Subject: [PATCH v3] SUNRPC: protect service sockets lists during per-net shutdown
Posted by Stanislav Kinsbursky on Tue, 03 Jul 2012 12:58:57 GMT
View Forum Message <> Reply to Message

v3:
1) rebased on 3.5-rc3 kernel.

v2: destruction of currently processing transport added:
1) Added marking of currently processing transports with XPT_CLOSE on per-net shutdown. These transports will be destroyed in svc_xprt_enqueue() (instead of enqueueing).
2) newly created temporary transport in svc_recv() will be destroyed, if it's "parent" was marked with XPT_CLOSE.
3) spin_lock(&serv->sv_lock) was replaced by spin_lock_bh() in svc_close_net(&serv->sv_lock).

Service sv_tempsocks and sv_permsocks lists are accessible by tasks with different network namespaces, and thus per-net service destruction must be protected.
These lists are protected by service sv_lock. So lets wrap list munipulations with this lock and move tranports destruction outside wrapped area to prevent deadlocks.

Signed-off-by: Stanislav Kinsbursky <skinsbursky@parallels.com>
---
 net/sunrpc/svc_xprt.c | 56 ++---
 1 files changed, 52 insertions(+), 4 deletions(-)

```
diff --git a/net/sunrpc/svc_xprt.c b/net/sunrpc/svc_xprt.c
index 88f2bf6..4af2114 100644
--- a/net/sunrpc/svc_xprt.c
+++ b/net/sunrpc/svc_xprt.c
@@ -320,6 +320,7 @@ void svc_xprt_enqueue(struct svc_xprt *xprt)
  struct svc_pool *pool;
  struct svc_rqst *rqstp;
  int cpu;
+ int destroy = 0;

  if (!svc_xprt_has_something_to_do(xprt))
   return;
@@ -338,6 +339,17 @@ void svc_xprt_enqueue(struct svc_xprt *xprt)

  pool->sp_stats.packets++;

+ /*
+  * Check transport close flag. It could be marked as closed on per-net
+  * service shutdown.
```

```
+  */
+ if (test_bit(XPT_CLOSE, &xprt->xpt_flags)) {
+  /* Don't enqueue transport if it has to be destroyed. */
+  dprintk("svc: transport %p have to be closed\n", xprt);
+  destroy++;
+  goto out_unlock;
+ }
+
   /* Mark transport as busy. It will remain in this state until
    * the provider calls svc_xprt_received. We update XPT_BUSY
    * atomically because it also guards against trying to enqueue
@@ -374,6 +386,8 @@ void svc_xprt_enqueue(struct svc_xprt *xprt)

 out_unlock:
  spin_unlock_bh(&pool->sp_lock);
+ if (destroy)
+  svc_delete_xprt(xprt);
 }
 EXPORT_SYMBOL_GPL(svc_xprt_enqueue);

@@ -714,6 +728,13 @@ int svc_recv(struct svc_rqst *rqstp, long timeout)
    __module_get(newxpt->xpt_class->xcl_owner);
    svc_check_conn_limits(xprt->xpt_server);
    spin_lock_bh(&serv->sv_lock);
+   if (test_bit(XPT_CLOSE, &xprt->xpt_flags)) {
+    dprintk("svc_recv: found XPT_CLOSE on listener\n");
+    set_bit(XPT_DETACHED, &newxpt->xpt_flags);
+    spin_unlock_bh(&pool->sp_lock);
+    svc_delete_xprt(newxpt);
+    goto out_closed;
+   }
    set_bit(XPT_TEMP, &newxpt->xpt_flags);
    list_add(&newxpt->xpt_list, &serv->sv_tempsocks);
    serv->sv_tmpcnt++;
@@ -739,6 +760,7 @@ int svc_recv(struct svc_rqst *rqstp, long timeout)
    len = xprt->xpt_ops->xpo_recvfrom(rqstp);
   dprintk("svc: got len=%d\n", len);
  }
+out_closed:
  svc_xprt_received(xprt);

  /* No data, incomplete (TCP) read, or accept() */
@@ -936,6 +958,7 @@ static void svc_clear_pools(struct svc_serv *serv, struct net *net)
  struct svc_pool *pool;
  struct svc_xprt *xprt;
  struct svc_xprt *tmp;
+ struct svc_rqst *rqstp;
  int i;
```

```
  for (i = 0; i < serv->sv_nrpools; i++) {
@@ -947,11 +970,16 @@ static void svc_clear_pools(struct svc_serv *serv, struct net *net)
      continue;
     list_del_init(&xprt->xpt_ready);
    }
+   list_for_each_entry(rqstp, &pool->sp_all_threads, rq_all) {
+    if (rqstp->rq_xprt && rqstp->rq_xprt->xpt_net == net)
+     set_bit(XPT_CLOSE, &rqstp->rq_xprt->xpt_flags);
+   }
    spin_unlock_bh(&pool->sp_lock);
   }
  }

-static void svc_clear_list(struct list_head *xprt_list, struct net *net)
+static void svc_clear_list(struct list_head *xprt_list, struct net *net,
+       struct list_head *kill_list)
  {
   struct svc_xprt *xprt;
   struct svc_xprt *tmp;
@@ -959,7 +987,8 @@ static void svc_clear_list(struct list_head *xprt_list, struct net *net)
   list_for_each_entry_safe(xprt, tmp, xprt_list, xpt_list) {
    if (xprt->xpt_net != net)
     continue;
-   svc_delete_xprt(xprt);
+   list_move(&xprt->xpt_list, kill_list);
+   set_bit(XPT_DETACHED, &xprt->xpt_flags);
   }
   list_for_each_entry(xprt, xprt_list, xpt_list)
    BUG_ON(xprt->xpt_net == net);
@@ -967,6 +996,15 @@ static void svc_clear_list(struct list_head *xprt_list, struct net *net)

  void svc_close_net(struct svc_serv *serv, struct net *net)
  {
+ struct svc_xprt *xprt, *tmp;
+ LIST_HEAD(kill_list);
+
+ /*
+  * Protect the lists, since they can be by tasks with different network
+  * namespace contexts.
+  */
+ spin_lock_bh(&serv->sv_lock);
+
   svc_close_list(&serv->sv_tempsocks, net);
   svc_close_list(&serv->sv_permsocks, net);

@@ -976,8 +1014,18 @@ void svc_close_net(struct svc_serv *serv, struct net *net)
    * svc_xprt_enqueue will not add new entries without taking the
```

```
   * sp_lock and checking XPT_BUSY.
   */
- svc_clear_list(&serv->sv_tempsocks, net);
- svc_clear_list(&serv->sv_permsocks, net);
+ svc_clear_list(&serv->sv_tempsocks, net, &kill_list);
+ svc_clear_list(&serv->sv_permsocks, net, &kill_list);
+
+ spin_unlock_bh(&serv->sv_lock);
+
+ /*
+  * Destroy collected transports.
+  * Note: tranports has been marked as XPT_DETACHED on svc_clear_list(),
+  * so no need to protect againt list_del() in svc_delete_xprt().
+  */
+ list_for_each_entry_safe(xprt, tmp, &kill_list, xpt_list)
+  svc_delete_xprt(xprt);
 }

 /*
```

Subject: Re: [PATCH v3] SUNRPC: protect service sockets lists during per-net shutdown
Posted by bfields on Tue, 24 Jul 2012 19:40:37 GMT
View Forum Message <> Reply to Message

On Tue, Jul 03, 2012 at 04:58:57PM +0400, Stanislav Kinsbursky wrote:
> v3:
> 1) rebased on 3.5-rc3 kernel.
>
> v2: destruction of currently processing transport added:
> 1) Added marking of currently processing transports with XPT_CLOSE on per-net
> shutdown. These transports will be destroyed in svc_xprt_enqueue() (instead of
> enqueueing).

