SYSTEM DESIGN OF MULTITHREADED WEB SERVER WITH BACKEND DATABASE CONNECTIONS

SENG, L. K.

Department of Electrical Engineering, National University of Singapore, Lower Kent Ridge Rd, Singapore.
e-mail: lksark[at]hotmail.com; lksark[at]gmail.com

(Received 12th May 2021; accepted 24th June 2021)

Abstract. Web server application creates multiple duplicated subprocesses. Each subprocess software threads listen to a dedicated web socket. Server portal subprocess uses shared list to inform clients which web socket to establish connection with. Moreover, it uses shared queue to aggregate the clients' database queries. When queue has DB queries, multiple DB connection threads will be invoked to read or write data concurrently to the database. When shared queue is empty, shared queue will put the DB connections threads into wait state to idle instead of disconnecting the connections.
Keywords: *multithreaded web application, server-client, concurrency, concurrent connections, database connections pool*

Short communication

Web server applications are supposed to response to multiple clients. This article will demonstrate a web server application system design, capable of handling multiple clients concurrently without need to use excessive hardware. This server application able to listen & accept multiple clients network connections concurrently; able to maintain communications and duplex data transmission with multiple clients concurrently using multiple web sockets; able to read/write multiple data to database concurrently.

Computers' CPU nowadays is multi-cores and multithreaded. Therefore, we should design our web server application to be multithreaded to response to multiple clients concurrently without the need to use excessive hardware.

Materials and Methods

Download the source code from my web URL homepage, compile it using Microsoft Visual Studio and execute to reproduce the reported outcome. The source codes are written in Microsoft C#.

After compilations, execute single server application and multiple client applications simultaneously. We can see that server application is able maintain communications and duplex data transmission with multiple clients' application concurrently. If number of clients is more than server application available web sockets, server application will inform new clients to wait awhile then re-attempt to connect.

The homepage for My online source code URL are:

(i) https://www.algoonline.net/MultithreadingWorkloadsDistribution/Multi-threading_workloads_distribution.htm
(ii) https://github.com/lksark/Multithreaded-TCP-server-with-multiple-database-connections-pool

(iii) https://github.com/lksark/-Multithreaded-Chat-Server-communicates-with-multiple-clients

Source code (ii) is simulated TCP chat server-client applications. Via chat server application, port number '50001' chat-client chat with port number '50002' chat-client; port number '50003' chat-client chat with port number '50004' chat-client.

Source code (iii) is simulated employee clock-in & clock-out applications. Multiple client applications are able to read or write clock-in or clock-out time concurrently from-or-to database via a single server application.

Results and Discussion

Modern day computers and operating systems have 16-bit unsigned integer network ports, ranging from 0 to 65535 (Hall, 2019; Kozierok, 2005; Makofske et al., 2004; Bonner, 1995). Higher-numbered ports are available for general use by applications and are known as ephemeral ports.

In computer networking, a port is a communication endpoint. When server application established connection with a client using a network port (Hall, 2019; Kozierok, 2005; Makofske et al., 2004; Deitel et al., 2002; Bonner, 1995), it takes times to process client requests. Hence during this period, other clients will not able to establish connection with the server application using the same network port.

A web application can create multiple duplicated subprocesses. Each subprocesses executed by a dedicated software thread, listens to a dedicated web socket. This enables the web application to handle equivalent count of clients concurrently. However, this method has a challenge. How clients know which web socket should they connect to? All web application's duplicated subprocesses are performing the same tasks.

A simple solution for concurrent connection is we can bind web server's network port to a known client, though this will result in we cannot reuse this network port for other clients.

Instead of binding a network port number to a known client during concurrent connection, in many scenarios we would want to re-use the network port numbers for any known or unknown clients.

This article illustrates a method of a web application or network device uses queue/list/stack/priority queue to inform clients which web socket they should establish connection with during concurrent connection. Moreover, within the web application, it uses queue/list/stack/priority queue to aggregate the database queries it received from concurrently connected clients.

In server-client model, a web application listens to multiple web sockets. When clients visit the web application, clients firstly go to portal network port. Application's portal server subprocess will inform client which worker web socket to connect to using queue/list/stack/priority queue and give a handshake passcode. Client connects to the pre-assigned web socket and sends the handshake passcode to maintain the connection.

