQ()
>   __queue_work(x)
>
> But it won't, will it?

Yes, I think this should be OK. schedule_delayed_work() will notice
_PENDING and abort, so the last "x->work()" doesn't happen.

What can happen is

```
    <timer expires>
 cancel_delayed_work(x) == 0
    -->delayed_work_timer_fn(x)
    __queue_work(x)
    <keventd scheduled>
    x->work()
 schedule_delayed_work(x,0)
 <the work is scheduled again>
```

, so we can have an "unneeded schedule", but this is very unlikely.

Oleg.

Subject: Re: Getting the new RxRPC patches upstream
Posted by David Howells on Wed, 25 Apr 2007 08:10:12 GMT
View Forum Message <> Reply to Message

Oleg Nesterov <oleg@tv-sign.ru> wrote:

> Yes sure. Note that this is documented:
>
> /*
>   * Kill off a pending schedule_delayed_work(). Note that the work callback

> * function may still be running on return from cancel_delayed_work(). Run
> * flush_workqueue() or cancel_work_sync() to wait on it.
> */

No, it isn't documented. It says that the *work* callback may be running, but
does not mention the timer callback. However, just looking at the
cancellation function source made it clear that this would wait for the timer
handler to return first.

However, is it worth just making cancel_delayed_work() a void function and not
returning anything? I'm not sure the return value is very useful.

David

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: Getting the new RxRPC patches upstream
Posted by Oleg Nesterov on Wed, 25 Apr 2007 10:41:45 GMT
View Forum Message <> Reply to Message

On 04/25, David Howells wrote:
>
> Oleg Nesterov <oleg@tv-sign.ru> wrote:
>
> > Yes sure. Note that this is documented:
> >
> > /*
> > * Kill off a pending schedule_delayed_work(). Note that the work callback
> > * function may still be running on return from cancel_delayed_work(). Run
> > * flush_workqueue() or cancel_work_sync() to wait on it.
> > */
>
> No, it isn't documented. It says that the *work* callback may be running, but
> does not mention the timer callback. However, just looking at the
> cancellation function source made it clear that this would wait for the timer
> handler to return first.

Ah yes, it says nothing about what the returned value means...

> However, is it worth just making cancel_delayed_work() a void function and not
> returning anything? I'm not sure the return value is very useful.

cancel_rearming_delayed_work() needs this, tty_io.c, probably somebody else.

Oleg.

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: Getting the new RxRPC patches upstream
Posted by David Howells on Wed, 25 Apr 2007 10:45:44 GMT
View Forum Message <> Reply to Message

Oleg Nesterov <oleg@tv-sign.ru> wrote:

> Ah yes, it says nothing about what the returned value means...

Yeah... If you could amend that as part of your patch, that'd be great.

David

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: Getting the new RxRPC patches upstream
Posted by David Howells on Wed, 25 Apr 2007 13:48:39 GMT
View Forum Message <> Reply to Message

David Miller <davem@davemloft.net> wrote:

> Is it possible for your changes to be purely networking
> and not need those changes outside of the networking?

See my latest patchset release. I've reduced the dependencies on
non-networking changes to:

(1) Oleg Nesterov's patch to change cancel_delayed_work() to use del_timer()
rather than del_timer_sync() [patch 02/16].

This patch can be discarded without compilation failure at the expense of
making AFS slightly less efficient. It also makes AF_RXRPC slightly less
efficient, but only in the rmmod path.

(2) A symbol export in the keyring stuff plus a proliferation of the types
available in the struct key::type_data union [patch 03/16]. This does
not conflict with any other patches that I know about.

(3) A symbol export in the timer stuff [patch 04/16].

Everything else that remains after the reduction is confined to the AF_RXRPC or AFS code, save for a couple of networking patches in my patchset that you already have and I just need to make the thing compile.

I'm not sure that I can make the AF_RXRPC patches totally independent of the AFS patches as the two sets need to interleave since the last AF_RXRPC patch deletes the old RxRPC code - which the old AFS code depends on.

David

____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: [PATCH] ia64 sn xpc: Convert to use kthread API.
Posted by Dean Nelson on Thu, 26 Apr 2007 20:00:47 GMT
View Forum Message <> Reply to Message

On Thu, Apr 19, 2007 at 01:58:44AM -0600, Eric W. Biederman wrote:
> From: Eric W. Biederman <ebiederm@xmission.com>
>
> This patch starts the xpc kernel threads using kthread_run
> not a combination of kernel_thread and daemonize. Resuling
> in slightly simpler and more maintainable code.
>
> Cc: Jes Sorensen <jes@sgi.com>
> Cc: Tony Luck <tony.luck@intel.com>
> Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
> ---
> arch/ia64/sn/kernel/xpc_main.c | 31 +++++++++++++------------------
> 1 files changed, 13 insertions(+), 18 deletions(-)

Acked-by: Dean Nelson <dcn@sgi.com>

Andrew, I've tested Eric's patch in conjunction with a fix from Christoph Lameter, which you've already got (it cleaned up a couple of compiler errors), and the following patch that I'd like added (it cleans up a couple of compiler warning errors and makes a few cosmetic changes).

Thanks,
Dean

Signed-off-by: Dean Nelson <dcn@sgi.com>

```
Index: mm-tree/arch/ia64/sn/kernel/xpc_main.c
===================================================================
--- mm-tree.orig/arch/ia64/sn/kernel/xpc_main.c 2007-04-25 14:04:51.701213426 -0500
+++ mm-tree/arch/ia64/sn/kernel/xpc_main.c 2007-04-26 06:29:02.447330438 -0500
@@ -568,7 +568,6 @@

  task = kthread_run(xpc_activating, (void *) ((u64) partid),
      "xpc%02d", partid);
-
  if (unlikely(IS_ERR(task))) {
   spin_lock_irqsave(&part->act_lock, irq_flags);
   part->act_state = XPC_P_INACTIVE;
@@ -808,7 +807,6 @@
    int ignore_disconnecting)
 {
  unsigned long irq_flags;
- pid_t pid;
  u64 args = XPC_PACK_ARGS(ch->partid, ch->number);
  struct xpc_partition *part = &xpc_partitions[ch->partid];
  struct task_struct *task;
@@ -840,7 +838,7 @@
   (void) xpc_part_ref(part);
   xpc_msgqueue_ref(ch);

-  task = kthread_run(xpc_daemonize_kthread, args,
+  task = kthread_run(xpc_daemonize_kthread, (void *) args,
       "xpc%02dc%d", ch->partid, ch->number);
   if (IS_ERR(task)) {
    /* the fork failed */
@@ -1381,7 +1379,8 @@
   * activate based on info provided by SAL. This new thread is short
   * lived and will exit once discovery is complete.
   */
- task = kthread_run(xpc_initiate_discovery, NULL, XPC_DISCOVERY_THREAD_NAME);
+ task = kthread_run(xpc_initiate_discovery, NULL,
+      XPC_DISCOVERY_THREAD_NAME);
  if (IS_ERR(task)) {
   dev_err(xpc_part, "failed while forking discovery thread\n");
```

Subject: Re: [PATCH] ia64 sn xpc: Convert to use kthread API.

