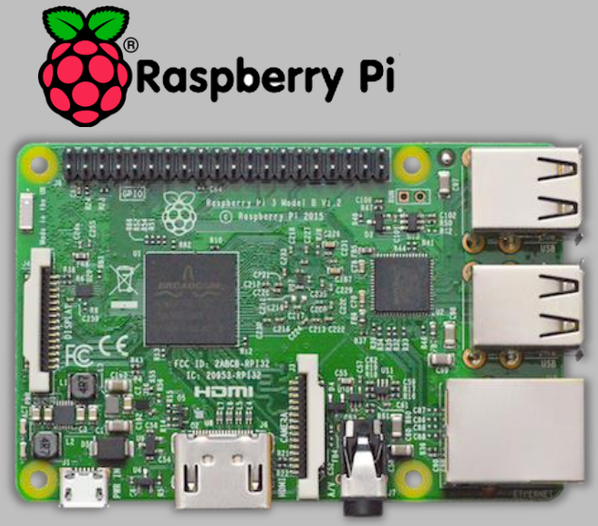
Módulo de 16 relés com Raspberry Pi 3 utilizando Socket

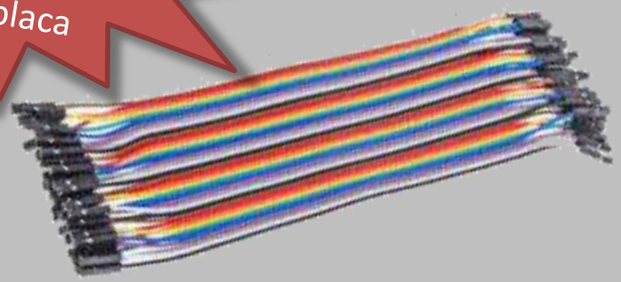


Por Fernando Koyanagi

Recursos usados

- 18 jumpers fêmea x fêmea
- Módulo de 16 relés com optoacoplador
- Raspberry Pi 3
- Rede interna (TCP/IP)

Pode usar Leds com resistor de 330 ohm no lugar da placa



Intenção dessa aula

1. Iniciar automação com Raspberry Pi
2. Introduzir conceitos de socket e Layer
3. Mostrar um cliente-servidor
4. Introduzir programação em C Unix / Linux
5. Utilizar funções da Lib WiringPI

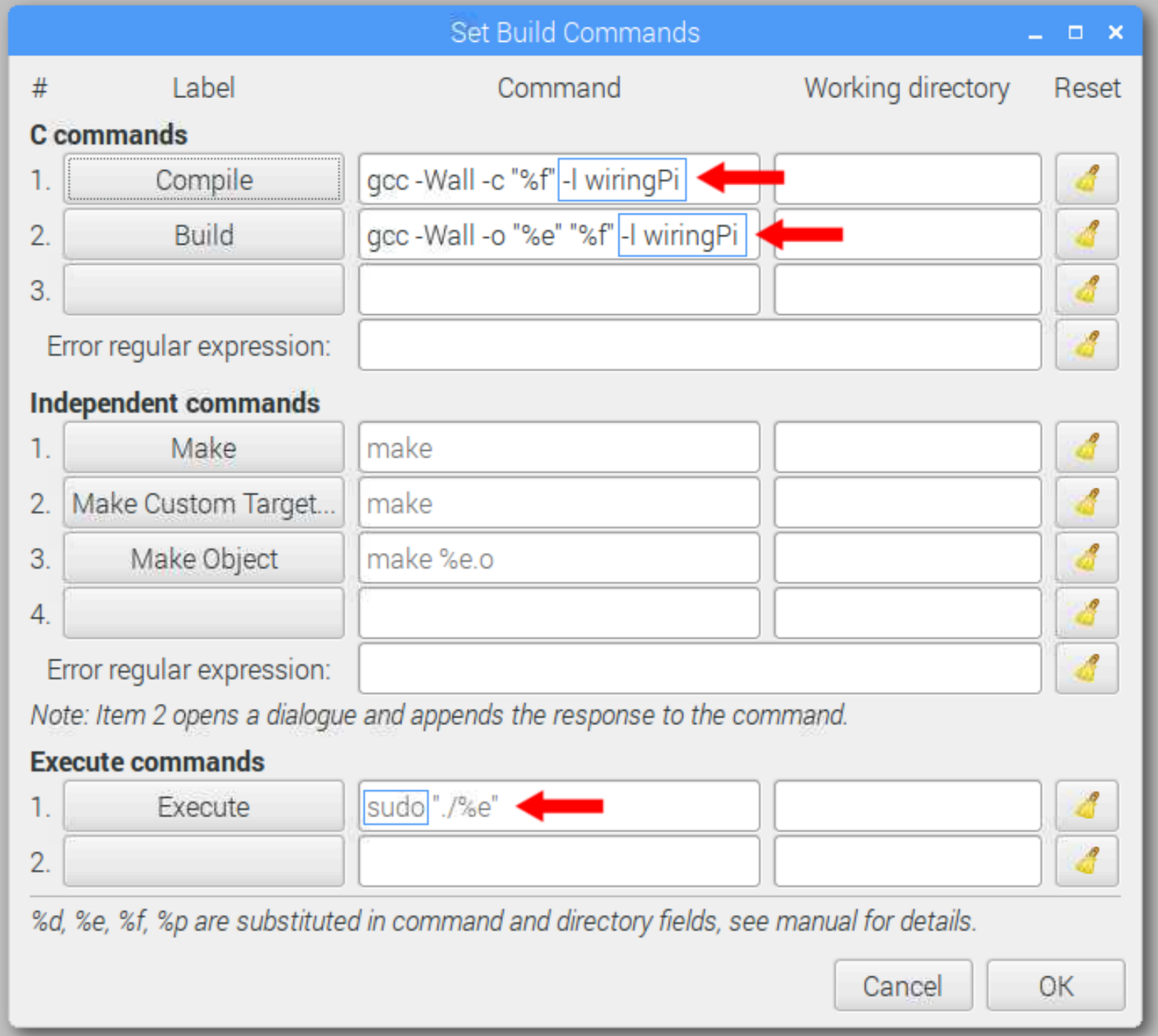
Configuração Geany

É necessário configurar a IDE do Raspberry antes de compilar os códigos.

1. Clique no menu
Build -> Set Build Commands
(Construir -> Definir Comandos de Construção);

2. Adicione os seguintes comandos:

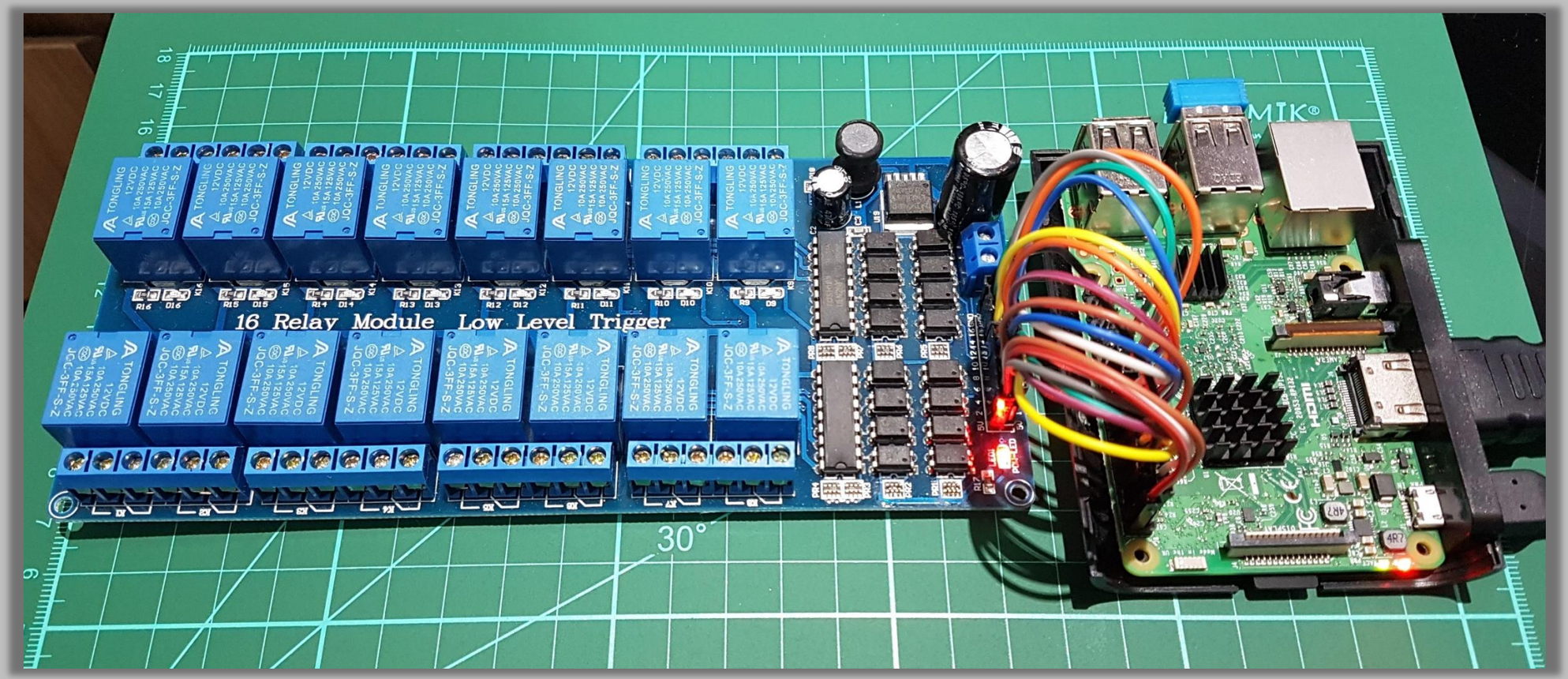
Em “Compile” adicione: “-lwiringPi” e “-lpthread”
Em “Build” adicione: “-lwiringPi” e “-lpthread”
Em “Execute” adicione: “sudo” como na imagem

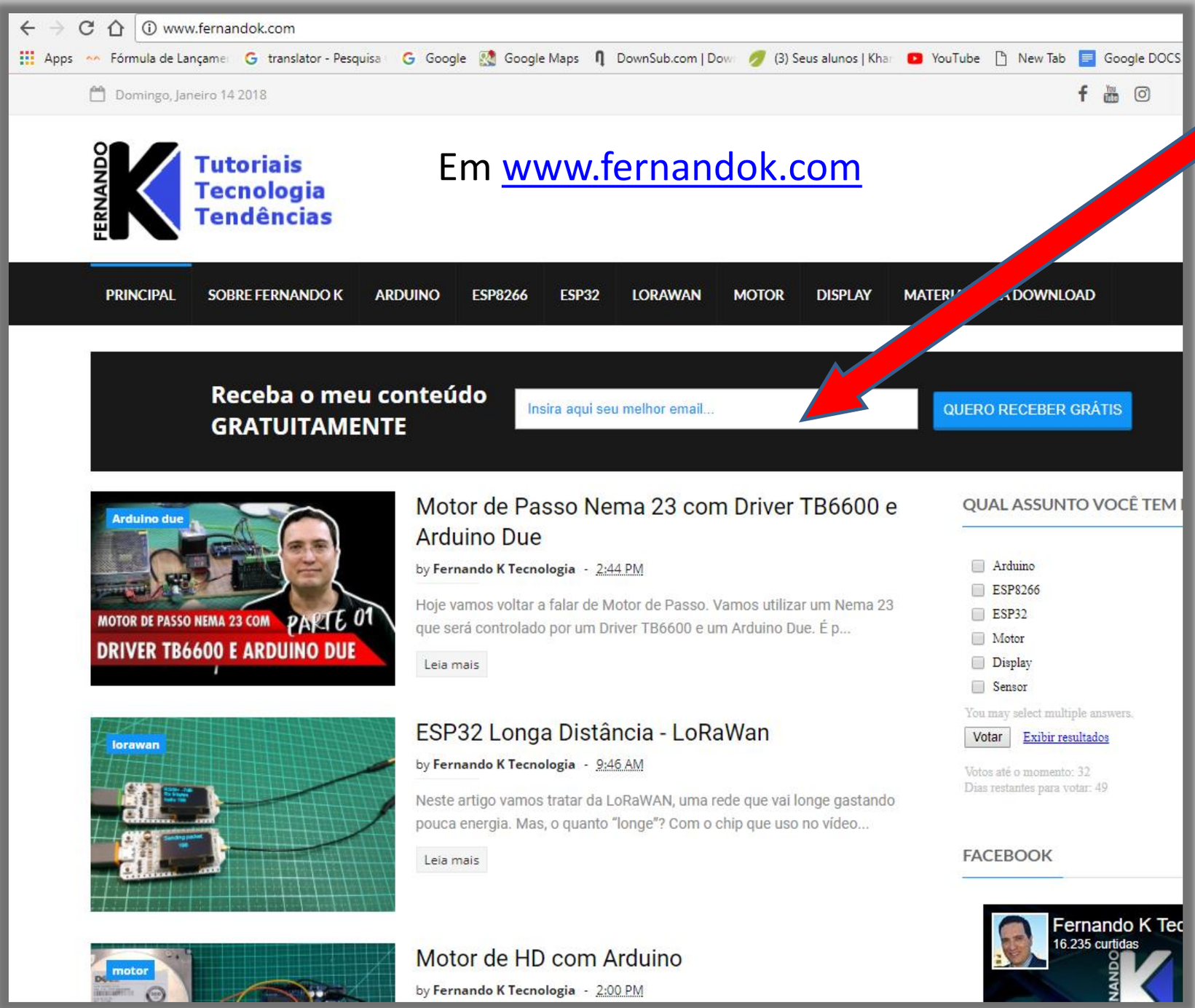


Programas Client.c e Server.c

Os dois vão rodar dentro do Raspberry PI
Compilados no C do Linux (gcc) com WiringPI
Rodam dentro do próprio Raspberry e o Client pode rodar fora mudando o IP "127.0.0.1"

Ligação dos jumpers...

