Project 2: Understanding Read Copy Update

Goal

The goal of this project is to understand the mechanism of read copy update. You are required to write an application that accesses a shared data structure through the network. Multiple instances of the application will be running on different machines. A standalone server controls the shared data structure.

General Information

- Read the paper "Read-Copy Update: Using Execution History to Solve Concurrency Problems" from the syllabus first.
- Read this assignment carefully, in its entirety.
- You are free to discuss the paper with anybody. But the design for your project must be done individually.
- The format of the turn-in is strictly enforced.

The Project Requirement

The project consists of three steps.

1. Implement a simple server that manages shared data structures.
2. Write an application that communicates with the server.
3. Implement your read copy update primitives to control the synchronization.

Step One

First, you will write a server that accepts incoming requests, and sends proper responses. The server listens to a non-block socket. It maintains multiple connections at the same time. It should use 'select' to determine whether there are new connection requests, or incoming messages from established connections.

The server is used to simulate the access to a shared data structure. For this purpose, the server will allocate 1000 triplets indexed by 0..999 at the beginning. Each triplet consists of three integers (A, B, C) where A equals to B and B equals to C. You should initialize the array of triplets using random numbers.
The server should handle two types of messages:

1. Read the Nth (N=1, 2 or 3) element of the Mth (M=0, 1... or 999) triplet where N and M should be included in the incoming message.
2. Write value X to the Nth element of the Mth triplet where X, N and M should be included in the incoming message.

The format of the message and the response is up to you.

**Step Two**

Write an application that can talk to the above server. You should specify the server name and port in the command line. The application takes two additional parameters: \(\text{dividend}\) and \(\text{remainder}\). The application should be programmed in the following way:

```c
while ( exit condition not reached. ) {
    for (i=0; i<1000; i++) {
        Send a message to the server to read the 1st element(A) of the ith triplet.
        Receive the response to recover value of A.
        Send a message to the server to read the 2nd element(B) of the ith triplet.
        Receive the response to recover value of B.
        Send a message to the server to read the 3rd element(C) of the ith triplet.
        Receive the response to recover value of C.

        Check whether A==B && B==C. // **

        if ( A % dividend == remainder
             // B % dividend == remainder
             // C % dividend == remainder ) {
            A++; B++; C++;
            Send a message to the server to update the 3rd element of the ith triplet with value C.
            Receive the response from the server that indicates success.
            Send a message to the server to update the 2nd element of the ith triplet with value B.
            Receive the response from the server that indicates success.
            Send a message to the server to update the 1st element of the ith triplet with value A.
            Receive the response from the server that indicates success.
        } }
}
```

You can define the exit condition of your program using any proper method (for example, a certain time, a termination signal etc.). You will run \#\(\text{dividend}\) instances of your application on different machines. The instances should have different \(\text{remainder}\) value from 0 to \(\text{dividend} - 1\).
When the server exits, it should output the value of each triplets (one triplet per line of text). It should check whether the property of 'A equals to B and C' keeps for each of the triples. Since the read/write operation is not guarded by any protection mechanism, the property may not hold at that time. Also at step** of the above pseudo code, the property may not hold.

**Step Three**

Based on your understanding of the paper, define the quiescent state and the quiescent period. Design a read-copy update implementation for this particular problem. Add new request/response primitives to the server as necessary. Then update your application with your RCU implementation.

Write a report to explain your design and implementation in detail.

If your mechanism works properly, the server output at the end should keep the 'A equals to B and C' property for all triplets. And your application will always observe that A, B and C have the same value. Please note that you have to use 'read-copy-update', not other lock mechanisms.