Posted by Dean Nelson on Fri, 27 Apr 2007 17:41:42 GMT
View Forum Message <> Reply to Message

On Thu, Apr 19, 2007 at 04:51:03PM -0700, Andrew Morton wrote:
> Another driver which should be fully converted to the kthread API:
> kthread_stop() and kthread_should_stop().
>
> And according to my logs, this driver was added to the tree more than
> a year _after_ the kthread interface was made available.
>
> This isn't good.

On Sun, Apr 22, 2007 at 09:36:47PM +0100, Christoph Hellwig wrote:
> On Thu, Apr 19, 2007 at 01:58:44AM -0600, Eric W. Biederman wrote:
> > From: Eric W. Biederman <ebiederm@xmission.com>
> >
> > This patch starts the xpc kernel threads using kthread_run
> > not a combination of kernel_thread and daemonize. Resuling
> > in slightly simpler and more maintainable code.
>
> This driver is a really twisted maze. It has a lot of threads,
> some of them running through the whole lifetime of the driver,
> some short-lived and some in a sort of a pool.
>
> The patch below fixes up the long-lived thread as well as fixing
> gazillions of leaks in the init routine by switching to proper
> goto-based unwinding.

I see that the setting of 'xpc_rsvd_page->vars_pa = 0;' in xpc_init() is considered a leak by Christoph (hch), but it really is not. If you look at xpc_rsvd_page_init() where it is set up, you might see that the reserved page is something XPC gets from SAL who created it at system boot time. If XPC is rmmod'd and insmod'd again, it will be handed the same page of memory by SAL. So there is no memory leak here.

As for the other suspected leaks mentioned I'm not sure what they could be. It may be the fact that XPC continues coming up when presented with error returns from register_reboot_notifier() and register_die_notifier(). There is no leak in this, just simply a loss of an early notification to other SGI system partitions that this partition is going down. A fact they will sooner or later discover on their own. A notice of this degraded functionality is written to the console. And the likelyhood that these functions should ever return an error is very, very small (currently, neither of them has an error return).

>From my experience the goto-based unwinding of error returns is not necessarily a superior approach. I spent a month tracking down a difficult to reproduce problem that ended up being an error return jumping to the wrong label in a goto-based unwind. Problems can arise with either approach. I've also seen

the compiler generate less code for the non-goto approach. I'm not a compiler person so I can't explain this, nor can I say that it's always the case, but at one time when I did a bake off between the two approaches the non-goto approach generated less code.

Having said this I have no problem with switching to a goto-based unwinding of errors if that is what the community prefers. I personally find it more readable than the non-goto approach.

> Note that thread pools are something we have in a few places,
> and might be worth handling in the core kthread infrastructure,
> as tearing down pools will get a bit complicated using the
> kthread APIs.

Christoph is correct in that XPC has a single thread that exists throughout its lifetime, another set of threads that exist for the time that active contact with other XPCs running on other SGI system partitions exists, and finally there is a pool of threads that exist on an as needed basis once a channel connection has been established between two partitions.

In principle I approve of the kthread API and its use as opposed to what XPC currently does (calls kernel_thread(), daemonize(), wait_for_completion(), and complete()). So Christoph's patch that changes the single long-lived thread to use kthread_stop() and kthread_should_stop() is appreciated.

But the fact that another thread, started at the xpc_init() time, that does discovery of other SGI system partitions wasn't converted points out a weekness in either my thinking or the kthread API. This discovery thread does its job and then exits. Should XPC be rmmod'd while the discovery thread is still running we would need to do a kthread_stop() against it. But kthread_stop() isn't set up to deal with a task that has already exited. And if what once was the task structure of this exited task has been reallocated to another new task, we'd end up stopping it should it be operating under the kthread API, or possibly waiting a very long time for it to exit if it is not.

I'm also a little uneasy that kthread_stop() has an "only one thread can stop another thread at a time" design. It's a potential bottleneck on very large systems where threads are blocked and unable to respond to a kthread_should_stop() for some period of time.

XPC is in need of threads that can block indefinitely, which is why XPC is in the business of maintaining a pool of threads. Currently there is no such capability (that I know of) that is provided by linux. Workqueues can't block indefinitely.

And for performance reasons these threads need to be able to be created quickly. These threads are involved in delivering messages to XPC's users

(like XPNET) and we had latency issues that led us to use kernel_thread()
directly instead of the kthread API. Additionally, XPC may need to have
hundreds of these threads active at any given time.

I think it would be great if the kthread API (or underlying implementation)
could be changed to handle these issues. I'd love for XPC to not have to
maintain this sort of thing itself.

Dean

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: [PATCH] ia64 sn xpc: Convert to use kthread API.
Posted by ebiederm on Fri, 27 Apr 2007 18:34:02 GMT
View Forum Message <> Reply to Message

Dean Nelson <dcn@sgi.com> writes:
>
> Christoph is correct in that XPC has a single thread that exists throughout
> its lifetime, another set of threads that exist for the time that active
> contact with other XPCs running on other SGI system partitions exists, and
> finally there is a pool of threads that exist on an as needed basis once
> a channel connection has been established between two partitions.
>
> In principle I approve of the kthread API and its use as opposed to what
> XPC currently does (calls kernel_thread(), daemonize(), wait_for_completion(),
> and complete()). So Christoph's patch that changes the single long-lived
> thread to use kthread_stop() and kthread_should_stop() is appreciated.
>
> But the fact that another thread, started at the xpc_init() time, that does
> discovery of other SGI system partitions wasn't converted points out a
> weekness in either my thinking or the kthread API. This discovery thread
> does its job and then exits. Should XPC be rmmod'd while the discovery
> thread is still running we would need to do a kthread_stop() against it.
> But kthread_stop() isn't set up to deal with a task that has already exited.
> And if what once was the task structure of this exited task has been
> reallocated to another new task, we'd end up stopping it should it be
> operating under the kthread API, or possibly waiting a very long time
> for it to exit if it is not.

