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