That worries me:

 - Why did we originally defer close until svc_recv?
 - Are we sure there's no risk to performing it immediately in
   svc_enqueue? Is it safe to call from the socket callbacks and
   wherever else we call svc_enqueue?

And in the past I haven't been good at testing for problems
here--instead they tend to show up when a use somewhere tries shutting
down a server that's under load.

I'll look more closely. Meanwhile you could split out that change as a
separate patch and convince me why it's right....

--b.

> 2) newly created temporary transport in svc_recv() will be destroyed, if it's
> "parent" was marked with XPT_CLOSE.
> 3) spin_lock(&serv->sv_lock) was replaced by spin_lock_bh() in
> svc_close_net(&serv->sv_lock).
>
> Service sv_tempsocks and sv_permsocks lists are accessible by tasks with
> different network namespaces, and thus per-net service destruction must be
> protected.
> These lists are protected by service sv_lock. So lets wrap list munipulations
> with this lock and move tranports destruction outside wrapped area to prevent
> deadlocks.
>
> Signed-off-by: Stanislav Kinsbursky <skinsbursky@parallels.com>
> ---
> net/sunrpc/svc_xprt.c | 56 ++---
> 1 files changed, 52 insertions(+), 4 deletions(-)
>
> diff --git a/net/sunrpc/svc_xprt.c b/net/sunrpc/svc_xprt.c
> index 88f2bf6..4af2114 100644
> --- a/net/sunrpc/svc_xprt.c
> +++ b/net/sunrpc/svc_xprt.c
> @@ -320,6 +320,7 @@ void svc_xprt_enqueue(struct svc_xprt *xprt)
>   struct svc_pool *pool;
>   struct svc_rqst *rqstp;
>   int cpu;
> + int destroy = 0;
>
>   if (!svc_xprt_has_something_to_do(xprt))
>   return;
> @@ -338,6 +339,17 @@ void svc_xprt_enqueue(struct svc_xprt *xprt)
>
>   pool->sp_stats.packets++;
>
> + /*
> + * Check transport close flag. It could be marked as closed on per-net
> + * service shutdown.
> + */
> + if (test_bit(XPT_CLOSE, &xprt->xpt_flags)) {
> + /* Don't enqueue transport if it has to be destroyed. */
> + dprintk("svc: transport %p have to be closed\n", xprt);
> + destroy++;
> + goto out_unlock;
> + }
> +
>   /* Mark transport as busy. It will remain in this state until

```
>    * the provider calls svc_xprt_received. We update XPT_BUSY
>    * atomically because it also guards against trying to enqueue
> @@ -374,6 +386,8 @@ void svc_xprt_enqueue(struct svc_xprt *xprt)
>
>  out_unlock:
>   spin_unlock_bh(&pool->sp_lock);
> + if (destroy)
> +  svc_delete_xprt(xprt);
>  }
>  EXPORT_SYMBOL_GPL(svc_xprt_enqueue);
>
> @@ -714,6 +728,13 @@ int svc_recv(struct svc_rqst *rqstp, long timeout)
>     __module_get(newxpt->xpt_class->xcl_owner);
>     svc_check_conn_limits(xprt->xpt_server);
>     spin_lock_bh(&serv->sv_lock);
> +   if (test_bit(XPT_CLOSE, &xprt->xpt_flags)) {
> +    dprintk("svc_recv: found XPT_CLOSE on listener\n");
> +    set_bit(XPT_DETACHED, &newxpt->xpt_flags);
> +    spin_unlock_bh(&pool->sp_lock);
> +    svc_delete_xprt(newxpt);
> +    goto out_closed;
> +   }
>     set_bit(XPT_TEMP, &newxpt->xpt_flags);
>     list_add(&newxpt->xpt_list, &serv->sv_tempsocks);
>     serv->sv_tmpcnt++;
> @@ -739,6 +760,7 @@ int svc_recv(struct svc_rqst *rqstp, long timeout)
>     len = xprt->xpt_ops->xpo_recvfrom(rqstp);
>    dprintk("svc: got len=%d\n", len);
>   }
> +out_closed:
>   svc_xprt_received(xprt);
>
>   /* No data, incomplete (TCP) read, or accept() */
> @@ -936,6 +958,7 @@ static void svc_clear_pools(struct svc_serv *serv, struct net *net)
>   struct svc_pool *pool;
>   struct svc_xprt *xprt;
>   struct svc_xprt *tmp;
> + struct svc_rqst *rqstp;
>   int i;
>
>   for (i = 0; i < serv->sv_nrpools; i++) {
> @@ -947,11 +970,16 @@ static void svc_clear_pools(struct svc_serv *serv, struct net *net)
>      continue;
>     list_del_init(&xprt->xpt_ready);
>    }
> +  list_for_each_entry(rqstp, &pool->sp_all_threads, rq_all) {
> +   if (rqstp->rq_xprt && rqstp->rq_xprt->xpt_net == net)
> +    set_bit(XPT_CLOSE, &rqstp->rq_xprt->xpt_flags);
```

```
> +  }
>    spin_unlock_bh(&pool->sp_lock);
>   }
>  }
>
> -static void svc_clear_list(struct list_head *xprt_list, struct net *net)
> +static void svc_clear_list(struct list_head *xprt_list, struct net *net,
> +       struct list_head *kill_list)
>  {
>   struct svc_xprt *xprt;
>   struct svc_xprt *tmp;
> @@ -959,7 +987,8 @@ static void svc_clear_list(struct list_head *xprt_list, struct net *net)
>   list_for_each_entry_safe(xprt, tmp, xprt_list, xpt_list) {
>    if (xprt->xpt_net != net)
>     continue;
> -  svc_delete_xprt(xprt);
> +  list_move(&xprt->xpt_list, kill_list);
> +  set_bit(XPT_DETACHED, &xprt->xpt_flags);
>   }
>   list_for_each_entry(xprt, xprt_list, xpt_list)
>    BUG_ON(xprt->xpt_net == net);
> @@ -967,6 +996,15 @@ static void svc_clear_list(struct list_head *xprt_list, struct net *net)
>
>  void svc_close_net(struct svc_serv *serv, struct net *net)
>  {
> + struct svc_xprt *xprt, *tmp;
> + LIST_HEAD(kill_list);
> +
> + /*
> +  * Protect the lists, since they can be by tasks with different network
> +  * namespace contexts.
> +  */
> + spin_lock_bh(&serv->sv_lock);
> +
>   svc_close_list(&serv->sv_tempsocks, net);
>   svc_close_list(&serv->sv_permsocks, net);
>
> @@ -976,8 +1014,18 @@ void svc_close_net(struct svc_serv *serv, struct net *net)
>    * svc_xprt_enqueue will not add new entries without taking the
>    * sp_lock and checking XPT_BUSY.
>    */
> - svc_clear_list(&serv->sv_tempsocks, net);
> - svc_clear_list(&serv->sv_permsocks, net);
> + svc_clear_list(&serv->sv_tempsocks, net, &kill_list);
> + svc_clear_list(&serv->sv_permsocks, net, &kill_list);
> +
> + spin_unlock_bh(&serv->sv_lock);
> +
```

> + /*
> + * Destroy collected transports.
> + * Note: tranports has been marked as XPT_DETACHED on svc_clear_list(),
> + * so no need to protect againt list_del() in svc_delete_xprt().
> + */
> + list_for_each_entry_safe(xprt, tmp, &kill_list, xpt_list)
> + svc_delete_xprt(xprt);
> }
>
> /*
>

Subject: Re: [PATCH v3] SUNRPC: protect service sockets lists during per-net shutdown
Posted by Neil Brown on Tue, 31 Jul 2012 05:28:48 GMT
View Forum Message <> Reply to Message

On Tue, 24 Jul 2012 15:40:37 -0400 "J. Bruce Fields" <bfields@fieldses.org>
wrote:

> On Tue, Jul 03, 2012 at 04:58:57PM +0400, Stanislav Kinsbursky wrote:
> > > v3:
> > > 1) rebased on 3.5-rc3 kernel.
> > >
> > > v2: destruction of currently processing transport added:
> > > 1) Added marking of currently processing transports with XPT_CLOSE on per-net
> > > shutdown. These transports will be destroyed in svc_xprt_enqueue() (instead of
> > > enqueueing).
>
> That worries me:
>
> - Why did we originally defer close until svc_recv?