Patches are currently under development to allow kthreads to exit
before kthread_stop is called. The big thing is that once we allow
kernel threads that exited by themselves to be reaped by kthread_stop
we have some significant work to do.

> I'm also a little uneasy that kthread_stop() has an "only one thread can
> stop another thread at a time" design. It's a potential bottleneck on
> very large systems where threads are blocked and unable to respond to a
> kthread_should_stop() for some period of time.

There are already patches out there to fix this issue.

> XPC is in need of threads that can block indefinitely, which is why XPC
> is in the business of maintaining a pool of threads. Currently there is
> no such capability (that I know of) that is provided by linux. Workqueues
> can't block indefinitely.

I'm not certain I understand this requirement. Do you mean block indefinitely
unless requested to stop?

> And for performance reasons these threads need to be able to be created
> quickly. These threads are involved in delivering messages to XPC's users
> (like XPNET) and we had latency issues that led us to use kernel_thread()
> directly instead of the kthread API. Additionally, XPC may need to have
> hundreds of these threads active at any given time.

Ugh. Can you tell me a little more about the latency issues?

Is having a non-halting kthread_create enough to fix this?
So you don't have to context switch several times to get the
thread running?

Or do you need more severe latency reductions?

The more severe fix would require some significant changes to copy_process
and every architecture would need to be touched to fix up copy_thread.
It is possible, it is a lot of work, and the reward is far from obvious.

> I think it would be great if the kthread API (or underlying implementation)
> could be changed to handle these issues. I'd love for XPC to not have to
> maintain this sort of thing itself.

Currently daemonize is a serious maintenance problem.

Using daemonize and kernel_thread to create kernel threads is a blocker
in implementing the pid namespace because of their use of pid_t.

So I am motivated to get this fixed.

Eric

_____

Containers mailing list
Containers@lists.linux-foundation.org

Subject: Re: [PATCH] ia64 sn xpc: Convert to use kthread API.
Posted by Dean Nelson on Fri, 27 Apr 2007 20:12:11 GMT
View Forum Message <> Reply to Message

On Fri, Apr 27, 2007 at 12:34:02PM -0600, Eric W. Biederman wrote:
> Dean Nelson <dcn@sgi.com> writes:
> >
> > Christoph is correct in that XPC has a single thread that exists throughout
> > its lifetime, another set of threads that exist for the time that active
> > contact with other XPCs running on other SGI system partitions exists, and
> > finally there is a pool of threads that exist on an as needed basis once
> > a channel connection has been established between two partitions.
> >
> > In principle I approve of the kthread API and its use as opposed to what
> > XPC currently does (calls kernel_thread(), daemonize(), wait_for_completion(),
> > and complete()). So Christoph's patch that changes the single long-lived
> > thread to use kthread_stop() and kthread_should_stop() is appreciated.
> >
> > But the fact that another thread, started at the xpc_init() time, that does
> > discovery of other SGI system partitions wasn't converted points out a
> > weekness in either my thinking or the kthread API. This discovery thread
> > does its job and then exits. Should XPC be rmmod'd while the discovery
> > thread is still running we would need to do a kthread_stop() against it.
> > But kthread_stop() isn't set up to deal with a task that has already exited.
> > And if what once was the task structure of this exited task has been
> > reallocated to another new task, we'd end up stopping it should it be
> > operating under the kthread API, or possibly waiting a very long time
> > for it to exit if it is not.
>
> Patches are currently under development to allow kthreads to exit
> before kthread_stop is called. The big thing is that once we allow
> kernel threads that exited by themselves to be reaped by kthread_stop
> we have some significant work to do.
>
> > I'm also a little uneasy that kthread_stop() has an "only one thread can
> > stop another thread at a time" design. It's a potential bottleneck on
> > very large systems where threads are blocked and unable to respond to a
> > kthread_should_stop() for some period of time.
>
> There are already patches out there to fix this issue.
>
> > XPC is in need of threads that can block indefinitely, which is why XPC
> > is in the business of maintaining a pool of threads. Currently there is
> > no such capability (that I know of) that is provided by linux. Workqueues
> > can't block indefinitely.

>
> I'm not certain I understand this requirement. Do you mean block indefinitely
> unless requested to stop?

These threads can block waiting for a hardware DMA engine, which has a 28
second timeout setpoint.

> > And for performance reasons these threads need to be able to be created
> > quickly. These threads are involved in delivering messages to XPC's users
> > (like XPNET) and we had latency issues that led us to use kernel_thread()
> > directly instead of the kthread API. Additionally, XPC may need to have
> > hundreds of these threads active at any given time.
>
> Ugh. Can you tell me a little more about the latency issues?

After placing a message in a local message queue, one SGI system partition
will interrupt another to retrieve the message. We need to minimize the
time from entering XPC's interrupt handler to the time that the message
can be DMA transferred and delivered to the consumer (like XPNET) to
whom it was sent.

> Is having a non-halting kthread_create enough to fix this?
> So you don't have to context switch several times to get the
> thread running?
>
> Or do you need more severe latency reductions?
>
> The more severe fix would require some significant changes to copy_process
> and every architecture would need to be touched to fix up copy_thread.
> It is possible, it is a lot of work, and the reward is far from obvious.

I think a non-halting kthread_create() should be sufficient. It is in
effect what XPC has now in calling kernel_thread() directly.

Taking it one step further, if you added the notion of a thread pool,
where upon exit, a thread isn't destroyed but rather is queued ready to
handle the next kthread_create_quick() request.

> > I think it would be great if the kthread API (or underlying implementation)
> > could be changed to handle these issues. I'd love for XPC to not have to
> > maintain this sort of thing itself.
>
> Currently daemonize is a serious maintenance problem.
>
> Using daemonize and kernel_thread to create kernel threads is a blocker
> in implementing the pid namespace because of their use of pid_t.
>
> So I am motivated to get this fixed.

This would also address the problems we see with huge pid spaces for
kernel threads on our largest machines. In the example from last week,
we had 10 threads each on 4096 cpus. If we reworked work_queues to use
the kthread_create_nonblocking() thread pool, we could probably collapse
the need for having all of those per-task, per-cpu work queues.

