

Módulo de 16 relés com Raspberry Pi 3 utilizando Socket



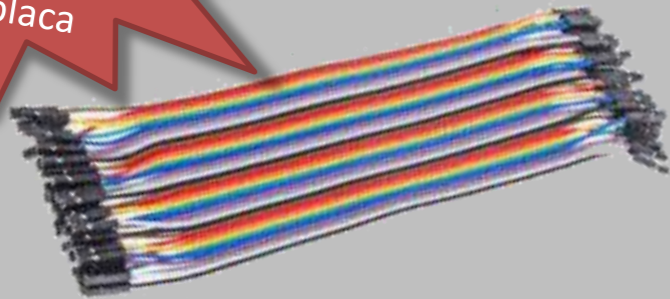
Por Fernando Koyanagi

Recursos usados

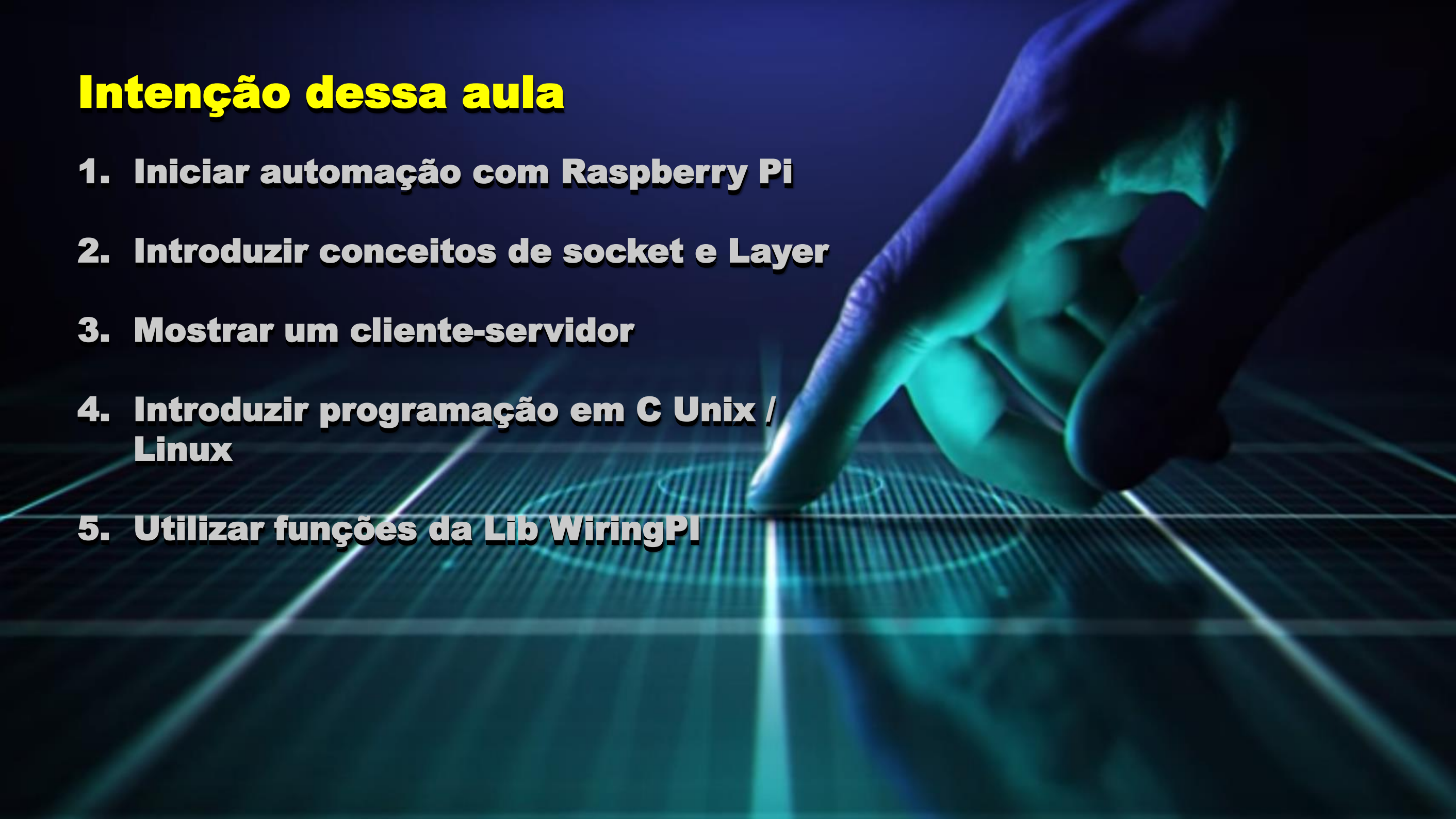
- 18 jumpers fêmea x fêmea
- Módulo de 16 relés com optoacoplador
- Raspberry Pi 3
- Rede interna (TCP/IP)



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**
- 
- A hand is shown in a dark, futuristic environment, pointing towards a glowing digital interface. The interface consists of a grid of lines and a circular highlight, suggesting a selection or focus point. The overall color scheme is dark blue and green, with bright highlights from the interface elements.