I don't think there was any obscure reason - it was just the natural place do
to it. In svc_recv we are absolutely sure that the socket is idle. There
are a number of things we might want to do, so we find the highest-priority
one and do it. "state machine" pattern?

> - Are we sure there's no risk to performing it immediately in
>   svc_enqueue? Is it safe to call from the socket callbacks and
>   wherever else we call svc_enqueue?

The latter point is the one I'd want to see verified. If svc_xprt_enqueue
gets called in 'bh' content, and calls svc_delete_xprt which then calls
svc_deferred_dequeue and that takes ->xpt_lock - does that mean that all
lock/unlock of ->xpt_lock needs to be changed to use the _bh variants?

NeilBrown

>
> And in the past I haven't been good at testing for problems
> here--instead they tend to show up when a use somewhere tries shutting
> down a server that's under load.
>
> I'll look more closely. Meanwhile you could split out that change as a
> separate patch and convince me why it's right....
>
> --b.
>
> > 2) newly created temporary transport in svc_recv() will be destroyed, if it's
> > "parent" was marked with XPT_CLOSE.
> > 3) spin_lock(&serv->sv_lock) was replaced by spin_lock_bh() in
> > svc_close_net(&serv->sv_lock).
> >
> > Service sv_tempsocks and sv_permsocks lists are accessible by tasks with
> > different network namespaces, and thus per-net service destruction must be
> > protected.
> > These lists are protected by service sv_lock. So lets wrap list munipulations
> > with this lock and move tranports destruction outside wrapped area to prevent
> > deadlocks.
> >
> > Signed-off-by: Stanislav Kinsbursky <skinsbursky@parallels.com>
> > ---
> > net/sunrpc/svc_xprt.c | 56 ++---
> > 1 files changed, 52 insertions(+), 4 deletions(-)
> >
> > diff --git a/net/sunrpc/svc_xprt.c b/net/sunrpc/svc_xprt.c
> > index 88f2bf6..4af2114 100644
> > --- a/net/sunrpc/svc_xprt.c
> > +++ b/net/sunrpc/svc_xprt.c
> > @@ -320,6 +320,7 @@ void svc_xprt_enqueue(struct svc_xprt *xprt)
> >   struct svc_pool *pool;
> >   struct svc_rqst *rqstp;
> >   int cpu;
> > + int destroy = 0;
> >
> >   if (!svc_xprt_has_something_to_do(xprt))
> >   return;
> > @@ -338,6 +339,17 @@ void svc_xprt_enqueue(struct svc_xprt *xprt)
> >
> >   pool->sp_stats.packets++;
> >
> > + /*

```
> > +  * Check transport close flag. It could be marked as closed on per-net
> > +  * service shutdown.
> > +  */
> > + if (test_bit(XPT_CLOSE, &xprt->xpt_flags)) {
> > +  /* Don't enqueue transport if it has to be destroyed. */
> > +  dprintk("svc: transport %p have to be closed\n", xprt);
> > +  destroy++;
> > +  goto out_unlock;
> > + }
> > +
> >    /* Mark transport as busy. It will remain in this state until
> >     * the provider calls svc_xprt_received. We update XPT_BUSY
> >     * atomically because it also guards against trying to enqueue
> > @@ -374,6 +386,8 @@ void svc_xprt_enqueue(struct svc_xprt *xprt)
> >
> >  out_unlock:
> >   spin_unlock_bh(&pool->sp_lock);
> > + if (destroy)
> > +  svc_delete_xprt(xprt);
> >  }
> >  EXPORT_SYMBOL_GPL(svc_xprt_enqueue);
> >
> > @@ -714,6 +728,13 @@ int svc_recv(struct svc_rqst *rqstp, long timeout)
> >     __module_get(newxpt->xpt_class->xcl_owner);
> >     svc_check_conn_limits(xprt->xpt_server);
> >     spin_lock_bh(&serv->sv_lock);
> > +   if (test_bit(XPT_CLOSE, &xprt->xpt_flags)) {
> > +    dprintk("svc_recv: found XPT_CLOSE on listener\n");
> > +    set_bit(XPT_DETACHED, &newxpt->xpt_flags);
> > +    spin_unlock_bh(&pool->sp_lock);
> > +    svc_delete_xprt(newxpt);
> > +    goto out_closed;
> > +   }
> >     set_bit(XPT_TEMP, &newxpt->xpt_flags);
> >     list_add(&newxpt->xpt_list, &serv->sv_tempsocks);
> >     serv->sv_tmpcnt++;
> > @@ -739,6 +760,7 @@ int svc_recv(struct svc_rqst *rqstp, long timeout)
> >     len = xprt->xpt_ops->xpo_recvfrom(rqstp);
> >    dprintk("svc: got len=%d\n", len);
> >   }
> > +out_closed:
> >   svc_xprt_received(xprt);
> >
> >   /* No data, incomplete (TCP) read, or accept() */
> > @@ -936,6 +958,7 @@ static void svc_clear_pools(struct svc_serv *serv, struct net *net)
> >   struct svc_pool *pool;
> >   struct svc_xprt *xprt;
> >   struct svc_xprt *tmp;
```

```
> > + struct svc_rqst *rqstp;
> >   int i;
> >
> >   for (i = 0; i < serv->sv_nrpools; i++) {
> > @@ -947,11 +970,16 @@ static void svc_clear_pools(struct svc_serv *serv, struct net *net)
> >       continue;
> >     list_del_init(&xprt->xpt_ready);
> >    }
> > +  list_for_each_entry(rqstp, &pool->sp_all_threads, rq_all) {
> > +   if (rqstp->rq_xprt && rqstp->rq_xprt->xpt_net == net)
> > +    set_bit(XPT_CLOSE, &rqstp->rq_xprt->xpt_flags);
> > +  }
> >    spin_unlock_bh(&pool->sp_lock);
> >   }
> >  }
> >
> > -static void svc_clear_list(struct list_head *xprt_list, struct net *net)
> > +static void svc_clear_list(struct list_head *xprt_list, struct net *net,
> > +       struct list_head *kill_list)
> >  {
> >   struct svc_xprt *xprt;
> >   struct svc_xprt *tmp;
> > @@ -959,7 +987,8 @@ static void svc_clear_list(struct list_head *xprt_list, struct net *net)
> >   list_for_each_entry_safe(xprt, tmp, xprt_list, xpt_list) {
> >    if (xprt->xpt_net != net)
> >     continue;
> > -  svc_delete_xprt(xprt);
> > +  list_move(&xprt->xpt_list, kill_list);
> > +  set_bit(XPT_DETACHED, &xprt->xpt_flags);
> >   }
> >   list_for_each_entry(xprt, xprt_list, xpt_list)
> >    BUG_ON(xprt->xpt_net == net);
> > @@ -967,6 +996,15 @@ static void svc_clear_list(struct list_head *xprt_list, struct net *net)
> >
> >  void svc_close_net(struct svc_serv *serv, struct net *net)
> >  {
> > + struct svc_xprt *xprt, *tmp;
> > + LIST_HEAD(kill_list);
> > +
> > + /*
> > +  * Protect the lists, since they can be by tasks with different network
> > +  * namespace contexts.
> > +  */
> > + spin_lock_bh(&serv->sv_lock);
> > +
> >   svc_close_list(&serv->sv_tempsocks, net);
> >   svc_close_list(&serv->sv_permsocks, net);
> >
```