Dean

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: [PATCH] ia64 sn xpc: Convert to use kthread API.
Posted by ebiederm on Fri, 27 Apr 2007 20:33:32 GMT
View Forum Message <> Reply to Message

Dean Nelson <dcn@sgi.com> writes:

> On Fri, Apr 27, 2007 at 12:34:02PM -0600, Eric W. Biederman wrote:
>> Dean Nelson <dcn@sgi.com> writes:
>> >
>> > Christoph is correct in that XPC has a single thread that exists throughout
>> > its lifetime, another set of threads that exist for the time that active
>> > contact with other XPCs running on other SGI system partitions exists, and
>> > finally there is a pool of threads that exist on an as needed basis once
>> > a channel connection has been established between two partitions.
>> >
>> > In principle I approve of the kthread API and its use as opposed to what
>> > XPC currently does (calls kernel_thread(), daemonize(),
> wait_for_completion(),
>> > and complete()). So Christoph's patch that changes the single long-lived
>> > thread to use kthread_stop() and kthread_should_stop() is appreciated.
>> >
>> > But the fact that another thread, started at the xpc_init() time, that does
>> > discovery of other SGI system partitions wasn't converted points out a
>> > weekness in either my thinking or the kthread API. This discovery thread
>> > does its job and then exits. Should XPC be rmmod'd while the discovery
>> > thread is still running we would need to do a kthread_stop() against it.
>> > But kthread_stop() isn't set up to deal with a task that has already exited.
>> > And if what once was the task structure of this exited task has been
>> > reallocated to another new task, we'd end up stopping it should it be
>> > operating under the kthread API, or possibly waiting a very long time
>> > for it to exit if it is not.
>>
>> Patches are currently under development to allow kthreads to exit
>> before kthread_stop is called. The big thing is that once we allow

>> kernel threads that exited by themselves to be reaped by kthread_stop
>> we have some significant work to do.
>>
>> > I'm also a little uneasy that kthread_stop() has an "only one thread can
>> > stop another thread at a time" design. It's a potential bottleneck on
>> > very large systems where threads are blocked and unable to respond to a
>> > kthread_should_stop() for some period of time.
>>
>> There are already patches out there to fix this issue.
>>
>> > XPC is in need of threads that can block indefinitely, which is why XPC
>> > is in the business of maintaining a pool of threads. Currently there is
>> > no such capability (that I know of) that is provided by linux. Workqueues
>> > can't block indefinitely.
>>
>> I'm not certain I understand this requirement. Do you mean block indefinitely
>> unless requested to stop?
>
> These threads can block waiting for a hardware DMA engine, which has a 28
> second timeout setpoint.

Ok. So this is an interruptible sleep?
Do you have any problems being woken up out of that interruptible sleep
by kthread_stop?

I am in the process of modifying kthread_stop to wake up thread in an
interruptible sleep and set signal_pending, so they will break out.

>> > And for performance reasons these threads need to be able to be created
>> > quickly. These threads are involved in delivering messages to XPC's users
>> > (like XPNET) and we had latency issues that led us to use kernel_thread()
>> > directly instead of the kthread API. Additionally, XPC may need to have
>> > hundreds of these threads active at any given time.
>>
>> Ugh. Can you tell me a little more about the latency issues?
>
> After placing a message in a local message queue, one SGI system partition
> will interrupt another to retrieve the message. We need to minimize the
> time from entering XPC's interrupt handler to the time that the message
> can be DMA transferred and delivered to the consumer (like XPNET) to
> whom it was sent.
>
>> Is having a non-halting kthread_create enough to fix this?
>> So you don't have to context switch several times to get the
>> thread running?
>>
>> Or do you need more severe latency reductions?
>>

>> The more severe fix would require some significant changes to copy_process
>> and every architecture would need to be touched to fix up copy_thread.
>> It is possible, it is a lot of work, and the reward is far from obvious.
>
> I think a non-halting kthread_create() should be sufficient. It is in
> effect what XPC has now in calling kernel_thread() directly.

A little different but pretty close.

We call kthread_create() it prepares everything and places it on
a queue and wakes up kthreadd.

kthreadd then wakes up and forks the thread.

After the thread has finishing setting up it will call complete on
a completion so kthread_create can continue on it's merry way
but it should not need to go to sleep waiting for someone to
call kthread_bind.

But if you can live with what I have just described that will
be easy to code up.

It is a little slower then kernel_thread but hopefully not much.

> Taking it one step further, if you added the notion of a thread pool,
> where upon exit, a thread isn't destroyed but rather is queued ready to
> handle the next kthread_create_quick() request.

That might happen. So far I am avoiding the notion of a thread pool for
as long as I can. There is some sense in it, especially in generalizing
the svc thread pool code from nfs. But if I don't have to go there I would
prefer it.

>> > I think it would be great if the kthread API (or underlying implementation)
>> > could be changed to handle these issues. I'd love for XPC to not have to
>> > maintain this sort of thing itself.
>>
>> Currently daemonize is a serious maintenance problem.
>>
>> Using daemonize and kernel_thread to create kernel threads is a blocker
>> in implementing the pid namespace because of their use of pid_t.
>>
>> So I am motivated to get this fixed.
>
> This would also address the problems we see with huge pid spaces for
> kernel threads on our largest machines. In the example from last week,
> we had 10 threads each on 4096 cpus. If we reworked work_queues to use
> the kthread_create_nonblocking() thread pool, we could probably collapse

> the need for having all of those per-task, per-cpu work queues.

Patches have already been sent (and I don't think found problems with)
that make kthreadd pid == 2, and they also modify daemonize to reparent
to kthreadd instead of init.

Eric

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

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

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: [PATCH] s390/scsi/zfcp_erp: Convert to use the kthread API
Posted by Swen Schillig on Mon, 30 Apr 2007 10:41:23 GMT
View Forum Message <> Reply to Message

On Sunday 22 April 2007 22:17, Christoph Hellwig wrote:
> On Thu, Apr 19, 2007 at 01:58:42AM -0600, Eric W. Biederman wrote:
> > From: Eric W. Biederman <ebiederm@xmission.com>
> >

> > Modify zfcperp%s to be started with kthread_run not
> > a combination of kernel_thread, daemonize and siginitsetinv
> > making the code slightly simpler and more maintainable.
>
> This driver would also benefit from a full kthread conversion.
> Unfortunately it has a strange dual-use semaphore (->erp_ready_sem)
> that hinders a straight conversion. Maybe the maintainer can take
> a look whether there's a nice way to get rid of that one?
>
I know and we have it on our schedule,
but it's not as easy as it might look like .