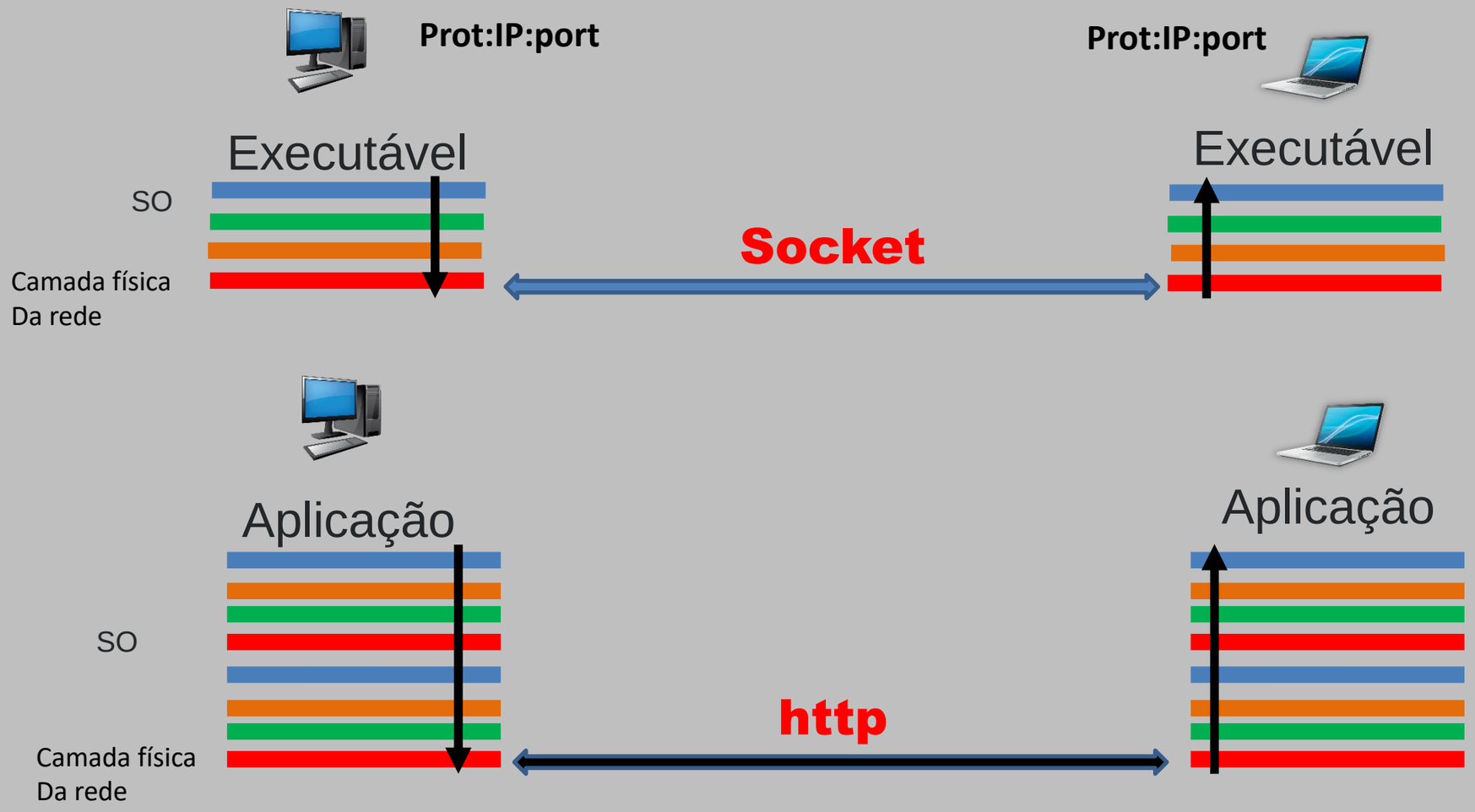


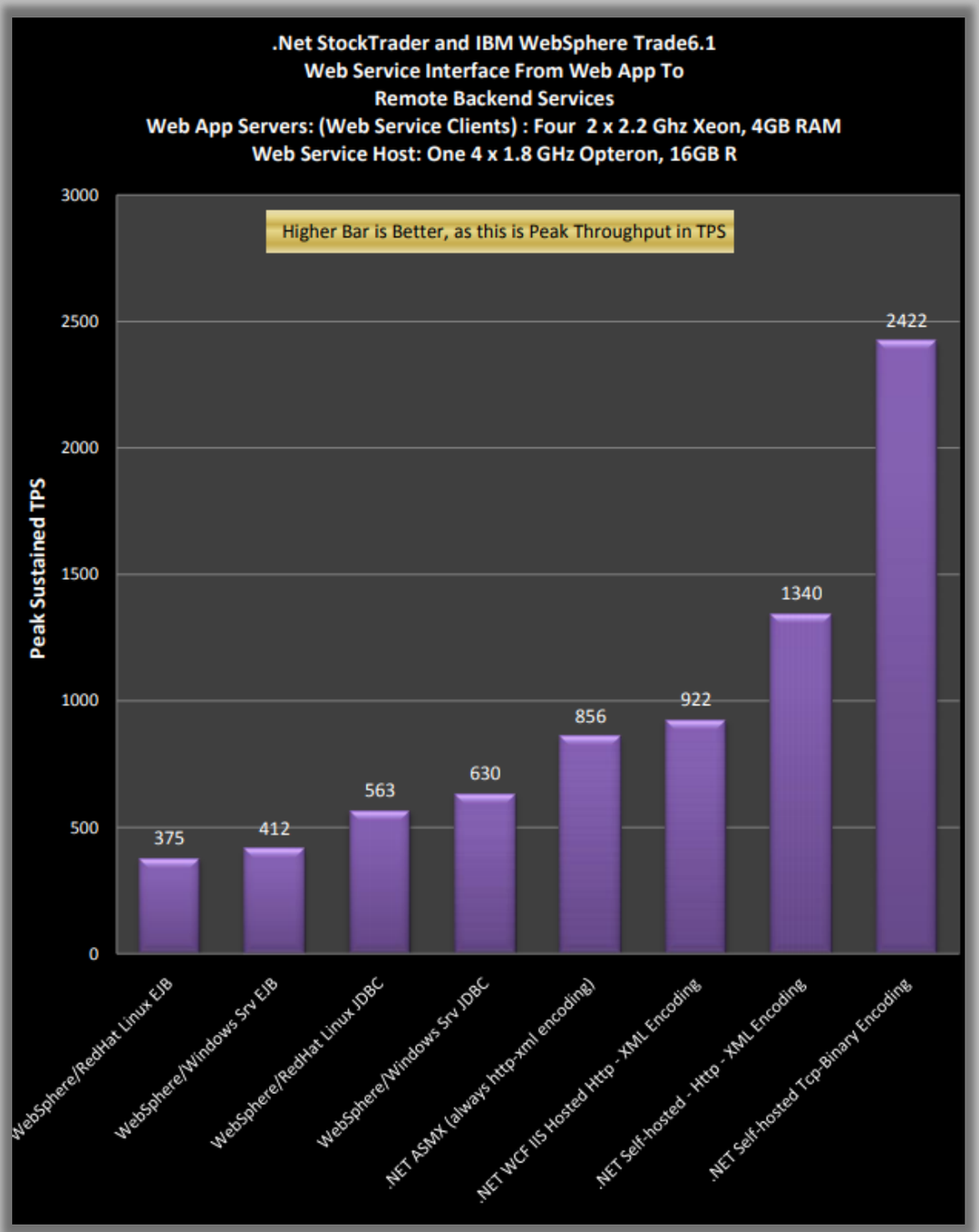

www.fernandok.com
Apps
Fórmula de Lançame
translator - Pesquisa
Google
Google Maps
DownSub.com | Dow
(3) Seus alunos | Kha
YouTube
New Tab
Google DOCS
Domingo, Janeiro 14 2018
FERNANDO K
Tutoriais
Tecnologia
Tendências
Em www.fernandok.com
PRINCIPAL
SOBRE FERNANDO K
ARDUINO
ESP8266
ESP32
LORAWAN
MOTOR
DISPLAY
DOWNLOAD
Receba o meu conteúdo GRATUITAMENTE
Insira aqui seu melhor email...
QUERO RECEBER GRÁTIS
Arduino due
MOTOR DE PASSO NEMA 23 COM DRIVER TB6600 E ARDUINO DUE
PARTE 01
Motor de Passo Nema 23 com Driver TB6600 e Arduino Due
by Fernando K Tecnologia - 2:44 PM
Hoje vamos voltar a falar de Motor de Passo. Vamos utilizar um Nema 23 que será controlado por um Driver TB6600 e um Arduino Due. É p...
Leia mais
lorawan
ESP32 Longa Distância - LoRaWan
by Fernando K Tecnologia - 9:46 AM
Neste artigo vamos tratar da LoRaWAN, uma rede que vai longe gastando pouca energia. Mas, o quanto "longe"? Com o chip que uso no vídeo...
Leia mais
motor
Motor de HD com Arduino
by Fernando K Tecnologia - 2:00 PM
QUAL ASSUNTO VOCÊ TEM
Arduino
ESP8266
ESP32
Motor
Display
Sensor
You may select multiple answers.
Votar
Exibir resultados
Votos até o momento: 32
Dias restantes para votar: 49
FACEBOOK
Fernando K Tec
16.235 curtidas

Seu e-mail

Socket ou http ?

É preferivelmente usado em sistemas orientados para o desempenho e segurança , nos setores militar ou financeiro.





Socket ou HTTP ?
Performance ?
Segurança ?

