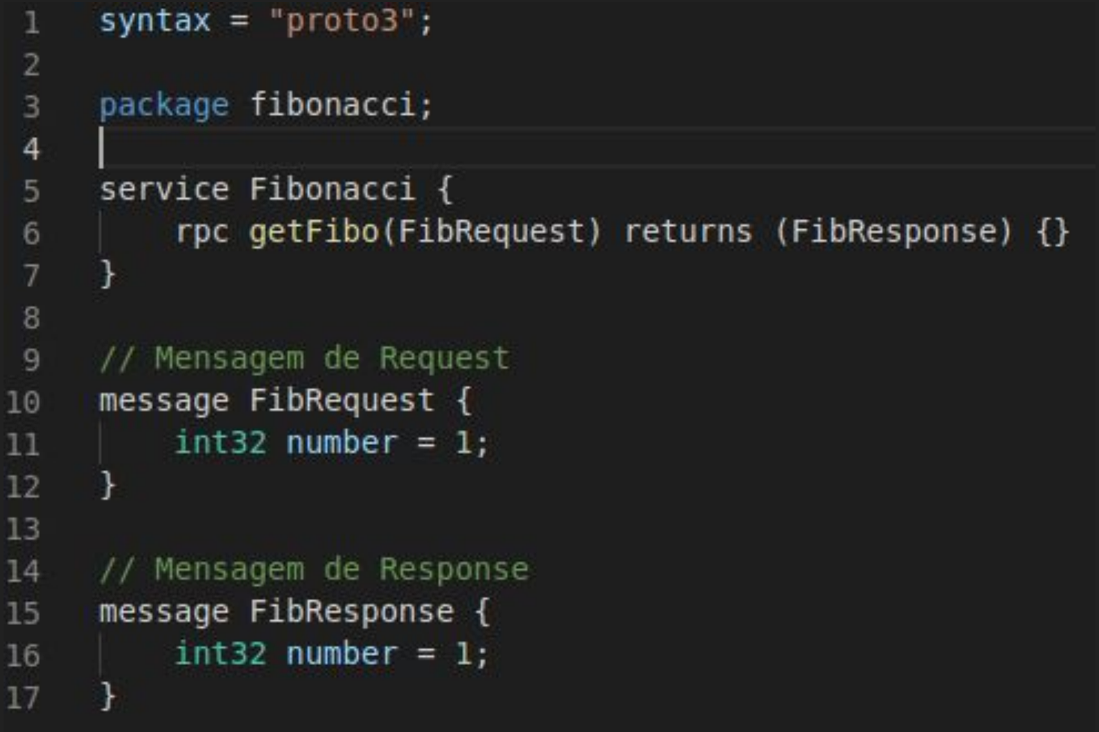
Exercício 3
MOO e gRPC

Aplicação e Avaliação de Desempenho

Implementação gRPC

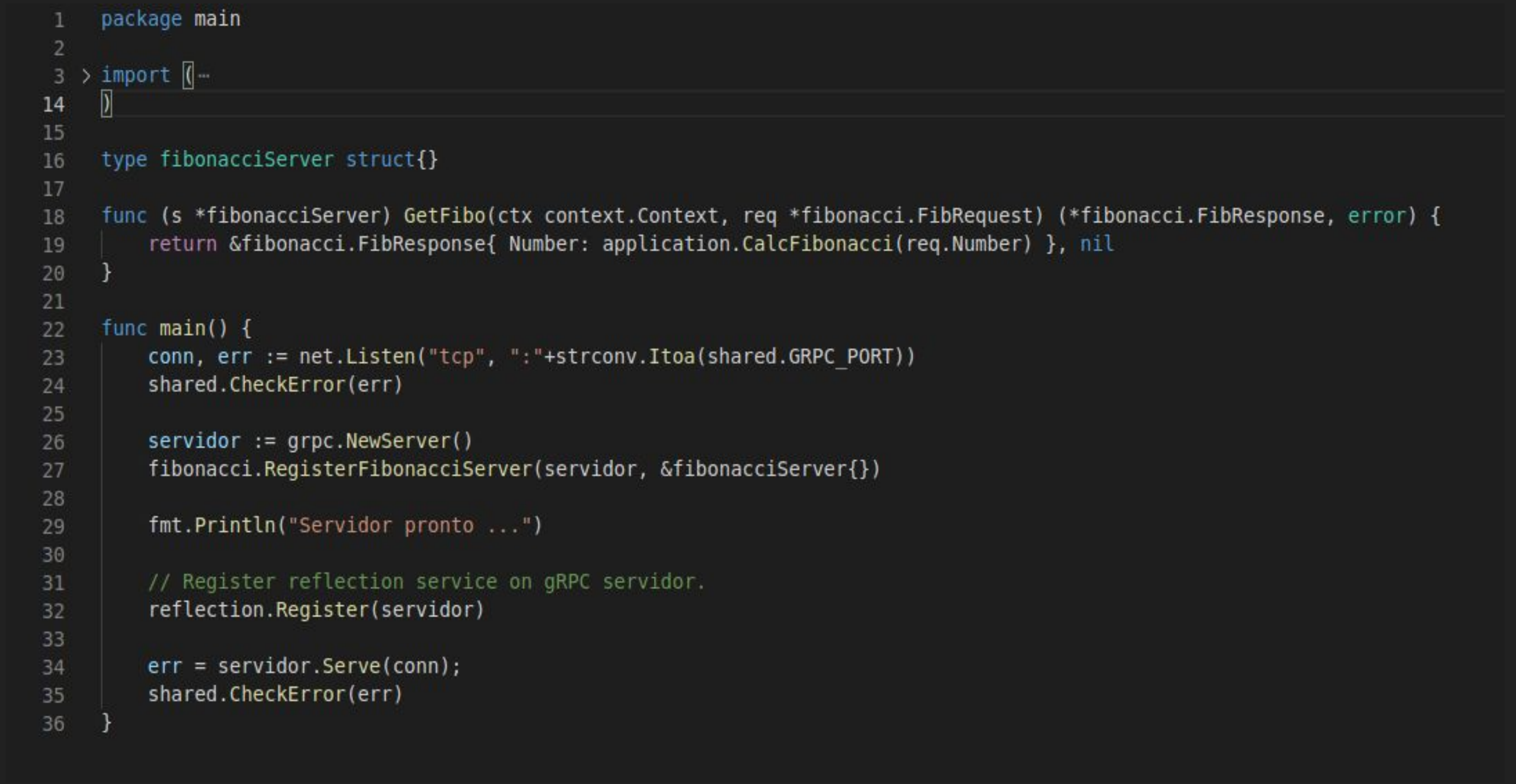
Código - Serviço

- ProtoBuffs e Plugins em Go



```
syntax = "proto3";

package fibonacci;

service Fibonacci {
    rpc getFibo(FibRequest) returns (FibResponse) {}
}

// Mensagem de Request
message FibRequest {
    int32 number = 1;
}

// Mensagem de Response
message FibResponse {
    int32 number = 1;
}
```

