Subject: [RFC][PATCH] Add child reaper to struct pspace
Posted by Sukadev Bhattiprolu on Thu, 07 Sep 2006 00:48:52 GMT
View Forum Message <> Reply to Message

Add per-pid-space child-reaper. This is needed so processes are reaped within
the same pid space and do not spill over to the parent pid space. Its also
needed so containers preserve existing semantic that pid == 1 would reap
orphaned children.

This is based on Eric Biederman's patch: http://lkml.org/lkml/2006/2/6/285

Signed-off-by: Sukadev Bhattiprolu <sukadev@us.ibm.com>
Cc: Eric Biederman <ebiederm@xmission.com>
Cc: Cedric Le Goater <clg@fr.ibm.com>
Cc: Dave Hansen <haveblue@us.ibm.com>
Cc: Serge Hallyn <serue@us.ibm.com>
Cc: containers@lists.osdl.org

```
 fs/exec.c             |    4 ++--
 include/linux/pid.h   |    5 +++--
 include/linux/pspace.h |    1 +
 include/linux/sched.h  |    2 +-
 init/main.c           |    5 ++---
 kernel/exit.c         |   15 ++++++++-------
 kernel/signal.c       |    8 ++++++--
 7 files changed, 23 insertions(+), 17 deletions(-)

Index: lx26-18-rc5/include/linux/pspace.h
===================================================================
--- lx26-18-rc5.orig/include/linux/pspace.h 2006-09-06 17:04:03.000000000 -0700
+++ lx26-18-rc5/include/linux/pspace.h 2006-09-06 17:39:37.000000000 -0700
@@ -16,6 +16,7 @@ typedef struct pidmap
 struct pspace {
        struct pidmap pidmap[PIDMAP_ENTRIES];
        int last_pid;
+       struct task_struct * child_reaper;
 };

 extern struct pspace init_pspace;
Index: lx26-18-rc5/init/main.c
===================================================================
--- lx26-18-rc5.orig/init/main.c 2006-09-06 17:04:03.000000000 -0700
+++ lx26-18-rc5/init/main.c 2006-09-06 17:05:00.000000000 -0700
@@ -51,6 +51,7 @@
 #include <linux/buffer_head.h>
 #include <linux/debug_locks.h>
 #include <linux/lockdep.h>
+#include <linux/pspace.h>
```

```
 #include <asm/io.h>
 #include <asm/bugs.h>
@@ -596,8 +597,6 @@ static int __init initcall_debug_setup(c
 }
 __setup("initcall_debug", initcall_debug_setup);

-struct task_struct *child_reaper = &init_task;
-
 extern initcall_t __initcall_start[], __initcall_end[];

 static void __init do_initcalls(void)
@@ -697,7 +696,7 @@ static int init(void * unused)
   * assumptions about where in the task array this
   * can be found.
   */
- child_reaper = current;
+ init_pspace.child_reaper = current;

  smp_prepare_cpus(max_cpus);

Index: lx26-18-rc5/fs/exec.c
===================================================================
--- lx26-18-rc5.orig/fs/exec.c 2006-09-06 17:04:03.000000000 -0700
+++ lx26-18-rc5/fs/exec.c 2006-09-06 17:39:29.000000000 -0700
@@ -620,8 +620,8 @@ static int de_thread(struct task_struct
   * Reparenting needs write_lock on tasklist_lock,
   * so it is safe to do it under read_lock.
   */
- if (unlikely(current->group_leader == child_reaper))
-  child_reaper = current;
+ if (unlikely(current->group->leader == current->pspace->child_reaper)
+  current->pspace->child_reaper = current;

  zap_other_threads(current);
  read_unlock(&tasklist_lock);
Index: lx26-18-rc5/include/linux/pid.h
===================================================================
--- lx26-18-rc5.orig/include/linux/pid.h 2006-09-06 17:04:03.000000000 -0700
+++ lx26-18-rc5/include/linux/pid.h 2006-09-06 17:05:00.000000000 -0700
@@ -35,8 +35,9 @@ enum pid_type
  *
  * Holding a reference to struct pid solves both of these problems.
  * It is small so holding a reference does not consume a lot of
- * resources, and since a new struct pid is allocated when the numeric
- * pid value is reused we don't mistakenly refer to new processes.
+ * resources, and since a new struct pid is allocated when the numeric pid
+ * value is reused (when pids wrap around) we don't mistakenly refer to new
```

```
+ * processes.
  */

 struct pid
Index: lx26-18-rc5/include/linux/sched.h