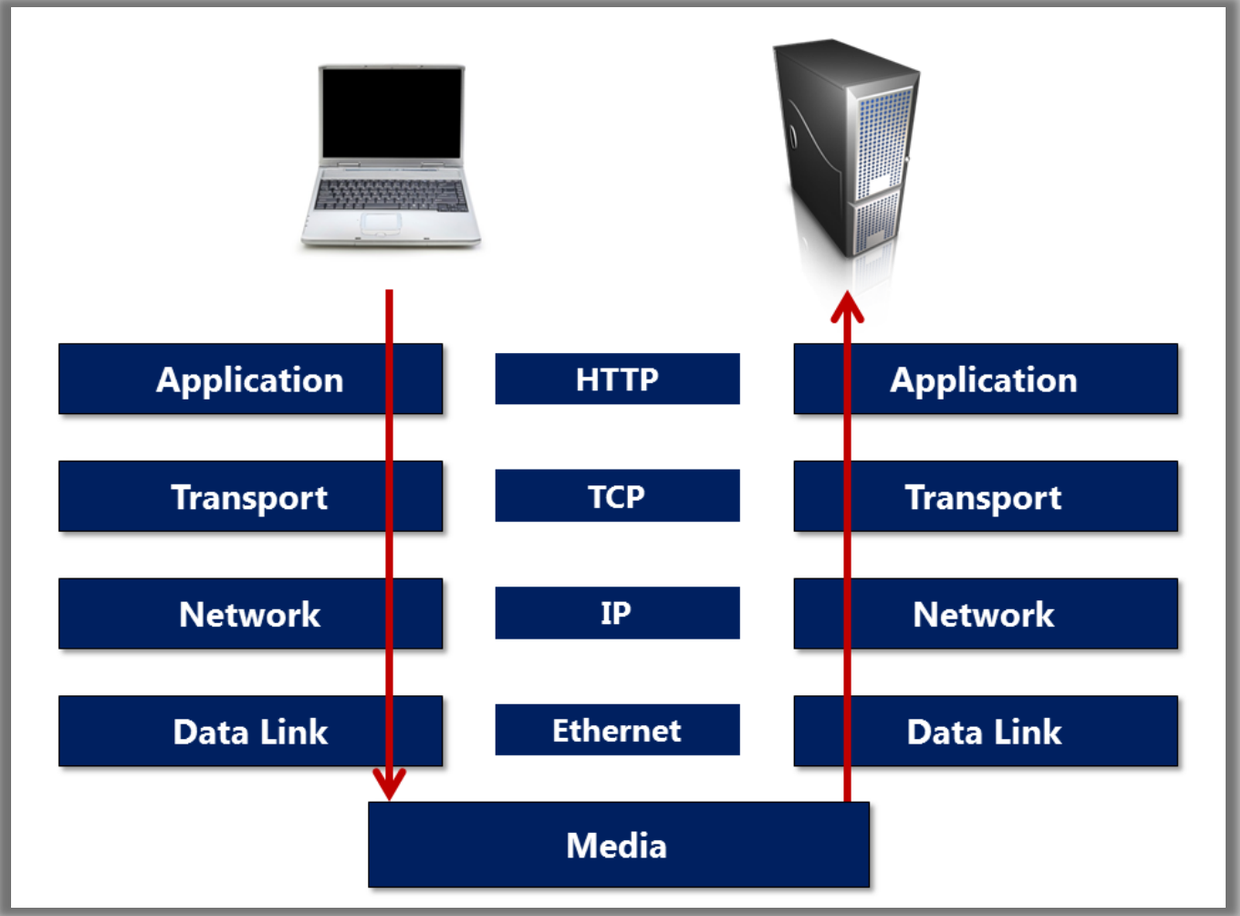
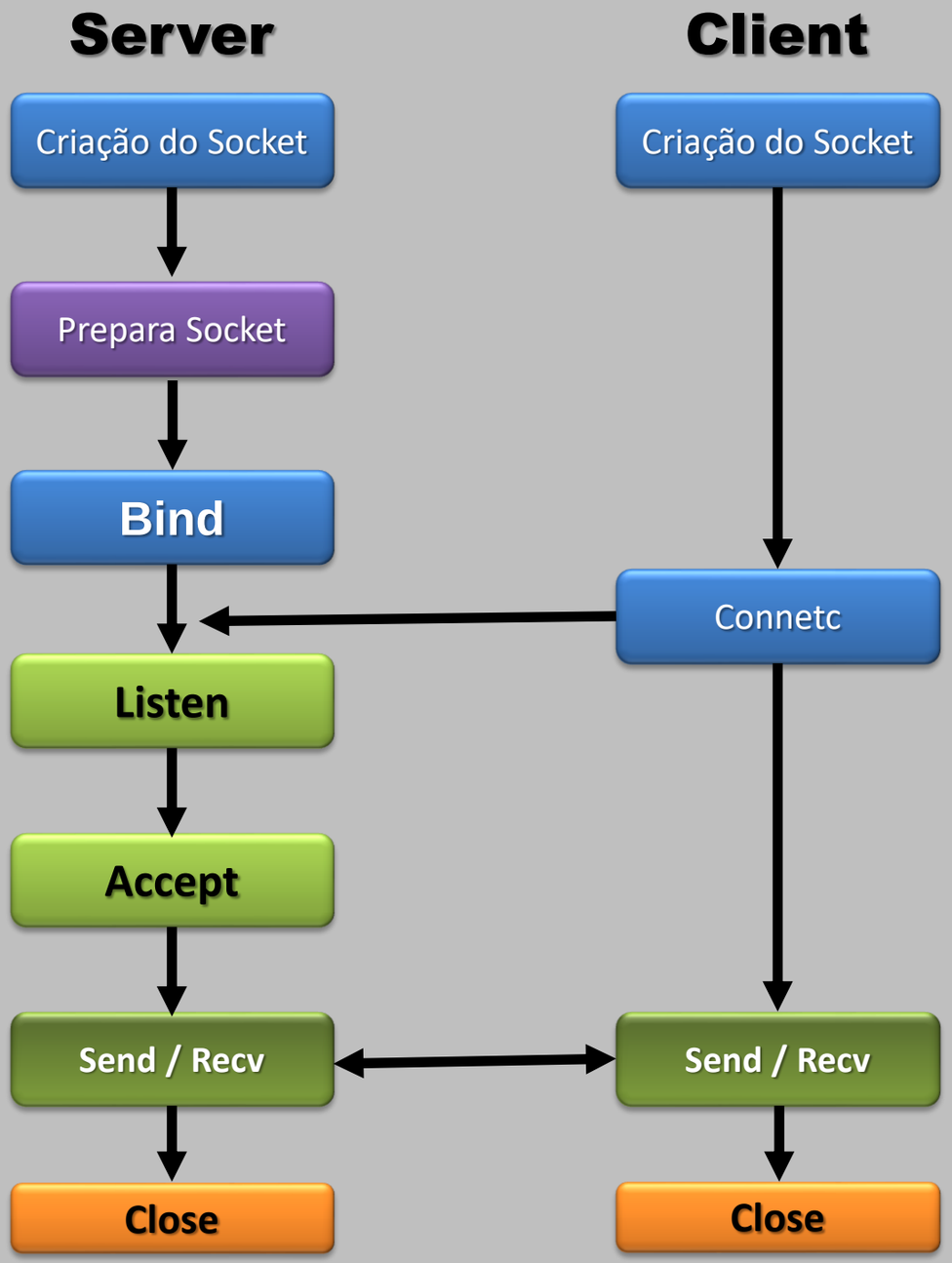
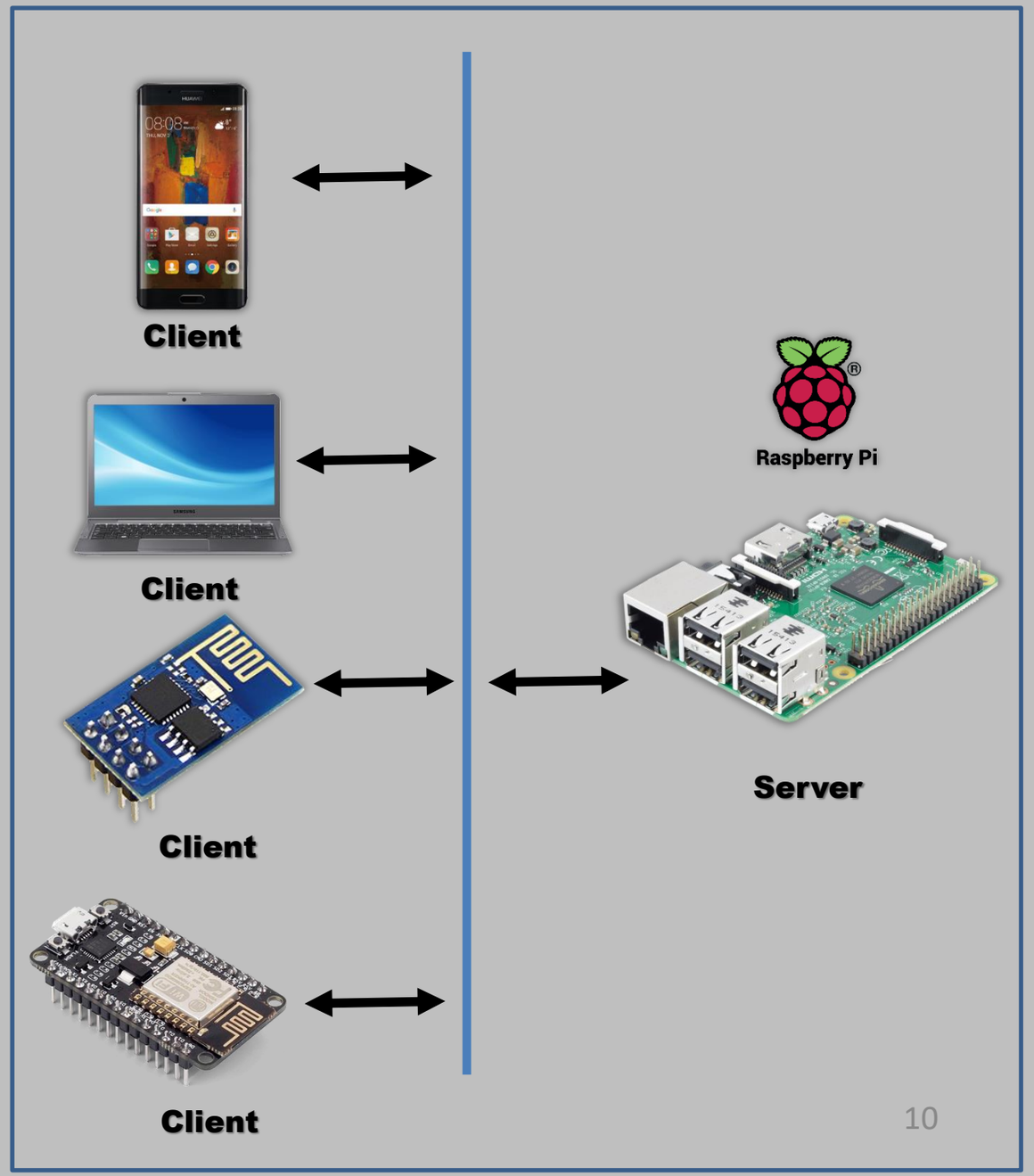


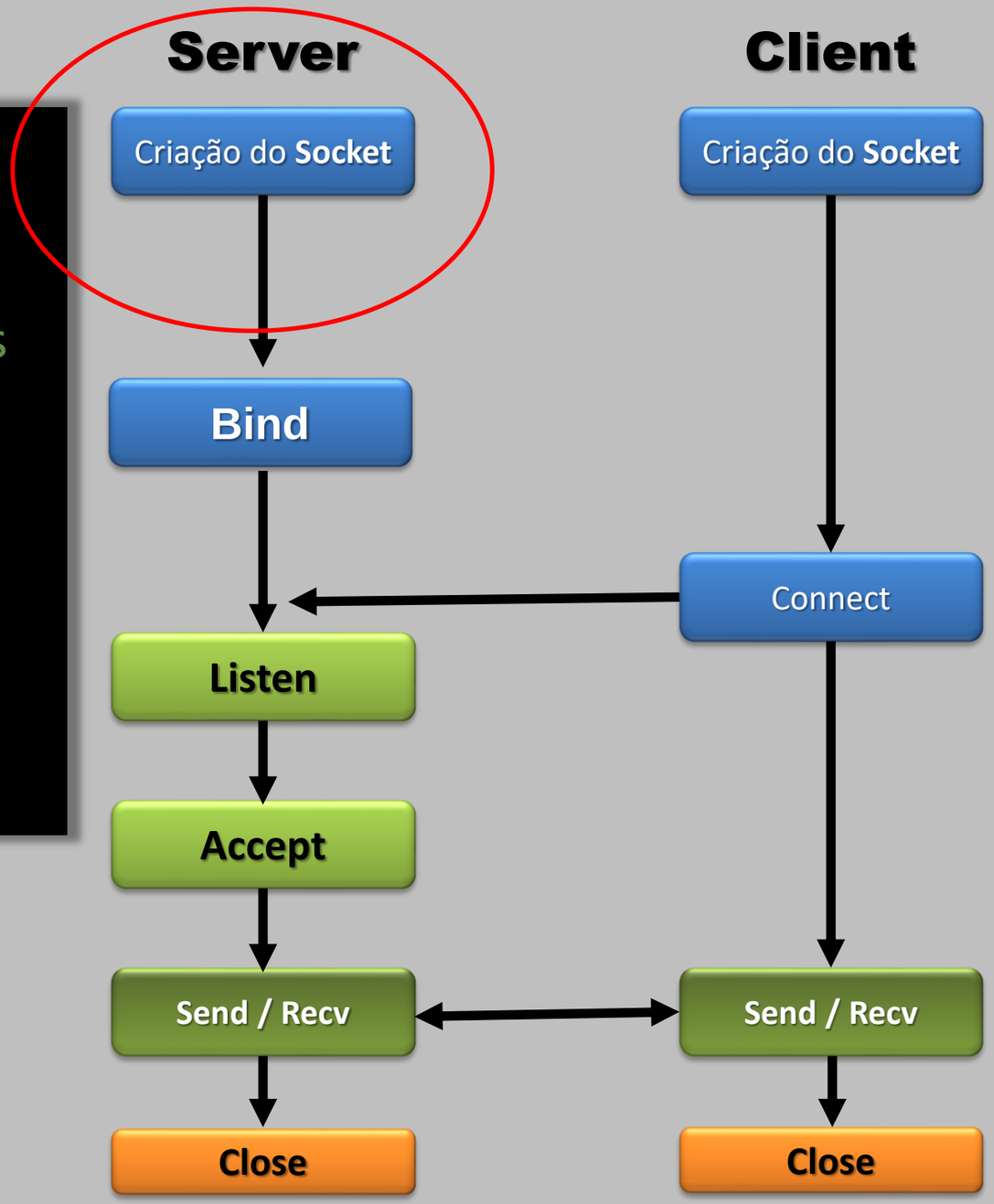
Diagrama de estado do modelo Client / Server





Código Servidor main()

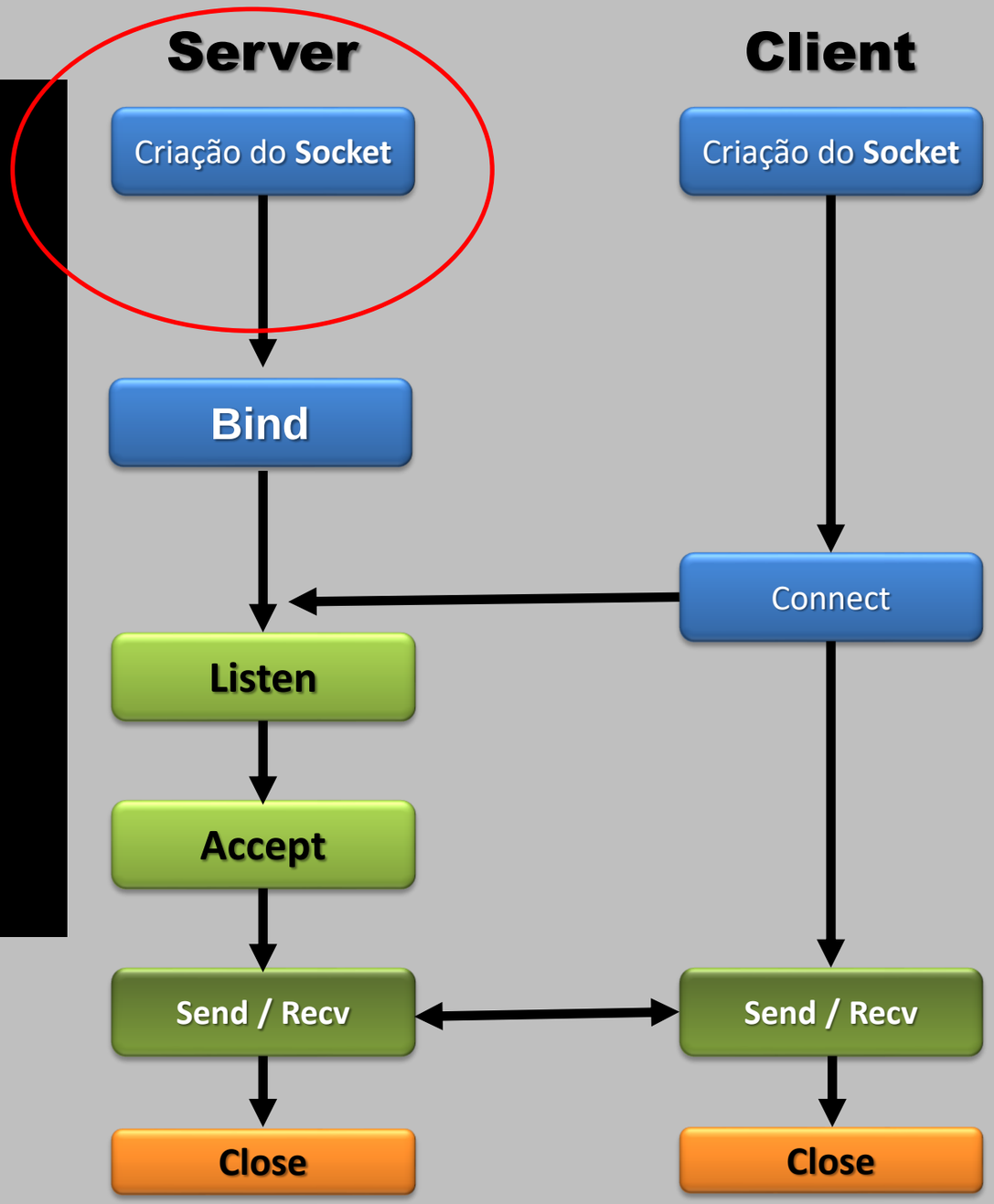
```c
int main(int argc , char *argv[])
{
    int socket_servidor;
    setaPinos();  //seta os pinos dos reles
    //cria e prepara o socket
    if(!preparaSocket(&socket_servidor))
        return 0;
    iniciaServidor(socket_servidor);
    //inicia o servidor e executa
    um loop de recepção de mensagens
}
```



Código Servidor

```
bool preparaSocket(int *conexao)
{
struct sockaddr_in servidor;
//cria conexão socket
*conexao = socket(AF_INET, SOCK_STREAM , 0);
if (*conexao == -1)
{
puts("Não foi possível criar o socket");
return false;
}
puts("Socket criado");
//Prepara a estrutura sockaddr_in
servidor.sin_family = AF_INET;
servidor.sin_addr.s_addr = INADDR_ANY;
servidor.sin_port = htons(PORTA);
```

