RemoteDictionary Task

Overview

You should implement a client-server application that uses a minimalist remote dictionary access protocol, defined bellow. Not to mention, the project requires you to handle both C files and CPP files altogether within the same Cmake project.

Necessary Files to develop

C Files

1. tcp_client.c & tcp_client.h files for client-side socket implementation.
2. tcp_server.c & tcp_server.h files for server-side concurrent socket implementation.

CPP Files

3. dictionary.cpp & dictionary.h files for data storage.
4. main.cpp file to initialize server/client and process request messages.

Build

Use CmakeLists.txt file to build the project and make sure to compile both C & CPP files independently using same cmakelist file.

Protocol

The protocol defines a set of operations over the remote dictionary object RemoteDictionary.

The data types used are specified only at the requirements level. You may implement them as you wish, as long as the implementation meets the requirements.

Networking

The protocol must be implemented using TCP.

Request-response model

The client sends a request

The server handles it

The server passes a response to the client.

Serialization

Any serialization method may be used as long as the RemoteDictionary requirements are met.

Possible options are: FlatBuffers, Protocol Buffers, Json, etc.

Server

The server should implement the RemoteDictionary data structure and the network exposed access protocol with client handling.

The implementation should be non-blocking. Libraries such as libuv or boost may be used.

tcp_server.c file must include socket_init(), socket_bind(), socket_listen(), socket_accept(), socket_recv(), socket_send() and server_start() functions. Here, server_start() function should traverse in an infinite loop listening on a port for a remote client to connect. Once a remote client connects, it should create a child thread for incoming client and start processing his request messages.

RemoteDictionary

In this section we define the RemoteDictionary requirements per operation.

RemoteDictionary::get(key: string) GetResponse

Lookup the specified key and pass back a GetResponse.

The GetResponse type must be able to convey either success or failure.

The successful case includes the value found in the dictionary at the specified key.

The failure case includes the reason behind it.

RemoteDictionary::set(key: string, value: string) SetResponse

Store the value in the dictionary at the specified key.

The SetResponse type must be able to convey either success or failure.

In case of failure the reason behind it must be included.

RemoteDictionary::stats() StatsResponse

Get dictionary statistics, should include:

- total number of get operations

- total number of successful get operations

- total number of failed get operations

Important: You have to call these dictionary methods directly from tcp_server.c file right after you receive a request over tcp socket and finish processing it.