> > @@ -976,8 +1014,18 @@ void svc_close_net(struct svc_serv *serv, struct net *net)
> > * svc_xprt_enqueue will not add new entries without taking the
> > * sp_lock and checking XPT_BUSY.
> > */
> > - svc_clear_list(&serv->sv_tempsocks, net);
> > - svc_clear_list(&serv->sv_permsocks, net);
> > + svc_clear_list(&serv->sv_tempsocks, net, &kill_list);
> > + svc_clear_list(&serv->sv_permsocks, net, &kill_list);
> > +
> > + spin_unlock_bh(&serv->sv_lock);
> > +
> > + /*
> > + * Destroy collected transports.
> > + * Note: tranports has been marked as XPT_DETACHED on svc_clear_list(),
> > + * so no need to protect againt list_del() in svc_delete_xprt().
> > + */
> > + list_for_each_entry_safe(xprt, tmp, &kill_list, xpt_list)
> > + svc_delete_xprt(xprt);
> > }
> >
> > /*
> >

Subject: Re: [PATCH v3] SUNRPC: protect service sockets lists during per-net shutdown
Posted by bfields on Thu, 16 Aug 2012 19:29:03 GMT
View Forum Message <> Reply to Message

On Tue, Jul 24, 2012 at 03:40:37PM -0400, J. Bruce Fields wrote:
> On Tue, Jul 03, 2012 at 04:58:57PM +0400, Stanislav Kinsbursky wrote:
> > v3:
> > 1) rebased on 3.5-rc3 kernel.
> >
> > v2: destruction of currently processing transport added:
> > 1) Added marking of currently processing transports with XPT_CLOSE on per-net
> > shutdown. These transports will be destroyed in svc_xprt_enqueue() (instead of
> > enqueueing).
>
> That worries me:
>
> - Why did we originally defer close until svc_recv?
> - Are we sure there's no risk to performing it immediately in
>   svc_enqueue? Is it safe to call from the socket callbacks and
>   wherever else we call svc_enqueue?
>
> And in the past I haven't been good at testing for problems
> here--instead they tend to show up when a use somewhere tries shutting

> down a server that's under load.
>
> I'll look more closely. Meanwhile you could split out that change as a
> separate patch and convince me why it's right....

Looking back at this:

 - adding the sv_lock looks like the right thing to do anyway
   independent of containers, because svc_age_temp_xprts may
   still be running.

 - I'm increasingly unhappy about sharing rpc servers between
   network namespaces. Everything would be easier to understand
   if they were independent. Can we figure out how to do that?

>
> --b.
>
> > 2) newly created temporary transport in svc_recv() will be destroyed, if it's
> > "parent" was marked with XPT_CLOSE.
> > 3) spin_lock(&serv->sv_lock) was replaced by spin_lock_bh() in
> > svc_close_net(&serv->sv_lock).
> >
> > Service sv_tempsocks and sv_permsocks lists are accessible by tasks with
> > different network namespaces, and thus per-net service destruction must be
> > protected.
> > These lists are protected by service sv_lock. So lets wrap list munipulations
> > with this lock and move tranports destruction outside wrapped area to prevent
> > deadlocks.
> >
> > Signed-off-by: Stanislav Kinsbursky <skinsbursky@parallels.com>
> > ---
> > net/sunrpc/svc_xprt.c | 56 ++---
> > 1 files changed, 52 insertions(+), 4 deletions(-)
> >
> > diff --git a/net/sunrpc/svc_xprt.c b/net/sunrpc/svc_xprt.c
> > index 88f2bf6..4af2114 100644
> > --- a/net/sunrpc/svc_xprt.c
> > +++ b/net/sunrpc/svc_xprt.c
> > @@ -320,6 +320,7 @@ void svc_xprt_enqueue(struct svc_xprt *xprt)
> >   struct svc_pool *pool;
> >   struct svc_rqst *rqstp;
> >   int cpu;
> > + int destroy = 0;
> >
> >   if (!svc_xprt_has_something_to_do(xprt))
> >   return;
> > @@ -338,6 +339,17 @@ void svc_xprt_enqueue(struct svc_xprt *xprt)

```
> >
> >   pool->sp_stats.packets++;
> >
> > + /*
> > +  * Check transport close flag. It could be marked as closed on per-net
> > +  * service shutdown.
> > +  */
> > + if (test_bit(XPT_CLOSE, &xprt->xpt_flags)) {
> > +  /* Don't enqueue transport if it has to be destroyed. */
> > +  dprintk("svc: transport %p have to be closed\n", xprt);
> > +  destroy++;
> > +  goto out_unlock;
> > + }
> > +
> >   /* Mark transport as busy. It will remain in this state until
> >    * the provider calls svc_xprt_received. We update XPT_BUSY
> >    * atomically because it also guards against trying to enqueue
> > @@ -374,6 +386,8 @@ void svc_xprt_enqueue(struct svc_xprt *xprt)
> >
> >  out_unlock:
> >   spin_unlock_bh(&pool->sp_lock);
> > + if (destroy)
> > +  svc_delete_xprt(xprt);
> >  }
> >  EXPORT_SYMBOL_GPL(svc_xprt_enqueue);
> >
> > @@ -714,6 +728,13 @@ int svc_recv(struct svc_rqst *rqstp, long timeout)
> >     __module_get(newxpt->xpt_class->xcl_owner);
> >     svc_check_conn_limits(xprt->xpt_server);
> >     spin_lock_bh(&serv->sv_lock);
> > +   if (test_bit(XPT_CLOSE, &xprt->xpt_flags)) {
> > +    dprintk("svc_recv: found XPT_CLOSE on listener\n");
> > +    set_bit(XPT_DETACHED, &newxpt->xpt_flags);
> > +    spin_unlock_bh(&pool->sp_lock);
> > +    svc_delete_xprt(newxpt);
> > +    goto out_closed;
> > +   }
> >     set_bit(XPT_TEMP, &newxpt->xpt_flags);
> >     list_add(&newxpt->xpt_list, &serv->sv_tempsocks);
> >     serv->sv_tmpcnt++;
> > @@ -739,6 +760,7 @@ int svc_recv(struct svc_rqst *rqstp, long timeout)
> >     len = xprt->xpt_ops->xpo_recvfrom(rqstp);
> >    dprintk("svc: got len=%d\n", len);
> >   }
> > +out_closed:
> >   svc_xprt_received(xprt);
> >
> >   /* No data, incomplete (TCP) read, or accept() */
```

```
> > @@ -936,6 +958,7 @@ static void svc_clear_pools(struct svc_serv *serv, struct net *net)
> >   struct svc_pool *pool;
> >   struct svc_xprt *xprt;
> >   struct svc_xprt *tmp;
> > + struct svc_rqst *rqstp;
> >   int i;
> >
> >   for (i = 0; i < serv->sv_nrpools; i++) {
> > @@ -947,11 +970,16 @@ static void svc_clear_pools(struct svc_serv *serv, struct net *net)
> >      continue;
> >     list_del_init(&xprt->xpt_ready);
> >    }
> > +  list_for_each_entry(rqstp, &pool->sp_all_threads, rq_all) {
> > +   if (rqstp->rq_xprt && rqstp->rq_xprt->xpt_net == net)
> > +    set_bit(XPT_CLOSE, &rqstp->rq_xprt->xpt_flags);
> > +  }
> >    spin_unlock_bh(&pool->sp_lock);
> >   }
> >  }
> >
> > -static void svc_clear_list(struct list_head *xprt_list, struct net *net)
> > +static void svc_clear_list(struct list_head *xprt_list, struct net *net,
> > +      struct list_head *kill_list)
> >  {
> >   struct svc_xprt *xprt;
> >   struct svc_xprt *tmp;
> > @@ -959,7 +987,8 @@ static void svc_clear_list(struct list_head *xprt_list, struct net *net)
> >   list_for_each_entry_safe(xprt, tmp, xprt_list, xpt_list) {
> >    if (xprt->xpt_net != net)
> >     continue;
> > -  svc_delete_xprt(xprt);
> > +  list_move(&xprt->xpt_list, kill_list);
> > +  set_bit(XPT_DETACHED, &xprt->xpt_flags);
> >   }
> >   list_for_each_entry(xprt, xprt_list, xpt_list)
> >    BUG_ON(xprt->xpt_net == net);
> > @@ -967,6 +996,15 @@ static void svc_clear_list(struct list_head *xprt_list, struct net *net)
> >
> >  void svc_close_net(struct svc_serv *serv, struct net *net)
> >  {
> > + struct svc_xprt *xprt, *tmp;
> > + LIST_HEAD(kill_list);
> > +
> > + /*
> > +  * Protect the lists, since they can be by tasks with different network
> > +  * namespace contexts.
> > +  */
> > + spin_lock_bh(&serv->sv_lock);
```

