Subject: [PATCH] pid: sys_wait... fixes
Posted by ebiederm on Thu, 06 Dec 2007 03:14:49 GMT
View Forum Message <> Reply to Message

This modifies do_wait and eligible_child to take a pair of
enum pid_type and struct pid *pid to precisely specify what
set of processes are eligible to be waited for, instead of the
raw pid_t value from sys_wait4.

This fixes a bug in sys_waitid where you could not wait for children
in just process group 1.

This fixes a pid namespace crossing case in eligible_child. Allowing
us to wait for a processes in our current process group even if
our current process group == 0.

This allows the no child with this pid case to be optimized.
This allows us to optimize the pid membership test in eligible
child to be optimized.

This even closes a theoretical pid wraparound race where in
a threaded parent if two threads are waiting for the same child
and one thread picks up the child and the pid numbers wrap around
and generate another child with that same pid before the other
thread is scheduled (teribly insanely unlikely) we could end
up waiting on the second child with the same pid# and not discover
that the specific child we were waiting for has exited.

Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>

```
---
 kernel/exit.c |   72 ++++++++++++++++++++++++++++++++++++-------------------
 1 files changed, 48 insertions(+), 24 deletions(-)

diff --git a/kernel/exit.c b/kernel/exit.c
index aede730..9e4e22a 100644
--- a/kernel/exit.c
+++ b/kernel/exit.c
@@ -1106,20 +1106,13 @@ asmlinkage void sys_exit_group(int error_code)
  do_group_exit((error_code & 0xff) << 8);
 }

-static int eligible_child(pid_t pid, int options, struct task_struct *p)
+static int eligible_child(enum pid_type type, struct pid *pid, int options,
+     struct task_struct *p)
 {
  int err;
- struct pid_namespace *ns;
```

```
- ns = current->nsproxy->pid_ns;
- if (pid > 0) {
-  if (task_pid_nr_ns(p, ns) != pid)
-   return 0;
- } else if (!pid) {
-  if (task_pgrp_nr_ns(p, ns) != task_pgrp_vnr(current))
-   return 0;
- } else if (pid != -1) {
-  if (task_pgrp_nr_ns(p, ns) != -pid)
+ if (type < PIDTYPE_MAX) {
+  if (p->pids[type].pid != pid)
    return 0;
  }

@@ -1143,7 +1136,7 @@ static int eligible_child(pid_t pid, int options, struct task_struct *p)
  if (likely(!err))
   return 1;

- if (pid <= 0)
+ if (type != PIDTYPE_PID)
   return 0;
  /* This child was explicitly requested, abort */
  read_unlock(&tasklist_lock);
@@ -1463,8 +1456,9 @@ static int wait_task_continued(struct task_struct *p, int noreap,
  return retval;
 }

-static long do_wait(pid_t pid, int options, struct siginfo __user *infop,
-      int __user *stat_addr, struct rusage __user *ru)
+static long do_wait(enum pid_type type, struct pid *pid, int options,
+      struct siginfo __user *infop, int __user *stat_addr,
+      struct rusage __user *ru)
 {
  DECLARE_WAITQUEUE(wait, current);
  struct task_struct *tsk;
@@ -1472,6 +1466,11 @@ static long do_wait(pid_t pid, int options, struct siginfo __user *infop,

  add_wait_queue(&current->signal->wait_chldexit,&wait);
 repeat:
+ /* If there is nothing that can match our critier just get out */
+ retval = -ECHILD;
+ if ((type < PIDTYPE_MAX) && (!pid || hlist_empty(&pid->tasks[type])))
+  goto end;
+
  /*
   * We will set this flag if we see any child that might later
   * match our criteria, even if we are not able to reap it yet.
@@ -1484,7 +1483,7 @@ repeat:
```

```
   struct task_struct *p;

   list_for_each_entry(p, &tsk->children, sibling) {
-    int ret = eligible_child(pid, options, p);
+    int ret = eligible_child(type, pid, options, p);
     if (!ret)
      continue;

@@ -1531,7 +1530,7 @@ repeat:
    if (!flag) {
     list_for_each_entry(p, &tsk->ptrace_children,
         ptrace_list) {
-     flag = eligible_child(pid, options, p);
+     flag = eligible_child(type, pid, options, p);
      if (!flag)
       continue;
      if (likely(flag > 0))
@@ -1586,10 +1585,12 @@ end:
  return retval;
 }

-asmlinkage long sys_waitid(int which, pid_t pid,
+asmlinkage long sys_waitid(int which, pid_t upid,
        struct siginfo __user *infop, int options,
        struct rusage __user *ru)
 {
+ struct pid *pid = NULL;
+ enum pid_type type;
  long ret;

  if (options & ~(WNOHANG|WNOWAIT|WEXITED|WSTOPPED|WCONTINUED))
@@ -1599,37 +1600,60 @@ asmlinkage long sys_waitid(int which, pid_t pid,

  switch (which) {
  case P_ALL:
-  pid = -1;
+  type = PIDTYPE_MAX;
    break;
  case P_PID:
-  if (pid <= 0)
+  type = PIDTYPE_PID;
+  if (upid <= 0)
     return -EINVAL;
    break;
  case P_PGID:
-  if (pid <= 0)
+  type = PIDTYPE_PGID;
+  if (upid <= 0)
```

```
    return -EINVAL;
-   pid = -pid;
    break;
   default:
    return -EINVAL;
   }

- ret = do_wait(pid, options, infop, NULL, ru);
+ if (type < PIDTYPE_MAX)
+   pid = find_get_pid(upid);
+ ret = do_wait(type, pid, options, infop, NULL, ru);
+ put_pid(pid);

   /* avoid REGPARM breakage on x86: */
   prevent_tail_call(ret);
   return ret;
 }

-asmlinkage long sys_wait4(pid_t pid, int __user *stat_addr,
+asmlinkage long sys_wait4(pid_t upid, int __user *stat_addr,
       int options, struct rusage __user *ru)
 {
+ struct pid *pid = NULL;
+ enum pid_type type;
   long ret;

   if (options & ~(WNOHANG|WUNTRACED|WCONTINUED|
     __WNOTHREAD|__WCLONE|__WALL))
    return -EINVAL;
- ret = do_wait(pid, options | WEXITED, NULL, stat_addr, ru);
+
+ if (upid == -1)
+   type = PIDTYPE_MAX;
+ else if (upid < 0) {
+   type = PIDTYPE_PGID;
+   pid = find_get_pid(-upid);
+ }
+ else if (upid == 0) {
+   type = PIDTYPE_PGID;
+   pid = get_pid(task_pgrp(current));
+ }
+ else /* upid > 0 */ {
+   type= PIDTYPE_PID;
+   pid = find_get_pid(upid);
+ }
+
+ ret = do_wait(type, pid, options | WEXITED, NULL, stat_addr, ru);
+ put_pid(pid);
```

```
  /* avoid REGPARM breakage on x86: */
  prevent_tail_call(ret);
--
1.5.3.rc6.17.g1911
```

_____

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