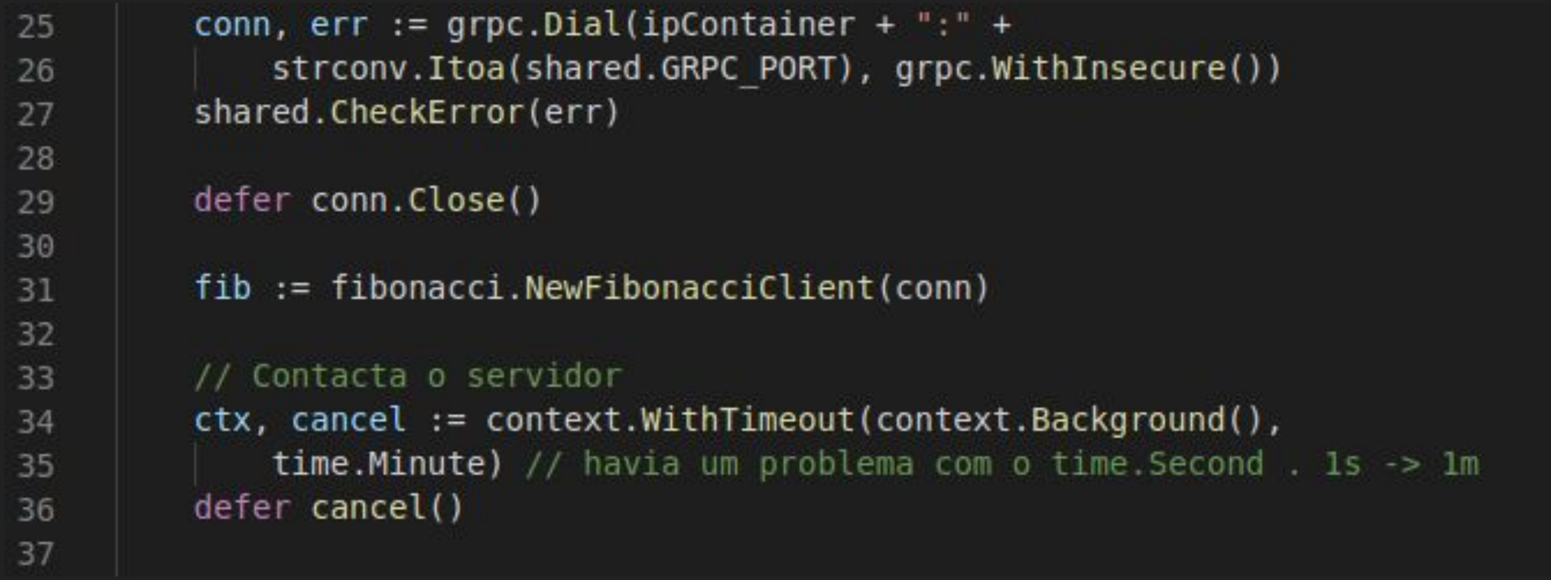
Código - Servidor gRPC



```go
package main

import (…
)

type fibonacciServer struct{}

func (s *fibonacciServer) GetFibo(ctx context.Context, req *fibonacci.FibRequest) (*fibonacci.FibResponse, error) {
    return &fibonacci.FibResponse{ Number: application.CalcFibonacci(req.Number) }, nil
}

func main() {
    conn, err := net.Listen("tcp", ":"+strconv.Itoa(shared.GRPC_PORT))
    shared.CheckError(err)

    servidor := grpc.NewServer()
    fibonacci.RegisterFibonacciServer(servidor, &fibonacciServer{})

    fmt.Println("Servidor pronto ...")

    // Register reflection service on gRPC servidor.
    reflection.Register(servidor)

    err = servidor.Serve(conn);
    shared.CheckError(err)
}
```

Código - Cliente gRPC

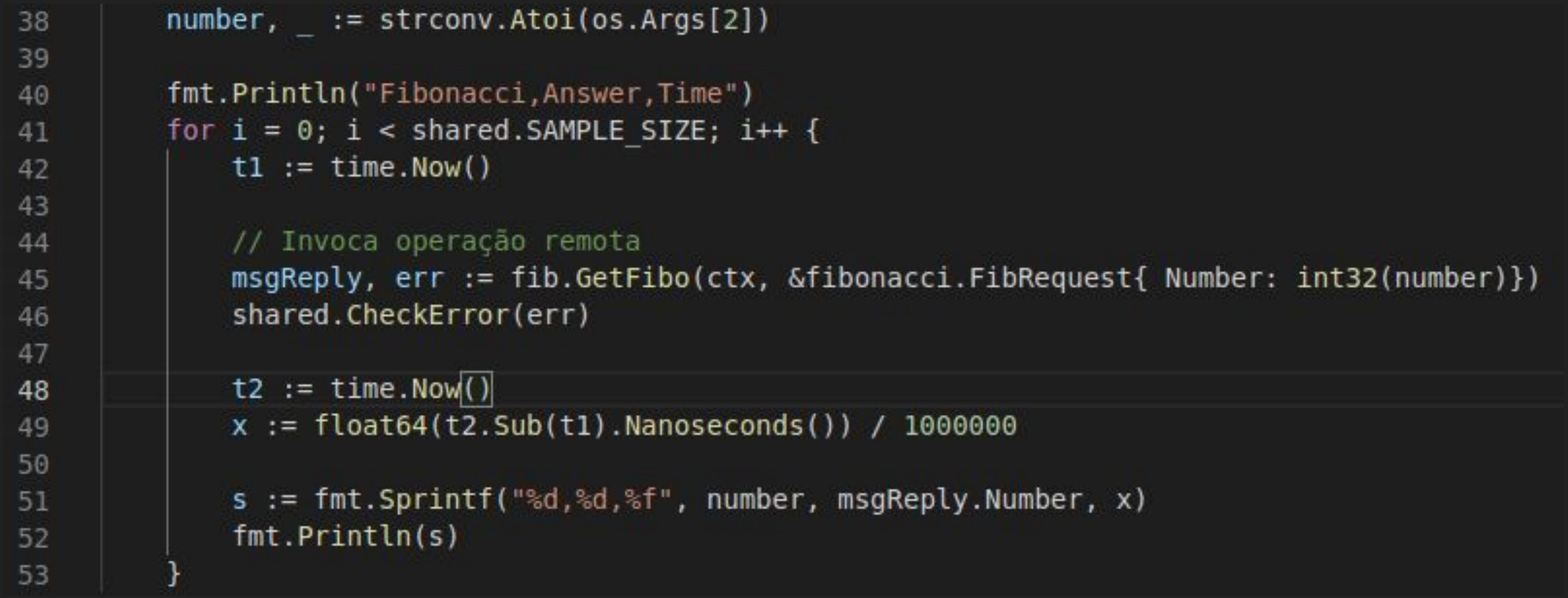
- Estabelecendo conexão
- Criando contexto com informações sobre Request



```go
	conn, err := grpc.Dial(ipContainer + ":" +
		strconv.Itoa(shared.GRPC_PORT), grpc.WithInsecure())
	shared.CheckError(err)

	defer conn.Close()

	fib := fibonacci.NewFibonacciClient(conn)

	// Contacta o servidor
	ctx, cancel := context.WithTimeout(context.Background(),
		time.Minute) // havia um problema com o time.Second . 1s -> 1m
	defer cancel()

```

Código - Cliente gRPC

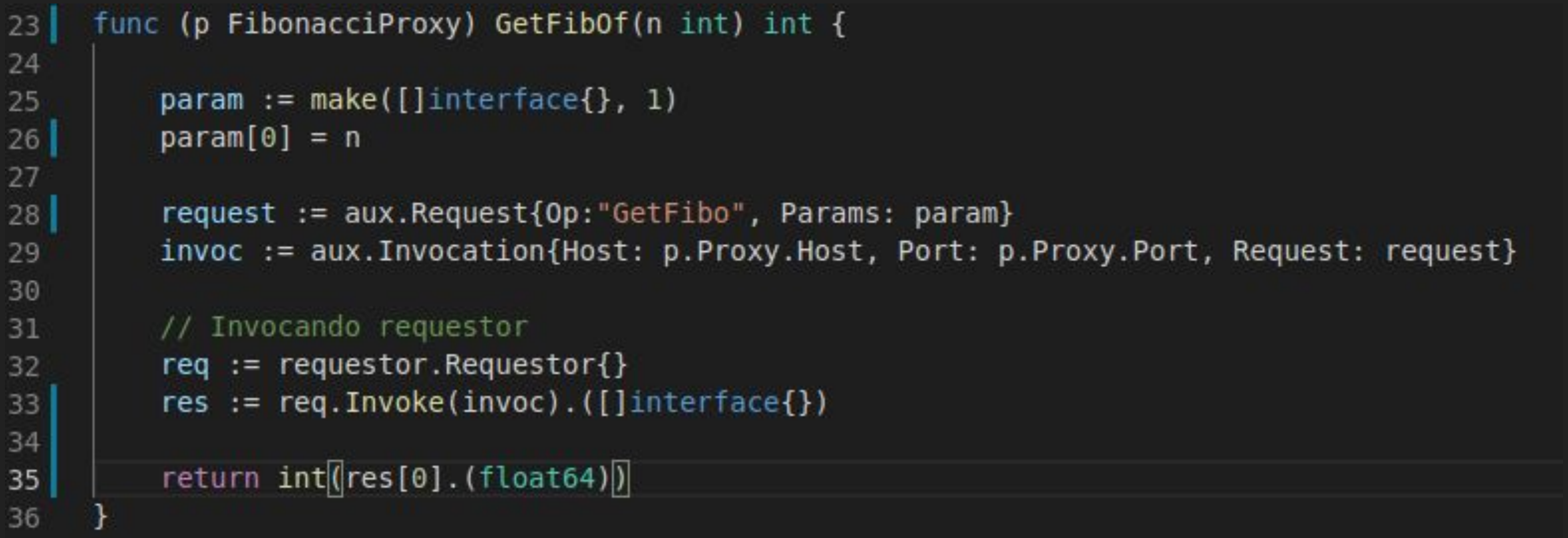
- Fazendo requisições
- Invocando operação remota



```go
    number, _ := strconv.Atoi(os.Args[2])

    fmt.Println("Fibonacci,Answer,Time")
    for i = 0; i < shared.SAMPLE_SIZE; i++ {
        t1 := time.Now()

        // Invoca operação remota
        msgReply, err := fib.GetFibo(ctx, &fibonacci.FibRequest{ Number: int32(number)})
        shared.CheckError(err)

        t2 := time.Now()
        x := float64(t2.Sub(t1).Nanoseconds()) / 1000000

        s := fmt.Sprintf("%d,%d,%f", number, msgReply.Number, x)
        fmt.Println(s)
    }
```