> > +
> > svc_close_list(&serv->sv_tempsocks, net);
> > svc_close_list(&serv->sv_permsocks, net);
> >
> > @@ -976,8 +1014,18 @@ void svc_close_net(struct svc_serv *serv, struct net *net)
> > * svc_xprt_enqueue will not add new entries without taking the
> > * sp_lock and checking XPT_BUSY.
> > */
> > - svc_clear_list(&serv->sv_tempsocks, net);
> > - svc_clear_list(&serv->sv_permsocks, net);
> > + svc_clear_list(&serv->sv_tempsocks, net, &kill_list);
> > + svc_clear_list(&serv->sv_permsocks, net, &kill_list);
> > +
> > + spin_unlock_bh(&serv->sv_lock);
> > +
> > + /*
> > + * Destroy collected transports.
> > + * Note: tranports has been marked as XPT_DETACHED on svc_clear_list(),
> > + * so no need to protect againt list_del() in svc_delete_xprt().
> > + */
> > + list_for_each_entry_safe(xprt, tmp, &kill_list, xpt_list)
> > + svc_delete_xprt(xprt);
> > }
> >
> > /*
> >

Subject: Re: [PATCH v3] SUNRPC: protect service sockets lists during per-net shutdown
Posted by Stanislav Kinsbursky on Mon, 20 Aug 2012 11:05:49 GMT
View Forum Message <> Reply to Message

> On Tue, Jul 24, 2012 at 03:40:37PM -0400, J. Bruce Fields wrote:
>> On Tue, Jul 03, 2012 at 04:58:57PM +0400, Stanislav Kinsbursky wrote:
>>> v3:
>>> 1) rebased on 3.5-rc3 kernel.
>>>
>>> v2: destruction of currently processing transport added:
>>> 1) Added marking of currently processing transports with XPT_CLOSE on per-net
>>> shutdown. These transports will be destroyed in svc_xprt_enqueue() (instead of
>>> enqueueing).
>>
>> That worries me:
>>
>> - Why did we originally defer close until svc_recv?

The problem I was trying to solve is shutting down of transports in use.
I.e. some transport was dequeued from pool in svc_recv() and some process called
xpo_accept(), trying to create new socket, new transport and so on.
How to shutdown such transports properly?
The best idea I had was to check all such active transports (rqstp->rq_xprt) and
mark the with XPT_CLOSE. So then new transport will be destroyed without adding
to service lists.
Probably, I've missed some points and would be glad to hear your opinion on this.

>> - Are we sure there's no risk to performing it immediately in
>>   svc_enqueue? Is it safe to call from the socket callbacks and
>>   wherever else we call svc_enqueue?
>>
>> And in the past I haven't been good at testing for problems
>> here--instead they tend to show up when a use somewhere tries shutting
>> down a server that's under load.
>>
>> I'll look more closely. Meanwhile you could split out that change as a
>> separate patch and convince me why it's right....
>
> Looking back at this:
>
> - adding the sv_lock looks like the right thing to do anyway
>   independent of containers, because svc_age_temp_xprts may
>   still be running.
>
> - I'm increasingly unhappy about sharing rpc servers between
>   network namespaces. Everything would be easier to understand
>   if they were independent. Can we figure out how to do that?
>

Could you, please, elaborate on your your unhappiness?
I.e. I don't like it too. But the problem here, is that rpc server is tied with
kernel threads creation and destruction. And these threads can be only a part of
initial pid namespace (because we have only one kthreadd). And we decided do not
create new kernel thread per container when were discussing the problem last time.

