Subject: [RFC] [PATCH] Queue blocked signals to container-init separately
Posted by Sukadev Bhattiprolu on Wed, 10 Oct 2007 06:00:18 GMT
View Forum Message <> Reply to Message

When a process in a container signals its container-init, we want
to ensure that the signal does not terminate the container-init.
i.e if the container-init has no handler for the signal, and the
signal is fatal, we want to ignore the signal.

Patches at the following URL check whether the signal will terminate
the cinit (i.e is a fatal signal and the cinit has no handler for it).

https://lists.linux-foundation.org/pipermail/containers/2007-September/007019.html

But if the fatal signal is currently blocked by the container-init,
those checks fail since we cannot ignore the signals even though it
may be fatal. This is because the container-init may install a handler
for that signal, before unblocking it. So we must post the signal.

But if the container-init unblocks the signal without setting up
a handler, the signal would terminate the container-init.

This patch (very lightly tested :-) attempts to queue the blocked
signals to the container init separately and then requeue them
on the container-init's normal queue when the signal is unblocked.

This patch is just a prototype and to indicate the basic idea and
scope of changes. If it looks interesting/feasible I could do more
testing and possibly some reorg.

---
 fs/binfmt_elf.c | 5
 fs/proc/array.c | 10 +
 include/linux/pid_namespace.h | 2
 kernel/pid.c | 12 ++
 kernel/signal.c | 211 ++--
 5 files changed, 230 insertions(+), 10 deletions(-)

Index: 2.6.23-rc8-mm2/kernel/signal.c
===
--- 2.6.23-rc8-mm2.orig/kernel/signal.c 2007-10-09 22:41:29.000000000 -0700
+++ 2.6.23-rc8-mm2/kernel/signal.c 2007-10-09 22:48:28.000000000 -0700
@@ -46,7 +46,7 @@ static int sig_init_ignore(struct task_s