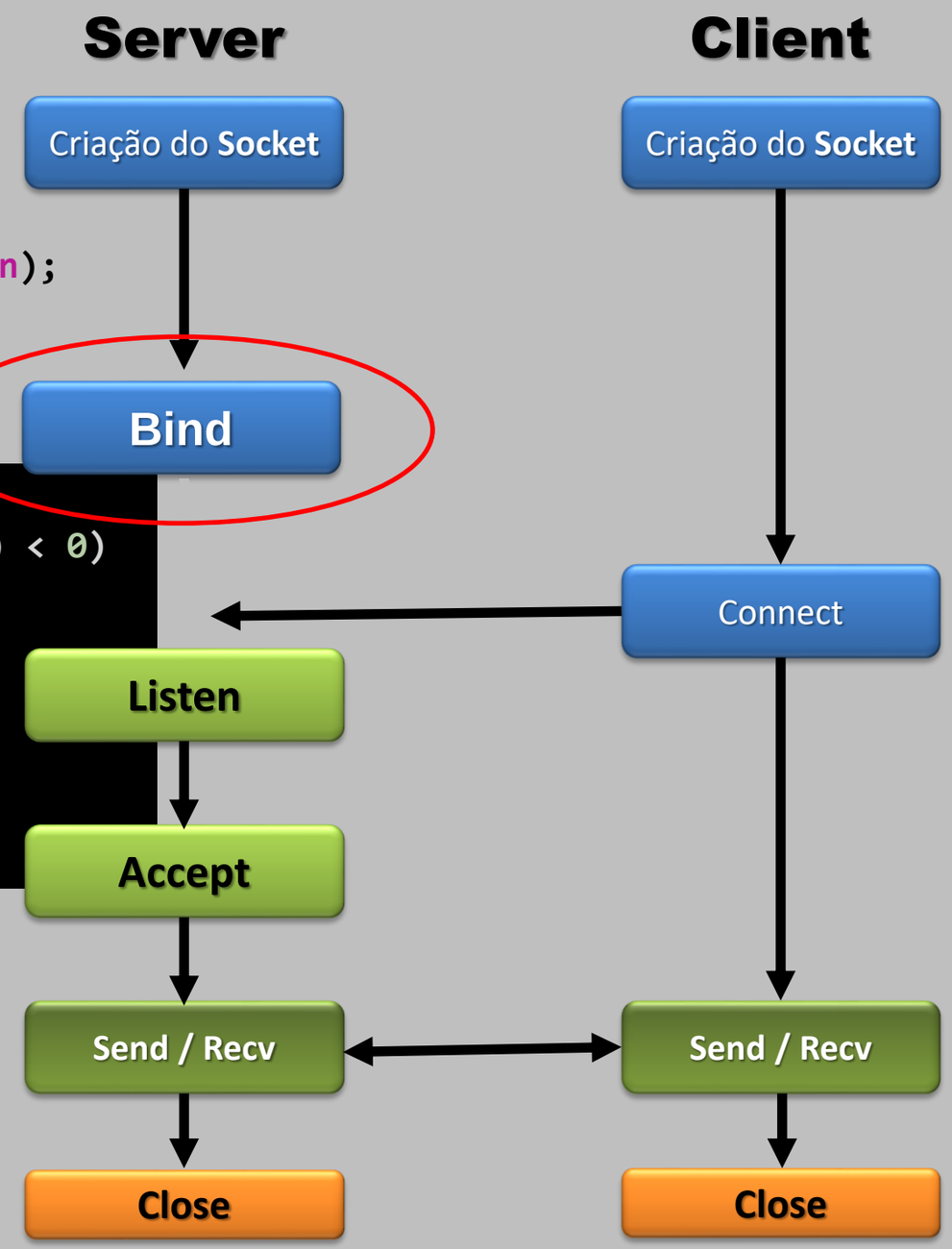
domínio: inteiro, domínio de comunicação, por exemplo, protocolo AF_INET (protocolo IPv4), AF_INET6 (protocolo IPv6)
Quando **INADDR_ANY** é especificado na chamada de ligação, o soquete será vinculado a *todas* as interfaces locais



Código Servidor

```
int bind (int sockfd, const struct sockaddr * addr, socklen_t addrlen);
```

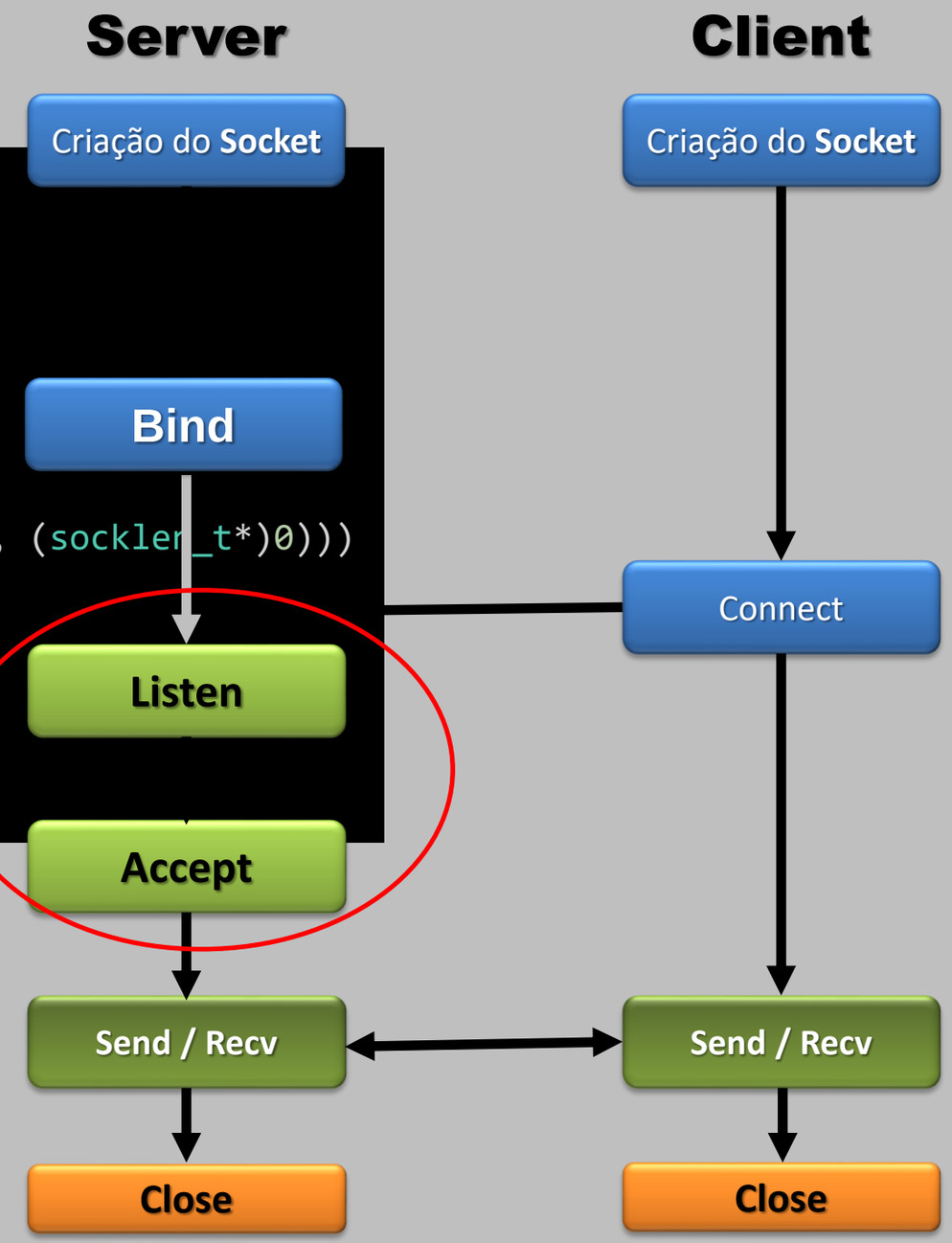
```
//efetua ligação
if(bind((*conexao),(struct sockaddr *)&(servidor) , sizeof(servidor)) < 0)
{
     puts("Falha na ligaçao");
     return false;
}
puts("Ligação efetuada");
return true;
```



Código Servidor

void iniciaServidor(int socket_servidor)

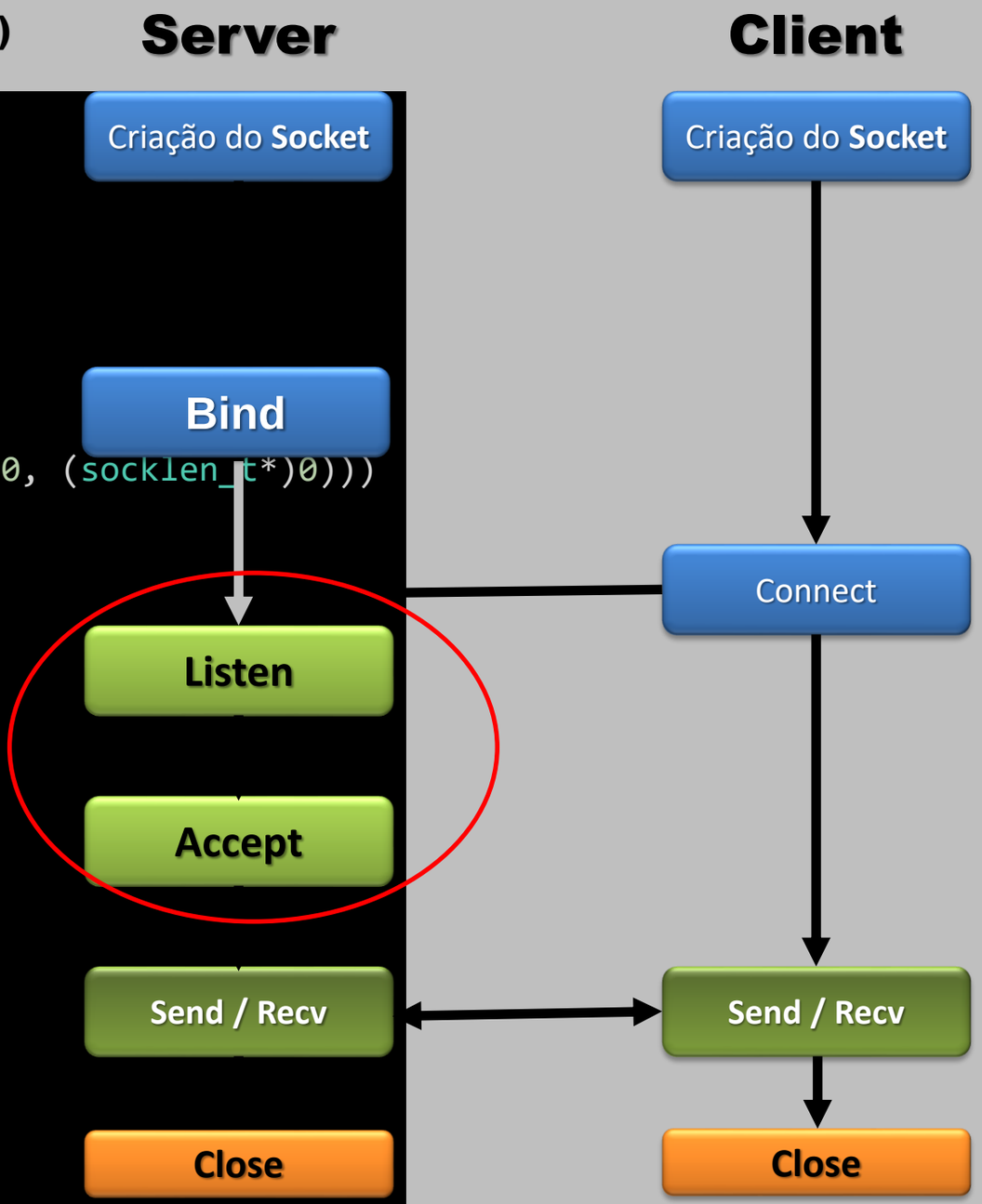
```c
void iniciaServidor(int socket_servidor)
{
 int socket_cliente, *novo_socket;
 //inicia a recepção de mensagens
 listen(socket_servidor , 3);
 puts("Aguardando novas conexões...");
 //loop infinito, para cada conexão é criada uma thread que recebe
 as mensagens
while((socket_cliente = accept(socket_servidor, (struct sockaddr *)0, (sockler_t*)0)))
{
      puts("Conexão aceita");
      //cria thread
      pthread_t thread;
      novo_socket = malloc(1);
      *novo_socket = socket_cliente;
```



Código Servidor

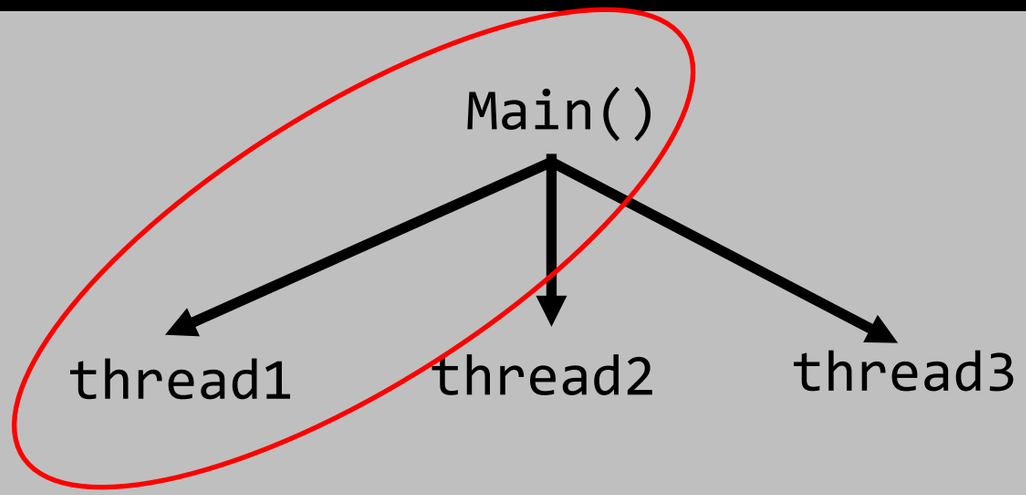
void iniciaServidor(int socket_servidor)

```c
void iniciaServidor(int socket_servidor)
{
 int socket_cliente, *novo_socket;
 //inicia a recepção de mensagens
 listen(socket_servidor , 3);
 puts("Aguardando novas conexões...");
 //loop infinito, para cada conexão é criada uma thread que recebe
 as mensagens
while((socket_cliente = accept(socket_servidor, (struct sockaddr *)0, (socklen_t*)0)))
{
      puts("Conexão aceita");
      //cria thread
      pthread_t thread;
      novo_socket = malloc(sizeof(*novo_socket));
      *novo_socket = socket_cliente;
      //caso ocorra algum erro neste momento o loop é abortado
      if(pthread_create(&thread,NULL , comunicacaoSocket,
      (void*) novo_socket) < 0)
      {
            puts("Não foi possível criar a thread.");
            close(*novo_socket);
            free(novo_socket);
            return;
      }
      puts("Thread criada");
}
```