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: [PATCH] ia64 sn xpc: Convert to use kthread API.
Posted by Dean Nelson on Mon, 30 Apr 2007 15:22:30 GMT
View Forum Message <> Reply to Message

On Fri, Apr 27, 2007 at 02:33:32PM -0600, Eric W. Biederman wrote:
> Dean Nelson <dcn@sgi.com> writes:
>
> > On Fri, Apr 27, 2007 at 12:34:02PM -0600, Eric W. Biederman wrote:
> >> Dean Nelson <dcn@sgi.com> writes:
> >> >
> >> > XPC is in need of threads that can block indefinitely, which is why XPC
> >> > is in the business of maintaining a pool of threads. Currently there is
> >> > no such capability (that I know of) that is provided by linux. Workqueues
> >> > can't block indefinitely.
> >>
> >> I'm not certain I understand this requirement. Do you mean block indefinitely
> >> unless requested to stop?
> >
> > These threads can block waiting for a hardware DMA engine, which has a 28
> > second timeout setpoint.
>
> Ok. So this is an interruptible sleep?

No, the hardware DMA engine's software interface, doesn't sleep
nor relinquish the CPU. But there are other spots where we do sleep
interruptibly.

> Do you have any problems being woken up out of that interruptible sleep
> by kthread_stop?
>
> I am in the process of modifying kthread_stop to wake up thread in an

> interruptible sleep and set signal_pending, so they will break out.

No, this is fine, just avoid designing the kthread stop mechanism
to require a thread being requested to stop to actually stop in some
finite amount of time, and that by it not stopping other kthread stop
requests are held off. Allow the thread to take as much time as it needs
to respond to the kthread stop request.

> >> > And for performance reasons these threads need to be able to be created
> >> > quickly. These threads are involved in delivering messages to XPC's users
> >> > (like XPNET) and we had latency issues that led us to use kernel_thread()
> >> > directly instead of the kthread API. Additionally, XPC may need to have
> >> > hundreds of these threads active at any given time.
> >>
> >> Ugh. Can you tell me a little more about the latency issues?
> >
> > After placing a message in a local message queue, one SGI system partition
> > will interrupt another to retrieve the message. We need to minimize the
> > time from entering XPC's interrupt handler to the time that the message
> > can be DMA transferred and delivered to the consumer (like XPNET) to
> > whom it was sent.
> >
> >> Is having a non-halting kthread_create enough to fix this?
> >> So you don't have to context switch several times to get the
> >> thread running?
> >>
> >> Or do you need more severe latency reductions?
> >>
> >> The more severe fix would require some significant changes to copy_process
> >> and every architecture would need to be touched to fix up copy_thread.
> >> It is possible, it is a lot of work, and the reward is far from obvious.
> >
> > I think a non-halting kthread_create() should be sufficient. It is in
> > effect what XPC has now in calling kernel_thread() directly.
>
> A little different but pretty close.
>
> We call kthread_create() it prepares everything and places it on
> a queue and wakes up kthreadd.
>
> kthreadd then wakes up and forks the thread.
>
> After the thread has finishing setting up it will call complete on
> a completion so kthread_create can continue on it's merry way
> but it should not need to go to sleep waiting for someone to
> call kthread_bind.
>
> But if you can live with what I have just described that will

> be easy to code up.
>
> It is a little slower then kernel_thread but hopefully not much.

I was aware of this behavior of kthread_create(), which I consider 'halting' in that the thread doing the kthread_create() blocks waiting for kthreadd to get scheduled, call kernel_thread(), and then call complete(). By your mentioning a 'non-halting' kthread_create() I thought you were planning to create a new flavor of kthread_create() that called kernel_thread() directly and reparented the child thread to kthreadd. My mistake.

So there will be more overhead (time-wise) for XPC in calling kthread_run() as opposed to it formerly calling kernel_thread() directly. Thus requiring XPC to utilize a pool of kthread_create()'d threads.

> > Taking it one step further, if you added the notion of a thread pool,
> > where upon exit, a thread isn't destroyed but rather is queued ready to
> > handle the next kthread_create_quick() request.
>
> That might happen. So far I am avoiding the notion of a thread pool for
> as long as I can. There is some sense in it, especially in generalizing
> the svc thread pool code from nfs. But if I don't have to go there I would
> prefer it.

This means that XPC will have to retain its thread pool, but I can understand you not wanting to go there.

Thanks,
Dean

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: [PATCH] md: Remove broken SIGKILL support
Posted by Neil Brown on Tue, 01 May 2007 00:47:24 GMT
View Forum Message <> Reply to Message

On Thursday April 19, ebiederm@xmission.com wrote:
> From: Eric W. Biederman <ebiederm@xmission.com>
>
> Currently md_thread calls allow_signal so it can receive a
> SIGKILL but then does nothing with it except flush the
> sigkill so that it not can use an interruptible sleep.
>

> This whole dance is silly so remove the unnecessary
> and broken signal handling logic.

(sorry of the delay in replying)

You missed some related code which should help you see that it is -
maybe - not completely 'silly' (though I confess it might be slightly
broken).
In md_check_recovery:

```
 if (signal_pending(current)) {
  if (mddev->pers->sync_request) {
   printk(KERN_INFO "md: %s in immediate safe mode\n",
          mdname(mddev));
   mddev->safemode = 2;
  }
  flush_signals(current);
 }
```

The idea is that alt-sysrq-K will send SIGKILL to all processes
including the md support threads, which will cause them to enter
"immediate safe mode" so that the metadata will be marked clean
immediately at every opportunity. That way you can use alt-sysrq:
  sync,unmount,kill,reboot
and be fairly sure that you md array will be shut down cleanly.

I'd be just as happy to link this into Unmount (aka
do_emergency_remount), but that doesn't seem at all straight forward,
and in any case should be done before the current code is ripped out.

While we do have a reboot_notifier which tries to stop all arrays,
I've never been comfortable with that. A reboot really should just
reboot...

What I would REALLY like is for the block device to know whether it is
open read-only or read-write. Then I could mark it clean when it
becomes read-only as would happen when do_emergency_remount remounts
it read-only.

I might see how hard that would be...

NeilBrown

Subject: Re: [PATCH] md: Remove broken SIGKILL support
Posted by ebiederm on Tue, 01 May 2007 06:13:45 GMT
View Forum Message <> Reply to Message

Neil Brown <neilb@suse.de> writes:

> On Thursday April 19, ebiederm@xmission.com wrote:
>> From: Eric W. Biederman <ebiederm@xmission.com>
>>
>> Currently md_thread calls allow_signal so it can receive a
>> SIGKILL but then does nothing with it except flush the
>> sigkill so that it not can use an interruptible sleep.
>>
>> This whole dance is silly so remove the unnecessary
>> and broken signal handling logic.
>
> (sorry of the delay in replying)
>
> You missed some related code which should help you see that it is -
> maybe - not completely 'silly' (though I confess it might be slightly
> broken).
> In md_check_recovery:
>
> if (signal_pending(current)) {
>   if (mddev->pers->sync_request) {
>   printk(KERN_INFO "md: %s in immediate safe mode\n",
>   mdname(mddev));
>   mddev->safemode = 2;
>   }
>   flush_signals(current);
> }

Thanks.