  // Currently this check is a bit racy with exec(),
  // we can _simplify_ de_thread and close the race.
- if (likely(!is_cgroup_init(tsk->group_leader)))

```
+ if (likely(!is_container_init(tsk->group_leader)))
   return 0;

  if (is_current_in_ancestor_pid_ns(tsk) && !in_interrupt())
@@ -119,11 +119,28 @@ static inline int has_pending_signals(si

 #define PENDING(p,b) has_pending_signals(&(p)->signal, (b))

+static int task_has_pending_signals(struct task_struct *t)
+{
+ if (PENDING(&t->pending, &t->blocked) ||
+     PENDING(&t->signal->shared_pending, &t->blocked))
+  return 1;
+
+ if (is_container_init(t)) {
+  struct pid_namespace *ns = task_active_pid_ns(t);
+
+  if (!ns)
+   return 0;
+
+  return PENDING(&ns->cinit_blocked_pending, &t->blocked) ||
+   PENDING(&ns->cinit_blocked_shared_pending, &t->blocked);
+ }
+ return 0;
+}
+
+
 static int recalc_sigpending_tsk(struct task_struct *t)
 {
- if (t->signal->group_stop_count > 0 ||
-     PENDING(&t->pending, &t->blocked) ||
-     PENDING(&t->signal->shared_pending, &t->blocked)) {
+ if (t->signal->group_stop_count > 0 || task_has_pending_signals(t)) {
   set_tsk_thread_flag(t, TIF_SIGPENDING);
   return 1;
  }
@@ -235,6 +252,20 @@ void flush_sigqueue(struct sigpending *q
  }
 }

+static void flush_cinit_signals(struct task_struct *t)
+{
+ struct pid_namespace *ns;
+
+ if (!is_container_init(t))
+  return;
+ ns = task_active_pid_ns(t);
+ if (ns) {
```

```
+  flush_sigqueue(&ns->cinit_blocked_pending);
+  flush_sigqueue(&ns->cinit_blocked_shared_pending);
+
+ }
+}
+
 /*
  * Flush all pending signals for a task.
  */
@@ -246,6 +277,7 @@ void flush_signals(struct task_struct *t
  clear_tsk_thread_flag(t,TIF_SIGPENDING);
  flush_sigqueue(&t->pending);
  flush_sigqueue(&t->signal->shared_pending);
+ flush_cinit_signals(t);
  spin_unlock_irqrestore(&t->sighand->siglock, flags);
 }

@@ -400,6 +432,12 @@ int dequeue_signal(struct task_struct *t
  /* We only dequeue private signals from ourselves, we don't let
   * signalfd steal them
   */
+ /*
+  * Note: We can ignore any signals for container init that are
+  * queued in the pid namespace since they are only queued there
+  * when signals are blocked (and we don't dequeue blocked signals
+  * here anyway)
+  */
  signr = __dequeue_signal(&tsk->pending, mask, info);
  if (!signr) {
   signr = __dequeue_signal(&tsk->signal->shared_pending,
@@ -579,6 +617,21 @@ static void do_notify_parent_cldstop(str
  * actual continuing for SIGCONT, but not the actual stopping for stop
  * signals.  The process stop is done as a signal action for SIG_DFL.
  */
+static void rm_from_cinit_queue(unsigned long mask, struct task_struct *t)
+{
+ struct pid_namespace *ns;
+
+ if (!is_container_init(t))
+  return;
+
+ ns = task_active_pid_ns(t);
+
+ rm_from_queue(mask, &ns->cinit_blocked_pending);
+ rm_from_queue(mask, &ns->cinit_blocked_shared_pending);
+
+ return;
+}
```

```
+
 static void handle_stop_signal(int sig, struct task_struct *p)
 {
  struct task_struct *t;
@@ -597,6 +650,7 @@ static void handle_stop_signal(int sig,
    * This is a stop signal.  Remove SIGCONT from all queues.
    */
   rm_from_queue(sigmask(SIGCONT), &p->signal->shared_pending);
+  rm_from_cinit_queue(sigmask(SIGCONT), p);
   t = p;
   do {
    rm_from_queue(sigmask(SIGCONT), &t->pending);
@@ -627,6 +681,7 @@ static void handle_stop_signal(int sig,
    spin_lock(&p->sighand->siglock);
   }
   rm_from_queue(SIG_KERNEL_STOP_MASK, &p->signal->shared_pending);
+  rm_from_cinit_queue(SIG_KERNEL_STOP_MASK, p);
   t = p;
   do {
    unsigned int state;
@@ -802,6 +857,7 @@ static int
 specific_send_sig_info(int sig, struct siginfo *info, struct task_struct *t)
 {
  int ret = 0;
+ struct sigpending *pending;

  BUG_ON(!irqs_disabled());
  assert_spin_locked(&t->sighand->siglock);
@@ -816,13 +872,75 @@ specific_send_sig_info(int sig, struct s
  if (LEGACY_QUEUE(&t->pending, sig))
   goto out;

- ret = send_signal(sig, info, t, &t->pending);
+ pending = &t->pending;
+ if (is_container_init(t) && sigismember(&t->blocked, sig) &&
+   !is_current_in_ancestor_pid_ns(t)) {
+  struct pid_namespace *ns = task_active_pid_ns(t);
+
+  /*
+   * Hmm. If container init is exiting (ns == NULL) and we
+   * are trying to post a blocked signal.  Post to the normal
+   * queue for now.
+   */
+  if (ns && LEGACY_QUEUE(&ns->cinit_blocked_pending, sig))
+   goto out;
+  else if (ns)
+   pending = &ns->cinit_blocked_pending;
+ }
```

```
+
+ ret = send_signal(sig, info, t, pending);
  if (!ret && !sigismember(&t->blocked, sig))
   signal_wake_up(t, sig == SIGKILL);
 out:
  return ret;
 }

+static void requeue_cinit_signals(struct task_struct *t, sigset_t *set)
+{
+ struct sigqueue *q, *n;
+ struct pid_namespace *ns = task_active_pid_ns(t);
+ struct sigpending *ns_pending;
+ struct sigpending *ns_shpending;
+ struct sigpending *tsk_pending;
+ struct sigpending *tsk_shpending;
+
+ /*
+  * Unblocking a signal while a process is exitng ?
+  */
+ if (!ns)
+  return;
+
+ /*
+  * For each signal being unblocked, remove it from the namespace
+  * pending list and add it to task's pending list
+  */