>>
>> --b.
>>
>>> 2) newly created temporary transport in svc_recv() will be destroyed, if it's
>>> "parent" was marked with XPT_CLOSE.
>>> 3) spin_lock(&serv->sv_lock) was replaced by spin_lock_bh() in
>>> svc_close_net(&serv->sv_lock).
>>>
>>> Service sv_tempsocks and sv_permsocks lists are accessible by tasks with
>>> different network namespaces, and thus per-net service destruction must be

```
>>> protected.
>>> These lists are protected by service sv_lock. So lets wrap list munipulations
>>> with this lock and move tranports destruction outside wrapped area to prevent
>>> deadlocks.
>>>
>>> Signed-off-by: Stanislav Kinsbursky <skinsbursky@parallels.com>
>>> ---
>>>   net/sunrpc/svc_xprt.c |   56 ++++++++++++++++++++++++++++++++++++++++++++---
>>>   1 files changed, 52 insertions(+), 4 deletions(-)
>>>
>>> diff --git a/net/sunrpc/svc_xprt.c b/net/sunrpc/svc_xprt.c
>>> index 88f2bf6..4af2114 100644
>>> --- a/net/sunrpc/svc_xprt.c
>>> +++ b/net/sunrpc/svc_xprt.c
>>> @@ -320,6 +320,7 @@ void svc_xprt_enqueue(struct svc_xprt *xprt)
>>>    struct svc_pool *pool;
>>>    struct svc_rqst *rqstp;
>>>    int cpu;
>>> + int destroy = 0;
>>>
>>>    if (!svc_xprt_has_something_to_do(xprt))
>>>     return;
>>> @@ -338,6 +339,17 @@ void svc_xprt_enqueue(struct svc_xprt *xprt)
>>>
>>>    pool->sp_stats.packets++;
>>>
>>> + /*
>>> +  * Check transport close flag. It could be marked as closed on per-net
>>> +  * service shutdown.
>>> +  */
>>> + if (test_bit(XPT_CLOSE, &xprt->xpt_flags)) {
>>> +  /* Don't enqueue transport if it has to be destroyed. */
>>> +  dprintk("svc: transport %p have to be closed\n", xprt);
>>> +  destroy++;
>>> +  goto out_unlock;
>>> + }
>>> +
>>>    /* Mark transport as busy. It will remain in this state until
>>>     * the provider calls svc_xprt_received. We update XPT_BUSY
>>>     * atomically because it also guards against trying to enqueue
>>> @@ -374,6 +386,8 @@ void svc_xprt_enqueue(struct svc_xprt *xprt)
>>>
>>>   out_unlock:
>>>    spin_unlock_bh(&pool->sp_lock);
>>> + if (destroy)
>>> +  svc_delete_xprt(xprt);
>>>   }
>>>   EXPORT_SYMBOL_GPL(svc_xprt_enqueue);
```

```
>>>
>>> @@ -714,6 +728,13 @@ int svc_recv(struct svc_rqst *rqstp, long timeout)
>>>       __module_get(newxpt->xpt_class->xcl_owner);
>>>       svc_check_conn_limits(xprt->xpt_server);
>>>       spin_lock_bh(&serv->sv_lock);
>>> +     if (test_bit(XPT_CLOSE, &xprt->xpt_flags)) {
>>> +      dprintk("svc_recv: found XPT_CLOSE on listener\n");
>>> +      set_bit(XPT_DETACHED, &newxpt->xpt_flags);
>>> +      spin_unlock_bh(&pool->sp_lock);
>>> +      svc_delete_xprt(newxpt);
>>> +      goto out_closed;
>>> +     }
>>>       set_bit(XPT_TEMP, &newxpt->xpt_flags);
>>>       list_add(&newxpt->xpt_list, &serv->sv_tempsocks);
>>>       serv->sv_tmpcnt++;
>>> @@ -739,6 +760,7 @@ int svc_recv(struct svc_rqst *rqstp, long timeout)
>>>       len = xprt->xpt_ops->xpo_recvfrom(rqstp);
>>>      dprintk("svc: got len=%d\n", len);
>>>     }
>>> +out_closed:
>>>     svc_xprt_received(xprt);
>>>
>>>     /* No data, incomplete (TCP) read, or accept() */
>>> @@ -936,6 +958,7 @@ static void svc_clear_pools(struct svc_serv *serv, struct net *net)
>>>     struct svc_pool *pool;
>>>     struct svc_xprt *xprt;
>>>     struct svc_xprt *tmp;
>>> + struct svc_rqst *rqstp;
>>>     int i;
>>>
>>>     for (i = 0; i < serv->sv_nrpools; i++) {
>>> @@ -947,11 +970,16 @@ static void svc_clear_pools(struct svc_serv *serv, struct net *net)
>>>       continue;
>>>       list_del_init(&xprt->xpt_ready);
>>>      }
>>> +   list_for_each_entry(rqstp, &pool->sp_all_threads, rq_all) {
>>> +    if (rqstp->rq_xprt && rqstp->rq_xprt->xpt_net == net)
>>> +     set_bit(XPT_CLOSE, &rqstp->rq_xprt->xpt_flags);
>>> +   }
>>>      spin_unlock_bh(&pool->sp_lock);
>>>     }
>>>    }
>>>
>>> -static void svc_clear_list(struct list_head *xprt_list, struct net *net)
>>> +static void svc_clear_list(struct list_head *xprt_list, struct net *net,
>>> +       struct list_head *kill_list)
>>>    {
>>>     struct svc_xprt *xprt;
```

```
>>>    struct svc_xprt *tmp;
>>> @@ -959,7 +987,8 @@ static void svc_clear_list(struct list_head *xprt_list, struct net *net)
>>>    list_for_each_entry_safe(xprt, tmp, xprt_list, xpt_list) {
>>>     if (xprt->xpt_net != net)
>>>      continue;
>>> -  svc_delete_xprt(xprt);
>>> +  list_move(&xprt->xpt_list, kill_list);
>>> +  set_bit(XPT_DETACHED, &xprt->xpt_flags);
>>>    }
>>>    list_for_each_entry(xprt, xprt_list, xpt_list)
>>>     BUG_ON(xprt->xpt_net == net);
>>> @@ -967,6 +996,15 @@ static void svc_clear_list(struct list_head *xprt_list, struct net *net)
>>>
>>>   void svc_close_net(struct svc_serv *serv, struct net *net)
>>>   {
>>> + struct svc_xprt *xprt, *tmp;
>>> + LIST_HEAD(kill_list);
>>> +
>>> + /*
>>> +  * Protect the lists, since they can be by tasks with different network
>>> +  * namespace contexts.
>>> +  */
>>> + spin_lock_bh(&serv->sv_lock);
>>> +
>>>    svc_close_list(&serv->sv_tempsocks, net);
>>>    svc_close_list(&serv->sv_permsocks, net);
>>>
>>> @@ -976,8 +1014,18 @@ void svc_close_net(struct svc_serv *serv, struct net *net)
>>>     * svc_xprt_enqueue will not add new entries without taking the
>>>     * sp_lock and checking XPT_BUSY.
>>>     */
>>> - svc_clear_list(&serv->sv_tempsocks, net);
>>> - svc_clear_list(&serv->sv_permsocks, net);
>>> + svc_clear_list(&serv->sv_tempsocks, net, &kill_list);
>>> + svc_clear_list(&serv->sv_permsocks, net, &kill_list);
>>> +
>>> + spin_unlock_bh(&serv->sv_lock);
>>> +
>>> + /*
>>> +  * Destroy collected transports.
>>> +  * Note: tranports has been marked as XPT_DETACHED on svc_clear_list(),
>>> +  * so no need to protect againt list_del() in svc_delete_xprt().
>>> +  */
>>> + list_for_each_entry_safe(xprt, tmp, &kill_list, xpt_list)
>>> +  svc_delete_xprt(xprt);
>>>   }
>>>
>>>   /*
```

>>>

--
Best regards,
Stanislav Kinsbursky

Subject: Re: [PATCH v3] SUNRPC: protect service sockets lists during per-net shutdown
Posted by bfields on Mon, 20 Aug 2012 14:56:56 GMT
View Forum Message <> Reply to Message

On Mon, Aug 20, 2012 at 03:05:49PM +0400, Stanislav Kinsbursky wrote:

> >Looking back at this:
> >
> > - adding the sv_lock looks like the right thing to do anyway
> >   independent of containers, because svc_age_temp_xprts may
> >   still be running.
> >
> > - I'm increasingly unhappy about sharing rpc servers between
> >   network namespaces. Everything would be easier to understand
> >   if they were independent. Can we figure out how to do that?
> >
>
> Could you, please, elaborate on your your unhappiness?

It seems like you're having to do a lot of work on each individual rpc server (callback server, lockd, etc.) to make per-net startup/shutdown work. And then we still don't have it quite right (see the shutdown races).)

In general whenever we have the opportunity to have entirely separate data structures, I'd expect that to simplify things: it should eliminate some locking and reference-counting issues.

> I.e. I don't like it too. But the problem here, is that rpc server
> is tied with kernel threads creation and destruction. And these
> threads can be only a part of initial pid namespace (because we have
> only one kthreadd). And we decided do not create new kernel thread
> per container when were discussing the problem last time.

There really should be some way to create a kernel thread in a specific namespace, shouldn't there?

Until we have that, could the threads be taught to fix their namespace on startup?

