

Forum:[Hack Forums](#)

Thread: [Multithreading implementation pattern problem](#)

morphine1337 - January 25, 2017, 10:51 am

Hello there. First of all, i'm using pthreads. So the scenario is this: There are servers of a game that send logs over UDP to an ip and port you give them. I'm building an application that will receive those logs, process them and insert them in a mysql database. Since i'm using blocking sockets because the number of servers will never go over 20-30, I'm thinking that i will create a thread for each socket that will receive and process logs for that socket. All the mysql information that needs to be inserted in the database will be send to a redis queue where it will get processed by another php running. Is this ok, or better, is it reliable ?

Haxalot - January 26, 2017, 1:59 am

The short answer is: it depends.

Now for the longer answer...

You didn't really specify whether you're building a server-based (web) application or a CLI-based application. If you opted for the former, then the firm answer is no, this would not be a good idea. Threading in a server-based environment in PHP is not a good idea because the threading model (which is 1:n, where 1 web request creates n threads) simply does not scale. If your application is receiving 1,000 requests per second, then it will also be creating 1,000 threads per second. This is why in pthreads v3, you simply cannot use the extension in a non-CLI-based environment (it's enforced by the extension itself). Instead, you should opt for performing operations asynchronously, via something like [Amphp](#).

If you're looking to build a CLI-based application, then pthreads could work for your scenario. It's Pool class makes it quite trivial to create and use a pool of (lazily-created) threads. Note that the documentation around pthreads v3 still isn't that up-to-date. I've been gradually updating it over the past few weeks, but it's still going to take another couple of weeks (since the changes in pthreads v3 were quite dramatic). The docs on pthreads.org are even more outdated, and I can't touch them either, so I'd stick to the docs in the PHP manual for now.

As for your setup, it sounds like you're over-complicating things by sending data that needs inserting into MySQL to Redis first, before sending it off to another PHP instance. Why not simply insert that data into the MySQL database from the threads? If you're using the mysqli API, then it exposes asynchronous queries to you (though they have little documentation around them...). Or simply use an asynchronous framework to handle the asynchronous operations (like Amphp mentioned above). Also, sending logs over UDP alone doesn't sound particularly reliable...

morphine1337 - January 26, 2017, 8:22 am

(Jan 26, 2017 02:59 AM)Haxalot Wrote: [\[CITE\]](#) The short answer is: it depends.

Now for the longer answer...

You didn't really specify whether you're building a server-based (web) application or a CLI-based application. If you opted for the former, then the firm answer is no, this would not be a good idea.

Threading in a server-based environment in PHP is not a good idea because the threading model (which is 1:n, where 1 web request creates n threads) simply does not scale. If your application is receiving 1,000 requests per second, then it will also be creating 1,000 threads per second. This is why in pthreads v3, you simply cannot use the extension in a non-CLI-based environment (it's enforced by the extension itself). Instead, you should opt for performing operations asynchronously, via something like [Amphp](#).

If you're looking to build a CLI-based application, then pthreads could work for your scenario. It's Pool class makes it quite trivial to create and use a pool of (lazily-created) threads. Note that the documentation around pthreads v3 still isn't that up-to-date. I've been gradually updating it over the past few weeks, but it's still going to take another couple of weeks (since the changes in pthreads v3 were quite dramatic). The docs on pthreads.org are even more outdated, and I can't touch them either, so I'd stick to the docs in the PHP manual for now.

As for your setup, it sounds like you're over-complicating things by sending data that needs inserting into MySQL to Redis first, before sending it off to another PHP instance. Why not simply insert that data into the MySQL database from the threads? If you're using the mysqli API, then it exposes asynchronous queries to you (though they have little documentation around them...). Or simply use an asynchronous framework to handle the asynchronous operations (like Amphp mentioned above). Also, sending logs over UDP alone doesn't sound particularly reliable...

So the scenario is: CSGO servers have an option to send logs over UDP sockets given a specific ip and port. I have a cronjob that checks every minute if another server has been added and sends this data over ZeroMQ to my CLI based application. On the application side, when receiving a new server ip and port it will create a new thread where a new socket will be created it will bind to that ip and port and start receiving. There will be a maximum of 20 servers so maximum 20 threads. So each thread will bind to a socket and start receiving data from the server that it has been assigned to. When new data arrives I want to process it and do some queries to a mysql database (insert, update, delete) my question is: can i have a single mysql connection and all threads use it ? And would this architecture of a long running php process work ?

The whole redis thing was that in case i can't use the same mysql connection in all threads or if i cant just start a new connection to mysql in each thread so there will be a single application that will manage mysql queries.

Haxalot - January 26, 2017, 10:12 am

(Jan 26, 2017 09:22 AM)morphine1337 Wrote: [\[CITE\]](#) my question is: can i have a single mysql connection and all threads use it ? And would this architecture of a long running php process work ?

To answer you first question, no, threads cannot share the same connection to MySQL. This is because only Threaded objects may be passed between threads (since they employ various hacks internally to make this possible in PHP). Therefore, you will need to create a new DB connection per thread. This really isn't that bad though.

As for you second question, yes, it should work fine to have an application as a long running PHP process. It's difficult to comment on the overall details of whether the way you specified above will be the best way to do this, but that's something you will be able to see clearer as you progress through the project. In any case, if you have any problems (or if you find any problems with PHP or the pthreads extension (i.e. leaking)), then by all means come back here and ask away again!

morphine1337 - January 26, 2017, 11:35 am

(Jan 26, 2017 11:12 AM)Haxalot Wrote: [\[CITE\]](#) To answer you first question, no, threads cannot share the same connection to MySQL. This is because only Threaded objects may be passed between threads (since they employ various hacks internally to make this possible in PHP). Therefore, you will need to create a new DB connection per thread. This really isn't that bad though.

As for your second question, yes, it should work fine to have an application as a long running PHP process. It's difficult to comment on the overall details of whether the way you specified above will be the best way to do this, but that's something you will be able to see clearer as you progress through the project. In any case, if you have any problems (or if you find any problems with PHP or the pthreads extension (i.e. leaking)), then by all means come back here and ask away again!

Ok, so what I will do because pthreads and composer does not work very well and I need classes inside threads I will do the following :

1. The threads that will each bind to a socket and recv from the server. All of the data received will be put in a logqueue
2. In the main thread I will do a loop and check if there are logs in the queue and process them

Like this: <http://pastebin.com/qH8fzh5Z>