+ tsk_pending = &t->pending;
+ ns_pending = &ns->cinit_blocked_pending;
+
+ list_for_each_entry_safe(q, n, &ns_pending->list, list) {
+  if (sigismember(set, q->info.si_signo)) {
+   list_del_init(&q->list);
+   sigdelset(&ns_pending->signal, q->info.si_signo);
+
+   list_add_tail(&q->list, &tsk_pending->list);
+   sigaddset(&tsk_pending->signal, q->info.si_signo);
+  }
+ }
+
+ /* Repeat for the shared-pending signals */
+ tsk_shpending = &t->signal->shared_pending;
+ ns_shpending = &ns->cinit_blocked_shared_pending;
+ list_for_each_entry_safe(q, n, &ns_shpending->list, list) {
+  if (sigismember(set, q->info.si_signo)) {
+   list_del_init(&q->list);
+   sigdelset(&ns_shpending->signal, q->info.si_signo);
+
```

```
+   list_add_tail(&q->list, &tsk_shpending->list);
+   sigaddset(&tsk_shpending->signal, q->info.si_signo);
+  }
+ }
+}
+
 /*
  * Force a signal that the process can't ignore: if necessary
  * we unblock the signal and change any SIG_IGN to SIG_DFL.
@@ -848,6 +966,20 @@ force_sig_info(int sig, struct siginfo *
   action->sa.sa_handler = SIG_DFL;
   if (blocked) {
    sigdelset(&t->blocked, sig);
+   if (is_container_init(t)) {
+    struct pid_namespace *ns;
+    sigset_t set;
+    /*
+     * We just unblocked a signal. Requeue any
+     * pending instances of the signal on the
+     * namespace queue to the task's pending
+     * queue.
+     */
+    sigemptyset(&set);
+    sigaddset(&set, sigmask(sig));
+    ns = task_active_pid_ns(t);
+    requeue_cinit_signals(t, &set);
+   }
    recalc_sigpending_and_wake(t);
   }
  }
@@ -890,6 +1022,13 @@ __group_complete_signal(int sig, struct
  struct task_struct *t;

  /*
+  * If signal came from same or descendant namespace and is a
+  * blocked signal, we process the signal when we unblock it
+  */
+ if (!is_current_in_ancestor_pid_ns(p) && sigismember(&p->blocked, sig))
+  return;
+
+ /*
   * Now find a thread we can wake up to take the signal off the queue.
   *
   * If the main thread wants the signal, it gets first crack.
@@ -987,6 +1126,7 @@ int
 __group_send_sig_info(int sig, struct siginfo *info, struct task_struct *p)
 {
  int ret = 0;
```

```
+ struct sigpending *pending;

  assert_spin_locked(&p->sighand->siglock);
  handle_stop_signal(sig, p);
@@ -999,12 +1139,28 @@ __group_send_sig_info(int sig, struct si
   /* This is a non-RT signal and we already have one queued.  */
   return ret;

+ pending = &p->signal->shared_pending;
+ if (is_container_init(p) && sigismember(&p->blocked, sig) &&
+    !is_current_in_ancestor_pid_ns(p)) {
+  struct pid_namespace *ns = task_active_pid_ns(p);
+
+  /*
+   * Hmm. If container init is exiting (ns == NULL) and we
+   * are trying to post a blocked signal, what should we do ?
+   * Post to the normal queue for now.
+   */
+  if (ns && LEGACY_QUEUE(&ns->cinit_blocked_shared_pending, sig))
+   return ret;
+  else if (ns)
+   pending = &ns->cinit_blocked_shared_pending;
+ }
+
   /*
    * Put this signal on the shared-pending queue, or fail with EAGAIN.
    * We always use the shared queue for process-wide signals,
    * to avoid several races.
    */
- ret = send_signal(sig, info, p, &p->signal->shared_pending);
+ ret = send_signal(sig, info, p, pending);
   if (unlikely(ret))
    return ret;

@@ -1365,6 +1521,7 @@ int send_sigqueue(int sig, struct sigque
 {
   unsigned long flags;
   int ret = 0;
+ struct sigpending *pending;

   BUG_ON(!(q->flags & SIGQUEUE_PREALLOC));

@@ -1397,14 +1554,23 @@ int send_sigqueue(int sig, struct sigque
    ret = 1;
    goto out;
   }
+
+ pending = &p->pending;
```

```
+ if (is_container_init(p) && sigismember(&p->blocked, sig) &&
+    !is_current_in_ancestor_pid_ns(p)) {
+  struct pid_namespace *ns = task_active_pid_ns(p);
+  if (ns)
+   pending = &ns->cinit_blocked_pending;
+ }
+
   /*
    * Deliver the signal to listening signalfds. This must be called
    * with the sighand lock held.
    */
  signalfd_notify(p, sig);

- list_add_tail(&q->list, &p->pending.list);
- sigaddset(&p->pending.signal, sig);
+ list_add_tail(&q->list, &pending->list);
+ sigaddset(&pending->signal, sig);
  if (!sigismember(&p->blocked, sig))
   signal_wake_up(p, sig == SIGKILL);

@@ -1421,6 +1587,7 @@ send_group_sigqueue(int sig, struct sigq
 {
  unsigned long flags;
  int ret = 0;
+ struct sigpending *shpending;

  BUG_ON(!(q->flags & SIGQUEUE_PREALLOC));

@@ -1435,6 +1602,14 @@ send_group_sigqueue(int sig, struct sigq
   goto out;
  }

+ shpending = &p->signal->shared_pending;
+ if (is_container_init(p) && sigismember(&p->blocked, sig) &&
+    !is_current_in_ancestor_pid_ns(p)) {
+  struct pid_namespace *ns = task_active_pid_ns(p);
+  if (ns)
+   shpending = &ns->cinit_blocked_shared_pending;
+ }
+
  if (unlikely(!list_empty(&q->list))) {
   /*
    * If an SI_TIMER entry is already queue just increment
@@ -1456,8 +1631,8 @@ send_group_sigqueue(int sig, struct sigq
   * We always use the shared queue for process-wide signals,
   * to avoid several races.
   */
- list_add_tail(&q->list, &p->signal->shared_pending.list);
```

```
- sigaddset(&p->signal->shared_pending.signal, sig);
+ list_add_tail(&q->list, &shpending->list);
