Subject: [RFC][PATCH][LLC] Use existing sock refcnt debugging
Posted by Pavel Emelianov on Fri, 09 Nov 2007 14:43:56 GMT
View Forum Message <> Reply to Message

Hi, Arnaldo.

I've grep-ed through the code and found one more place, where
the sk refcnt debugging is required, but is still performed in
an old fashion - this is the LLC2.

The problem in using the sk_refcnt_debug_xxx here is that these
socks do not provide the sk_destruct callback to catch the
moment of the sock destruction.

Making this callback mandatory is not a good solution, as most
often it will be empty and thus useless. Making this callback
be set under the #ifdef SOCK_REFCNT_DEBUG is even more ugly
than the previous one.

So, I propose to extend the sk_refcnt_debug_xxx set of helperf
for those socks not having the sk_destruct callback by default,
like the LLC2 ones.

The new helper is sk_refcnt_debug_inc_undo(sk) sets the
sk_destruct callback into the sk_refcnt_debug_dec() in case the
SOCK_REFCNT_DEBUG is on.

What do you think about it?

Signed-off-by: Pavel Emelyanov <xemul@openvz.org>

```
diff --git a/include/net/sock.h b/include/net/sock.h
index 5504fb9..1404ab9 100644
--- a/include/net/sock.h
+++ b/include/net/sock.h
@@ -654,10 +654,21 @@ static inline void sk_refcnt_debug_release(const struct sock *sk)
   printk(KERN_DEBUG "Destruction of the %s socket %p delayed, refcnt=%d\n",
          sk->sk_prot->name, sk, atomic_read(&sk->sk_refcnt));
 }
+
+/*
+ * this one is to be used *only* for thouse socks, that
+ * do not have their own sk_destruct callback
+ */
+static inline void sk_refcnt_debug_inc_undo(struct sock *sk)
+{
```

```
+ sk_refcnt_debug_inc(sk);
+ sk->sk_destruct = sk_refcnt_debug_dec;
+}
 #else /* SOCK_REFCNT_DEBUG */
 #define sk_refcnt_debug_inc(sk) do { } while (0)
 #define sk_refcnt_debug_dec(sk) do { } while (0)
 #define sk_refcnt_debug_release(sk) do { } while (0)
+#define sk_refcnt_debug_inc_undo(sk) do { } while (0)
 #endif /* SOCK_REFCNT_DEBUG */

 /* Called with local bh disabled */
diff --git a/net/llc/llc_conn.c b/net/llc/llc_conn.c
index 5c0b484..6ee8778 100644
--- a/net/llc/llc_conn.c
+++ b/net/llc/llc_conn.c
@@ -775,11 +775,6 @@ drop_unlock:
  goto out;
 }

-#undef LLC_REFCNT_DEBUG
-#ifdef LLC_REFCNT_DEBUG
-static atomic_t llc_sock_nr;
-#endif
-
 /**
  * llc_backlog_rcv - Processes rx frames and expired timers.
  * @sk: LLC sock (p8022 connection)
@@ -875,11 +870,7 @@ struct sock *llc_sk_alloc(struct net *net, int family, gfp_t priority, struct pr
   goto out;
  llc_sk_init(sk);
  sock_init_data(NULL, sk);
-#ifdef LLC_REFCNT_DEBUG
- atomic_inc(&llc_sock_nr);
- printk(KERN_DEBUG "LLC socket %p created in %s, now we have %d alive\n", sk,
-  __FUNCTION__, atomic_read(&llc_sock_nr));
-#endif
+ sk_refcnt_debug_inc_undo(sk);
 out:
  return sk;
 }
@@ -905,18 +896,7 @@ void llc_sk_free(struct sock *sk)
  skb_queue_purge(&sk->sk_receive_queue);
  skb_queue_purge(&sk->sk_write_queue);
  skb_queue_purge(&llc->pdu_unack_q);
-#ifdef LLC_REFCNT_DEBUG
- if (atomic_read(&sk->sk_refcnt) != 1) {
-  printk(KERN_DEBUG "Destruction of LLC sock %p delayed in %s, cnt=%d\n",
-   sk, __FUNCTION__, atomic_read(&sk->sk_refcnt));
```

```
-  printk(KERN_DEBUG "%d LLC sockets are still alive\n",
-   atomic_read(&llc_sock_nr));
- } else {
-  atomic_dec(&llc_sock_nr);
-  printk(KERN_DEBUG "LLC socket %p released in %s, %d are still alive\n", sk,
-   __FUNCTION__, atomic_read(&llc_sock_nr));
- }
-#endif
+ sk_refcnt_debug_release(sk);
  sock_put(sk);
 }
```