> The idea is that alt-sysrq-K will send SIGKILL to all processes
> including the md support threads, which will cause them to enter
> "immediate safe mode" so that the metadata will be marked clean
> immediately at every opportunity. That way you can use alt-sysrq:
>   sync,unmount,kill,reboot
> and be fairly sure that you md array will be shut down cleanly.
>
> I'd be just as happy to link this into Unmount (aka
> do_emergency_remount), but that doesn't seem at all straight forward,
> and in any case should be done before the current code is ripped out.
>
> While we do have a reboot_notifier which tries to stop all arrays,
> I've never been comfortable with that. A reboot really should just
> reboot...
>

> What I would REALLY like is for the block device to know whether it is
> open read-only or read-write. Then I could mark it clean when it
> becomes read-only as would happen when do_emergency_remount remounts
> it read-only.
>
> I might see how hard that would be...

My goal to get signals to kernel threads out of the user space interface
especially for non-privileged processes, so everything that we do with
kernel threads can just be an unimportant implementation detail to user
space.

Eric

Subject: Re: [PATCH] ia64 sn xpc: Convert to use kthread API.
Posted by Dean Nelson on Wed, 02 May 2007 15:16:42 GMT
View Forum Message <> Reply to Message

On Mon, Apr 30, 2007 at 10:22:30AM -0500, Dean Nelson wrote:
> On Fri, Apr 27, 2007 at 02:33:32PM -0600, Eric W. Biederman wrote:
> > Dean Nelson <dcn@sgi.com> writes:
> >
> > > Taking it one step further, if you added the notion of a thread pool,
> > > where upon exit, a thread isn't destroyed but rather is queued ready to
> > > handle the next kthread_create_quick() request.
> >
> > That might happen. So far I am avoiding the notion of a thread pool for
> > as long as I can. There is some sense in it, especially in generalizing
> > the svc thread pool code from nfs. But if I don't have to go there I would
> > prefer it.
>
> This means that XPC will have to retain its thread pool, but I can
> understand you not wanting to go there.

On Thu, Apr 26, 2007 at 01:11:15PM -0600, Eric W. Biederman wrote:
>
> Ok.   Because of the module unloading issue, and because we don't have
> a lot of these threads running around, the current plan is to fix
> thread_create and kthread_stop so that they must always be paired,
> and so that kthread_stop will work correctly if the task has already
> exited.
>
> Basically that just involves calling get_task_struct in kthread_create

> and put_task_struct in kthread_stop.

Okay, so I need to expand upon Christoph Hellwig's patch so that all
the kthread_create()'d threads are kthread_stop()'d.

This is easy to do for the XPC thread that exists for the lifetime of XPC,
as well as for the threads created to manage the SGI system partitions.

XPC has the one discovery thread that is created when XPC is first started
and exits as soon as it has finished discovering all existing SGI system
partitions. With your forthcoming change to kthread_stop() that will allow
it to be called after the thread has exited, doing this one is also easy.
Note that the kthread_stop() for this discovery thread won't occur until
XPC is rmmod'd. This means that its task_struct will not be freed for
possibly a very long time (i.e., weeks). Is that a problem?

But then we come to XPC's pool of threads that deliver channel messages
to the appropriate consumer (like XPNET) and can block indefinitely. As
mentioned earlier there could be hundreds if not thousands of these
(our systems keep getting bigger). So now requiring a kthread_stop()
for each one of these becomes more of a problem, as it is a lot of
task_struct pointers to maintain.

Currently, XPC maintains these threads via a
wait_event_interruptible_exclusive() queue so that it can wakeup as many
or as few as needed at any given moment by calling wake_up_nr(). When XPC
is rmmod'd, a flag is set which causes them to exit and wake_up_all()
is called. Therefore XPC dosen't need to remember their pids or
task_struct pointers.

So what would you suggest we do for this pool of threads?

Is there any way to have a version of kthread_create() that doesn't
require a matching kthread_stop()? Or add a kthread_not_stopping()
that does the put_task_struct() call, so as to eliminate the need for
calling kthread_stop()? Or should we reconsider the kthread pool approach
(and get XPC out of the thread management business altogether)? Robin
Holt is putting together a proposal for how one could do a kthread pool,
it should provide a bit more justification for going down that road.

Thanks,
Dean

Subject: Re: [PATCH] ia64 sn xpc: Convert to use kthread API.
Posted by ebiederm on Wed, 02 May 2007 15:44:11 GMT
View Forum Message <> Reply to Message

Dean Nelson <dcn@sgi.com> writes:

> On Mon, Apr 30, 2007 at 10:22:30AM -0500, Dean Nelson wrote:
>> On Fri, Apr 27, 2007 at 02:33:32PM -0600, Eric W. Biederman wrote:
>> > Dean Nelson <dcn@sgi.com> writes:
>> >
>> > > Taking it one step further, if you added the notion of a thread pool,
>> > > where upon exit, a thread isn't destroyed but rather is queued ready to
>> > > handle the next kthread_create_quick() request.
>> >
>> > That might happen. So far I am avoiding the notion of a thread pool for
>> > as long as I can. There is some sense in it, especially in generalizing
>> > the svc thread pool code from nfs. But if I don't have to go there I would
>> > prefer it.
>>
>> This means that XPC will have to retain its thread pool, but I can
>> understand you not wanting to go there.
>
> On Thu, Apr 26, 2007 at 01:11:15PM -0600, Eric W. Biederman wrote:
>>
>> Ok.   Because of the module unloading issue, and because we don't have
>> a lot of these threads running around, the current plan is to fix
>> thread_create and kthread_stop so that they must always be paired,
>> and so that kthread_stop will work correctly if the task has already
>> exited.
>>
>> Basically that just involves calling get_task_struct in kthread_create
>> and put_task_struct in kthread_stop.
>
> Okay, so I need to expand upon Christoph Hellwig's patch so that all
> the kthread_create()'d threads are kthread_stop()'d.
>
> This is easy to do for the XPC thread that exists for the lifetime of XPC,
> as well as for the threads created to manage the SGI system partitions.
>
> XPC has the one discovery thread that is created when XPC is first started
> and exits as soon as it has finished discovering all existing SGI system
> partitions. With your forthcoming change to kthread_stop() that will allow
> it to be called after the thread has exited, doing this one is also easy.
> Note that the kthread_stop() for this discovery thread won't occur until
> XPC is rmmod'd. This means that its task_struct will not be freed for
> possibly a very long time (i.e., weeks). Is that a problem?