+ sigaddset(&shpending->signal, sig);

  __group_complete_signal(sig, p);
 out:
@@ -2029,6 +2204,7 @@ int sigprocmask(int how, sigset_t *set,
    break;
   case SIG_UNBLOCK:
    signandsets(&current->blocked, &current->blocked, set);
+   requeue_cinit_signals(current, set);
    break;
   case SIG_SETMASK:
    current->blocked = *set;
@@ -2082,15 +2258,25 @@ long do_sigpending(void __user *set, uns
 {
   long error = -EINVAL;
   sigset_t pending;
+ sigset_t ns_pending;
+ struct pid_namespace *ns;

   if (sigsetsize > sizeof(sigset_t))
    goto out;

+ sigemptyset(&ns_pending);
   spin_lock_irq(&current->sighand->siglock);
+ if (is_container_init(current)) {
+   ns = task_active_pid_ns(current);
+   sigorsets(&ns_pending, &ns->cinit_blocked_pending.signal,
+     &ns->cinit_blocked_shared_pending.signal);
+ }
   sigorsets(&pending, &current->pending.signal,
     &current->signal->shared_pending.signal);
   spin_unlock_irq(&current->sighand->siglock);

+ sigorsets(&pending, &pending, &ns_pending);
+
   /* Outside the lock because only this thread touches it.  */
   sigandsets(&pending, &current->blocked, &pending);

@@ -2387,6 +2573,11 @@ int do_sigaction(int sig, struct k_sigac
     rm_from_queue_full(&mask, &t->pending);
     t = next_thread(t);
    } while (t != current);
+   if (is_container_init(t)) {
+    struct pid_namespace *ns = task_active_pid_ns(t);
+    rm_from_queue_full(&mask, &ns->cinit_blocked_pending);
+    rm_from_queue_full(&mask, &ns->cinit_blocked_shared_pending);
```

```
+   }
  }
 }

Index: 2.6.23-rc8-mm2/fs/binfmt_elf.c
===================================================================
--- 2.6.23-rc8-mm2.orig/fs/binfmt_elf.c 2007-10-09 22:41:28.000000000 -0700
+++ 2.6.23-rc8-mm2/fs/binfmt_elf.c 2007-10-09 22:41:49.000000000 -0700
@@ -42,6 +42,7 @@
 #include <asm/uaccess.h>
 #include <asm/param.h>
 #include <asm/page.h>
+#include <linux/pid_namespace.h>

 static int load_elf_binary(struct linux_binprm *bprm, struct pt_regs *regs);
 static int load_elf_library(struct file *);
@@ -1449,6 +1450,10 @@ static void fill_prstatus(struct elf_prs
 {
  prstatus->pr_info.si_signo = prstatus->pr_cursig = signr;
  prstatus->pr_sigpend = p->pending.signal.sig[0];
+ if (is_container_init(p)) {
+  struct pid_namespace *ns = task_active_pid_ns(p);
+  prstatus->pr_sigpend |= ns->cinit_blocked_pending.signal.sig[0];
+ }
  prstatus->pr_sighold = p->blocked.sig[0];
  prstatus->pr_pid = task_pid_vnr(p);
  prstatus->pr_ppid = task_pid_vnr(p->parent);
Index: 2.6.23-rc8-mm2/fs/proc/array.c
===================================================================
--- 2.6.23-rc8-mm2.orig/fs/proc/array.c 2007-10-09 22:41:28.000000000 -0700
+++ 2.6.23-rc8-mm2/fs/proc/array.c 2007-10-09 22:41:49.000000000 -0700
@@ -264,6 +264,16 @@ static inline char *task_sig(struct task
  if (lock_task_sighand(p, &flags)) {
   pending = p->pending.signal;
   shpending = p->signal->shared_pending.signal;
+  if (is_container_init(p)) {
+   struct pid_namespace *ns = task_active_pid_ns(p);
+   if (ns) {
+    sigorsets(&pending, &pending,
+     &ns->cinit_blocked_pending.signal);
+    sigorsets(&shpending, &shpending,
+     &ns->cinit_blocked_shared_pending.signal);
+   }
+  }
+
   blocked = p->blocked;
   collect_sigign_sigcatch(p, &ignored, &caught);
   num_threads = atomic_read(&p->signal->count);
```

```
Index: 2.6.23-rc8-mm2/include/linux/pid_namespace.h
===================================================================
--- 2.6.23-rc8-mm2.orig/include/linux/pid_namespace.h 2007-10-09 22:41:28.000000000 -0700
+++ 2.6.23-rc8-mm2/include/linux/pid_namespace.h 2007-10-09 22:41:49.000000000 -0700
@@ -25,6 +25,8 @@ struct pid_namespace {
 #ifdef CONFIG_PROC_FS
  struct vfsmount *proc_mnt;
 #endif
+ struct sigpending cinit_blocked_pending;
+ struct sigpending cinit_blocked_shared_pending;
 };

 extern struct pid_namespace init_pid_ns;
Index: 2.6.23-rc8-mm2/kernel/pid.c
===================================================================
--- 2.6.23-rc8-mm2.orig/kernel/pid.c 2007-10-09 22:41:28.000000000 -0700
+++ 2.6.23-rc8-mm2/kernel/pid.c 2007-10-09 22:41:49.000000000 -0700
@@ -78,6 +78,12 @@ struct pid_namespace init_pid_ns = {
  .last_pid = 0,
  .level = 0,
  .child_reaper = &init_task,
+ .cinit_blocked_pending = {
+  .list = LIST_HEAD_INIT(init_pid_ns.cinit_blocked_pending.list),
+  .signal = {{0}}},
+ .cinit_blocked_shared_pending = {
+  .list = LIST_HEAD_INIT(init_pid_ns.cinit_blocked_shared_pending.list),
+  .signal = {{0}}}
 };
 EXPORT_SYMBOL_GPL(init_pid_ns);

@@ -621,6 +627,8 @@ static struct pid_namespace *create_pid_
  ns->last_pid = 0;
  ns->child_reaper = NULL;
  ns->level = level;
+ init_sigpending(&ns->cinit_blocked_pending);
+ init_sigpending(&ns->cinit_blocked_shared_pending);

  set_bit(0, ns->pidmap[0].page);
  atomic_set(&ns->pidmap[0].nr_free, BITS_PER_PAGE - 1);
@@ -646,6 +654,10 @@ static void destroy_pid_namespace(struct

  for (i = 0; i < PIDMAP_ENTRIES; i++)
   kfree(ns->pidmap[i].page);
+
+ flush_sigqueue(&ns->cinit_blocked_pending);
+ flush_sigqueue(&ns->cinit_blocked_shared_pending);
+
  kmem_cache_free(pid_ns_cachep, ns);
```