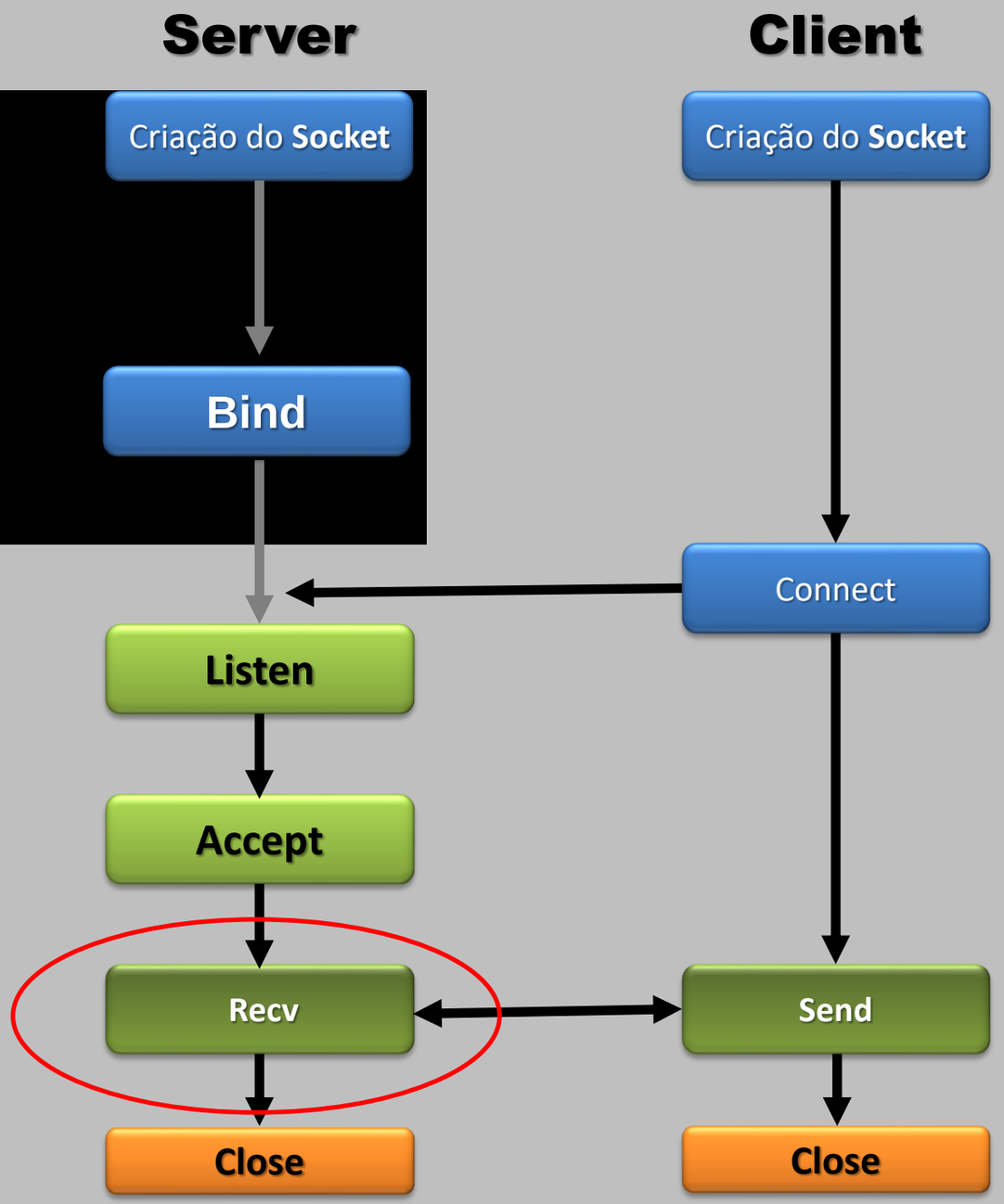


Código Servidor threads

```
      if(pthread_create(&thread,NULL , comunicacaoSocket,
      (void*) novo_socket) < 0)
      {
            puts("Não foi possível criar a thread.");
            close(*novo_socket);
            free(novo_socket);
            return;
      }
      puts("Thread criada");
}
```



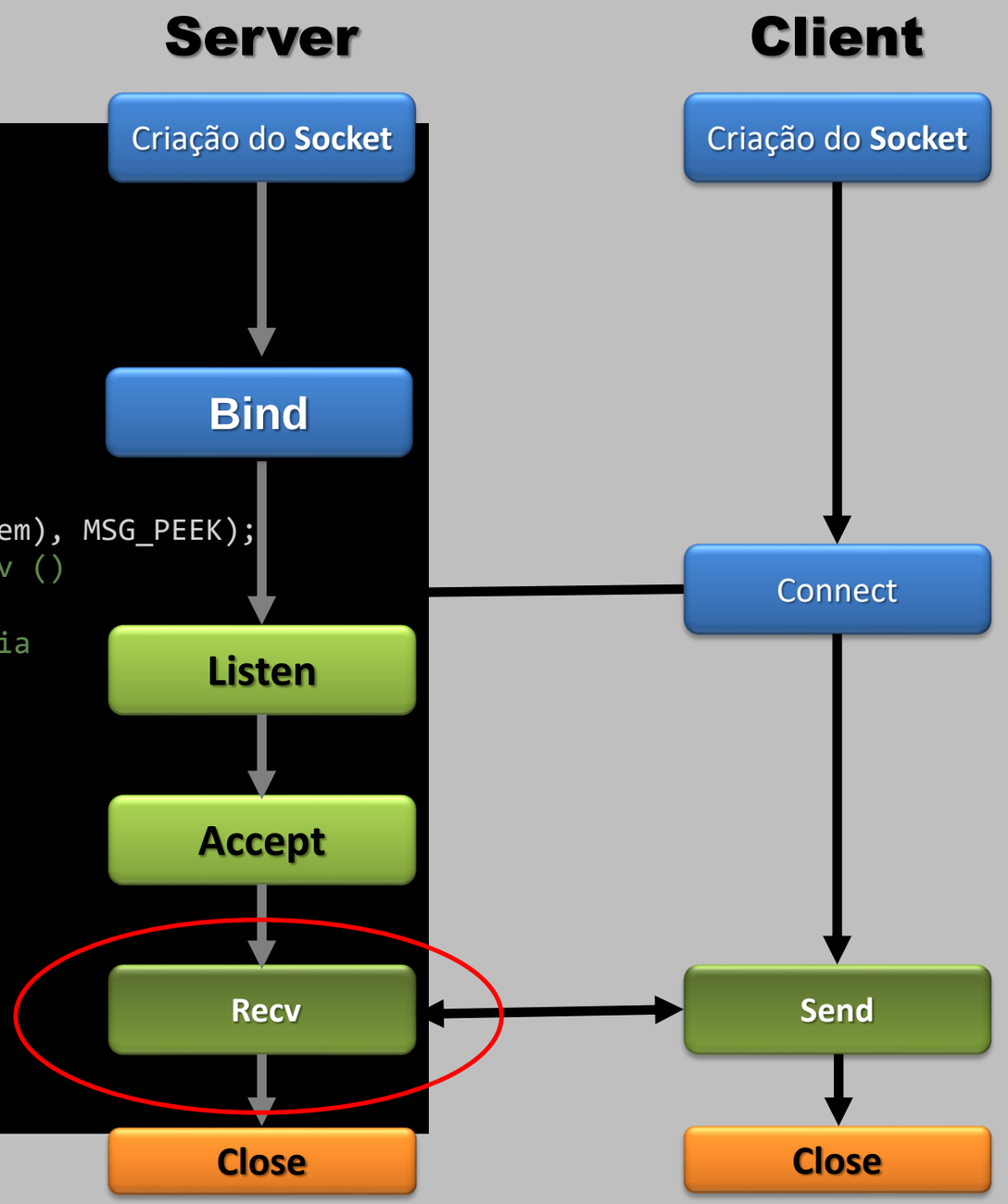

```
void *comunicacaoSocket(void *conexao)
```



Código Servidor

`void *comunicacaoSocket(void *conexao)`

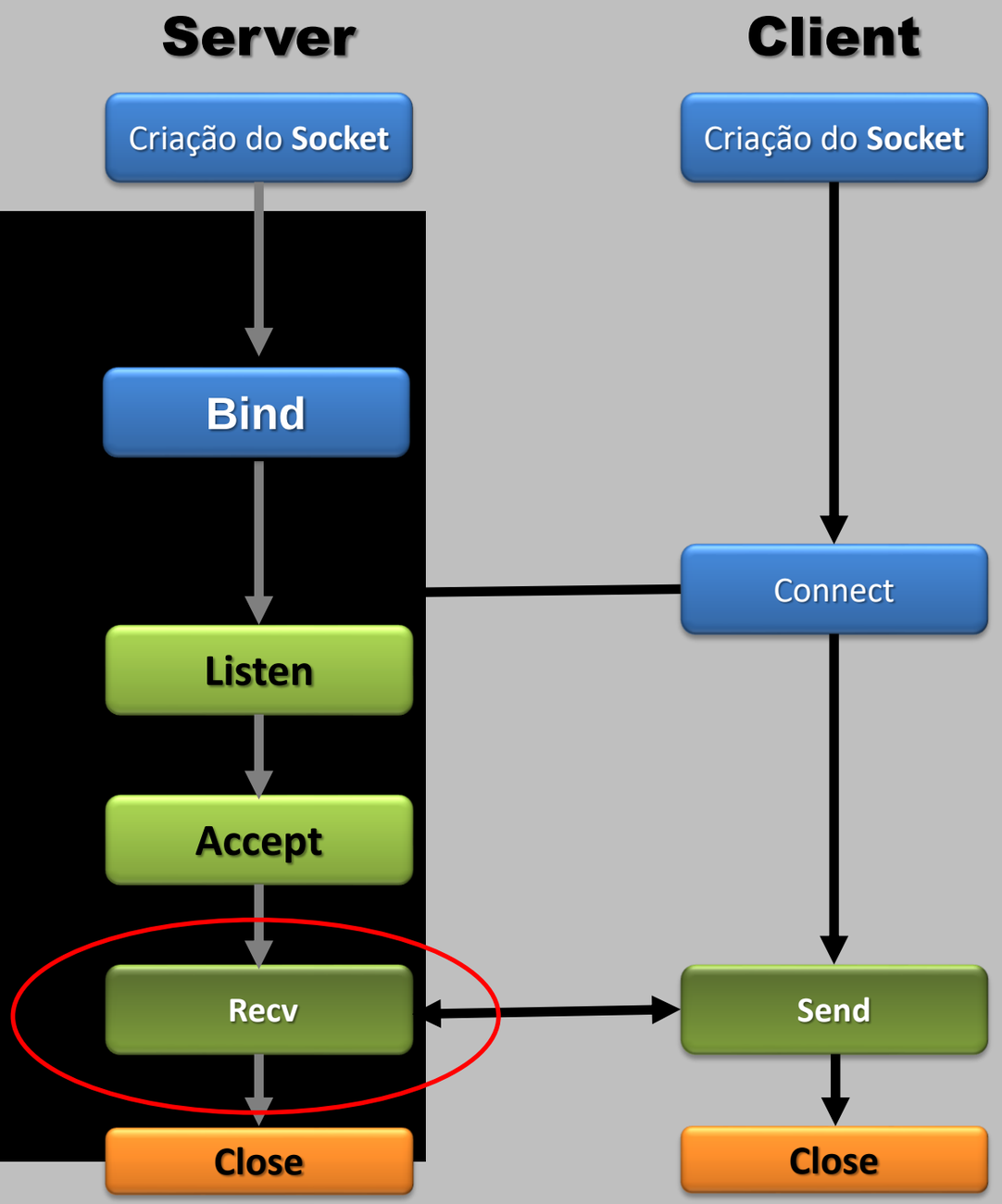
```
      void *comunicacaoSocket(void *conexao)
{
puts("Thread de conexão iniciada");
int tamanhoMensagemRecebida;
char mensagem[TF_MENSAGEM];
mensagem[0] = '\0';
//enquanto não for recebida nenhuma mensagem, permanece em loop
while(1)
{
   //recebe mensagem
   tamanhoMensagemRecebida = recv(*(int*)conexao ,&mensagem, sizeof(mensagem), MSG_PEEK);
   // MSG_PEEK Os dados são tratados como não lidos e a próxima função recv ()
      ou similar ainda deve retornar esses dados
   //define o final da mensagem (este comando impede que o 'lixo' de memória
     fique junto à  mensagem)
   mensagem[tamanhoMensagemRecebida] = '\0';

   //imprime mensagem recebida
   printf("\nRecebido: %s\n",mensagem);

   //deixa a mensagem em maíusculo, assim tanto faz "a1" ou "A1"
   for(int i=0; i<tamanhoMensagemRecebida; i++)
   {
            mensagem[i] = toupper(mensagem[i]);
   }
```



Código Servidor

`void *comunicacaoSocket(void *conexao)`

```
//acende ou apaga reles de acordo com a mensagem recebida
   if(strcmp(mensagem,"A1")==0)
    ativaRele(PINORELE_1);
   else
   if(strcmp(mensagem,"D1")==0)
    desativaRele(PINORELE_1);
   else
   if(strcmp(mensagem,"A2")==0)
    ativaRele(PINORELE_2);
   else
   if(strcmp(mensagem,"D2")==0)
    desativaRele(PINORELE_2);
   else
   if(strcmp(mensagem,"A3")==0)
    ativaRele(PINORELE_3);
   else
   if(strcmp(mensagem,"D3")==0)
    desativaRele(PINORELE_3);
// continua assim até o A16 e D16
```