As long as there is only one, not really. It would be good if we could
get rid of it though.

The practical problem is the race with rmmod, in particular if someone
calls rmmod while this thread is still running.

If I get clever I think this is likely solvable with something like.

```
kthread_maybe_stop(struct task_struct **loc)
{
        struct task_struct *tsk;
        tsk = xchg(loc, NULL);
        if (tsk)
         kthread_stop(tsk);
}

kthread_stop_self(struct task_struct **loc, int exit_code)
{
        struct task_struct *tsk;

        tsk = xchg(loc, NULL);
        if (tsk)
         put_task_struct(tsk);
 do_exit(tsk);
}
```

I'm not quite convinced that is a common enough paradigm to implement
that.

> But then we come to XPC's pool of threads that deliver channel messages
> to the appropriate consumer (like XPNET) and can block indefinitely. As
> mentioned earlier there could be hundreds if not thousands of these
> (our systems keep getting bigger). So now requiring a kthread_stop()
> for each one of these becomes more of a problem, as it is a lot of
> task_struct pointers to maintain.
>
> Currently, XPC maintains these threads via a
> wait_event_interruptible_exclusive() queue so that it can wakeup as many
> or as few as needed at any given moment by calling wake_up_nr(). When XPC
> is rmmod'd, a flag is set which causes them to exit and wake_up_all()
> is called. Therefore XPC dosen't need to remember their pids or
> task_struct pointers.
>
> So what would you suggest we do for this pool of threads?

Good question.

The whole concept of something that feels like a core part of the
platform code being modular I'm still looking at strange.

> Is there any way to have a version of kthread_create() that doesn't
> require a matching kthread_stop()? Or add a kthread_not_stopping()
> that does the put_task_struct() call, so as to eliminate the need for
> calling kthread_stop()?

Yes. I was thinking calling it kthread_orphan or something like that.
We can't make anything like that the default, because of the modular
remove problem, but it's not to hard.

> Or should we reconsider the kthread pool approach
> (and get XPC out of the thread management business altogether)? Robin
> Holt is putting together a proposal for how one could do a kthread pool,
> it should provide a bit more justification for going down that road.

Eric

_____

Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: [PATCH] cpci_hotplug: Convert to use the kthread API
Posted by Christoph Hellwig on Fri, 04 May 2007 11:12:16 GMT
View Forum Message <> Reply to Message

On Fri, Apr 27, 2007 at 06:07:49PM -0400, Scott Murray wrote:
> 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.

Can you forward the patches with your fix to Andrew to make sure he
picks it up?

_____

Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: [PATCH] cpci_hotplug: Convert to use the kthread API
Posted by Scott Murray on Mon, 07 May 2007 18:18:29 GMT
View Forum Message <> Reply to Message

On Fri, 4 May 2007, Christoph Hellwig wrote:

> On Fri, Apr 27, 2007 at 06:07:49PM -0400, Scott Murray wrote:
> > 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.
>
> Can you forward the patches with your fix to Andrew to make sure he
> picks it up?

Andrew, here is my updated version of Christoph's kthread conversion
patch for cpci_hotplug. I've CC'ed Kristen so she won't be surprised
when this eventually goes to mainline.

Signed-off-by: Christoph Hellwig <hch@lst.de>
Signed-off-by: Scott Murray <scottm@somanetworks.com>

```
diff --git a/drivers/pci/hotplug/cpci_hotplug_core.c b/drivers/pci/hotplug/cpci_hotplug_core.c
index 6845515..ed4d44e 100644
--- a/drivers/pci/hotplug/cpci_hotplug_core.c
+++ b/drivers/pci/hotplug/cpci_hotplug_core.c
@@ -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 +566,36 @@ 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);
+ thread_finished = 0;
  return 0;
 }

 static void
 cpci_stop_thread(void)
 {
+ kthread_stop(cpci_thread);
  thread_finished = 1;
- dbg("thread finish command given");
- if (controller->irq)
-  up(&event_semaphore);
- dbg("wait for thread to exit");
- down(&thread_exit);
 }

 int
```

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

Subject: Re: [PATCH] cpci_hotplug: Convert to use the kthread API
Posted by akpm on Wed, 09 May 2007 23:24:30 GMT
View Forum Message <> Reply to Message

On Mon, 7 May 2007 14:18:29 -0400 (EDT)
Scott Murray <scottm@somanetworks.com> wrote:

> On Fri, 4 May 2007, Christoph Hellwig wrote:
>
> > On Fri, Apr 27, 2007 at 06:07:49PM -0400, Scott Murray wrote:
> > > 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.
> >
> > Can you forward the patches with your fix to Andrew to make sure he
> > picks it up?
>
> Andrew, here is my updated version of Christoph's kthread conversion
> patch for cpci_hotplug. I've CC'ed Kristen so she won't be surprised
> when this eventually goes to mainline.

A patch in this area would normally go

```
 you->kristen->mainline
              |
              v
             -mm
```

or

```
 you->kristen->greg->mainline
                    |
                    v
                   -mm
```

or

you->me->greg->mainline (gets an Acked-by somewhere)
                |
                v
               -mm

or

you->kristen->Len->mainline
                  |
                  v
                 -mm

or something else.

Kristen, how do you want to play this?

Do you run a tree? If so, lemmeatit ;)

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: [PATCH] cpci_hotplug: Convert to use the kthread API
Posted by Kristen Carlson Accar on Thu, 10 May 2007 00:00:06 GMT
View Forum Message <> Reply to Message

On Wed, 9 May 2007 16:24:30 -0700
Andrew Morton <akpm@linux-foundation.org> wrote:

> On Mon, 7 May 2007 14:18:29 -0400 (EDT)
> Scott Murray <scottm@somanetworks.com> wrote:
>
> > On Fri, 4 May 2007, Christoph Hellwig wrote:
> >
> > > On Fri, Apr 27, 2007 at 06:07:49PM -0400, Scott Murray wrote:
> > > > 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.
> > >

> > > Can you forward the patches with your fix to Andrew to make sure he
> > > picks it up?
> >
> > Andrew, here is my updated version of Christoph's kthread conversion
> > patch for cpci_hotplug. I've CC'ed Kristen so she won't be surprised
> > when this eventually goes to mainline.
>
> A patch in this area would normally go
>
> you->kristen->mainline
> |
>   v
>   -mm
>
> or
>
> you->kristen->greg->mainline
> |
>   v
>   -mm
>
> or
>
> you->me->greg->mainline (gets an Acked-by somewhere)
> |
>   v
>   -mm
>
> or
>
> you->kristen->Len->mainline
> |
>   v
>   -mm
>
> or something else.
>
> Kristen, how do you want to play this?
>
> Do you run a tree? If so, lemmeatit ;)
>

So, we (Greg and I) have talked about this before - I do have a tree,
but I normally send things to Greg rather than directly to you. I should
probably change that (why add levels of indirection...), but for the
immediate purpose of getting this patch tested etc just go ahead and take
it in and when I get my act together I will put it in my tree.

