3/26/25, 4:11 PM clientMain

```
1
     package main
 2
 3
     import (
             "bufio"
 4
 5
             "fmt"
             "loa"
 6
 7
             "net/rpc"
             "os"
 8
 9
             "strconv"
10
             "time"
11
    )
12
13
14
    // Data Structures and RPC Arg/Reply
15
16
17
    type SetKeyArgs struct {
18
             BucketName string
19
             Key
                         int
20
             Value
                         string
21
    }
22
23
    type GetKeyArgs struct {
24
             BucketName string
25
             Key
                         int
26
    }
27
28
    type DeleteKeyArgs struct {
29
             BucketName string
30
             Key
                         int
    }
31
32
33
    type EmptyRequest struct{}
34
35
    type Response struct {
             Data
36
                    string
37
             Message string
38
    }
39
40
    // RPC Helper Functions
41
42
43
44
     func getKey(client *rpc.Client, args GetKeyArgs) string {
45
             var reply Response
46
             err := client.Call("LeaderRPC.GetKey", args, &reply)
             if err != nil {
47
48
                      return fmt.Sprintf("[ERROR getKey]: %v", err)
49
50
             if reply.Message != "" {
51
                      return fmt.Sprintf("[FAIL getKey]: %s", reply.Message)
52
53
             return reply.Data
54
    }
55
56
     func setKey(client *rpc.Client, args SetKeyArgs) string {
57
             var reply Response
58
             err := client.Call("LeaderRPC.SetKey", args, &reply)
59
             if err != nil {
60
                      return fmt.Sprintf("[ERROR setKey]: %v", err)
61
             if reply.Message != "OK" {
62
63
                      return fmt.Sprintf("[FAIL setKey]: %s", reply.Message)
```

3/26/25, 4:11 PM clientMain

```
64
              return "OK"
65
66
     }
67
     func deleteKey(client *rpc.Client, args DeleteKeyArgs) string {
68
69
              var reply Response
70
              err := client.Call("LeaderRPC.DeleteKey", args, &reply)
 71
              if err != nil {
72
                      return fmt.Sprintf("[ERROR deleteKey]: %v", err)
73
              }
74
              return reply.Message
75
     }
76
77
     // For the new test calls
 78
     func getStoreInfo(client *rpc.Client) string {
79
              var reply Response
80
              err := client.Call("LeaderRPC.GetStoreInfo", EmptyRequest{}, &reply)
81
              if err != nil {
82
                      return fmt.Sprintf("[ERROR getStoreInfo]: %v", err)
83
84
              if reply.Message != "" {
                      return fmt.Sprintf("[FAIL getStoreInfo]: %s", reply.Message)
85
86
              }
87
              return reply.Data
88
     }
89
90
91
     // Test Routines
92
93
94
     func testSetGetDelete(client *rpc.Client) {
95
              fmt.Println("== TestSetGetDelete on Leader ==")
96
              setRes := setKey(client, SetKeyArgs{"User", 1, "User A"})
97
98
              fmt.Println(" SetKey(User,1) =>", setRes)
99
100
              // 2) Get
101
              getRes := getKey(client, GetKeyArgs{"User", 1})
              fmt.Println(" GetKey(User,1) =>", getRes)
102
103
104
              // 3) Delete
              delRes := deleteKey(client, DeleteKeyArgs{"User", 1})
105
              fmt.Println(" DeleteKey(User,1) =>", delRes)
106
107
108
              // 4) Verify it's really deleted
109
              postDel := getKey(client, GetKeyArgs{"User", 1})
              fmt.Println(" GetKey(User,1) after Delete =>"
                                                             , postDel)
110
              fmt.Println("== Done TestSetGetDelete ==\n")
111
112
     }
113
     func testMassSetAndGet(client *rpc.Client, bucketName string, count int) {
114
              fmt.Printf("== Mass set/get test for %d items in bucket %q ==\n",
115
     count, bucketName)
116
              startTime := time.Now()
117
118
              // Bulk set
119
              for i := 0; i < count; i++ {
                      setRes := setKey(client, SetKeyArgs{
120
121
                              BucketName: bucketName,
122
                              Key:
                                           "Item " + strconv.Itoa(i),
123
                              Value:
                      })
124
                      if setRes != "OK" {
125
                               fmt.Printf(" Failed setKey at i=%d => %s\n", i,
126
     setRes)
```