}

_____

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

---

Subject: Re: [RFC] [PATCH] Queue blocked signals to container-init separately
Posted by serue on Tue, 16 Oct 2007 03:26:01 GMT
View Forum Message <> Reply to Message

Quoting sukadev@us.ibm.com (sukadev@us.ibm.com):
>
>
> When a process in a container signals its container-init, we want
> to ensure that the signal does not terminate the container-init.
> i.e if the container-init has no handler for the signal, and the
> signal is fatal, we want to ignore the signal.
>
> Patches at the following URL check whether the signal will terminate
> the cinit (i.e is a fatal signal and the cinit has no handler for it).
>
> https://lists.linux-foundation.org/pipermail/containers/2007-September/007019.html
>
> But if the fatal signal is currently blocked by the container-init,
> those checks fail since we cannot ignore the signals even though it
> may be fatal. This is because the container-init may install a handler
> for that signal, before unblocking it. So we must post the signal.
> But if the container-init unblocks the signal without setting up
> a handler, the signal would terminate the container-init.

So there is an init which actually does this? First forks some
potentially signaling children, then blocks signals temporarily
to install some signal handlers and then unblocks?

Any tasks in the container were forked by the init, so simply saying
that if it does that, it may die, doesn't seem unreasonable to me.

> This patch (very lightly tested :-) attempts to queue the blocked
> signals to the container init separately and then requeue them
> on the container-init's normal queue when the signal is unblocked.
>
> This patch is just a prototype and to indicate the basic idea and
> scope of changes. If it looks interesting/feasible I could do more
> testing and possibly some reorg.

It seems like a maintenance headache bc these "*_cinit_*" calls

seem randomly sprinkled about...

Still there isn't a better suggestion as of yet, right? So if you
have an updated version (preferably split into two patches with the
first containing helpers, and all the inline code which is still
under "if (is_container_init(p))" moved into helpers as well) please
send it.