Thanks,
Kristen

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: [PATCH] cpci_hotplug: Convert to use the kthread API
Posted by Scott Murray on Thu, 10 May 2007 18:29:29 GMT
View Forum Message <> Reply to Message

On Wed, 9 May 2007, Andrew Morton wrote:

> On Mon, 7 May 2007 14:18:29 -0400 (EDT)
> Scott Murray <scottm@somanetworks.com> wrote:
>
> > On Fri, 4 May 2007, Christoph Hellwig wrote:
> >
> > > On Fri, Apr 27, 2007 at 06:07:49PM -0400, Scott Murray wrote:
> > > > 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.
> > >
> > > Can you forward the patches with your fix to Andrew to make sure he
> > > picks it up?
> >
> > Andrew, here is my updated version of Christoph's kthread conversion
> > patch for cpci_hotplug. I've CC'ed Kristen so she won't be surprised
> > when this eventually goes to mainline.
>
> A patch in this area would normally go
>
> you->kristen->mainline
> |
> v
> -mm
>
> or
>

> you->kristen->greg->mainline
> |
>   v
>   -mm
>
> or
>
> you->me->greg->mainline (gets an Acked-by somewhere)
> |
>   v
>   -mm
>
> or
>
> you->kristen->Len->mainline
> |
>   v
>   -mm
>
> or something else.

My apologies, it wasn't entirely clear to me that that was happening with
the other kthread conversion patches in the big batch, so I didn't follow
the normal procedure. I'll try to stick to the regular flow up the chain
going forward.

Scott

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

Subject: Re: [PATCH] cpci_hotplug: Convert to use the kthread API
Posted by akpm on Thu, 10 May 2007 19:30:46 GMT
View Forum Message <> Reply to Message

On Thu, 10 May 2007 14:29:29 -0400 (EDT) Scott Murray <scottm@somanetworks.com> wrote:

> > or
> >

> > you->kristen->Len->mainline
> > |
> >   v
> >   -mm
> >
> > or something else.
>
> My apologies, it wasn't entirely clear to me that that was happening with
> the other kthread conversion patches in the big batch, so I didn't follow
> the normal procedure. I'll try to stick to the regular flow up the chain
> going forward.

oh that's OK. I'm kind of a catchall routing service. It's just that in
this case I wasn't sure which direction the next hop lay in.

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: [PATCH] ia64 sn xpc: Convert to use kthread API.
Posted by Dean Nelson on Thu, 17 May 2007 13:44:50 GMT
View Forum Message <> Reply to Message

On Wed, May 02, 2007 at 09:44:11AM -0600, Eric W. Biederman wrote:
> Dean Nelson <dcn@sgi.com> writes:
> > On Thu, Apr 26, 2007 at 01:11:15PM -0600, Eric W. Biederman wrote:
> >>
> >> Ok. Because of the module unloading issue, and because we don't have
> >> a lot of these threads running around, the current plan is to fix
> >> thread_create and kthread_stop so that they must always be paired,
> >> and so that kthread_stop will work correctly if the task has already
> >> exited.
> >>
> >> Basically that just involves calling get_task_struct in kthread_create
> >> and put_task_struct in kthread_stop.
> >
> > Okay, so I need to expand upon Christoph Hellwig's patch so that all
> > the kthread_create()'d threads are kthread_stop()'d.
> >
> > This is easy to do for the XPC thread that exists for the lifetime of XPC,
> > as well as for the threads created to manage the SGI system partitions.
> >
> > XPC has the one discovery thread that is created when XPC is first started
> > and exits as soon as it has finished discovering all existing SGI system
> > partitions. With your forthcoming change to kthread_stop() that will allow
> > it to be called after the thread has exited, doing this one is also easy.
> > Note that the kthread_stop() for this discovery thread won't occur until

> > XPC is rmmod'd. This means that its task_struct will not be freed for
> > possibly a very long time (i.e., weeks). Is that a problem?
>
> As long as there is only one, not really. It would be good if we could
> get rid of it though.
>
> The practical problem is the race with rmmod, in particular if someone
> calls rmmod while this thread is still running.

I guess I'm not seeing the race with rmmod that you're talking
about? In XPC's case, rmmod calls xpc_exit() which currently does a
wait_for_completion() on the discovery thread and on the other thread
mentioned above. These will be changed to kthread_stop() calls. And if
the discovery thread has already exited the kthread_stop() will return
immediately and if not it will wait until the discovery thread has
exited. rmmod won't return from xpc_exit() until both threads have exited.

Any thought as to when the changes to kthread_stop() that allow it to be
called for a kthread that has already exited will get into the -mm tree?

> > Is there any way to have a version of kthread_create() that doesn't
> > require a matching kthread_stop()? Or add a kthread_not_stopping()
> > that does the put_task_struct() call, so as to eliminate the need for
> > calling kthread_stop()?
>
> Yes. I was thinking calling it kthread_orphan or something like that.
> We can't make anything like that the default, because of the modular
> remove problem, but it's not to hard.

Again, when xpc_exit() is called by rmmod it waits for XPC's pool of
threads to exit before it returns, so not a problem.

Any thought as to when kthread_orphan() will get into the -mm tree? Once
kthread_stop() is changed and kthread_orphan() added I can proceed with
a patch to change XPC to use the kthread API.

> > Or should we reconsider the kthread pool approach
> > (and get XPC out of the thread management business altogether)? Robin
> > Holt is putting together a proposal for how one could do a kthread pool,
> > it should provide a bit more justification for going down that road.

Robin has changed his mind about tieing in the management of a pool
of threads with the kthread API, so there won't be the fore mentioned
proposal.

Thanks,
Dean