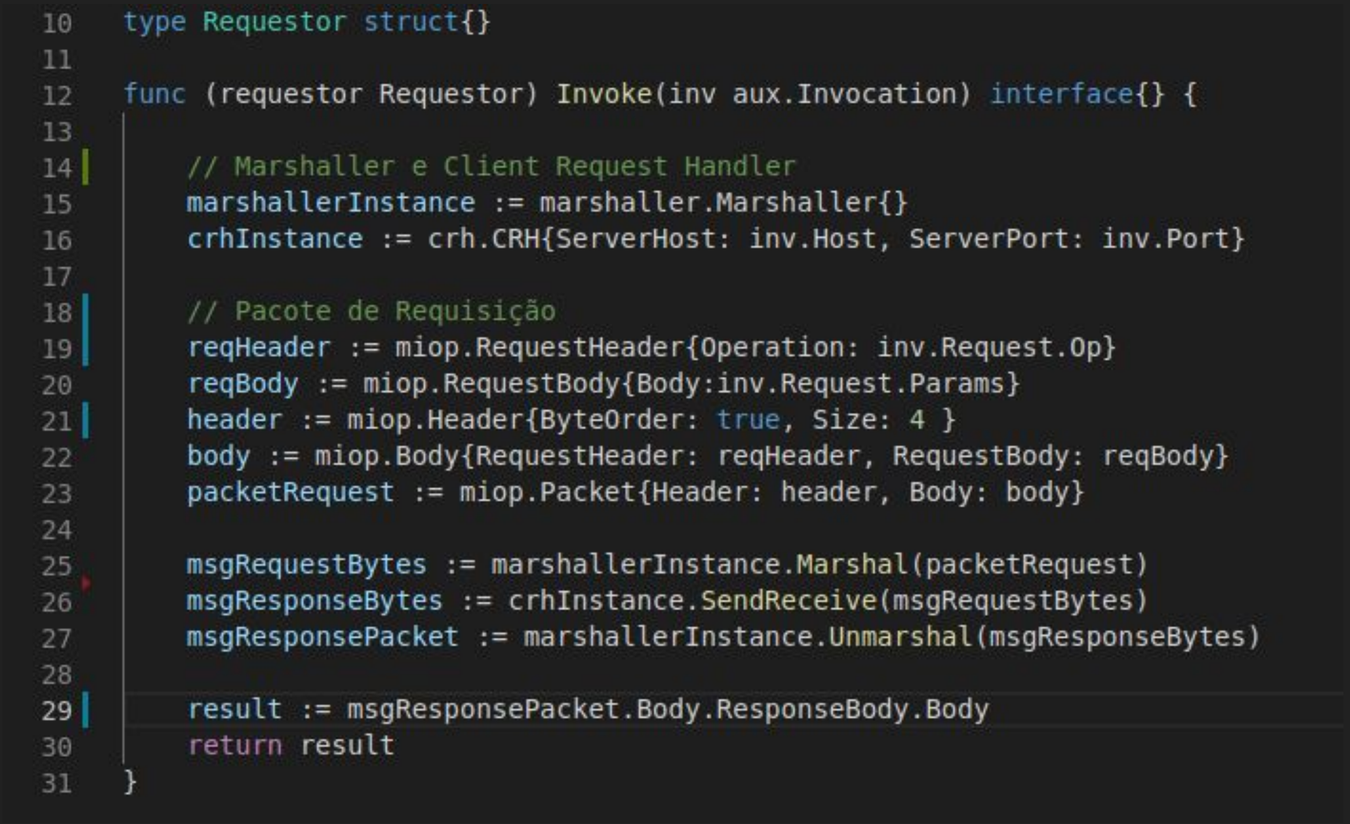
Implementação MOO

Código - ClientProxy



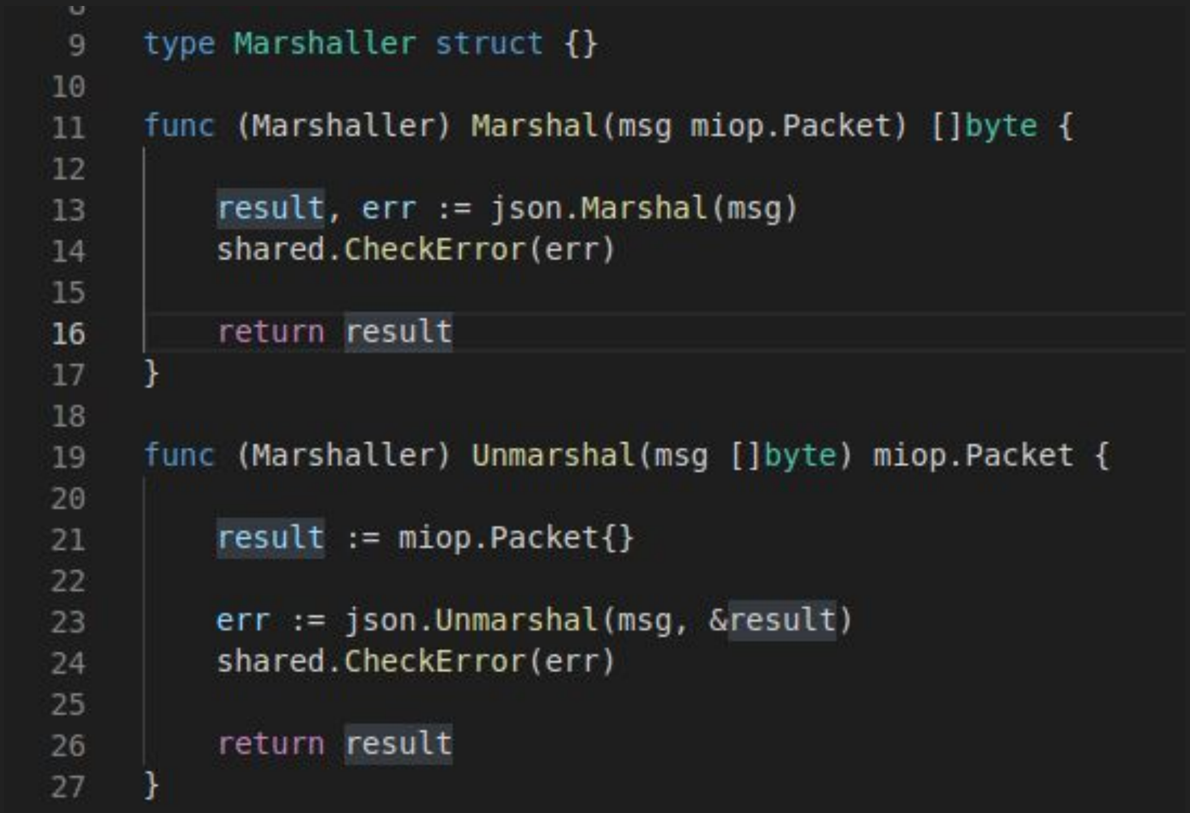
```go
func (p FibonacciProxy) GetFibOf(n int) int {

    param := make([]interface{}, 1)
    param[0] = n

    request := aux.Request{Op:"GetFibo", Params: param}
    invoc := aux.Invocation{Host: p.Proxy.Host, Port: p.Proxy.Port, Request: request}

    // Invocando requestor
    req := requestor.Requestor{}
    res := req.Invoke(invoc).([]interface{})

    return int(res[0].(float64))
}
```

