```
//
// *** a simple application using the lock service
//

func main() {
  primary_port := os.Args[1]
  backup_port := os.Args[2]
  clerk := lockservice.MakeClerk(primary_port,
                                 backup_port)

  for clerk.Lock("car keys") == false {
    // wait
  }

  // it's my turn to drive the car...

  clerk.Unlock("car keys")
}

```

```
//
// *** client.go -- the application calls
// these library "stubs"
//

type Clerk struct {
  servers [2]string // primary port, backup port
}

func MakeClerk(primary string, backup string) *Clerk {
  ck := new(Clerk)
  ck.servers[0] = primary
  ck.servers[1] = backup
  return ck
}

//
// ask the lock service for a lock.
// returns true if the lock service
// granted the lock, false otherwise.
//
func (ck *Clerk) Lock(lockname string) bool {
  args := &LockArgs{}        // RPC arguments
  args.Lockname = lockname
  var reply LockReply        // space for RPC reply

  // send an RPC request, wait for the reply.
  ok := call(ck.servers[0], "LockServer.Lock",
             args, &reply)
  return ok && reply.OK
}

```

```go
//
// *** server.go
//

//
// a lock server’s state
//
type LockServer struct {
  mu sync.Mutex
  l net.Listener

  am_primary bool // am I the primary?
  backup string   // backup’s port

  // for each lock name, is it locked?
  locks map[string]bool
}


//
// server Lock() RPC handler
//
func (ls *LockServer) Lock(args *LockArgs,
                           reply *LockReply) error {
  ls.mu.Lock()
  defer ls.mu.Unlock()

  locked, _ := ls.locks[args.Lockname]

  if locked {
    reply.OK = false
  } else {
    reply.OK = true
    ls.locks[args.Lockname] = true
  }

  return nil
}

```

```go
//
// start a lock server
//
func StartServer(primary string, backup string,
                 am_primary bool) *LockServer {
  ls := new(LockServer)
  ls.backup = backup
  ls.am_primary = am_primary
  ls.locks = map[string]bool{}

  // tell net/rpc about our RPC server and handlers.
  rpcs := rpc.NewServer()
  rpcs.Register(ls)

  my_port := ""
  if am_primary {
    my_port = primary
  } else {
    my_port = backup
  }

  // prepare to receive connections from clients.
  ls.l, _ := net.Listen("unix", my_port);

  // thread to accept RPC connections from clients.
  go func() {
    for {
      conn, _ := ls.l.Accept()
      go rpcs.ServeConn(conn)
    }
  }()

  return ls
}
```