thanks,
-serge

> ---
> fs/binfmt_elf.c | 5
> fs/proc/array.c | 10 +
> include/linux/pid_namespace.h | 2
> kernel/pid.c | 12 ++
> kernel/signal.c | 211 ++--
> 5 files changed, 230 insertions(+), 10 deletions(-)
>
> Index: 2.6.23-rc8-mm2/kernel/signal.c
> ===
> --- 2.6.23-rc8-mm2.orig/kernel/signal.c 2007-10-09 22:41:29.000000000 -0700
> +++ 2.6.23-rc8-mm2/kernel/signal.c 2007-10-09 22:48:28.000000000 -0700
> @@ -46,7 +46,7 @@ static int sig_init_ignore(struct task_s
>
> // Currently this check is a bit racy with exec(),
> // we can _simplify_ de_thread and close the race.
> - if (likely(!is_cgroup_init(tsk->group_leader)))
> + if (likely(!is_container_init(tsk->group_leader)))
> return 0;
>
> if (is_current_in_ancestor_pid_ns(tsk) && !in_interrupt())
> @@ -119,11 +119,28 @@ static inline int has_pending_signals(si
>
> #define PENDING(p,b) has_pending_signals(&(p)->signal, (b))
>
> +static int task_has_pending_signals(struct task_struct *t)
> +{
> + if (PENDING(&t->pending, &t->blocked) ||
> + PENDING(&t->signal->shared_pending, &t->blocked))
> + return 1;
> +
> + if (is_container_init(t)) {
> + struct pid_namespace *ns = task_active_pid_ns(t);
> +
> + if (!ns)
> + return 0;
> +

```
> +  return PENDING(&ns->cinit_blocked_pending, &t->blocked) ||
> +   PENDING(&ns->cinit_blocked_shared_pending, &t->blocked);
> + }
> + return 0;
> +}
> +
> +
>  static int recalc_sigpending_tsk(struct task_struct *t)
>  {
> - if (t->signal->group_stop_count > 0 ||
> -     PENDING(&t->pending, &t->blocked) ||
> -     PENDING(&t->signal->shared_pending, &t->blocked)) {
> + if (t->signal->group_stop_count > 0 || task_has_pending_signals(t)) {
>    set_tsk_thread_flag(t, TIF_SIGPENDING);
>    return 1;
>   }
> @@ -235,6 +252,20 @@ void flush_sigqueue(struct sigpending *q
>   }
>  }
>
> +static void flush_cinit_signals(struct task_struct *t)
> +{
> + struct pid_namespace *ns;
> +
> + if (!is_container_init(t))
> +  return;
> + ns = task_active_pid_ns(t);
> + if (ns) {
> +  flush_sigqueue(&ns->cinit_blocked_pending);
> +  flush_sigqueue(&ns->cinit_blocked_shared_pending);
> +
> + }
> +}
> +
>  /*
>   * Flush all pending signals for a task.
>   */
> @@ -246,6 +277,7 @@ void flush_signals(struct task_struct *t
>   clear_tsk_thread_flag(t,TIF_SIGPENDING);
>   flush_sigqueue(&t->pending);
>   flush_sigqueue(&t->signal->shared_pending);
> + flush_cinit_signals(t);
>   spin_unlock_irqrestore(&t->sighand->siglock, flags);
>  }
>
> @@ -400,6 +432,12 @@ int dequeue_signal(struct task_struct *t
>   /* We only dequeue private signals from ourselves, we don't let
>    * signalfd steal them
```

```
>    */
> + /*
> +  * Note: We can ignore any signals for container init that are
> +  * queued in the pid namespace since they are only queued there
> +  * when signals are blocked (and we don't dequeue blocked signals
> +  * here anyway)
> +  */
>   signr = __dequeue_signal(&tsk->pending, mask, info);
>   if (!signr) {
>    signr = __dequeue_signal(&tsk->signal->shared_pending,
> @@ -579,6 +617,21 @@ static void do_notify_parent_cldstop(str
>   * actual continuing for SIGCONT, but not the actual stopping for stop
>   * signals.  The process stop is done as a signal action for SIG_DFL.
>   */
> +static void rm_from_cinit_queue(unsigned long mask, struct task_struct *t)
> +{
> + struct pid_namespace *ns;
> +
> + if (!is_container_init(t))
> +  return;
> +
> + ns = task_active_pid_ns(t);
> +
> + rm_from_queue(mask, &ns->cinit_blocked_pending);
> + rm_from_queue(mask, &ns->cinit_blocked_shared_pending);
> +
> + return;
> +}
> +
>  static void handle_stop_signal(int sig, struct task_struct *p)
>  {
>   struct task_struct *t;
> @@ -597,6 +650,7 @@ static void handle_stop_signal(int sig,
>     * This is a stop signal.  Remove SIGCONT from all queues.
>     */
>    rm_from_queue(sigmask(SIGCONT), &p->signal->shared_pending);
> +  rm_from_cinit_queue(sigmask(SIGCONT), p);
>    t = p;
>    do {
>     rm_from_queue(sigmask(SIGCONT), &t->pending);
> @@ -627,6 +681,7 @@ static void handle_stop_signal(int sig,
>     spin_lock(&p->sighand->siglock);
>    }
>    rm_from_queue(SIG_KERNEL_STOP_MASK, &p->signal->shared_pending);
> +  rm_from_cinit_queue(SIG_KERNEL_STOP_MASK, p);
>    t = p;
>    do {
>     unsigned int state;
```

```
> @@ -802,6 +857,7 @@ static int
>  specific_send_sig_info(int sig, struct siginfo *info, struct task_struct *t)
>  {
>   int ret = 0;
> + struct sigpending *pending;
>
>   BUG_ON(!irqs_disabled());
>   assert_spin_locked(&t->sighand->siglock);
> @@ -816,13 +872,75 @@ specific_send_sig_info(int sig, struct s
>   if (LEGACY_QUEUE(&t->pending, sig))
>    goto out;
>
> - ret = send_signal(sig, info, t, &t->pending);
> + pending = &t->pending;
> + if (is_container_init(t) && sigismember(&t->blocked, sig) &&
> +   !is_current_in_ancestor_pid_ns(t)) {
> +  struct pid_namespace *ns = task_active_pid_ns(t);
> +
> +  /*
> +   * Hmm. If container init is exiting (ns == NULL) and we
> +   * are trying to post a blocked signal.  Post to the normal
> +   * queue for now.
> +   */
> +  if (ns && LEGACY_QUEUE(&ns->cinit_blocked_pending, sig))
> +   goto out;
> +  else if (ns)
> +   pending = &ns->cinit_blocked_pending;
> + }
> +
> + ret = send_signal(sig, info, t, pending);
>   if (!ret && !sigismember(&t->blocked, sig))
>    signal_wake_up(t, sig == SIGKILL);
>  out:
>   return ret;
>  }
>
> +static void requeue_cinit_signals(struct task_struct *t, sigset_t *set)
> +{
> + struct sigqueue *q, *n;
> + struct pid_namespace *ns = task_active_pid_ns(t);
> + struct sigpending *ns_pending;
> + struct sigpending *ns_shpending;