Código - Requestor MOO



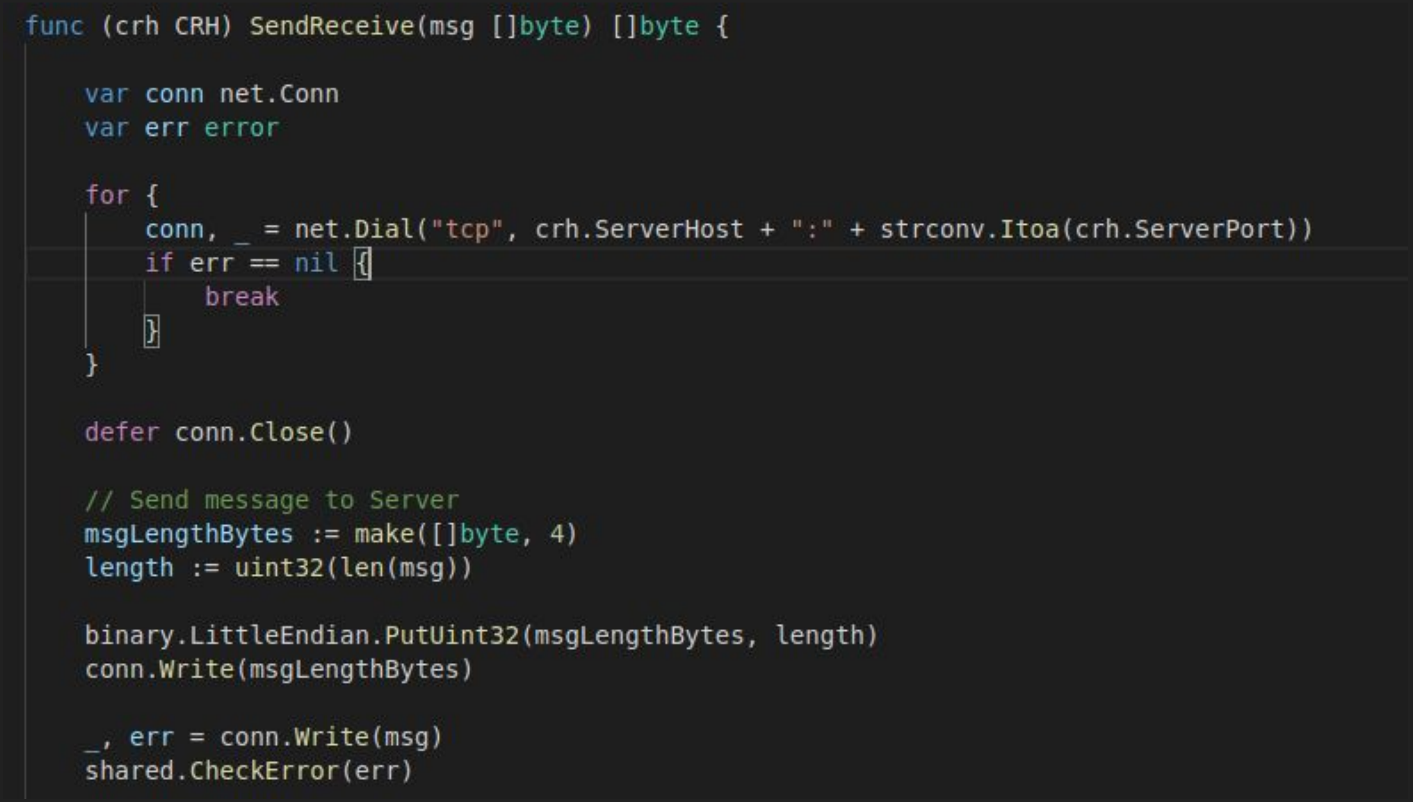
```go
type Requestor struct{}

func (requestor Requestor) Invoke(inv aux.Invocation) interface{} {

    // Marshaller e Client Request Handler
    marshallerInstance := marshaller.Marshaller{}
    crhInstance := crh.CRH{ServerHost: inv.Host, ServerPort: inv.Port}

    // Pacote de Requisição
    reqHeader := miop.RequestHeader{Operation: inv.Request.Op}
    reqBody := miop.RequestBody{Body:inv.Request.Params}
    header := miop.Header{ByteOrder: true, Size: 4 }
    body := miop.Body{RequestHeader: reqHeader, RequestBody: reqBody}
    packetRequest := miop.Packet{Header: header, Body: body}

    msgRequestBytes := marshallerInstance.Marshal(packetRequest)
    msgResponseBytes := crhInstance.SendReceive(msgRequestBytes)
    msgResponsePacket := marshallerInstance.Unmarshal(msgResponseBytes)

    result := msgResponsePacket.Body.ResponseBody.Body
    return result
}
```

Código - Marshaller MOO



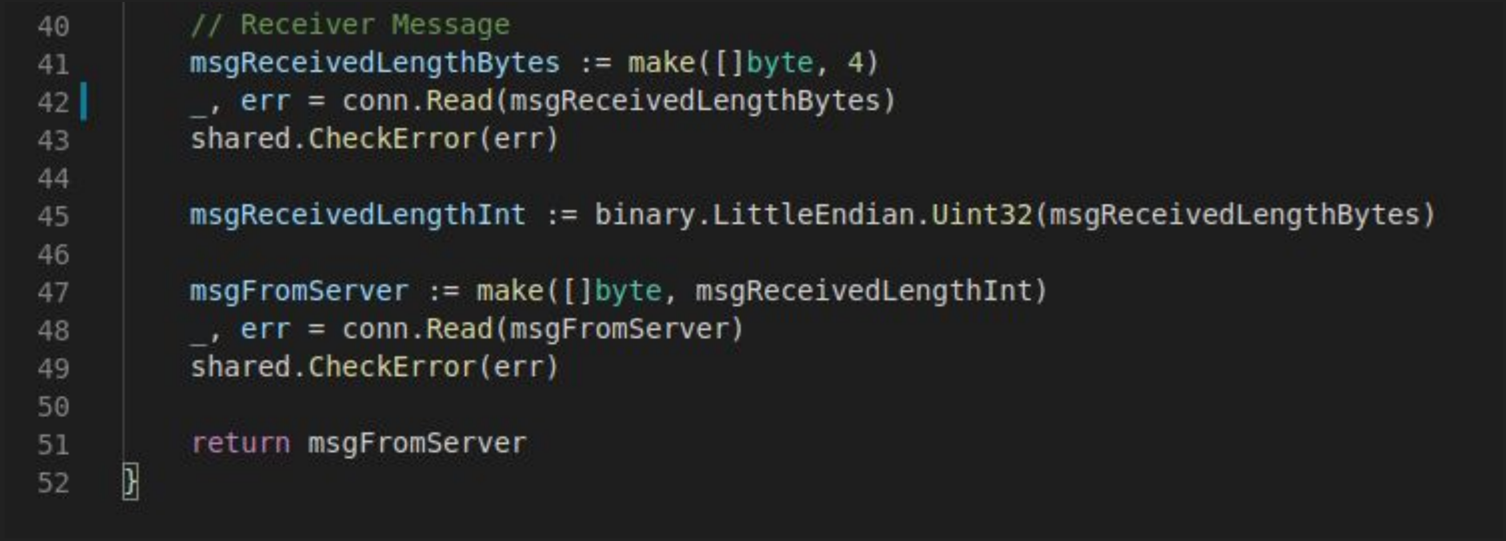
```go
type Marshaller struct {}

func (Marshaller) Marshal(msg miop.Packet) []byte {

    result, err := json.Marshal(msg)
    shared.CheckError(err)

    return result
}

func (Marshaller) Unmarshal(msg []byte) miop.Packet {

    result := miop.Packet{}

    err := json.Unmarshal(msg, &result)
    shared.CheckError(err)

    return result
}
```