===================================================================
--- lx26-18-rc5.orig/include/linux/sched.h 2006-09-06 17:04:03.000000000 -0700
+++ lx26-18-rc5/include/linux/sched.h 2006-09-06 17:05:00.000000000 -0700
@@ -80,6 +80,7 @@ struct sched_param {
 #include <linux/resource.h>
 #include <linux/timer.h>
 #include <linux/hrtimer.h>
+#include <linux/pspace.h>

 #include <asm/processor.h>

@@ -1297,7 +1298,6 @@ extern NORET_TYPE void do_group_exit(int
 extern void daemonize(const char *, ...);
 extern int allow_signal(int);
 extern int disallow_signal(int);
-extern struct task_struct *child_reaper;

 extern int do_execve(char *, char __user * __user *, char __user * __user *, struct pt_regs *);
 extern long do_fork(unsigned long, unsigned long, struct pt_regs *, unsigned long, int __user *,
int __user *);
Index: lx26-18-rc5/kernel/exit.c
===================================================================
--- lx26-18-rc5.orig/kernel/exit.c 2006-09-06 17:04:03.000000000 -0700
+++ lx26-18-rc5/kernel/exit.c 2006-09-06 17:18:47.000000000 -0700
@@ -45,7 +45,6 @@
 #include <asm/mmu_context.h>

 extern void sem_exit (void);
-extern struct task_struct *child_reaper;

 static void exit_mm(struct task_struct * tsk);

@@ -267,7 +266,8 @@ static int has_stopped_jobs(int pgrp)
 }

 /**
- * reparent_to_init - Reparent the calling kernel thread to the init task.
+ * reparent_to_init - Reparent the calling kernel thread to the init task
+ * of the pid space that the thread belongs to.
  *
  * If a kernel thread is launched as a result of a system call, or if
  * it ever exits, it should generally reparent itself to init so that
@@ -285,8 +285,8 @@ static void reparent_to_init(void)
```

```
  ptrace_unlink(current);
  /* Reparent to init */
  remove_parent(current);
- current->parent = child_reaper;
- current->real_parent = child_reaper;
+ current->parent = current->pspace->child_reaper;
+ current->real_parent = current->pspace->child_reaper;
  add_parent(current);

  /* Set the exit signal to SIGCHLD so we signal init on exit */
@@ -658,7 +658,8 @@ reparent_thread(struct task_struct *p, s
  * When we die, we re-parent all our children.
  * Try to give them to another thread in our thread
  * group, and if no such member exists, give it to
- * the global child reaper process (ie "init")
+ * the child reaper process (ie "init") in our pid
+ * space.
  */
 static void
 forget_original_parent(struct task_struct *father, struct list_head *to_release)
@@ -669,7 +670,7 @@ forget_original_parent(struct task_struc
  do {
   reaper = next_thread(reaper);
   if (reaper == father) {
-   reaper = child_reaper;
+   reaper = father->pspace->child_reaper;
    break;
   }
  } while (reaper->exit_state);
@@ -857,7 +858,7 @@ fastcall NORET_TYPE void do_exit(long co
   panic("Aiee, killing interrupt handler!");
  if (unlikely(!tsk->pid))
   panic("Attempted to kill the idle task!");
- if (unlikely(tsk == child_reaper))
+ if (unlikely(tsk == tsk->pspace->child_reaper))
   panic("Attempted to kill init!");

  if (unlikely(current->ptrace & PT_TRACE_EXIT)) {
Index: lx26-18-rc5/kernel/signal.c
===================================================================
--- lx26-18-rc5.orig/kernel/signal.c 2006-09-06 17:04:03.000000000 -0700
+++ lx26-18-rc5/kernel/signal.c 2006-09-06 17:05:00.000000000 -0700
@@ -1835,8 +1835,12 @@ relock:
   if (sig_kernel_ignore(signr)) /* Default is nothing. */
    continue;

-  /* Init gets no signals it doesn't want.  */
-  if (current == child_reaper)
```

```
+  /*
+   * Init of a pid space gets no signals it doesn't want from
+   * within that pid space. It can of course get signals from
+   * its parent pid space.
+   */
+  if (current == current->pspace->child_reaper)
    continue;

   if (sig_kernel_stop(signr)) {
```

Containers mailing list
Containers@lists.osdl.org
https://lists.osdl.org/mailman/listinfo/containers