--b.

Subject: Re: [PATCH v3] SUNRPC: protect service sockets lists during per-net shutdown
Posted by Stanislav Kinsbursky on Mon, 20 Aug 2012 15:11:00 GMT
View Forum Message <> Reply to Message

> On Mon, Aug 20, 2012 at 03:05:49PM +0400, Stanislav Kinsbursky wrote:

>>> Looking back at this:
>>>
>>> - adding the sv_lock looks like the right thing to do anyway
>>>   independent of containers, because svc_age_temp_xprts may
>>>   still be running.
>>>
>>> - I'm increasingly unhappy about sharing rpc servers between
>>>   network namespaces. Everything would be easier to understand
>>>   if they were independent. Can we figure out how to do that?
>>>
>>
>> Could you, please, elaborate on your your unhappiness?
>
> It seems like you're having to do a lot of work on each individual rpc
> server (callback server, lockd, etc.) to make per-net startup/shutdown
> work. And then we still don't have it quite right (see the shutdown
> races).)
>
> In general whenever we have the opportunity to have entirely separate
> data structures, I'd expect that to simplify things: it should eliminate
> some locking and reference-counting issues.
>

Agreed. But current solution still looks like the easies way to me to implement
desired functionality.

>> I.e. I don't like it too. But the problem here, is that rpc server
>> is tied with kernel threads creation and destruction. And these
>> threads can be only a part of initial pid namespace (because we have
>> only one kthreadd). And we decided do not create new kernel thread
>> per container when were discussing the problem last time.
>
> There really should be some way to create a kernel thread in a specific
> namespace, shouldn't there?
>

Kthreads support in a container is rather a "political" problem, than an implementation problem.

Currently, when you call kthread_create(), you add new job to kthreadd queue. Kthreadd is unique, starts right after init and lives in global initial environment. So, any kthread inherits namespaces from it.
Of course, we can start one kthread per environment and change it's root or even network namespace in kthread function. But pid namespace of this kthread will remain global.
It looks like not a big problem, when we shutdown kthread by some variable. But what about killable nfsd kthreads?
1) We can't kill them from nested pid namespace.
2) How we will differ nfsd kthreads in initial pid namespace?

In OpenVZ we have kthreadd per pid hamespace and it allows us to create kthreads (and thus services) per pid namespace.

> Until we have that, could the threads be taught to fix their namespace
> on startup?
>

Unfortunately, changing of pid namespace for kthreads doesn't look like an easy trick.

> --b.
>

--
Best regards,
Stanislav Kinsbursky

Subject: Re: [PATCH v3] SUNRPC: protect service sockets lists during per-net shutdown
Posted by bfields on Mon, 20 Aug 2012 16:58:52 GMT
View Forum Message <> Reply to Message

On Mon, Aug 20, 2012 at 07:11:00PM +0400, Stanislav Kinsbursky wrote:

> >On Mon, Aug 20, 2012 at 03:05:49PM +0400, Stanislav Kinsbursky wrote:

> >>>Looking back at this:
> >>>
> >>> - adding the sv_lock looks like the right thing to do anyway
> >>>   independent of containers, because svc_age_temp_xprts may
> >>>   still be running.

> >>>
> >>> - I'm increasingly unhappy about sharing rpc servers between
> >>> network namespaces. Everything would be easier to understand
> >>> if they were independent. Can we figure out how to do that?
> >>>
> >>
> >>Could you, please, elaborate on your your unhappiness?
> >
> >It seems like you're having to do a lot of work on each individual rpc
> >server (callback server, lockd, etc.) to make per-net startup/shutdown
> >work. And then we still don't have it quite right (see the shutdown
> >races).)
> >
> >In general whenever we have the opportunity to have entirely separate
> >data structures, I'd expect that to simplify things: it should eliminate
> >some locking and reference-counting issues.
> >
>
> Agreed. But current solution still looks like the easies way to me
> to implement desired functionality.
>
> >>I.e. I don't like it too. But the problem here, is that rpc server
> >>is tied with kernel threads creation and destruction. And these
> >>threads can be only a part of initial pid namespace (because we have
> >>only one kthreadd). And we decided do not create new kernel thread
> >>per container when were discussing the problem last time.
> >
> >There really should be some way to create a kernel thread in a specific
> >namespace, shouldn't there?
> >
>
>
> Kthreads support in a container is rather a "political" problem,
> than an implementation problem.

Is there a mail thread somewhere with a summary of the objections?

> Currently, when you call kthread_create(), you add new job to
> kthreadd queue. Kthreadd is unique, starts right after init and
> lives in global initial environment. So, any kthread inherits
> namespaces from it.
> Of course, we can start one kthread per environment and change it's
> root or even network namespace in kthread function. But pid
> namespace of this kthread will remain global.

OK. But the current implementation will leave all the server threads in
the initial pid namespace, too.

> It looks like not a big problem, when we shutdown kthread by some
> variable. But what about killable nfsd kthreads?

And we're stuck with that problem either way too, aren't we?

> 1) We can't kill them from nested pid namespace.
> 2) How we will differ nfsd kthreads in initial pid namespace?

I have to admit for my purposes I don't care too much about pid
namespaces or about signalling server threads. It'd be nice to get
those things right but it wouldn't bother me that much not to.

Another stupid idea: can we do our own implementation of something like
kthreadd just for the purpose of starting rpc server threads? It
doesn't seem that complicated.

--b.

> In OpenVZ we have kthreadd per pid hamespace and it allows us to
> create kthreads (and thus services) per pid namespace.

Subject: Re: [PATCH v3] SUNRPC: protect service sockets lists during per-net shutdown
Posted by Stanislav Kinsbursky on Tue, 21 Aug 2012 09:28:00 GMT
View Forum Message <> Reply to Message

> On Mon, Aug 20, 2012 at 07:11:00PM +0400, Stanislav Kinsbursky wrote:

>>> On Mon, Aug 20, 2012 at 03:05:49PM +0400, Stanislav Kinsbursky wrote:

>>>>> Looking back at this:
>>>>>
>>>>> - adding the sv_lock looks like the right thing to do anyway
>>>>>   independent of containers, because svc_age_temp_xprts may
>>>>>   still be running.
>>>>>
>>>>> - I'm increasingly unhappy about sharing rpc servers between
>>>>>   network namespaces. Everything would be easier to understand
>>>>>   if they were independent. Can we figure out how to do that?
>>>>>
>>>>
>>>> Could you, please, elaborate on your your unhappiness?
>>>
>>> It seems like you're having to do a lot of work on each individual rpc
>>> server (callback server, lockd, etc.) to make per-net startup/shutdown
>>> work. And then we still don't have it quite right (see the shutdown

>>> races).)
>>>
>>> In general whenever we have the opportunity to have entirely separate
>>> data structures, I'd expect that to simplify things: it should eliminate
>>> some locking and reference-counting issues.
>>>
>>
>> Agreed. But current solution still looks like the easies way to me
>> to implement desired functionality.
>>
>>>> I.e. I don't like it too. But the problem here, is that rpc server
>>>> is tied with kernel threads creation and destruction. And these
>>>> threads can be only a part of initial pid namespace (because we have
>>>> only one kthreadd). And we decided do not create new kernel thread
>>>> per container when were discussing the problem last time.
>>>
>>> There really should be some way to create a kernel thread in a specific
>>> namespace, shouldn't there?
>>>
>>
>>
>> Kthreads support in a container is rather a "political" problem,
>> than an implementation problem.
>
> Is there a mail thread somewhere with a summary of the objections?
>

I can't specify right now. Need to search over lkml history.
That's all what I've found for now:
 http://us.generation-nt.com/patch-cgroups-disallow-attaching -kthreadd-help-207003852.html

>> Currently, when you call kthread_create(), you add new job to
>> kthreadd queue. Kthreadd is unique, starts right after init and
>> lives in global initial environment. So, any kthread inherits
>> namespaces from it.
>> Of course, we can start one kthread per environment and change it's
>> root or even network namespace in kthread function. But pid
>> namespace of this kthread will remain global.
>
> OK. But the current implementation will leave all the server threads in
> the initial pid namespace, too.
>
>> It looks like not a big problem, when we shutdown kthread by some
>> variable. But what about killable nfsd kthreads?
>
> And we're stuck with that problem either way too, aren't we?
>

Yes, we are. But at least we are avoiding patching of task subsystem.

