```
package main

//
// toy RPC library
//

import "io"
import "fmt"
import "sync"
import "encoding/binary"

type ToyClient struct {
  mu sync.Mutex
  conn io.ReadWriteCloser      // connection to server
  xid int64                    // next unique request #
  pending map[int64]chan int32 // waiting calls [xid]
}

func MakeToyClient(conn io.ReadWriteCloser) *ToyClient {
  tc := &ToyClient{}
  tc.conn = conn
  tc.pending = map[int64]chan int32{}
  tc.xid = 1
  go tc.Listener()
  return tc
}

func (tc *ToyClient) WriteRequest(xid int64, procNum int32, arg int32) {
  binary.Write(tc.conn, binary.LittleEndian, xid)
  binary.Write(tc.conn, binary.LittleEndian, procNum)
  binary.Write(tc.conn, binary.LittleEndian, arg)
}

func (tc *ToyClient) ReadReply() (int64, int32) {
  var xid int64
  var arg int32
  binary.Read(tc.conn, binary.LittleEndian, &xid)
  binary.Read(tc.conn, binary.LittleEndian, &arg)
  return xid, arg
}

//
// client application uses Call() to make an RPC.
// client := MakeClient(server)
// reply := client.Call(procNum, arg)
//
func (tc *ToyClient) Call(procNum int32, arg int32) int32 {
  done := make(chan int32) // for tc.Listener()

  tc.mu.Lock()
  xid := tc.xid  // allocate a unique xid
  tc.xid++
  tc.pending[xid] = done  // for tc.Listener()
  tc.WriteRequest(xid, procNum, arg)  // send to server
  tc.mu.Unlock()

  reply := <- done  // wait for reply via tc.Listener()

  tc.mu.Lock()
  delete(tc.pending, xid)
  tc.mu.Unlock()

  return reply
}

//
// listen for replies from the server,
// send each reply to the right client Call() thread.
//
func (tc *ToyClient) Listener() {
  for {
    xid, reply := tc.ReadReply()
    tc.mu.Lock()
    ch, ok := tc.pending[xid]
    tc.mu.Unlock()
    if ok {
      ch <- reply
    }
  }
}

type ToyServer struct {
  mu sync.Mutex
  conn io.ReadWriteCloser  // connection from client
  handlers map[int32]func(int32)int32  // procedures
}

func MakeToyServer(conn io.ReadWriteCloser) *ToyServer {
  ts := &ToyServer{}
  ts.conn = conn
  ts.handlers = map[int32](func(int32)int32){}
  go ts.Dispatcher()
  return ts
}

func (ts *ToyServer) WriteReply(xid int64, arg int32) {
  binary.Write(ts.conn, binary.LittleEndian, xid)
  binary.Write(ts.conn, binary.LittleEndian, arg)
}

func (ts *ToyServer) ReadRequest() (int64, int32, int32) {
  var xid int64
  var procNum int32
  var arg int32
  binary.Read(ts.conn, binary.LittleEndian, &xid)
  binary.Read(ts.conn, binary.LittleEndian, &procNum)
  binary.Read(ts.conn, binary.LittleEndian, &arg)
  return xid, procNum, arg
}

//
// listen for client requests,
// dispatch each to the right handler function,
// send back reply.
//
func (ts *ToyServer) Dispatcher() {
  for {
    xid, procNum, arg := ts.ReadRequest()
    ts.mu.Lock()
    fn, ok := ts.handlers[procNum]
    ts.mu.Unlock()
    go func() {
      var reply int32
      if ok {
        reply = fn(arg)
      }
      ts.mu.Lock()
      ts.WriteReply(xid, reply)
      ts.mu.Unlock()
    }()
  }
}

type Pair struct {
  r *io.PipeReader
  w *io.PipeWriter
}
func (p Pair) Read(data []byte) (int, error) {
  return p.r.Read(data)
}
func (p Pair) Write(data []byte) (int, error) {
  return p.w.Write(data)
}
func (p Pair) Close() error {
  p.r.Close()
  return p.w.Close()
}

func main() {
  r1, w1 := io.Pipe()
  r2, w2 := io.Pipe()
  cp := Pair{r : r1, w : w2}
  sp := Pair{r : r2, w : w1}
  tc := MakeToyClient(cp)
  ts := MakeToyServer(sp)
  ts.handlers[22] = func(a int32) int32 { return a+1 }

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