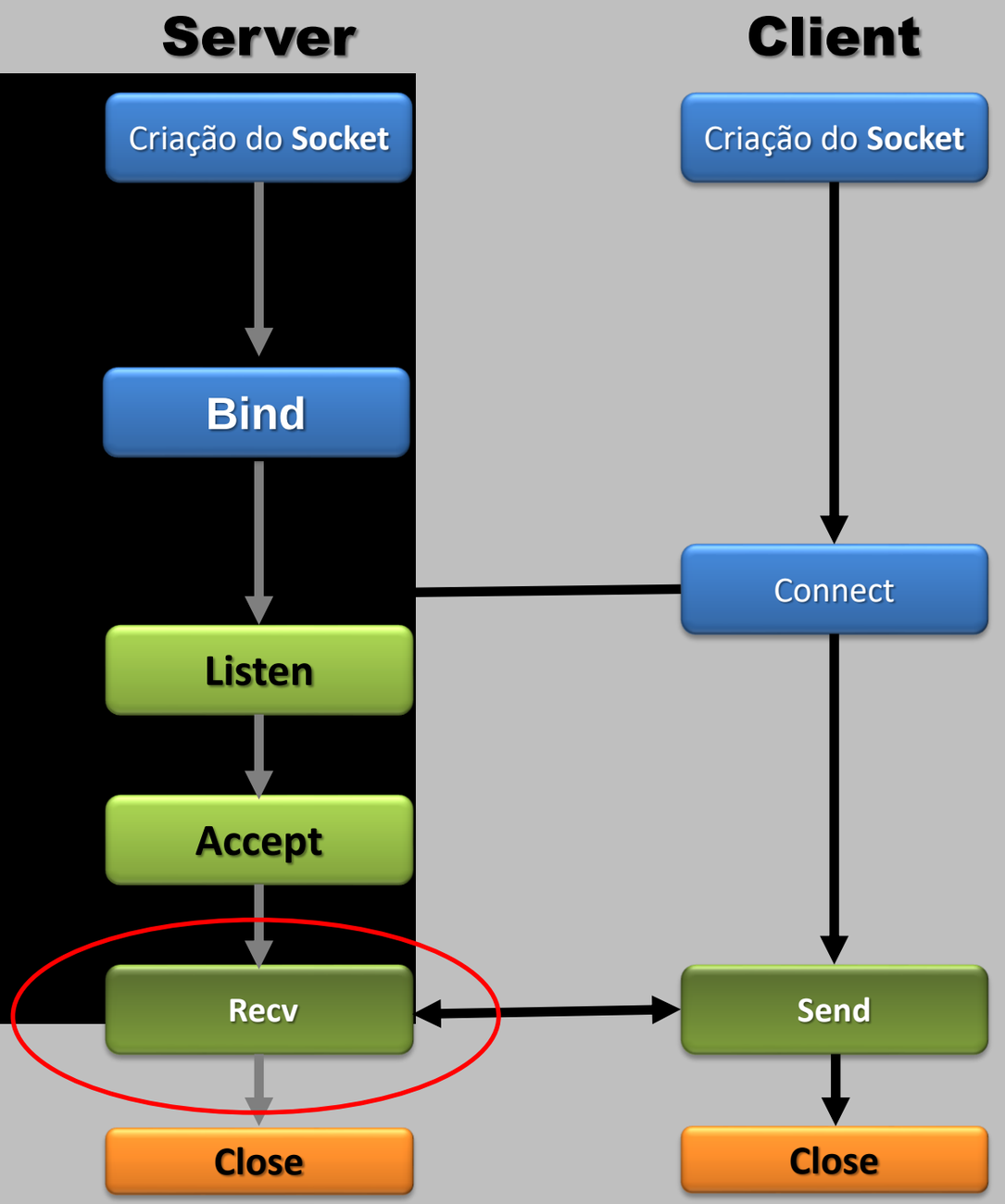


Código Servidor

`void *comunicacaoSocket(void *conexao)`

```
//acende ou apaga reles de acordo com a mensagem recebida
   if(strcmp(mensagem,"A1")==0)
    ativaRele(PINORELE_1);
   else
   if(strcmp(mensagem,"D1")==0)
    desativaRele(PINORELE_1);
   else
   if(strcmp(mensagem,"A2")==0)
    ativaRele(PINORELE_2);
   else
   if(strcmp(mensagem,"D2")==0)
    desativaRele(PINORELE_2);
   else
   if(strcmp(mensagem,"A3")==0)
    ativaRele(PINORELE_3);
   else
   if(strcmp(mensagem,"D3")==0)
    desativaRele(PINORELE_3);
// continua assim até o A16 e D16 e termina com um break;
```

```
puts("Thread finalizada e socket fechado.");
//libera memória desta conexão
free(conexao);
return 0;
}
```

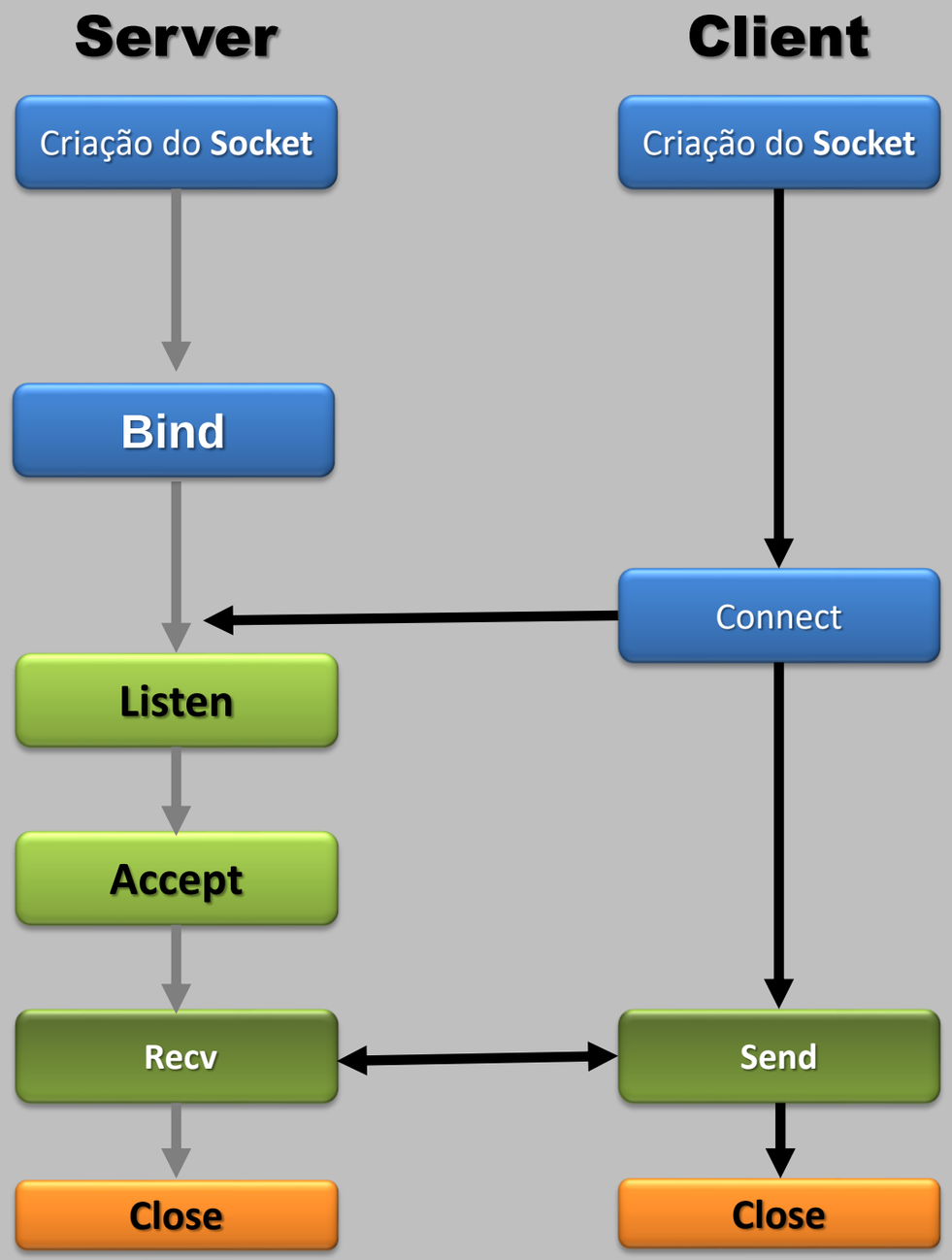


Código Servidor

`void ativaEmSequencia()`

```
void ativaEmSequencia()
{
  int vet[16];
  vet[0] = PINORELE_1;
  vet[1] = PINORELE_2;
  vet[2] = PINORELE_3;
  vet[3] = PINORELE_4;
  vet[4] = PINORELE_5;
  vet[5] = PINORELE_6;
  vet[6] = PINORELE_7;
  vet[7] = PINORELE_8;
  vet[8] = PINORELE_9;
  vet[9] = PINORELE_10;
  vet[10] = PINORELE_11;
  vet[11] = PINORELE_12;
  vet[12] = PINORELE_13;
  vet[13] = PINORELE_14;
  vet[14] = PINORELE_15;
  vet[15] = PINORELE_16;

  for(int i=0; i<16; i++)
  {
    digitalWrite(vet[i],LOW);
    delay(200);
    digitalWrite(vet[i],HIGH);
    delay(200);
  }
}
```

```
if(strcmp(mensagem,"SEQ")==0)
    ativaEmSequencia();
```

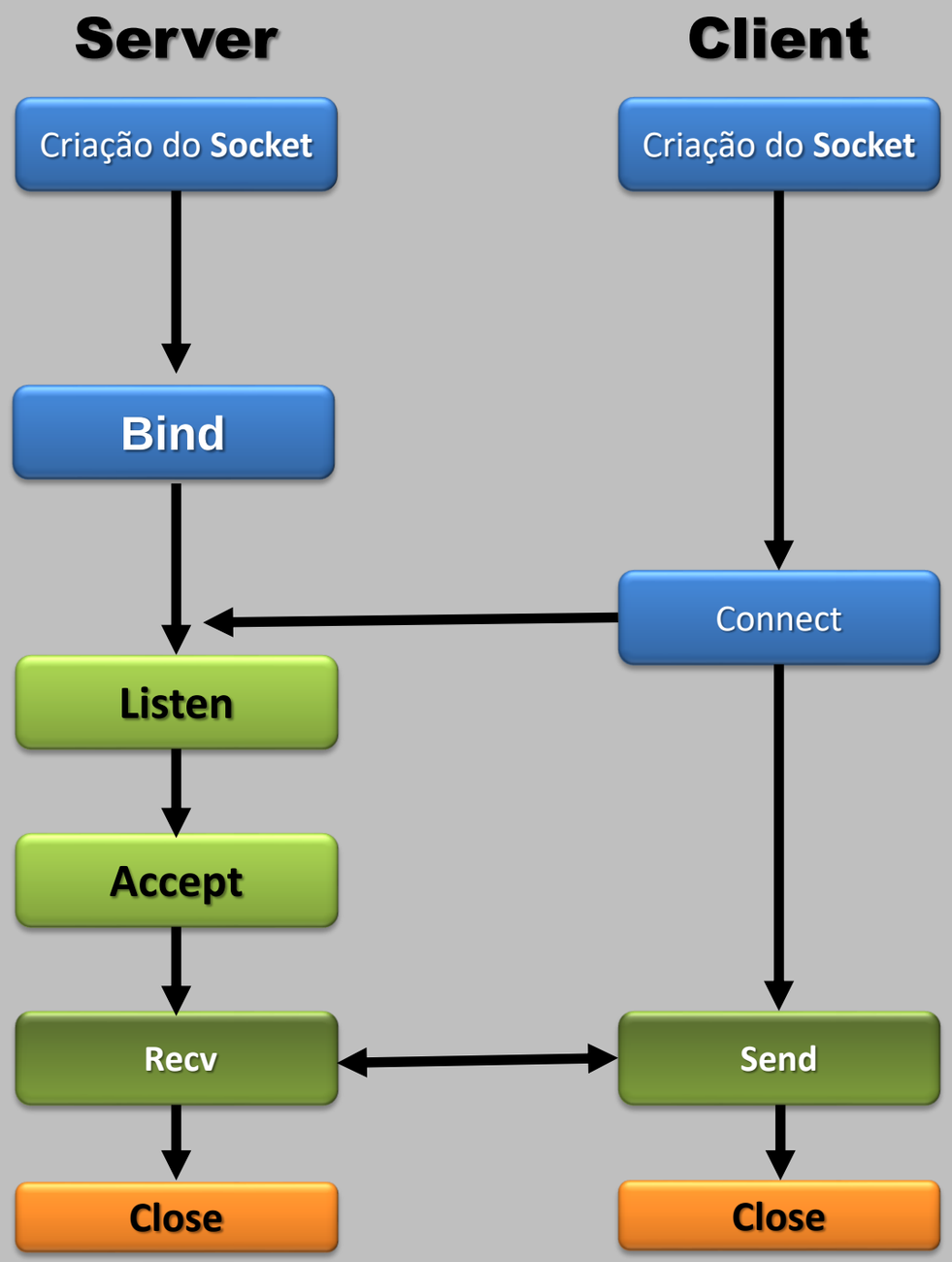


Código Servidor

`void ativaEmSequencia()`

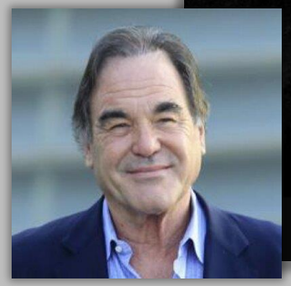
```
void ativaEmSequencia()
{
  int vet[16];
  vet[0] = PINORELE_1;
  vet[1] = PINORELE_2;
  vet[2] = PINORELE_3;
  vet[3] = PINORELE_4;
  vet[4] = PINORELE_5;
  vet[5] = PINORELE_6;
  vet[6] = PINORELE_7;
  vet[7] = PINORELE_8;
  vet[8] = PINORELE_9;
  vet[9] = PINORELE_10;
  vet[10] = PINORELE_11;
  vet[11] = PINORELE_12;
  vet[12] = PINORELE_13;
  vet[13] = PINORELE_14;
  vet[14] = PINORELE_15;
  vet[15] = PINORELE_16;

  for(int i=0; i<16; i++)
  {
    digitalWrite(vet[i],LOW);
    delay(200);
    digitalWrite(vet[i],HIGH);
    delay(200);
  }
}
```

```
if(strcmp(mensagem,"SEQ")==0)
    ativaEmSequencia();
```