3/26/25, 4:11 PM clientMain

```
127
                      }
              }
128
129
130
              // Bulk get
              missing := 0
131
              for i := 0; i < count; i++ {</pre>
132
                      val := getKey(client, GetKeyArgs{
133
134
                               BucketName: bucketName,
135
                              Key:
136
                      })
                      // If val is [FAIL or [ERROR, it indicates an issue
137
138
                      if len(val) >= 5 && (val[:5] == "[FAIL" || val[:6] == "
      [ERROR") {
139
                              missing++
                      }
140
141
              }
142
              elapsed := time.Since(startTime)
143
144
              fmt.Printf(" Mass set/get done in %v. Missing count=%d\n", elapsed,
     missing)
145
              fmt.Println("== Done Mass set/get ==\n")
146
147
148
     // main: orchestrates the test
149
150
151
152
      func main() {
              // Hard-coded addresses:
153
154
              // - Leader is on :8000
155
              // - Backup is on :8001
              leaderAddr := "localhost:8000"
156
              backupAddr := "localhost:8001"
157
158
159
              // 1) Connect to the leader
              leaderClient, err := rpc.Dial("tcp", leaderAddr)
160
161
              if err != nil {
162
                      log.Fatalf("Failed to connect leader @ %s: %v", leaderAddr,
     err)
163
              fmt.Println("[Connected to leader@", leaderAddr, "]")
164
165
              // 2) Basic Set/Get/Delete on Leader
166
167
              testSetGetDelete(leaderClient)
168
              // 3) Mass test
169
170
              testMassSetAndGet(leaderClient, "BulkBucket", 20)
171
172
              // ----- NEW TEST CASES FOR GetStoreInfo -----
173
              fmt.Println("== Test #1: GetStoreInfo from the leader (should
      succeed) ==")
174
              storeLeader := getStoreInfo(leaderClient)
              fmt.Println(" Leader's store info =>", storeLeader)
175
176
177
              fmt.Println("\n== Test #2: GetStoreInfo from backup (should fail or
      show 'Not the leader') ==")
178
              backupClient, err := rpc.Dial("tcp", backupAddr)
179
              if err != nil {
                      fmt.Printf("[WARN] Could not connect to backup @ %s: %v\n",
180
     backupAddr, err)
181
                      fmt.Println("Skipping backup storeInfo test.")
182
              } else {
183
                      storeBackup := getStoreInfo(backupClient)
                      fmt.Println(" Backup's store info =>", storeBackup)
184
185
                      backupClient.Close()
```

3/26/25, 4:11 PM clientMain

```
186
187
188
189
              // 4) Ask user to kill the leader
190
              fmt.Println("\n[ACTION REQUIRED] Please kill/stop the leader node.
     Then press ENTER.")
191
              bufio.NewReader(os.Stdin).ReadBytes('\n')
192
193
              // 5) Wait for failover
194
              fmt.Println("Waiting 5s to let cluster elect new leader on
      :8000...")
195
             time.Sleep(5 * time.Second)
196
197
              // 6) Connect to the new leader (still :8000)
              newLeaderClient, err := rpc.Dial("tcp", leaderAddr)
198
199
              if err != nil {
200
                      log.Fatalf("Failed to connect new leader @ %s: %v",
     leaderAddr, err)
201
              fmt.Println("[Connected to new leader @", leaderAddr, "]\n")
202
203
              // 7) Verify data is consistent
204
205
              fmt.Println("== Checking data from BulkBucket on new leader ==\n")
206
              lastVal := getKey(newLeaderClient, GetKeyArgs{"BulkBucket", 20})
207
              fmt.Println(" GetKey(BulkBucket,20) =>", lastVal)
208
209
              // 8) Check store info again on new leader
              fmt.Println("\n== Test #3: GetStoreInfo on new leader (post-
210
     failover) ==\n")
              storeNewLeader := getStoreInfo(newLeaderClient)
211
212
              fmt.Println(" NewLeader store info =>", storeNewLeader)
213
              // 9) Additional sets/deletes on new leader
214
215
              fmt.Println("\n== Testing another SET/DELETE on new leader ==\n")
216
              setRes := setKey(newLeaderClient, SetKeyArgs{"FailoverBucket", 1,
     "DataAfterFailover"})
217
              fmt.Println(" SetKey(FailoverBucket,1) =>", setRes)
218
219
              delRes := deleteKey(newLeaderClient, DeleteKeyArgs{"FailoverBucket",
     1})
220
              fmt.Println(" DeleteKey(FailoverBucket,1) =>", delRes)
221
222
              postDel := getKey(newLeaderClient, GetKeyArgs{"FailoverBucket", 1})
              fmt.Println(" GetKey(FailoverBucket,1) =>", postDel)
223
224
225
              newLeaderClient.Close()
              leaderClient.Close()
226
227
              fmt.Println("\n=== End of All Tests ===")
228
229
     }
230
```