Código - Client Request Handler MOO 1/2



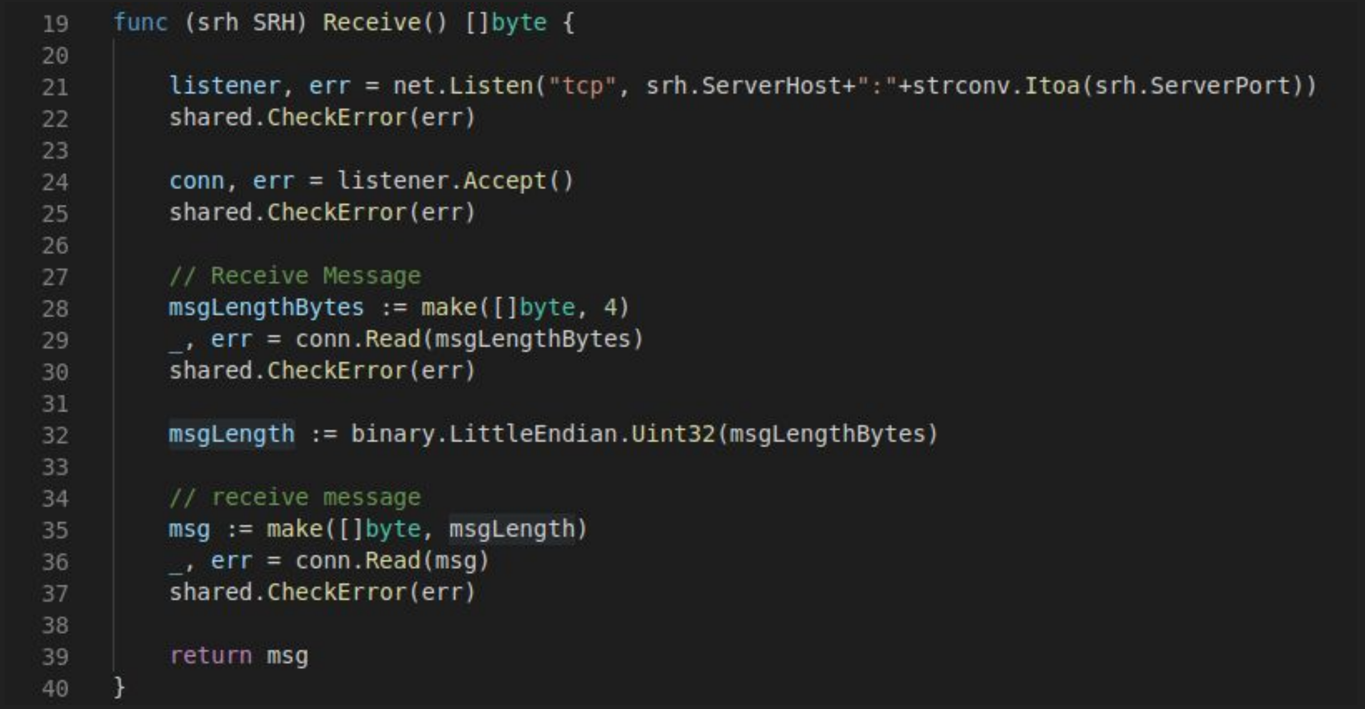
```go
func (crh CRH) SendReceive(msg []byte) []byte {

    var conn net.Conn
    var err error

    for {
        conn, _ = net.Dial("tcp", crh.ServerHost + ":" + strconv.Itoa(crh.ServerPort))
        if err == nil {
            break
        }
    }

    defer conn.Close()

    // Send message to Server
    msgLengthBytes := make([]byte, 4)
    length := uint32(len(msg))

    binary.LittleEndian.PutUint32(msgLengthBytes, length)
    conn.Write(msgLengthBytes)

    _, err = conn.Write(msg)
    shared.CheckError(err)
```

Código - Client Request Handler MOO 2/2



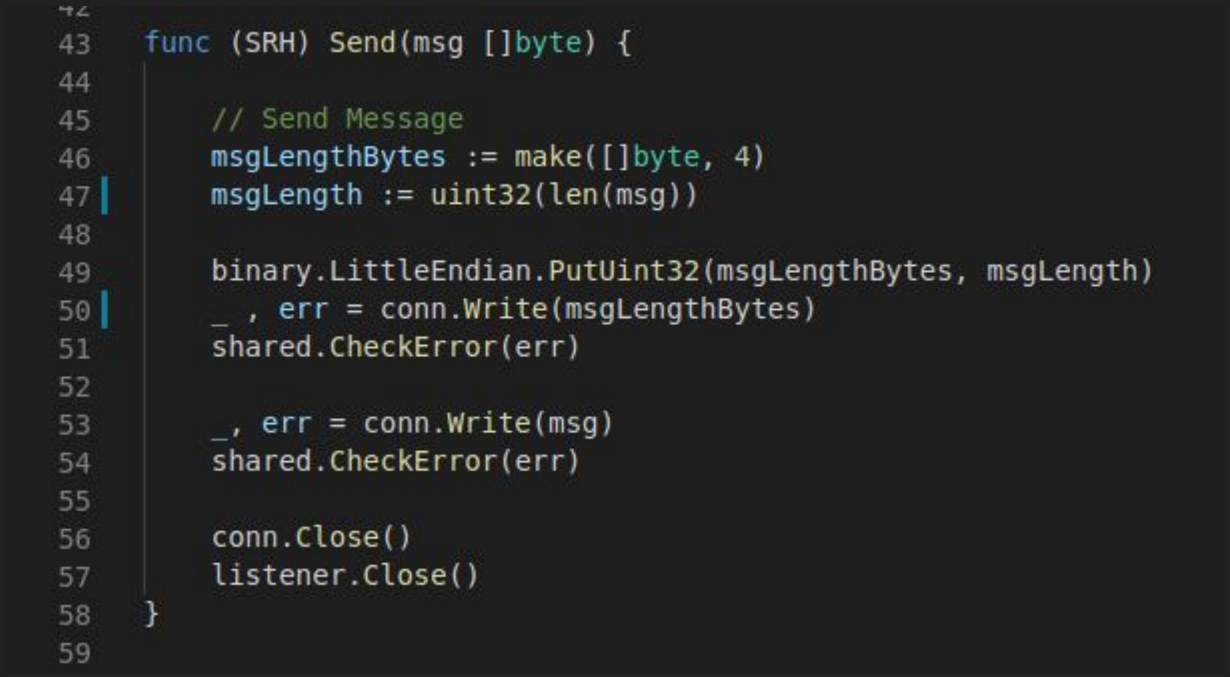
```
    // Receiver Message
    msgReceivedLengthBytes := make([]byte, 4)
    _, err = conn.Read(msgReceivedLengthBytes)
    shared.CheckError(err)

    msgReceivedLengthInt := binary.LittleEndian.Uint32(msgReceivedLengthBytes)

    msgFromServer := make([]byte, msgReceivedLengthInt)
    _, err = conn.Read(msgFromServer)
    shared.CheckError(err)

    return msgFromServer
}
```