> + struct sigpending *tsk_pending;
> + struct sigpending *tsk_shpending;
> +
> + /*
> +  * Unblocking a signal while a process is exitng ?
> +  */
```

```
> + if (!ns)
> +  return;
> +
> + /*
> +  * For each signal being unblocked, remove it from the namespace
> +  * pending list and add it to task's pending list
> +  */
> + tsk_pending = &t->pending;
> + ns_pending = &ns->cinit_blocked_pending;
> +
> + list_for_each_entry_safe(q, n, &ns_pending->list, list) {
> +  if (sigismember(set, q->info.si_signo)) {
> +   list_del_init(&q->list);
> +   sigdelset(&ns_pending->signal, q->info.si_signo);
> +
> +   list_add_tail(&q->list, &tsk_pending->list);
> +   sigaddset(&tsk_pending->signal, q->info.si_signo);
> +  }
> + }
> +
> + /* Repeat for the shared-pending signals */
> + tsk_shpending = &t->signal->shared_pending;
> + ns_shpending = &ns->cinit_blocked_shared_pending;
> + list_for_each_entry_safe(q, n, &ns_shpending->list, list) {
> +  if (sigismember(set, q->info.si_signo)) {
> +   list_del_init(&q->list);
> +   sigdelset(&ns_shpending->signal, q->info.si_signo);
> +
> +   list_add_tail(&q->list, &tsk_shpending->list);
> +   sigaddset(&tsk_shpending->signal, q->info.si_signo);
> +  }
> + }
> +}
> +
>  /*
>   * Force a signal that the process can't ignore: if necessary
>   * we unblock the signal and change any SIG_IGN to SIG_DFL.
> @@ -848,6 +966,20 @@ force_sig_info(int sig, struct siginfo *
>    action->sa.sa_handler = SIG_DFL;
>    if (blocked) {
>     sigdelset(&t->blocked, sig);
> +   if (is_container_init(t)) {
> +    struct pid_namespace *ns;
> +    sigset_t set;
> +    /*
> +     * We just unblocked a signal. Requeue any
> +     * pending instances of the signal on the
> +     * namespace queue to the task's pending
```

```
> +     * queue.
> +     */
> +    sigemptyset(&set);
> +    sigaddset(&set, sigmask(sig));
> +    ns = task_active_pid_ns(t);
> +    requeue_cinit_signals(t, &set);
> +   }
>     recalc_sigpending_and_wake(t);
>    }
>   }
> @@ -890,6 +1022,13 @@ __group_complete_signal(int sig, struct
>   struct task_struct *t;
>
>   /*
> +  * If signal came from same or descendant namespace and is a
> +  * blocked signal, we process the signal when we unblock it
> +  */
> + if (!is_current_in_ancestor_pid_ns(p) && sigismember(&p->blocked, sig))
> +  return;
> +
> + /*
>    * Now find a thread we can wake up to take the signal off the queue.
>    *
>    * If the main thread wants the signal, it gets first crack.
> @@ -987,6 +1126,7 @@ int
>  __group_send_sig_info(int sig, struct siginfo *info, struct task_struct *p)
>  {
>   int ret = 0;
> + struct sigpending *pending;
>
>   assert_spin_locked(&p->sighand->siglock);
>   handle_stop_signal(sig, p);
> @@ -999,12 +1139,28 @@ __group_send_sig_info(int sig, struct si
>    /* This is a non-RT signal and we already have one queued.  */
>    return ret;
>
> + pending = &p->signal->shared_pending;
> + if (is_container_init(p) && sigismember(&p->blocked, sig) &&
> +   !is_current_in_ancestor_pid_ns(p)) {
> +  struct pid_namespace *ns = task_active_pid_ns(p);
> +
> +  /*
> +   * Hmm. If container init is exiting (ns == NULL) and we
> +   * are trying to post a blocked signal, what should we do ?
> +   * Post to the normal queue for now.
> +   */
> +  if (ns && LEGACY_QUEUE(&ns->cinit_blocked_shared_pending, sig))
> +   return ret;
```

```
> +  else if (ns)
> +   pending = &ns->cinit_blocked_shared_pending;
> + }
> +
>   /*
>    * Put this signal on the shared-pending queue, or fail with EAGAIN.
>    * We always use the shared queue for process-wide signals,
>    * to avoid several races.
>    */
> - ret = send_signal(sig, info, p, &p->signal->shared_pending);
> + ret = send_signal(sig, info, p, pending);
>   if (unlikely(ret))
>    return ret;
>
> @@ -1365,6 +1521,7 @@ int send_sigqueue(int sig, struct sigque
>  {
>   unsigned long flags;
>   int ret = 0;
> + struct sigpending *pending;
>
>   BUG_ON(!(q->flags & SIGQUEUE_PREALLOC));
>
> @@ -1397,14 +1554,23 @@ int send_sigqueue(int sig, struct sigque
>    ret = 1;
>    goto out;
>   }
> +
> + pending = &p->pending;
> + if (is_container_init(p) && sigismember(&p->blocked, sig) &&
> +   !is_current_in_ancestor_pid_ns(p)) {
> +  struct pid_namespace *ns = task_active_pid_ns(p);
> +  if (ns)
> +   pending = &ns->cinit_blocked_pending;
> + }
> +
>   /*
>    * Deliver the signal to listening signalfds. This must be called
>    * with the sighand lock held.
>    */
>   signalfd_notify(p, sig);
>
> - list_add_tail(&q->list, &p->pending.list);
> - sigaddset(&p->pending.signal, sig);
> + list_add_tail(&q->list, &pending->list);
> + sigaddset(&pending->signal, sig);
>   if (!sigismember(&p->blocked, sig))
>    signal_wake_up(p, sig == SIGKILL);
>
```

```
> @@ -1421,6 +1587,7 @@ send_group_sigqueue(int sig, struct sigq
>  {
>   unsigned long flags;
>   int ret = 0;
> + struct sigpending *shpending;
>
>   BUG_ON(!(q->flags & SIGQUEUE_PREALLOC));
>
> @@ -1435,6 +1602,14 @@ send_group_sigqueue(int sig, struct sigq
>    goto out;
>   }
>
> + shpending = &p->signal->shared_pending;
> + if (is_container_init(p) && sigismember(&p->blocked, sig) &&
> +   !is_current_in_ancestor_pid_ns(p)) {
> +  struct pid_namespace *ns = task_active_pid_ns(p);
> +  if (ns)
> +   shpending = &ns->cinit_blocked_shared_pending;
> + }
> +
>   if (unlikely(!list_empty(&q->list))) {
>    /*
>     * If an SI_TIMER entry is already queue just increment
> @@ -1456,8 +1631,8 @@ send_group_sigqueue(int sig, struct sigq
>    * We always use the shared queue for process-wide signals,
>    * to avoid several races.
>    */