>> 1) We can't kill them from nested pid namespace.
>> 2) How we will differ nfsd kthreads in initial pid namespace?
>
> I have to admit for my purposes I don't care too much about pid
> namespaces or about signalling server threads. It'd be nice to get
> those things right but it wouldn't bother me that much not to.
>
> Another stupid idea: can we do our own implementation of something like
> kthreadd just for the purpose of starting rpc server threads? It
> doesn't seem that complicated.
>

Gm...
This idea is not stupid. If I understand you right, you suggest to implement a
service per network namespace (i.e. not only data, but also threads)?

> --b.
>
>> In OpenVZ we have kthreadd per pid hamespace and it allows us to
>> create kthreads (and thus services) per pid namespace.

--
Best regards,
Stanislav Kinsbursky

Subject: Re: [PATCH v3] SUNRPC: protect service sockets lists during per-net shutdown
Posted by bfields on Tue, 21 Aug 2012 12:25:44 GMT
View Forum Message <> Reply to Message

On Tue, Aug 21, 2012 at 01:28:00PM +0400, Stanislav Kinsbursky wrote:

> >On Mon, Aug 20, 2012 at 07:11:00PM +0400, Stanislav Kinsbursky wrote:
> >>Currently, when you call kthread_create(), you add new job to
> >>kthreadd queue. Kthreadd is unique, starts right after init and
> >>lives in global initial environment. So, any kthread inherits
> >>namespaces from it.
> >>Of course, we can start one kthread per environment and change it's
> >>root or even network namespace in kthread function. But pid
> >>namespace of this kthread will remain global.
> >
> >OK. But the current implementation will leave all the server threads in
> >the initial pid namespace, too.
> >

> >>It looks like not a big problem, when we shutdown kthread by some
> >>variable. But what about killable nfsd kthreads?
> >
> >And we're stuck with that problem either way too, aren't we?
> >
>
> Yes, we are. But at least we are avoiding patching of task subsystem.
>
> >>1) We can't kill them from nested pid namespace.
> >>2) How we will differ nfsd kthreads in initial pid namespace?
> >
> >I have to admit for my purposes I don't care too much about pid
> >namespaces or about signalling server threads. It'd be nice to get
> >those things right but it wouldn't bother me that much not to.
> >
> >Another stupid idea: can we do our own implementation of something like
> >kthreadd just for the purpose of starting rpc server threads? It
> >doesn't seem that complicated.
> >
>
> Gm...
> This idea is not stupid. If I understand you right, you suggest to
> implement a service per network namespace (i.e. not only data, but
> also threads)?

Some way or another, yes, entirely separate threads for the different
namespaces would be clearer, I think.

And if we can't get them in the right pid namespaces, I'm not sure I
care.

--b.

Subject: Re: [PATCH v3] SUNRPC: protect service sockets lists during per-net shutdown
Posted by bfields on Tue, 21 Aug 2012 19:06:46 GMT
View Forum Message <> Reply to Message

On Thu, Aug 16, 2012 at 03:29:03PM -0400, J. Bruce Fields wrote:
> Looking back at this:
>
> - adding the sv_lock looks like the right thing to do anyway
>   independent of containers, because svc_age_temp_xprts may
>   still be running.

This is what I've been testing with.

Or alternatively if you'd rather strip out the other stuff from your
patch I could take that instead.

--b.

```
commit 719f8bcc883e7992615f4d5625922e24995e2d98
Author: J. Bruce Fields <bfields@redhat.com>
Date:   Mon Aug 13 17:03:00 2012 -0400

    svcrpc: fix xpt_list traversal locking on shutdown

    Server threads are not running at this point, but svc_age_temp_xprts
    still may be, so we need this locking.

    Signed-off-by: J. Bruce Fields <bfields@redhat.com>

diff --git a/net/sunrpc/svc_xprt.c b/net/sunrpc/svc_xprt.c
index bac973a..e1810b9 100644
--- a/net/sunrpc/svc_xprt.c
+++ b/net/sunrpc/svc_xprt.c
@@ -917,16 +917,18 @@ void svc_close_xprt(struct svc_xprt *xprt)
 }
 EXPORT_SYMBOL_GPL(svc_close_xprt);

-static void svc_close_list(struct list_head *xprt_list, struct net *net)
+static void svc_close_list(struct svc_serv *serv, struct list_head *xprt_list, struct net *net)
 {
  struct svc_xprt *xprt;

+ spin_lock(&serv->sv_lock);
  list_for_each_entry(xprt, xprt_list, xpt_list) {
   if (xprt->xpt_net != net)
    continue;
   set_bit(XPT_CLOSE, &xprt->xpt_flags);
   set_bit(XPT_BUSY, &xprt->xpt_flags);
  }
+ spin_unlock(&serv->sv_lock);
 }

 static void svc_clear_pools(struct svc_serv *serv, struct net *net)
@@ -949,24 +951,28 @@ static void svc_clear_pools(struct svc_serv *serv, struct net *net)
  }
 }

-static void svc_clear_list(struct list_head *xprt_list, struct net *net)
+static void svc_clear_list(struct svc_serv *serv, struct list_head *xprt_list, struct net *net)
 {
  struct svc_xprt *xprt;
```

```
  struct svc_xprt *tmp;
+ LIST_HEAD(victims);

+ spin_lock(&serv->sv_lock);
  list_for_each_entry_safe(xprt, tmp, xprt_list, xpt_list) {
   if (xprt->xpt_net != net)
    continue;
-  svc_delete_xprt(xprt);
+  list_move(&xprt->xpt_list, &victims);
  }
- list_for_each_entry(xprt, xprt_list, xpt_list)
-  BUG_ON(xprt->xpt_net == net);
+ spin_unlock(&serv->sv_lock);
+
+ list_for_each_entry_safe(xprt, tmp, &victims, xpt_list)
+  svc_delete_xprt(xprt);
 }

 void svc_close_net(struct svc_serv *serv, struct net *net)
 {
- svc_close_list(&serv->sv_tempsocks, net);
- svc_close_list(&serv->sv_permsocks, net);
+ svc_close_list(serv, &serv->sv_tempsocks, net);
+ svc_close_list(serv, &serv->sv_permsocks, net);

  svc_clear_pools(serv, net);
  /*
@@ -974,8 +980,8 @@ void svc_close_net(struct svc_serv *serv, struct net *net)
   * svc_xprt_enqueue will not add new entries without taking the
   * sp_lock and checking XPT_BUSY.
   */
- svc_clear_list(&serv->sv_tempsocks, net);
- svc_clear_list(&serv->sv_permsocks, net);
+ svc_clear_list(serv, &serv->sv_tempsocks, net);
+ svc_clear_list(serv, &serv->sv_permsocks, net);
 }

 /*
```