```
1  package main
2
3  //
4  // toy RPC library
5  //
6
7  import "io"
8  import "fmt"
9  import "sync"
10 import "encoding/binary"
11
12 type ToyClient struct {
13         conn    io.ReadWriteCloser   // connection to server
14         xid     int64                // next unique request #
15         pending map[int64]chan int32 // waiting calls [xid]
16 }
17
18 func MakeToyClient(conn io.ReadWriteCloser) *ToyClient {
19         tc := &ToyClient{}
20         tc.conn = conn
21         tc.pending = map[int64]chan int32{}
22         tc.xid = 1
23         go tc.Listener()
24         return tc
25 }
26
27 func (tc *ToyClient) WriteRequest(xid int64, procNum int32, arg int32) {
28         binary.Write(tc.conn, binary.LittleEndian, xid)
29         binary.Write(tc.conn, binary.LittleEndian, procNum)
30         binary.Write(tc.conn, binary.LittleEndian, arg)
31 }
32
33 func (tc *ToyClient) ReadReply() (int64, int32) {
34         var xid int64
35         var arg int32
36         binary.Read(tc.conn, binary.LittleEndian, &xid)
37         binary.Read(tc.conn, binary.LittleEndian, &arg)
38         return xid, arg
39 }
40
41 //
42 // client application uses Call() to make an RPC.
43 // client := MakeClient(server)
44 // reply := client.Call(procNum, arg)
45 //
46 func (tc *ToyClient) Call(procNum int32, arg int32) int32 {
47         done := make(chan int32) // for tc.Listener()
48
49         xid := tc.xid // allocate a unique xid
50         tc.xid++
51         tc.pending[xid] = done             // for tc.Listener()
52         tc.WriteRequest(xid, procNum, arg) // send to server
53
54         reply := <-done // wait for reply via tc.Listener()
55
56         delete(tc.pending, xid)
57
58         return reply
59 }
60
61 //
62 // listen for replies from the server,
63 // send each reply to the right client Call() thread.
64 //
65 func (tc *ToyClient) Listener() {
66         for {
67                 xid, reply := tc.ReadReply()
68                 ch, ok := tc.pending[xid]
69                 if ok {
70                         ch <- reply
71                 }
72         }
73 }
```



```
74
75 type ToyServer struct {
76         conn     io.ReadWriteCloser          // connection from client
77         handlers map[int32]func(int32) int32 // procedures
78 }
79
80 func MakeToyServer(conn io.ReadWriteCloser) *ToyServer {
81         ts := &ToyServer{}
82         ts.conn = conn
83         ts.handlers = map[int32](func(int32) int32){}
84         go ts.Dispatcher()
85         return ts
86 }
87
88 func (ts *ToyServer) WriteReply(xid int64, arg int32) {
89         binary.Write(ts.conn, binary.LittleEndian, xid)
90         binary.Write(ts.conn, binary.LittleEndian, arg)
91 }
92
93 func (ts *ToyServer) ReadRequest() (int64, int32, int32) {
94         var xid int64
95         var procNum int32
96         var arg int32
97         binary.Read(ts.conn, binary.LittleEndian, &xid)
98         binary.Read(ts.conn, binary.LittleEndian, &procNum)
99         binary.Read(ts.conn, binary.LittleEndian, &arg)
100        return xid, procNum, arg
101 }
102
103 //
104 // listen for client requests,
105 // dispatch each to the right handler function,
106 // send back reply.
107 //
108 func (ts *ToyServer) Dispatcher() {
109         for {
110                 xid, procNum, arg := ts.ReadRequest()
111                 fn, ok := ts.handlers[procNum]
112                 go func() {
113                         var reply int32
114                         if ok {
115                                 reply = fn(arg)
116                         }
117                         ts.WriteReply(xid, reply)
118                 }()
119         }
120 }
121
122 type Pair struct {
123         r *io.PipeReader
124         w *io.PipeWriter
125 }
126
127 func (p Pair) Read(data []byte) (int, error) {
128         return p.r.Read(data)
129 }
130 func (p Pair) Write(data []byte) (int, error) {
131         return p.w.Write(data)
132 }
133 func (p Pair) Close() error {
134         p.r.Close()
135         return p.w.Close()
136 }
137
138 func main() {
139         r1, w1 := io.Pipe()
140         r2, w2 := io.Pipe()
141         cp := Pair{r: r1, w: w2}
142         sp := Pair{r: r2, w: w1}
143         tc := MakeToyClient(cp)
144         ts := MakeToyServer(sp)
145         ts.handlers[22] = func(a int32) int32 { return a + 1 }
146
```

```
147         reply := tc.Call(22, 100)
148         fmt.Printf("Call(22, 100) -> %v\n", reply)
149 }
```