Subject: [PATCH][NETNS] Use list_for_each_entry_continue_reverse in setup_net
Posted by Pavel Emelianov on Mon, 17 Sep 2007 13:54:33 GMT
View Forum Message <> Reply to Message

[snip]

> Pavel if you are going down this route. Could you look at
> cleanup_net as well. The reverse walk there could probably
> benefit from being list_for_each_entry_reverse.

Oh! Thanks a lot ;) Here's the new patch.

Log:

I proposed introducing a list_for_each_entry_continue_reverse
macro to be used in setup_net() when unrolling the failed
->init callback.

Here is the macro and some more cleanup in the setup_net() itself
to remove one variable from the stack :) Minor, but the code
looks nicer.

Signed-off-by: Pavel Emelyanov <xemul@openvz.org>
Acked-by: "Eric W. Biederman" <ebiederm@xmission.com>

```
diff --git a/include/linux/list.h b/include/linux/list.h
index f29fc9c..ad9dcb9 100644
--- a/include/linux/list.h
+++ b/include/linux/list.h
@@ -525,6 +525,20 @@ static inline void list_splice_init_rcu(
       pos = list_entry(pos->member.next, typeof(*pos), member))

 /**
+ * list_for_each_entry_continue_reverse - iterate backwards from the given point
+ * @pos: the type * to use as a loop cursor.
+ * @head: the head for your list.
+ * @member: the name of the list_struct within the struct.
+ *
+ * Start to iterate over list of given type backwards, continuing after
+ * the current position.
+ */
+#define list_for_each_entry_continue_reverse(pos, head, member)  \
+ for (pos = list_entry(pos->member.prev, typeof(*pos), member); \
+      prefetch(pos->member.prev), &pos->member != (head); \
+      pos = list_entry(pos->member.prev, typeof(*pos), member))
+
```

```
+/**
  * list_for_each_entry_from - iterate over list of given type from the current point
  * @pos: the type * to use as a loop cursor.
  * @head: the head for your list.
diff --git a/net/core/net_namespace.c b/net/core/net_namespace.c
index 1fc513c..0e6cb02 100644
--- a/net/core/net_namespace.c
+++ b/net/core/net_namespace.c
@@ -56,7 +56,6 @@ static void net_free(struct net *net)
 static void cleanup_net(struct work_struct *work)
 {
  struct pernet_operations *ops;
- struct list_head *ptr;
  struct net *net;

  net = container_of(work, struct net, work);
@@ -69,8 +68,7 @@ static void cleanup_net(struct work_stru
  net_unlock();

  /* Run all of the network namespace exit methods */
- list_for_each_prev(ptr, &pernet_list) {
-  ops = list_entry(ptr, struct pernet_operations, list);
+ list_for_each_entry_reverse(ops, &pernet_list, list) {
   if (ops->exit)
    ops->exit(net);
  }
@@ -102,7 +100,6 @@ static int setup_net(struct net *net)
 {
  /* Must be called with net_mutex held */
  struct pernet_operations *ops;
- struct list_head *ptr;
  int error;

  memset(net, 0, sizeof(struct net));
@@ -110,8 +107,7 @@ static int setup_net(struct net *net)
  atomic_set(&net->use_count, 0);

  error = 0;
- list_for_each(ptr, &pernet_list) {
-  ops = list_entry(ptr, struct pernet_operations, list);
+ list_for_each_entry(ops, &pernet_list, list) {
   if (ops->init) {
    error = ops->init(net);
    if (error < 0)
@@ -120,12 +116,12 @@ static int setup_net(struct net *net)
  }
 out:
  return error;
```

```
+
 out_undo:
  /* Walk through the list backwards calling the exit functions
   * for the pernet modules whose init functions did not fail.
   */
- for (ptr = ptr->prev; ptr != &pernet_list; ptr = ptr->prev) {
-  ops = list_entry(ptr, struct pernet_operations, list);
+ list_for_each_entry_continue_reverse(ops, &pernet_list, list) {
   if (ops->exit)
    ops->exit(net);
  }
```