> - list_add_tail(&q->list, &p->signal->shared_pending.list);
> - sigaddset(&p->signal->shared_pending.signal, sig);
> + list_add_tail(&q->list, &shpending->list);
> + sigaddset(&shpending->signal, sig);
>
>   __group_complete_signal(sig, p);
>  out:
> @@ -2029,6 +2204,7 @@ int sigprocmask(int how, sigset_t *set,
>    break;
>   case SIG_UNBLOCK:
>    signandsets(&current->blocked, &current->blocked, set);
> +  requeue_cinit_signals(current, set);
>    break;
>   case SIG_SETMASK:
>    current->blocked = *set;
> @@ -2082,15 +2258,25 @@ long do_sigpending(void __user *set, uns
>  {
>   long error = -EINVAL;
>   sigset_t pending;
> + sigset_t ns_pending;
> + struct pid_namespace *ns;
```

```
>
>   if (sigsetsize > sizeof(sigset_t))
>    goto out;
>
> + sigemptyset(&ns_pending);
>   spin_lock_irq(&current->sighand->siglock);
> + if (is_container_init(current)) {
> +  ns = task_active_pid_ns(current);
> +  sigorsets(&ns_pending, &ns->cinit_blocked_pending.signal,
> +    &ns->cinit_blocked_shared_pending.signal);
> + }
>   sigorsets(&pending, &current->pending.signal,
>      &current->signal->shared_pending.signal);
>   spin_unlock_irq(&current->sighand->siglock);
>
> + sigorsets(&pending, &pending, &ns_pending);
> +
>   /* Outside the lock because only this thread touches it.  */
>   sigandsets(&pending, &current->blocked, &pending);
>
> @@ -2387,6 +2573,11 @@ int do_sigaction(int sig, struct k_sigac
>      rm_from_queue_full(&mask, &t->pending);
>      t = next_thread(t);
>     } while (t != current);
> +   if (is_container_init(t)) {
> +    struct pid_namespace *ns = task_active_pid_ns(t);
> +    rm_from_queue_full(&mask, &ns->cinit_blocked_pending);
> +    rm_from_queue_full(&mask, &ns->cinit_blocked_shared_pending);
> +   }
>    }
>   }
>
> Index: 2.6.23-rc8-mm2/fs/binfmt_elf.c
> ===================================================================
> --- 2.6.23-rc8-mm2.orig/fs/binfmt_elf.c 2007-10-09 22:41:28.000000000 -0700
> +++ 2.6.23-rc8-mm2/fs/binfmt_elf.c 2007-10-09 22:41:49.000000000 -0700
> @@ -42,6 +42,7 @@
>  #include <asm/uaccess.h>
>  #include <asm/param.h>
>  #include <asm/page.h>
> +#include <linux/pid_namespace.h>
>
>  static int load_elf_binary(struct linux_binprm *bprm, struct pt_regs *regs);
>  static int load_elf_library(struct file *);
> @@ -1449,6 +1450,10 @@ static void fill_prstatus(struct elf_prs
>  {
>   prstatus->pr_info.si_signo = prstatus->pr_cursig = signr;
>   prstatus->pr_sigpend = p->pending.signal.sig[0];
```

```
> + if (is_container_init(p)) {
> +  struct pid_namespace *ns = task_active_pid_ns(p);
> +  prstatus->pr_sigpend |= ns->cinit_blocked_pending.signal.sig[0];
> + }
>   prstatus->pr_sighold = p->blocked.sig[0];
>   prstatus->pr_pid = task_pid_vnr(p);
>   prstatus->pr_ppid = task_pid_vnr(p->parent);
> Index: 2.6.23-rc8-mm2/fs/proc/array.c
> ===================================================================
> --- 2.6.23-rc8-mm2.orig/fs/proc/array.c 2007-10-09 22:41:28.000000000 -0700
> +++ 2.6.23-rc8-mm2/fs/proc/array.c 2007-10-09 22:41:49.000000000 -0700
> @@ -264,6 +264,16 @@ static inline char *task_sig(struct task
>   if (lock_task_sighand(p, &flags)) {
>    pending = p->pending.signal;
>    shpending = p->signal->shared_pending.signal;
> +  if (is_container_init(p)) {
> +   struct pid_namespace *ns = task_active_pid_ns(p);
> +   if (ns) {
> +    sigorsets(&pending, &pending,
> +     &ns->cinit_blocked_pending.signal);
> +    sigorsets(&shpending, &shpending,
> +     &ns->cinit_blocked_shared_pending.signal);
> +   }
> +  }
> +
>    blocked = p->blocked;
>    collect_sigign_sigcatch(p, &ignored, &caught);
>    num_threads = atomic_read(&p->signal->count);
> Index: 2.6.23-rc8-mm2/include/linux/pid_namespace.h
> ===================================================================
> --- 2.6.23-rc8-mm2.orig/include/linux/pid_namespace.h 2007-10-09 22:41:28.000000000 -0700
> +++ 2.6.23-rc8-mm2/include/linux/pid_namespace.h 2007-10-09 22:41:49.000000000 -0700
> @@ -25,6 +25,8 @@ struct pid_namespace {
>  #ifdef CONFIG_PROC_FS
>   struct vfsmount *proc_mnt;
>  #endif
> + struct sigpending cinit_blocked_pending;
> + struct sigpending cinit_blocked_shared_pending;
>  };
>
>  extern struct pid_namespace init_pid_ns;
> Index: 2.6.23-rc8-mm2/kernel/pid.c
> ===================================================================
> --- 2.6.23-rc8-mm2.orig/kernel/pid.c 2007-10-09 22:41:28.000000000 -0700
> +++ 2.6.23-rc8-mm2/kernel/pid.c 2007-10-09 22:41:49.000000000 -0700
> @@ -78,6 +78,12 @@ struct pid_namespace init_pid_ns = {
>   .last_pid = 0,
>   .level = 0,
```

```
>   .child_reaper = &init_task,
> + .cinit_blocked_pending = {
> +  .list = LIST_HEAD_INIT(init_pid_ns.cinit_blocked_pending.list),
> +  .signal = {{0}}},
> + .cinit_blocked_shared_pending = {
> +  .list = LIST_HEAD_INIT(init_pid_ns.cinit_blocked_shared_pending.list),
> +  .signal = {{0}}}
>  };
>  EXPORT_SYMBOL_GPL(init_pid_ns);
>
> @@ -621,6 +627,8 @@ static struct pid_namespace *create_pid_
>   ns->last_pid = 0;
>   ns->child_reaper = NULL;
>   ns->level = level;
> + init_sigpending(&ns->cinit_blocked_pending);
> + init_sigpending(&ns->cinit_blocked_shared_pending);
>
>   set_bit(0, ns->pidmap[0].page);
>   atomic_set(&ns->pidmap[0].nr_free, BITS_PER_PAGE - 1);
> @@ -646,6 +654,10 @@ static void destroy_pid_namespace(struct
>
>   for (i = 0; i < PIDMAP_ENTRIES; i++)
>    kfree(ns->pidmap[i].page);
> +
> + flush_sigqueue(&ns->cinit_blocked_pending);
> + flush_sigqueue(&ns->cinit_blocked_shared_pending);
> +
>   kmem_cache_free(pid_ns_cachep, ns);
>  }
```

_____

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