If client didn't connect to the pre-assigned web socket for a given period or sent a wrong handshake passcode to worker server, the web socket is pushed back to the queue. Respective web socket's handshake passcode will be changed too. Client would have to visit the portal web socket again to get to know which worker web socket to connect. This is to prevent stray clients connect to worker web socket directly without firstly visit the portal web socket.

After connected, if client idles too long or connected to worker server too long, worker server closes the current connection. Vice versa, when worker server idles too long or connected too long, client can close the connection. Client's timeout period would be slightly longer than worker server.

When all the server application's worker web sockets are occupied, portal server thread will inform the new clients to wait awhile then re-attempt the connection.

Web applications commonly have backend database. In this article, a multithreaded web application maintains multiple database connections. The multithreaded web application able to accept multiple incoming client connections concurrently using server's ephemeral ports and should able to R/W multiple data from/to backend database concurrently using DB's ephemeral ports.

The web application and DB program are two separate computer programs. Both programs can be resided in the same computer (localhost) or in separate computers. R/W data to database take times. Therefore, preferably to have multiple DB connections and multiple server-clients web-sockets concurrently to improve the DB efficiency and responsiveness to clients.

In simple SQL query procedure, process need to first establish connection with DB before the SQL query. The connection string includes login DB username and password, DB's IP address, selected DB name, port number & etc. After finished one SQL query, process will close the connection with DB. This process will repeatedly login & logout DB for every SQL query to DB.

To avoid frequently open & close connections to DB, this design uses the connection pool idea. A multithreaded application login & maintains multiple connections to DB using multiple software threads & DB ephemeral ports. When no query to DB, the DB connections will be put into wait the state to idle. Connection pool idea can be used in any type of connections that are allowed to be alive for extended period by the host; we can create multiple concurrent connections to the same source if permitted and if it would improve the efficiency.

In TCP server-to-DB coding portion, we will have a shared queue to save all the clients' DB queries. Established DB connection threads will execute the SQL queries whenever there are any in the shared queue. If there aren't any in the shared queue, shared queue will make DB connection threads to idle in Wait state. TCP server connections to DB are retained in Wait state. When new SQL queries are added into the shared queue, shared queue will invoke an idle DB connection thread to R/W data to DB. TCP server-to-DB code resides in TCP server program.

Worker server-client threads and DB connection threads are separate software threads. When worker server-client thread query DB via DB connection thread, worker server-client thread needs to wait for DB connection thread to get DB query result & notify him.

Client establishes a new TCP connection with server for every new query to the server. Server close the TCP connection after sent the client the query result. A connection to DB that does not last more than 1 minute is considered an exceptional connection error. When exceptional connection errors count more than 10 times, reattempt to connect stop. Multiple concurrent DB connections and multiple concurrent server-clients web-sockets improve the DB efficiency and responsiveness to clients.

Conclusion

Modern CPU are increasingly having more processing cores. Design a multithreaded web server with multiple backend DB connections will increase DB efficiency and responsiveness to client. Multithreaded web application makes concurrent communications with multiple clients possible with minimum hardware.

Multithreaded web applications are inevitable in critical web applications. All the clients' requests are expected to be attended and the connections are expected to be alive until web server reply. Web application should listen to sufficient count of web sockets, thus making sure that all concurrent incoming clients' requests are attended. No clients' requests should fail due to client timeout in attempting to connect to web server application.

Acknowledgement

This research is self-funded.

Conflict of interest

The author confims that there are no conflict of interest involve with any parties in this research study.

REFERENCES

[1] Bonner, P. (1995): Network Programming with Windows Sockets (Bk/Disk). – Prentice Hall Ptr 512p.
[2] Deitel, H.M., Deitel, P.J., Nieto, T.R. (2002): Visual Basic. NET: how to program. – Prentice Hall 1517p.
[3] Hall, B. (2019): Beej's guide to network programming: Using internet sockets. – Independent Published 174p.
[4] Kozierok, C.M. (2005): TCP/IP Guide: A Comprehensive, Illustrated Internet Protocols Reference. – No Starch Press 1616p.
[5] Makofske, D., Donahoo, M.J., Clavert, K.L. (2004):TCP/IP Sockets in C#: Practical Guide for Programmers (The Practical Guides). – Morgan Kaufmann 192p.