Hi,

I believe I have located another underlying flaw in GServer: I have observed server threads becoming orphaned, leading to resource allocation (when @connections.size > @@maxConnections).

The cause is that the new service thread is added to @connections from the tcpServerThread, but is removed from @connections from within itself as part of the teardown mechanism. Under certain circumstances (not investigated too deeply, but I work with low power embedded platforms), it is possible that the thread scheduler allows the service thread to run to completion (or at least as far as removing itself from @connections) before the tcpServerThread adds it @connections. The result is that the service thread is never removed from @connections. Over time, this can result in @connections becoming filled with orphaned threads, stopping new connections from being accepted.

There are also some issues in GServer relating to TCPSocket shutdown: There is a TCP timeout parameter called CLOSE_WAIT (default = 7200s = 2hours). This is the maximum time that the TCP socket is kept open if the socket closing protocol aborts incorrectly. This can happen under some circumstances if both ends try to close the socket simultaneously.

The way the GServer code is written, the service thread TCP socket is closed before the thread is removed from @connections. I am guessing, but I believe that the intention is to limit the number of actively running threads, preventing the system performance from degrading due to processor load. However, while this is implemented, it is done so in an (unintentionally) extremely conservative manner. The TCPSocket#close method can block while waiting for the CLOSE_WAIT timeout, which keeps the thread alive, and stops it being removed from @connections.

I suggest that the service TCPSocket#close method should be called after the service thread is removed from @connections. This has the effect of limiting the GServer to @@maxConnections actively serving threads (ie those that are within GServer#serve or the derived class #serve method).

I haven't included patches at this point, as the changes to GServer are actually quite extensive. I have a drop-in replacement class for GServer (called GenServer) that I have extensively tested (I use it as the basis for an actor model for embedded systems deployed in very remote places (think jungles, tops of mountains, etc). GenServer implements:

- changes required to avoid thread orphaning problem as explained above
- changes required to avoid server blocking while TCPSockets are in CLOSE_WAIT timeout
- non-blocking TCPServerThread loop with an (empty) #heartbeat method callout (which allows the server to check whether it is in a good state (whatever that might mean in your implementation!)
- solves #6369
- logging implemented via Logger rather than hand-rolled
- improved error handling : GServer will attempt to retry rather than shutting down on StandardError (but will shutdown on Exception)

However, while I believe that it is a significant improvement on the existing GServer, I am not yet fully convinced that it is good enough for the standard library (there is one issue in particular around TCPServer#accept that could do with a better solution).

Let me know if you would like the GenServer code, or a series of patches, or what.

Kind regards

Steve

## Associated revisions

**Revision 1d23123c - 02/02/2013 04:13 AM - mame (Yusuke Endoh)**

- lib/gserver.rb (GServer#start): fix a timing issue. patch from Charles Nutter. [Bug #7081]
Revision 39016 - 02/02/2013 04:13 AM - mame (Yusuke Endoh)

- lib/gserver.rb (GServer#start): fix a timing issue. patch from Charles Nutter. [Bug #7081]

Revision 39016 - 02/02/2013 04:13 AM - mame (Yusuke Endoh)

- lib/gserver.rb (GServer#start): fix a timing issue. patch from Charles Nutter. [Bug #7081]

Revision 39016 - 02/02/2013 04:13 AM - mame (Yusuke Endoh)

- lib/gserver.rb (GServer#start): fix a timing issue. patch from Charles Nutter. [Bug #7081]

Revision 39016 - 02/02/2013 04:13 AM - mame (Yusuke Endoh)

- lib/gserver.rb (GServer#start): fix a timing issue. patch from Charles Nutter. [Bug #7081]

Revision 39016 - 02/02/2013 04:13 AM - mame (Yusuke Endoh)

- lib/gserver.rb (GServer#start): fix a timing issue. patch from Charles Nutter. [Bug #7081]

Revision 39016 - 02/02/2013 04:13 AM - mame (Yusuke Endoh)

- lib/gserver.rb (GServer#start): fix a timing issue. patch from Charles Nutter. [Bug #7081]

History

#1 - 11/07/2012 11:46 PM - mame (Yusuke Endoh)

- Status changed from Open to Assigned
- Assignee set to mame (Yusuke Endoh)
- Target version set to 2.0.0

stevegoobermanhill (stephen gooberman-hill) wrote:

Hi,
I believe I have located another underlying flaw in GServer: I have observed server threads becoming orphaned, leading to resource allocation (when @connections.size > @@maxConnections).