Código - Server Request Handler MOO 1/2



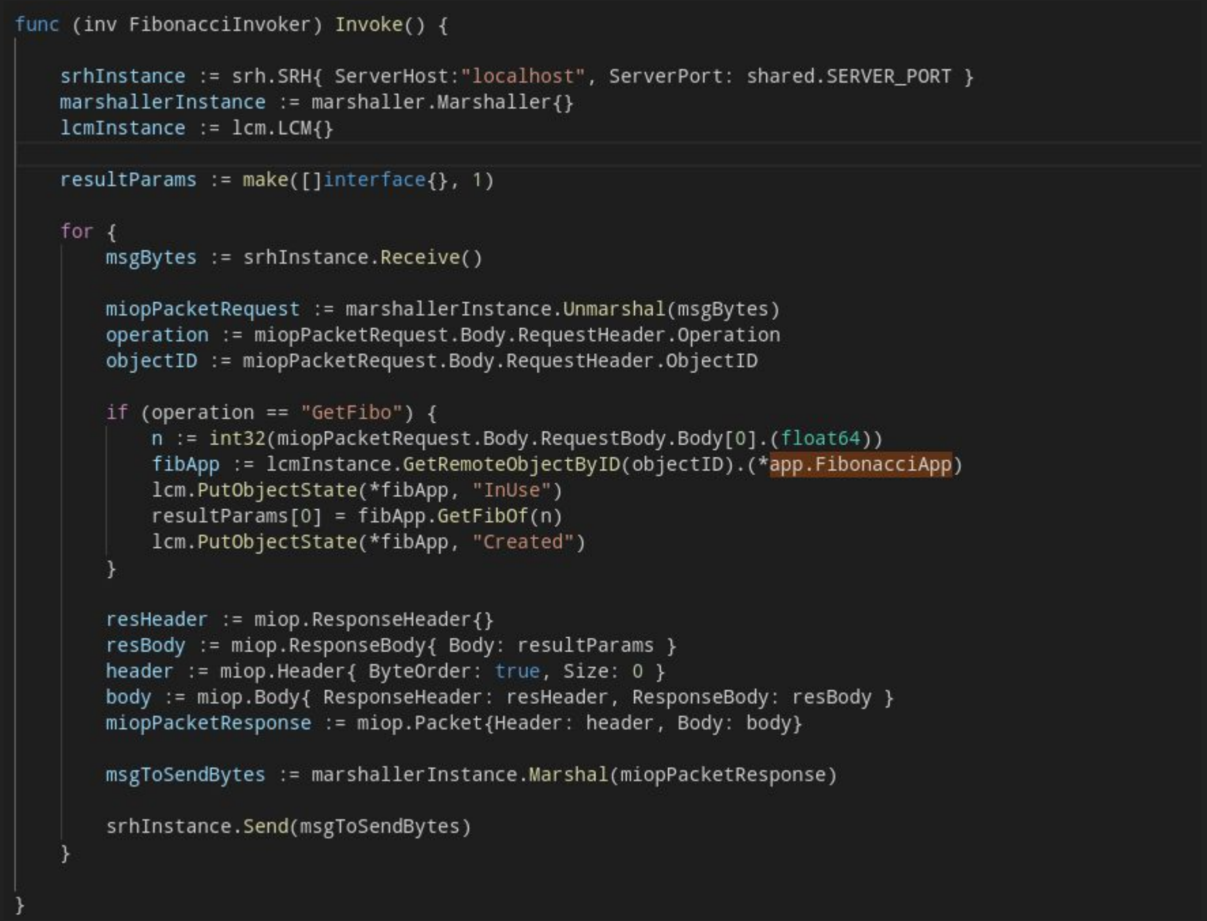
```go
func (srh SRH) Receive() []byte {

	listener, err = net.Listen("tcp", srh.ServerHost+":"+strconv.Itoa(srh.ServerPort))
	shared.CheckError(err)

	conn, err = listener.Accept()
	shared.CheckError(err)

	// Receive Message
	msgLengthBytes := make([]byte, 4)
	_, err = conn.Read(msgLengthBytes)
	shared.CheckError(err)

	msgLength := binary.LittleEndian.Uint32(msgLengthBytes)

	// receive message
	msg := make([]byte, msgLength)
	_, err = conn.Read(msg)
	shared.CheckError(err)

	return msg
}
```

Código - Server Request Handler MOO 2/2



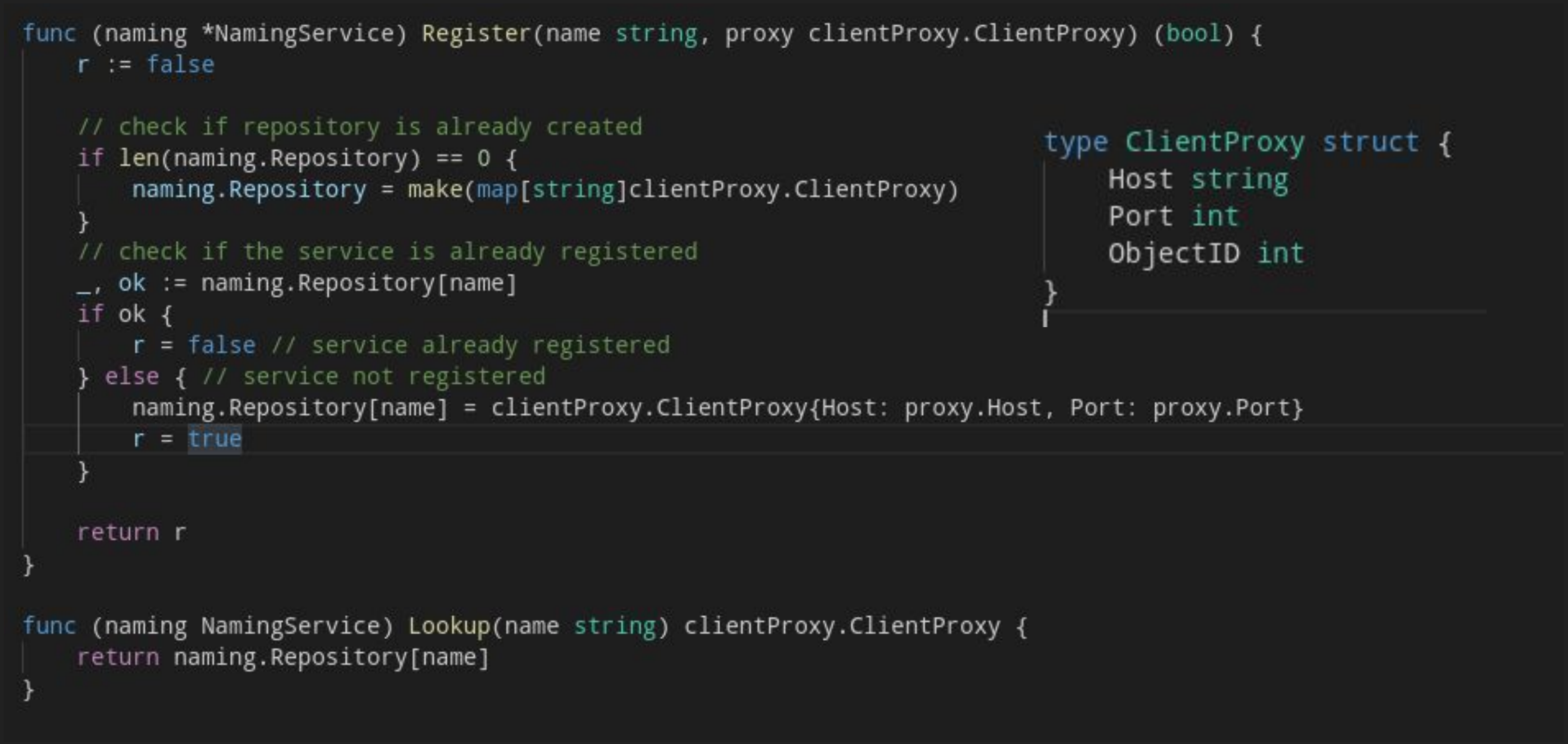
```go
func (SRH) Send(msg []byte) {

    // Send Message
    msgLengthBytes := make([]byte, 4)
    msgLength := uint32(len(msg))

    binary.LittleEndian.PutUint32(msgLengthBytes, msgLength)
    _ , err = conn.Write(msgLengthBytes)
    shared.CheckError(err)

    _, err = conn.Write(msg)
    shared.CheckError(err)

    conn.Close()
    listener.Close()
}
```

Código - Invoker MOO



```go
func (inv FibonacciInvoker) Invoke() {

    srhInstance := srh.SRH{ ServerHost:"localhost", ServerPort: shared.SERVER_PORT }
    marshallerInstance := marshaller.Marshaller{}
    lcmInstance := lcm.LCM{}

    resultParams := make([]interface{}, 1)

    for {
        msgBytes := srhInstance.Receive()

        miopPacketRequest := marshallerInstance.Unmarshal(msgBytes)
        operation := miopPacketRequest.Body.RequestHeader.Operation
        objectID := miopPacketRequest.Body.RequestHeader.ObjectID

        if (operation == "GetFibo") {
            n := int32(miopPacketRequest.Body.RequestBody.Body[0].(float64))
            fibApp := lcmInstance.GetRemoteObjectByID(objectID).(*app.FibonacciApp)
            lcm.PutObjectState(*fibApp, "InUse")
            resultParams[0] = fibApp.GetFibOf(n)
            lcm.PutObjectState(*fibApp, "Created")
        }

        resHeader := miop.ResponseHeader{}
        resBody := miop.ResponseBody{ Body: resultParams }
        header := miop.Header{ ByteOrder: true, Size: 0 }
        body := miop.Body{ ResponseHeader: resHeader, ResponseBody: resBody }
        miopPacketResponse := miop.Packet{Header: header, Body: body}

        msgToSendBytes := marshallerInstance.Marshal(miopPacketResponse)

        srhInstance.Send(msgToSendBytes)
    }

}
```

