How can I guarantee a "stay alive" heartbeat is sent?

Asked 15 years, 9 months ago Modified 15 years, 9 months ago Viewed 9k times

3

We have an RMI client application written in Java which needs to send periodic "stay alive" messages to a server application. We have implemented this as a separate heartbeat thread, which sends the stay alive message to the server, then sleeps for 15 seconds using Thread.sleep().

The thread is set to be high priority:

```
Thread heartbeatThread = new Thread(new HeartbeatRunnable(server));
heartbeatThread.setPriority(Thread.MAX_PRIORITY);
heartbeatThread.start();
```

However, when the box on which the client is running is using lots of CPU, we find that heartbeats are missed, which causes the server to assume our client application has died.

We have added Thread.yield() calls in my main thread, although this has not made the problem go away.

Is there any way of guaranteeing that heartbeats are sent on time while my application is still running?

java concurrency rmi heartbeat

Share
Improve this question
Follow

edited Mar 24, 2009 at 11:08

asked Mar 24, 2009 at 10:29

Simon Nickerson
43.1k • 20 • 106 • 127

6 Answers

Sorted by: Highest score (default)

3

You can not really guarantee it. You could send the heartbeat in a different thread to prevent the time it takes to send the heartbeat being added to your delay. It may also be advisable to set the delay between two heartbeats to half the time the server uses to decide a client is dead, i.e. if your server times out your client after 15 seconds, (try to) send a heartbeat every 7.5 seconds.

Share Improve this answer Follow

answered Mar 24, 2009 at 10:38

Bombe

83.7k ● 20 ● 127 ● 127

Thanks for the response. We are already sending heartbeats in a separate thread, and the server cuts us off after no heartbeats for 90 seconds, so sending every 15 seconds seemed conservative. – Simon Nickerson Mar 24, 2009 at 11:12

1 It is very conservative, indeed. But if you don't manage to send out a heartbeat for 90 seconds when trying to send one every 15 seconds your box is seriously overloaded and your client maybe really should be considered dead. See Pete's answer. – Bombe Mar 24, 2009 at 11:37

2

It depends what process is using the CPU.

If it's not your process, and so the client process really is unresponsive, then it is to all intents and purposes not alive, so not sending a heartbeart is appropriate. Having a heartbeat message which says 'I'm up and can process messages' when the box is too loaded to do that would be misleading.

If the intent of the heartbeat message is to say 'this process is running, but it might be half an hour until I get back to you', then have whatever is doing that processing send that message to the server. Or set the timeout to one that fits with the responsiveness of the client.

Share Improve this answer Follow

answered Mar 24, 2009 at 10:48

Pete Kirkham

49.3k ● 5 ● 94 ● 173

1

You can implement user-mode threading in a non-threaded environment by liberally scattering a self-written "yield" function in your code.

Similarly, you could liberally scatter heartbeat check function calls in your code. Forgo the thread, simply regularly call a heartbeat function which checks to see if a heartbeat yet needs to be sent.

It's a crude solution, but given you've tried the proper solution and it doesn't work, perhaps it's something you have to fall back to.

In fact what you could do is place a macro at the beginning of every function call, which does a quick check on the time and calls the heartbeat function when necessary.

(Ah, do you have macros in Java? I think not - but you get the idea).

Share Improve this answer Follow

answered Mar 28, 2009 at 14:20

user82238

1

You should configure the number of "missed heartbeats" that the server waits before deciding that the client is unavailable.

So, for example, if your heartbeat interval is 15 seconds and the number of missed heartbeats is 4, then the server will wait upto a maximum of 60 seconds (1 min) before deciding that the client is unreachable.

Share Improve this answer Follow

answered Mar 24, 2009 at 13:49

dogbane

274k ● 77 ● 404 ● 415

0

Perhaps the best solution would be to use Timer's scheduleAtFixedRate. With this, if one execution delays (which can't be avoided in Java), the subsequent calls won't be affected.

Share Improve this answer Follow

answered Mar 24, 2009 at 10:49

kgiannakakis

104k ● 28 ● 161 ● 197

0

If you want the server to announce that it's alive, you may be better off presenting an open socket. On your client simply read from that socket. It'll block (since your server isn't writing anything), and if the server disappears/shuts down your client will get an IOException indicating that the server socket/port has disappeared.

This won't be dependent on the server providing timely heartbeats. It uses few resources (a TCP port on the server end and next to no bandwidth) and it is timely in revealing when the server (or server machine) becomes unavailable.

Share Improve this answer Follow

answered Mar 24, 2009 at 11:39

Brian Agnew

272k ● 38 ● 339 ● 442

Doesn't that just track whether the server itself is alive? The target process could have crashed (or at least turned into a zombie) without the TCP/IP stack being any wiser.
– Pontus Gagge Mar 24, 2009 at 13:37

No. The operating system will look after that. It'll track whether the parent process of the socket/port combination is up – Brian Agnew Mar 24, 2009 at 16:35