As you know, there is no maintainer for GServer.
(I really want to remove GServer...)
So, I'd like to apply a minimum patch for 2.0.0. How about this?

```diff
diff --git a/lib/gserver.rb b/lib/gserver.rb
index f6f37d3..8a2ef4c 100644
--- a/lib/gserver.rb
+++ b/lib/gserver.rb
@@ -261,7 +261,10 @@ class GServer
end
} {myClient}
  begin            myPort = myClient.peeraddr[1]
-          serve(myClient) if @audit or connecting(myClient)
+          serve(myClient) if @audit or connecting(myClient)
@@ -279,6 +282,7 @@ class GServer
           disconnecting(myPort) if @audit
          end
        }
@@ -265,7 +268,7 @@ class GServer
           @connections << Thread.new(client) { |myClient|
-          @connections << Thread.new(client) { |myClient|
+          @connections << Thread.new(client) { |myClient|
            tmp_lock = Mutex.new
            tmp_lock.lock @connections << Thread.new(client) { |myClient| 
@@ -279,6 +282,7 @@ class GServer
           disconnecting(myPort) if @audit
          end

There are also some issues in GServer relating to TCPSocket shutdown: There is a TCP timeout parameter called CLOSE_WAIT (default = 7200s = 2hours).

If you need, please configure "/proc/sys/net/ipv4/tcp_keepalive_time" or what!

I haven't included patches at this point, as the changes to GServer are actually quite extensive. I have a drop-in replacement class for GServer (called GenServer) that I have extensively tested (I use it as the basis for an actor model for embedded systems deployed in very remote places.
Great. However, I doubt people really needs GServer or its compat.
AFAIK, the reason why GServer is bundled is just because it is used by the stdlib xmlrpc server, which is also little-used (in my personal opinion).

--
Yusuke Endoh mame@tsg.ne.jp

**#2 - 11/25/2012 01:39 AM - headius (Charles Nutter)**

FWIW, this could have been fixed by ko1’s proposal for threads that don’t start until you start them. This would guarantee the thread was created and in the list before started.

There’s an alternative fix:

```
 diff --git a/lib/ruby/1.9/gserver.rb b/lib/ruby/1.9/gserver.rb
index f6f37d3..8eac2a2 100644
--- a/lib/ruby/1.9/gserver.rb
+++ b/lib/ruby/1.9/gserver.rb
@@ -261,7 +261,8 @@ class GServer
 end
 }
 client = @tcpServer.accept
@@ -261,7 +261,8 @@ class GServer
 end
 }
 client = @tcpServer.accept
- @connections << Thread.new(client)  { |myClient|
+ @connections << Thread.new(client)  { |myClient|
    @connections << Thread.current          begin            myPort = myClient.peeraddr[1]
      serve(myClient) if @audit or connecting(myClient)
    end

This makes the thread responsible for both adding and removing itself, which feels cleaner than introducing locks to ensure the thread has been added before it starts.

Of course, on an implementation like JRuby, concurrently mutating the @connections array is not thread-safe anyway.

**#3 - 02/02/2013 01:13 PM - mame (Yusuke Endoh)**

- Status changed from Assigned to Closed
- % Done changed from 0 to 100

This issue was solved with changeset r39016.
stephen, thank you for reporting this issue.
Your contribution to Ruby is greatly appreciated.
May Ruby be with you.

- lib/gserver.rb (GServer#start): fix a timing issue. patch from Charles Nutter. [Bug #7081]