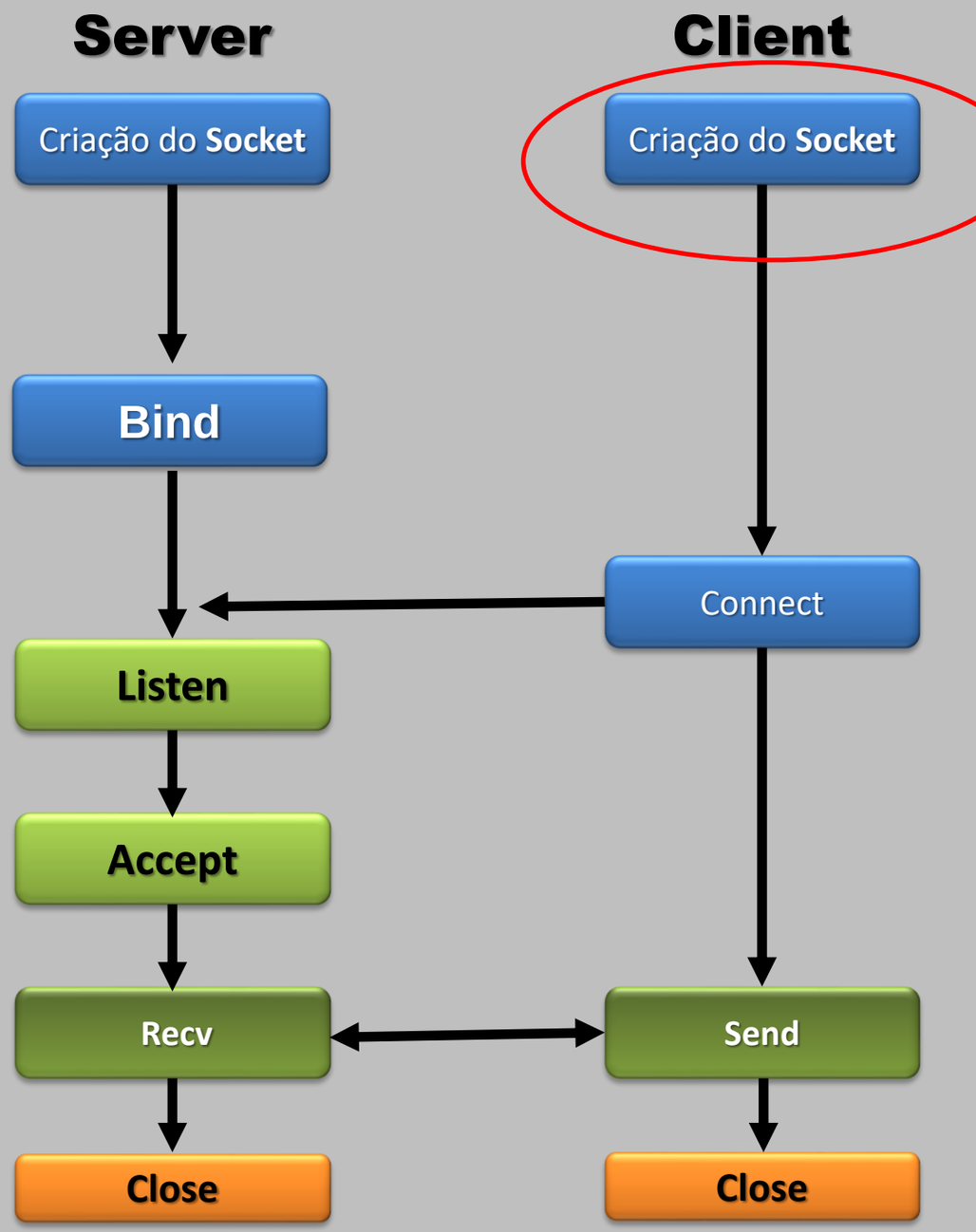
Código do Client

Dica de filme!



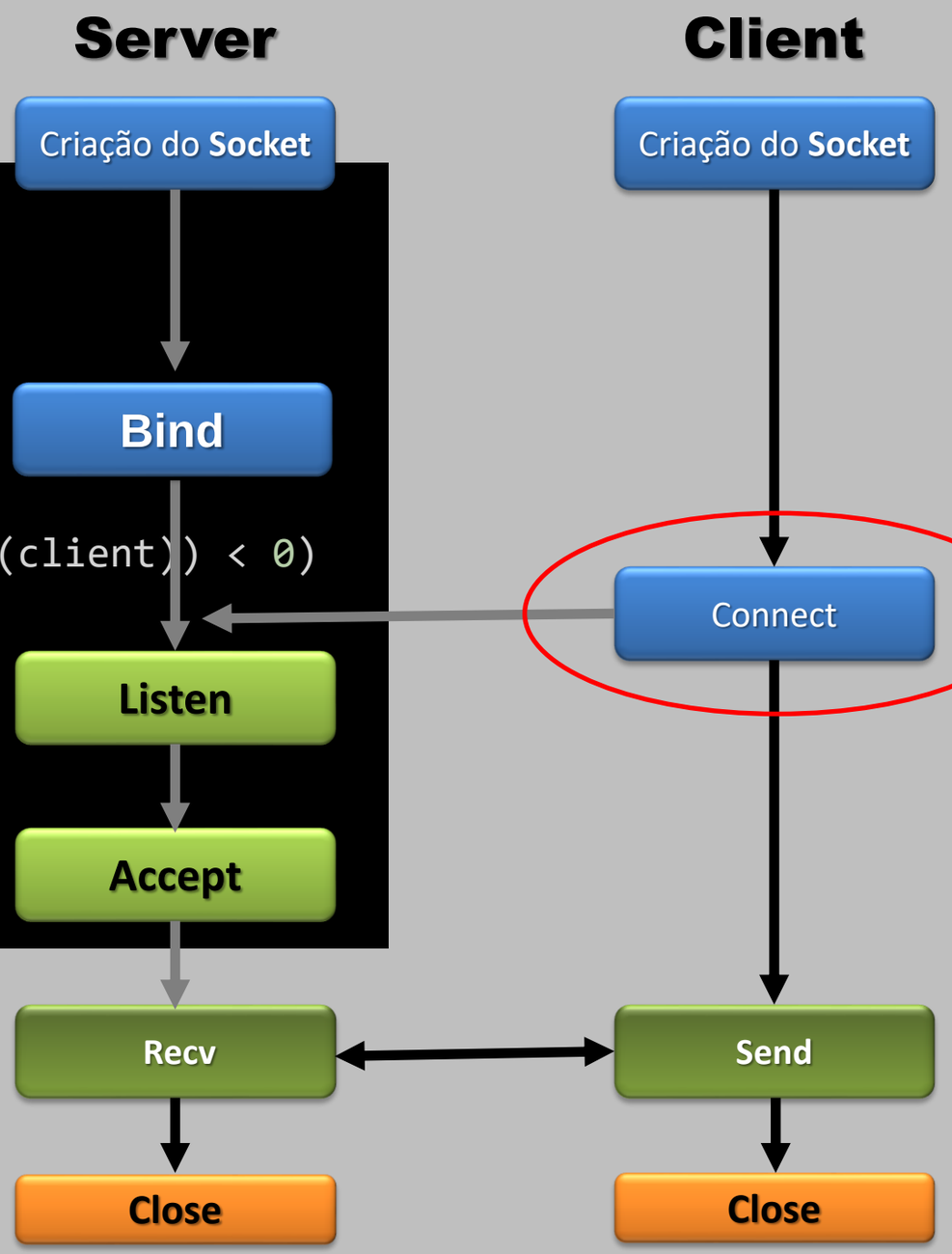
Código Cliente conecta()

```
//cria socket e conecta,
bool conecta(int *conexao)
{
    struct sockaddr_in client;

//cria conexão socket
    *conexao = socket(AF_INET , SOCK_STREAM , 0);

    //AF_INET faz parte de uma família de endereços que é
usada para designar o tipo de endereços com os quais seu
soquete pode se comunicar (neste caso, os endereços do
Internet Protocol v4)

    //caso ocorra um erro na criação, aborta a conexão
    if (*conexao == -1)
     return false;

    //imprime mensagem
    puts("Socket criado");
```



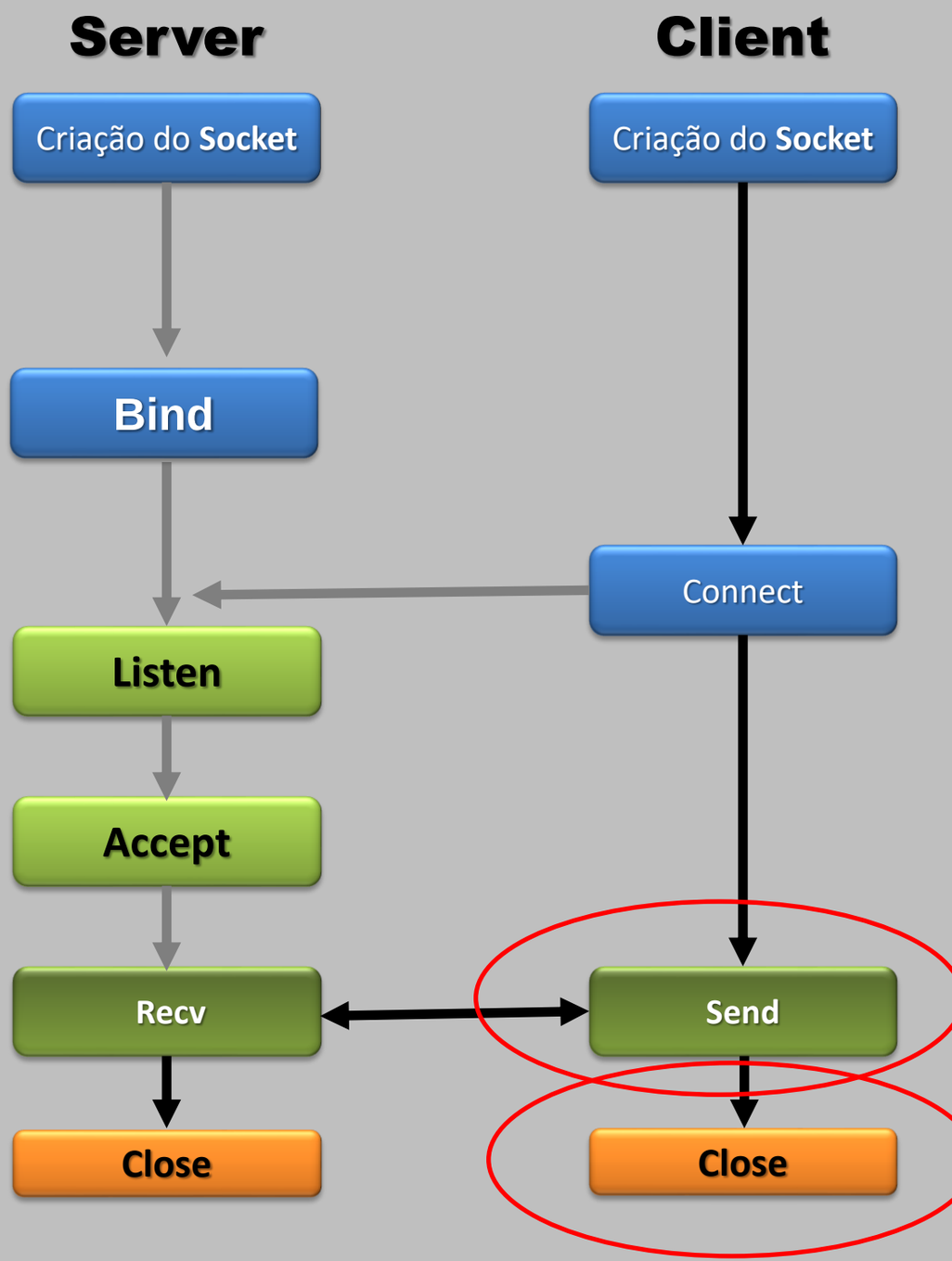
Código Cliente conecta()

```
    //prepara estrutura do cliente para conexão
    client.sin_addr.s_addr = inet_addr("127.0.0.1");
    client.sin_family = AF_INET;
    client.sin_port = htons(PORTA);

    //conecta o cliente
    if (connect(*conexao , (struct sockaddr *)&client, sizeof(client)) < 0)
     return false;

    //imprime mensagem
    puts("Conectado\n");

    return true;
}
```



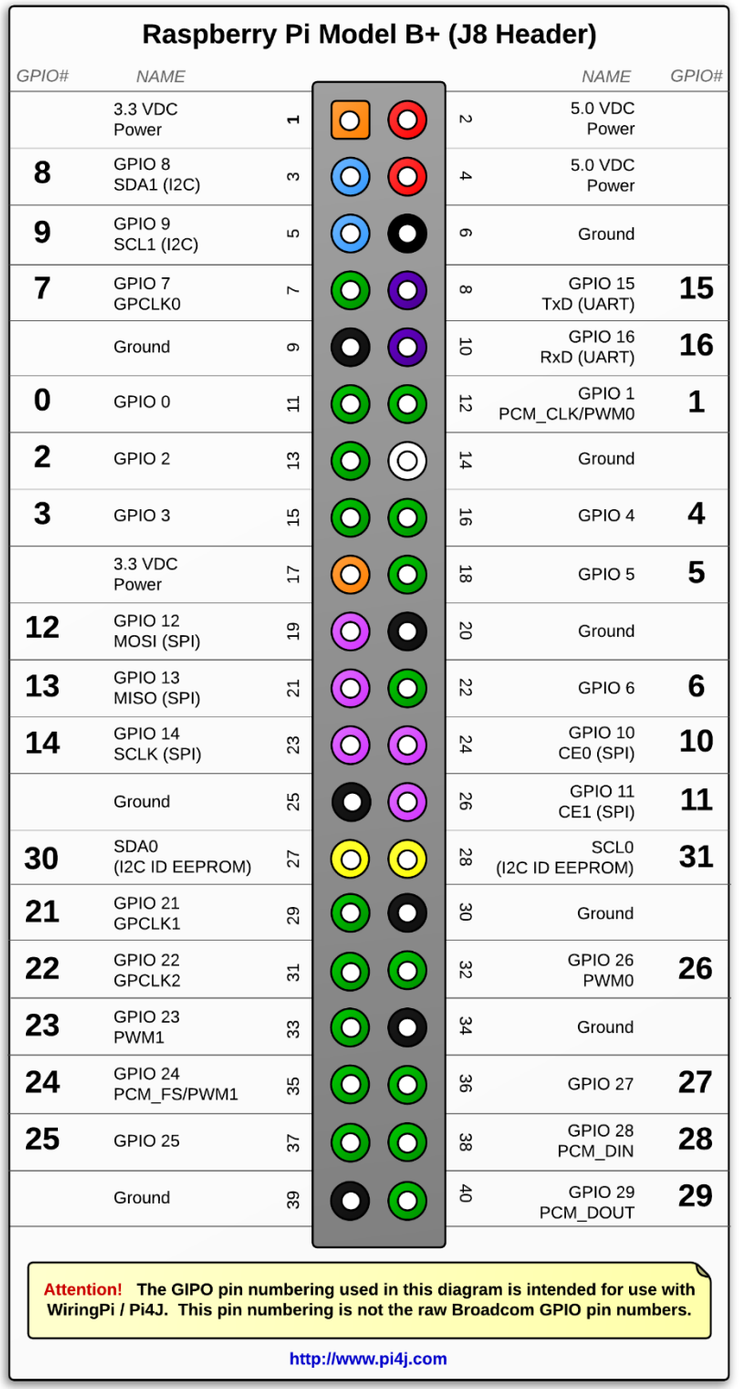
Código Cliente main()

```c
int main()
{
    int conexao;
    char msg[TF_MENSAGEM];

    //recebe mensagem
    puts("Digite a mensagem:\n[1] - Para sair\n");
    fflush(stdin);
    gets(msg);

    //envia mensagens até que "1" seja recebido
    while(strcmp(msg,"1")!=0)
    {
        //conecta cliente
        if(!conecta(&conexao))
        {
            puts("Erro de conexão");
            break;
        }
        //envia mensagem
        if(send(conexao ,&msg, TF_MENSAGEM, 0) < 0)
            puts("Falha no envio\n");
        else
            puts("Enviada!\n");

        puts("Digite a mensagem:\n[1] - Para sair\n");
        fflush(stdin);
        gets(msg);
    }
    //fecha o socket
    close(conexao);
    return 0;
}
```