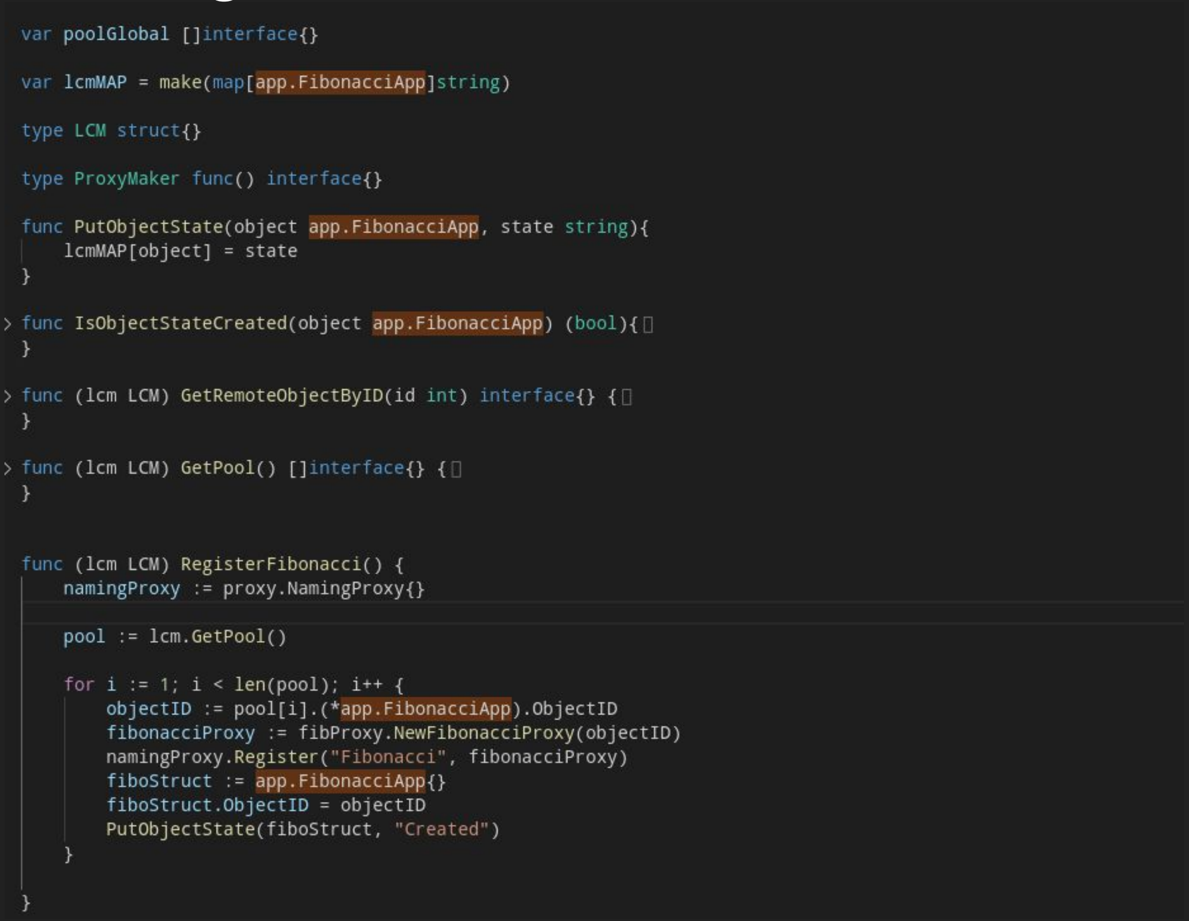
Lookup + AOR



```go
func (naming *NamingService) Register(name string, proxy clientProxy.ClientProxy) (bool) {
    r := false

    // check if repository is already created
    if len(naming.Repository) == 0 {
        naming.Repository = make(map[string]clientProxy.ClientProxy)
    }
    // check if the service is already registered
    _, ok := naming.Repository[name]
    if ok {
        r = false // service already registered
    } else { // service not registered
        naming.Repository[name] = clientProxy.ClientProxy{Host: proxy.Host, Port: proxy.Port}
        r = true
    }

    return r
}

func (naming NamingService) Lookup(name string) clientProxy.ClientProxy {
    return naming.Repository[name]
}
```

```go
type ClientProxy struct {
    Host string
    Port int
    ObjectID int
}
```

LCM + Pooling



```
var poolGlobal []interface{}

var lcmMAP = make(map[app.FibonacciApp]string)

type LCM struct{}

type ProxyMaker func() interface{}

func PutObjectState(object app.FibonacciApp, state string){
    lcmMAP[object] = state
}

> func IsObjectStateCreated(object app.FibonacciApp) (bool){
}

> func (lcm LCM) GetRemoteObjectByID(id int) interface{} {
}

> func (lcm LCM) GetPool() []interface{} {
}


func (lcm LCM) RegisterFibonacci() {
    namingProxy := proxy.NamingProxy{}

    pool := lcm.GetPool()

    for i := 1; i < len(pool); i++ {
        objectID := pool[i].(*app.FibonacciApp).ObjectID
        fibonacciProxy := fibProxy.NewFibonacciProxy(objectID)
        namingProxy.Register("Fibonacci", fibonacciProxy)
        fiboStruct := app.FibonacciApp{}
        fiboStruct.ObjectID = objectID
        PutObjectState(fiboStruct, "Created")
    }

}
```

Avaliação de Desempenho

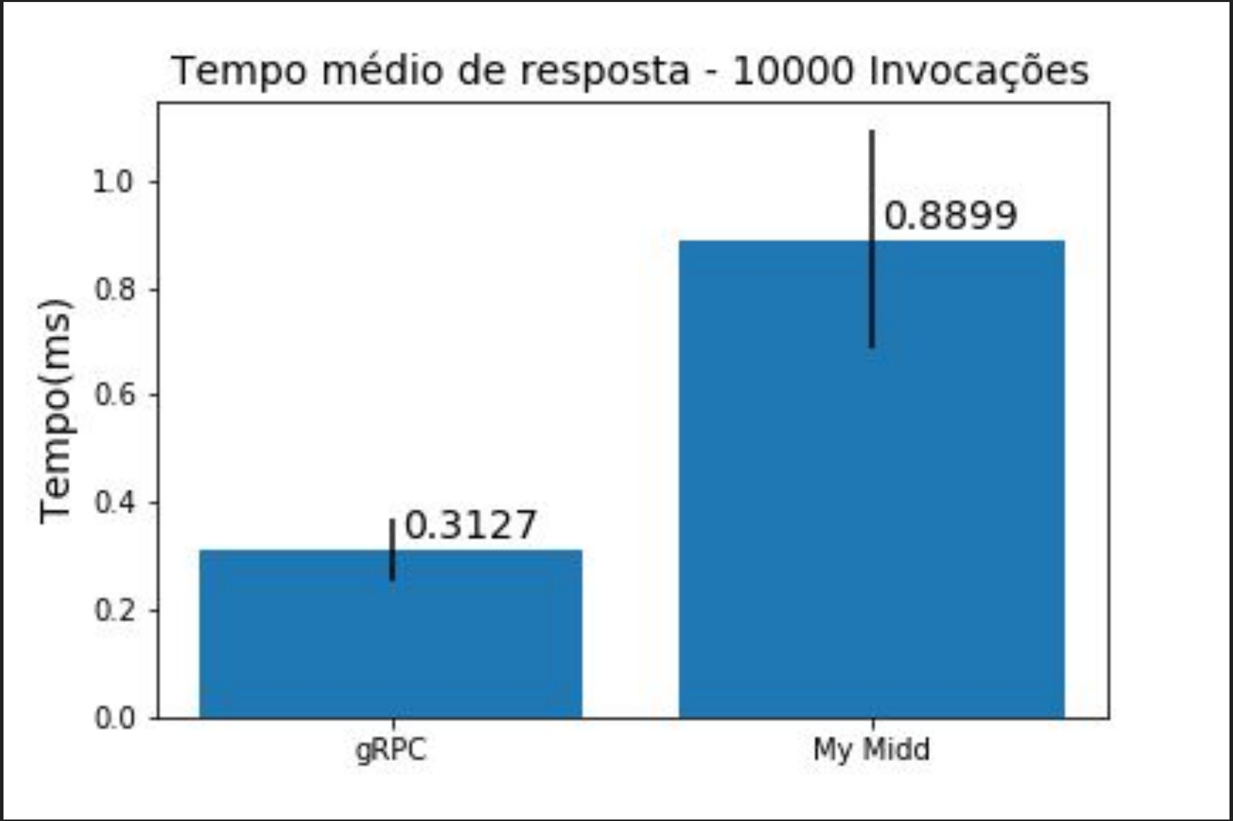
Máquina:

