

```
1 package main
2
3 import (
4     "bufio"
5     "fmt"
6     "log"
7     "net/rpc"
8     "os"
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 {
36     Data      string
37     Message   string
38 }
39
40 // -----
41 // RPC Helper Functions
42 // -----
43
44 func getKey(client *rpc.Client, args GetKeyArgs) string {
45     var reply Response
46     err := client.Call("LeaderRPC.GetKey", args, &reply)
47     if err != nil {
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     }
62     if reply.Message != "OK" {
63         return fmt.Sprintf("[FAIL setKey]: %s", reply.Message)
64     }
65 }
```

```

64     }
65     return "OK"
66 }
67
68 func deleteKey(client *rpc.Client, args DeleteKeyArgs) string {
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 != "" {
85         return fmt.Sprintf("[FAIL getStoreInfo]: %s", reply.Message)
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     // 1) Set
97     setRes := setKey(client, SetKeyArgs{"User", 1, "User A"})
98     fmt.Println(" SetKey(User,1) =>", setRes)
99
100    // 2) Get
101    getRes := getKey(client, GetKeyArgs{"User", 1})
102    fmt.Println(" GetKey(User,1) =>", getRes)
103
104    // 3) Delete
105    delRes := deleteKey(client, DeleteKeyArgs{"User", 1})
106    fmt.Println(" DeleteKey(User,1) =>", delRes)
107
108    // 4) Verify it's really deleted
109    postDel := getKey(client, GetKeyArgs{"User", 1})
110    fmt.Println(" GetKey(User,1) after Delete =>", postDel)
111    fmt.Println("== Done TestSetGetDelete ==\n")
112 }
113
114 func testMassSetAndGet(client *rpc.Client, bucketName string, count int) {
115     fmt.Printf("== Mass set/get test for %d items in bucket %q ==\n",
count, bucketName)
116     startTime := time.Now()
117
118     // Bulk set
119     for i := 0; i < count; i++ {
120         setRes := setKey(client, SetKeyArgs{
121             BucketName: bucketName,
122             Key:        i,
123             Value:       "Item " + strconv.Itoa(i),
124         })
125         if setRes != "OK" {
126             fmt.Printf(" Failed setKey at i=%d => %s\n", i,
setRes)

```

```

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

```

```

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)
198     newLeaderClient, err := rpc.Dial("tcp", leaderAddr)
199     if err != nil {
200         log.Fatalf("Failed to connect new leader @ %s: %v",
leaderAddr, err)
201     }
202     fmt.Println("[Connected to new leader @", leaderAddr, "]\n")
203
204     // 7) Verify data is consistent
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
210     fmt.Println("\n== Test #3: GetStoreInfo on new leader (post-
failover) ==\n")
211     storeNewLeader := getStoreInfo(newLeaderClient)
212     fmt.Println(" NewLeader store info =>", storeNewLeader)
213
214     // 9) Additional sets/deletes on new leader
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})
223     fmt.Println(" GetKey(FailoverBucket,1) =>", postDel)
224
225     newLeaderClient.Close()
226     leaderClient.Close()
227
228     fmt.Println("\n=== End of All Tests ===")
229 }
230

```