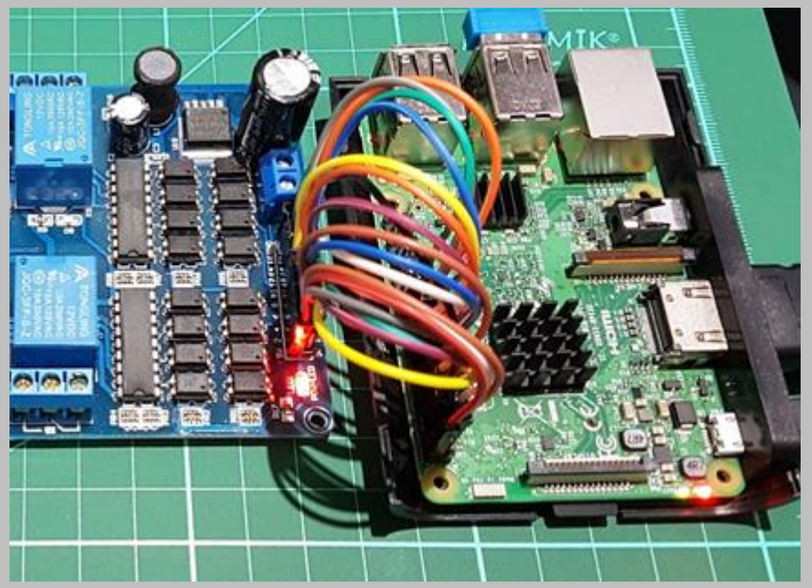
Tratamento de pinos (biblioteca wiringPi.h)

```
#define PINORELE_1    8
#define PINORELE_8    2
#define PINORELE_2    9
#define PINORELE_3    7
#define PINORELE_4    15
#define PINORELE_5    16
#define PINORELE_6    0
#define PINORELE_7    1
#define PINORELE_9    3
#define PINORELE_10   4
#define PINORELE_11   5
#define PINORELE_12   12
#define PINORELE_13   13
#define PINORELE_14   6
#define PINORELE_15   14
#define PINORELE_16   10
```



A sequência dos relés usados **neste exemplo** se dá a partir do pino 8 e até o pino 10, respeitando a seguinte sequência:

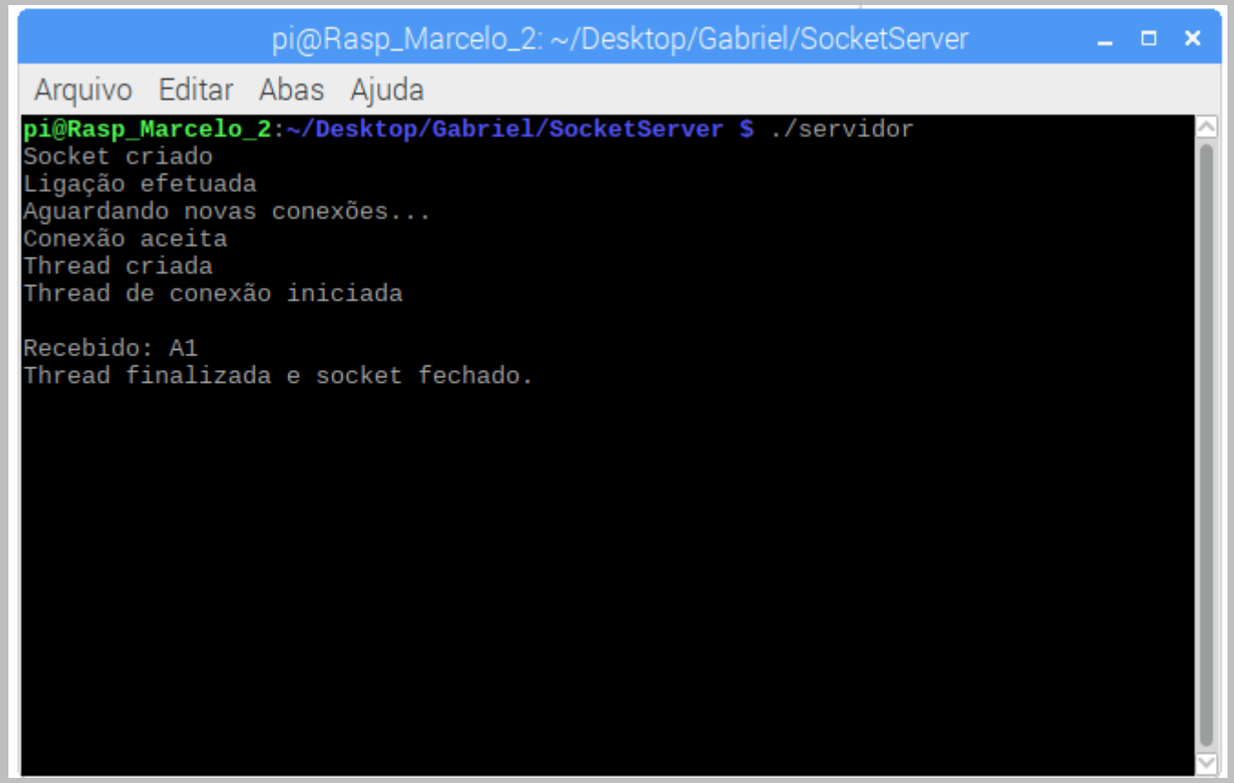
Ligação dos jumpers...



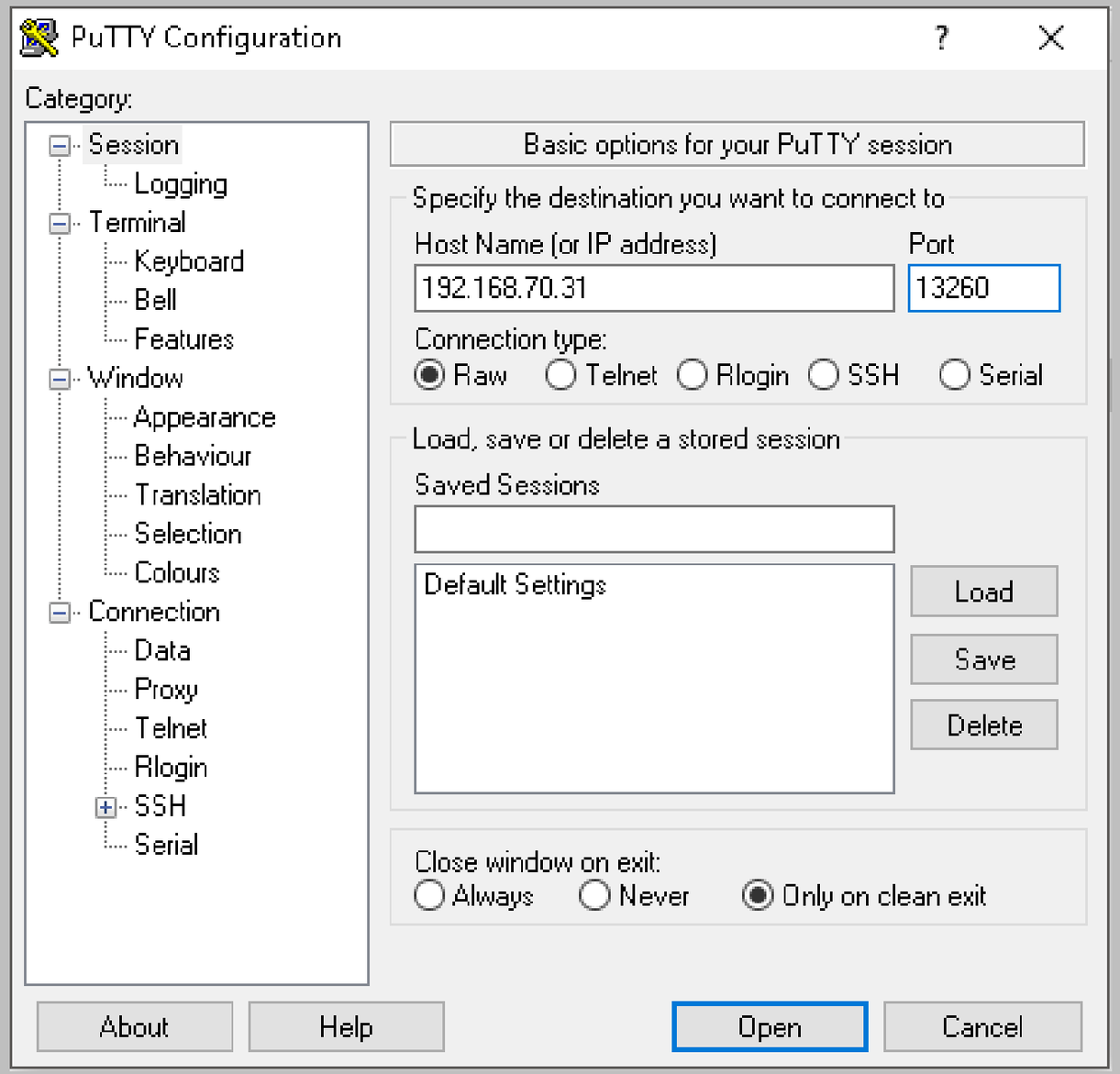
Pinagem Raspberry x Módulo		
Saída do Raspberry	Entrada do Módulo de Relés	Função
3	1	GPIO
5	2	GPIO
7	3	GPIO
8	4	GPIO
10	5	GPIO
11	6	GPIO
12	7	GPIO
13	8	GPIO
15	9	GPIO
16	10	GPIO
18	11	GPIO
18	12	GPIO
21	13	GPIO
22	14	GPIO
23	15	GPIO
24	16	GPIO
2	5 VDC	Alimentação (5 VDC)
6	GND	Ground

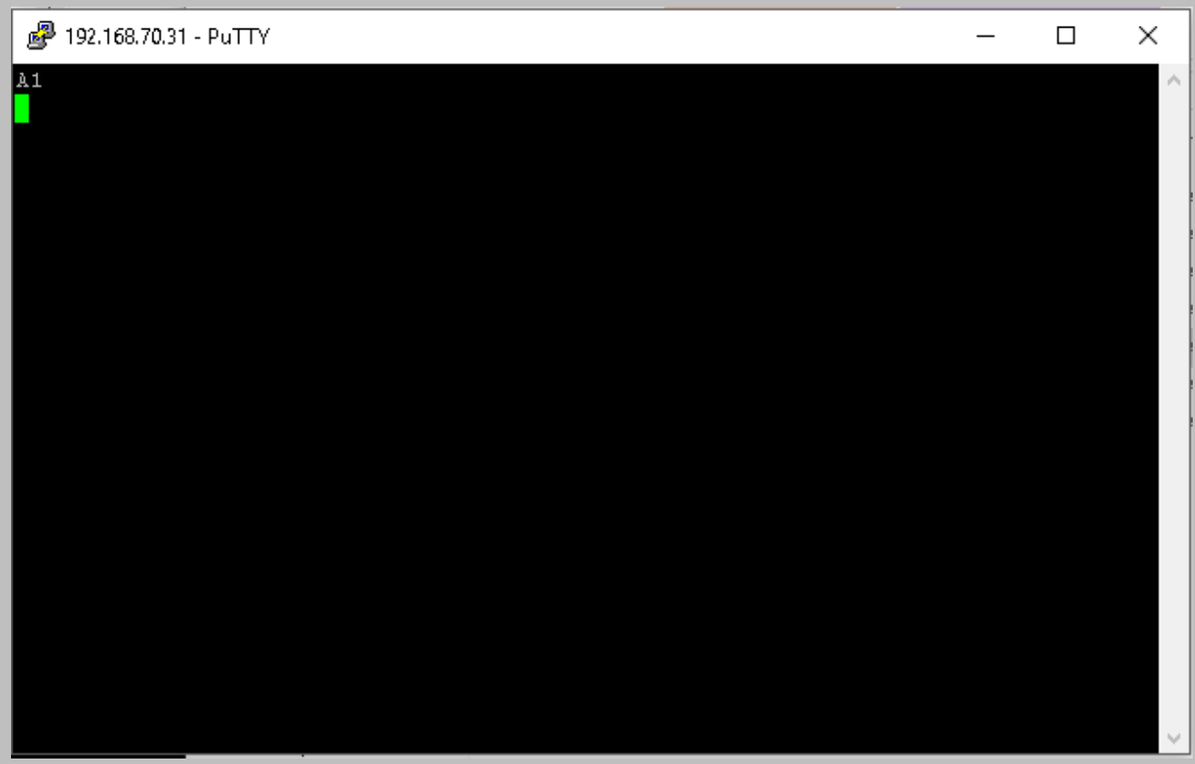
Iniciando o Servidor

Após compilar deve-se executar o programa com o comando “./servidor”, assim o servidor já será iniciado e todas as mensagens recebidas serão informadas no console



Enviando comandos pelo putty...





Digite o ip do Raspberry pi (para descobrir este ip digite ‘ifconfig’ no console) e a mesma porta utilizada no algorítmo.

Em www.fernandok.com

Download arquivos PDF e do código fonte