- Memória: 7,7 GiB
- Desktop
- OS: Manjaro xfce
- Processador: Intel® Core™ i7-3770 CPU @ 3.40GHz × 8

Preparação de Ambiente

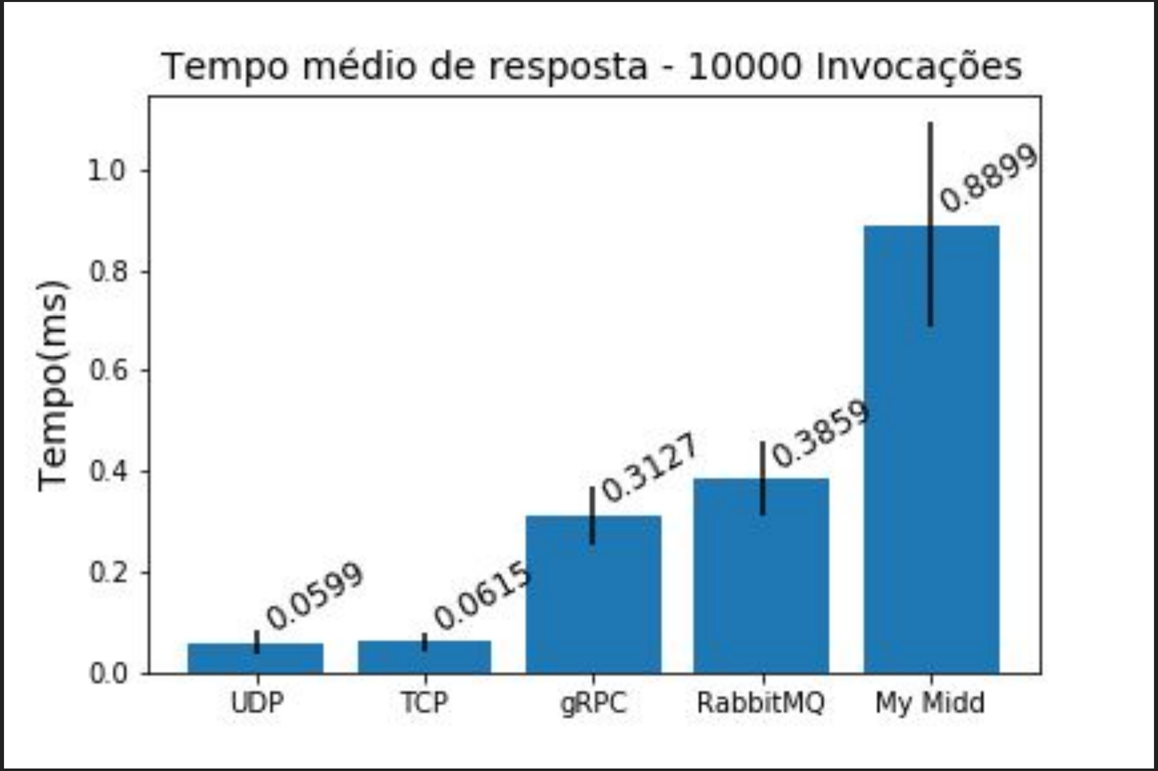
- Máquina recém inicializada
- Experimento realizado 30 vezes (Com e sem Warmup)

Resultados



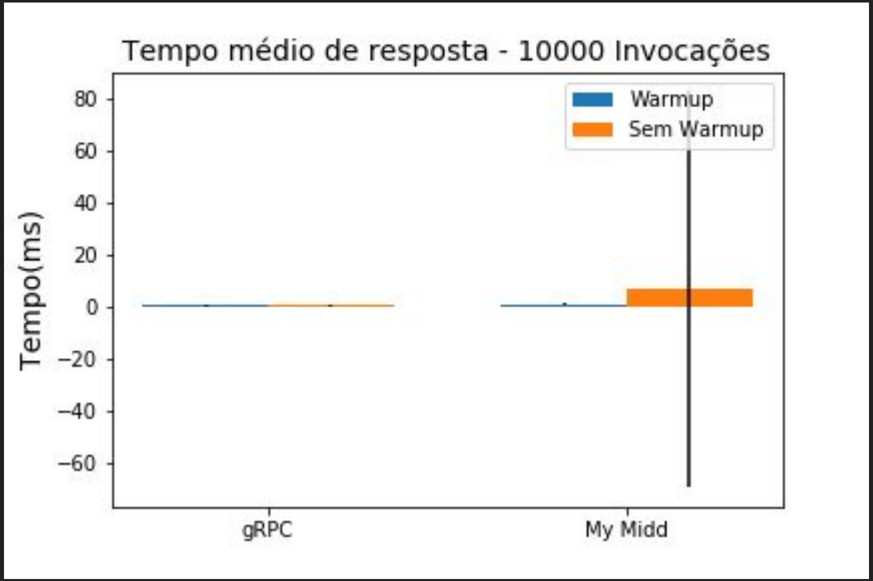
Tempo médio de resposta - 10000 Invocações
Tempo(ms)
0.0
0.2
0.4
0.6
0.8
1.0
0.3127
0.8899
gRPC
My Midd

Resultados

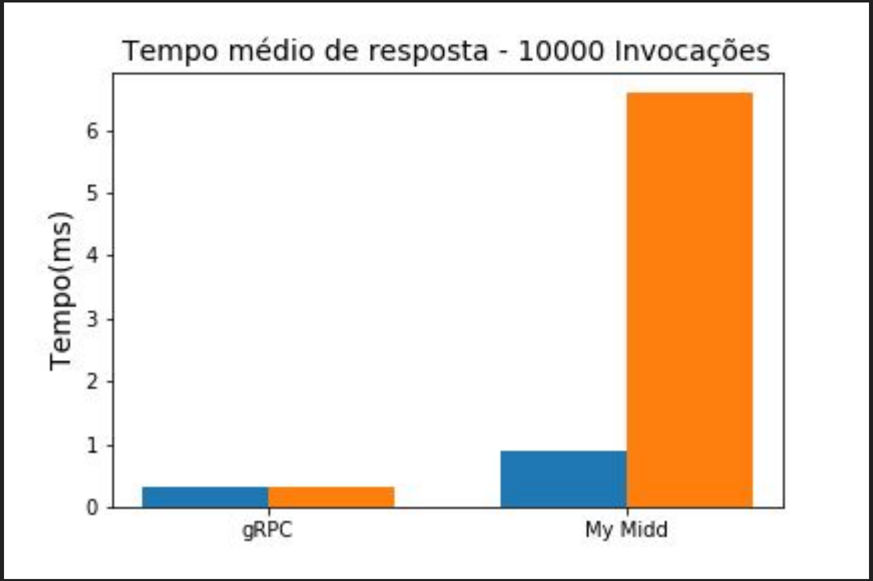


Tempo médio de resposta - 10000 Invocações
Tempo(ms)
0.0
0.2
0.4
0.6
0.8
1.0
0.0599
0.0615
0.3127
0.3859
0.8899
UDP
TCP
gRPC
RabbitMQ
My Midd

Resultados



Tempo médio de resposta - 10000 Invocações
Tempo(ms)
80
60
40
20
0
−20
−40
−60
gRPC
My Midd
Warmup
Sem Warmup



Tempo médio de resposta - 10000 Invocações
Tempo(ms)
6
5
4
3
2
1
0
gRPC
My Midd