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

| # | Label | Command | Working directory | Reset |
|---|-----------------------|-----------------------------------|-------------------|-------|
| C commands | | | | |
| 1. | Compile | gcc -Wall -c "%f" -lwiringPi | | |
| 2. | Build | gcc -Wall -o "%e" "%f" -lwiringPi | | |
| 3. | | | | |
| | | Error regular expression: | | |
| Independent commands | | | | |
| 1. | Make | make | | |
| 2. | Make Custom Target... | make | | |
| 3. | Make Object | make %e.o | | |
| 4. | | | | |
| | | Error regular expression: | | |
| <i>Note: Item 2 opens a dialogue and appends the response to the command.</i> | | | | |
| Execute commands | | | | |
| 1. | Execute | sudo "%e" | | |
| 2. | | | | |

%d, %e, %f, %p are substituted in command and directory fields, see manual for details.

Cancel OK

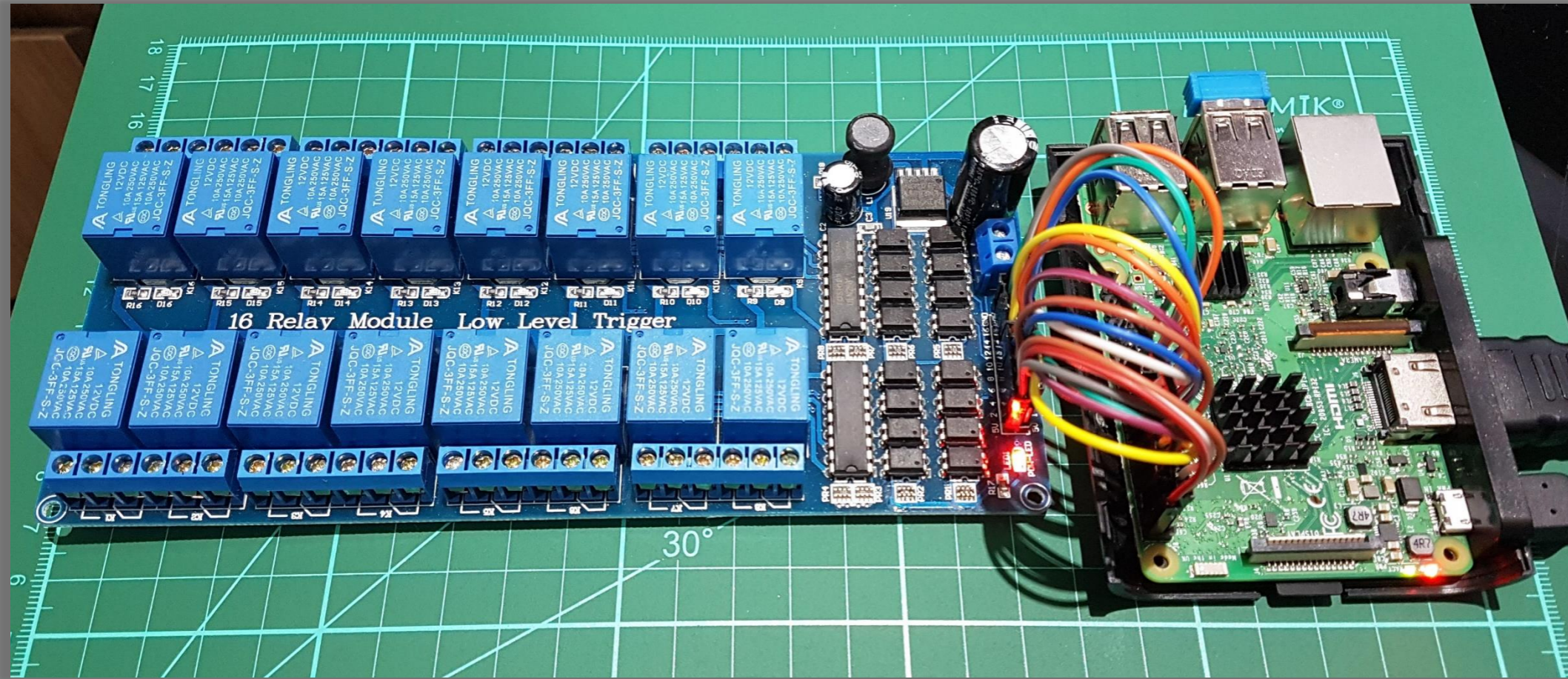


Programas Client.c e Server.c



**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...





Em www.fernandok.com

Seu e-mail



- PRINCIPAL
- SOBRE FERNANDO K
- ARDUINO
- ESP8266
- ESP32
- LORAWAN
- MOTOR
- DISPLAY
- MATERIAIS
- DOWNLOAD

Receba o meu conteúdo GRATUITAMENTE

Insira aqui seu melhor email...

QUERO RECEBER GRÁTIS



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



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 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

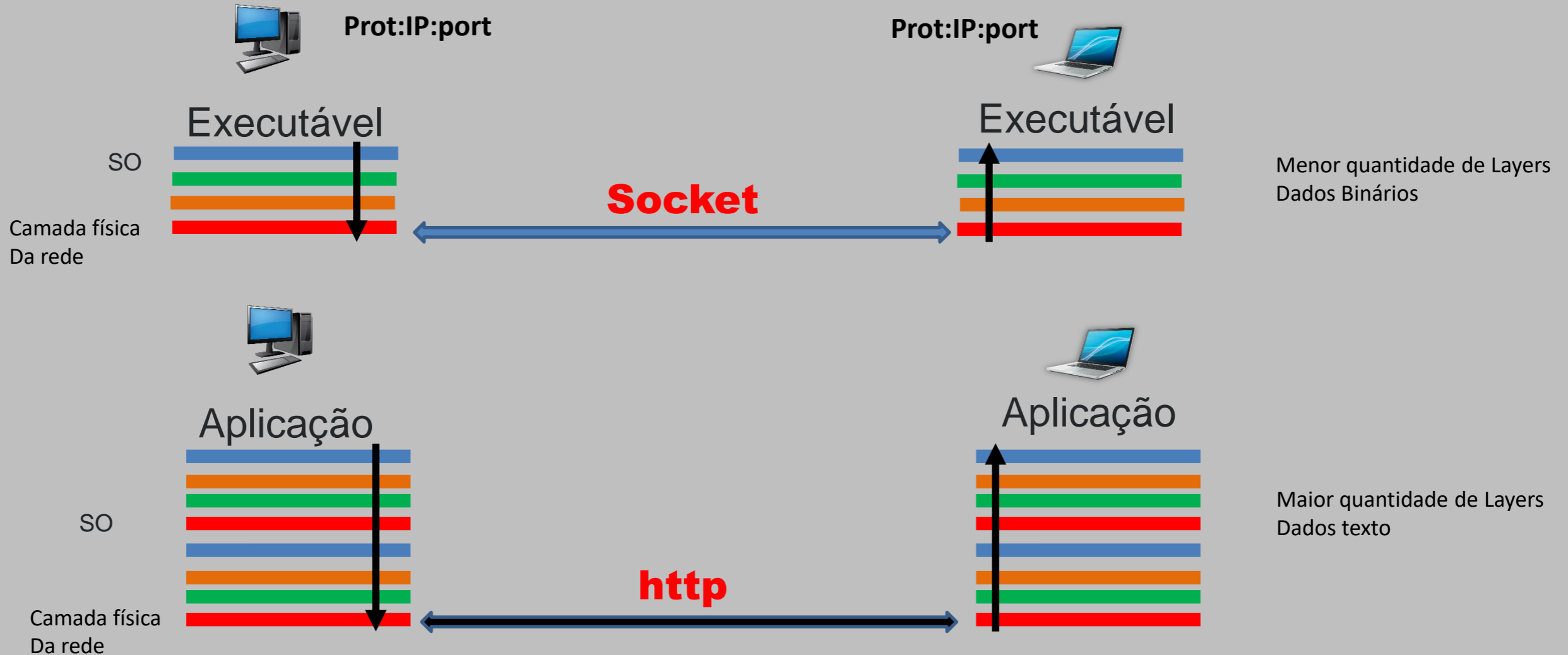
Votos até o momento: 32
Dias restantes para votar: 49

FACEBOOK

Fernando K Tec
16.235 curtidas

Socket ou http ?

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



Socket ou HTTP ?

Performance ?

Segurança ?

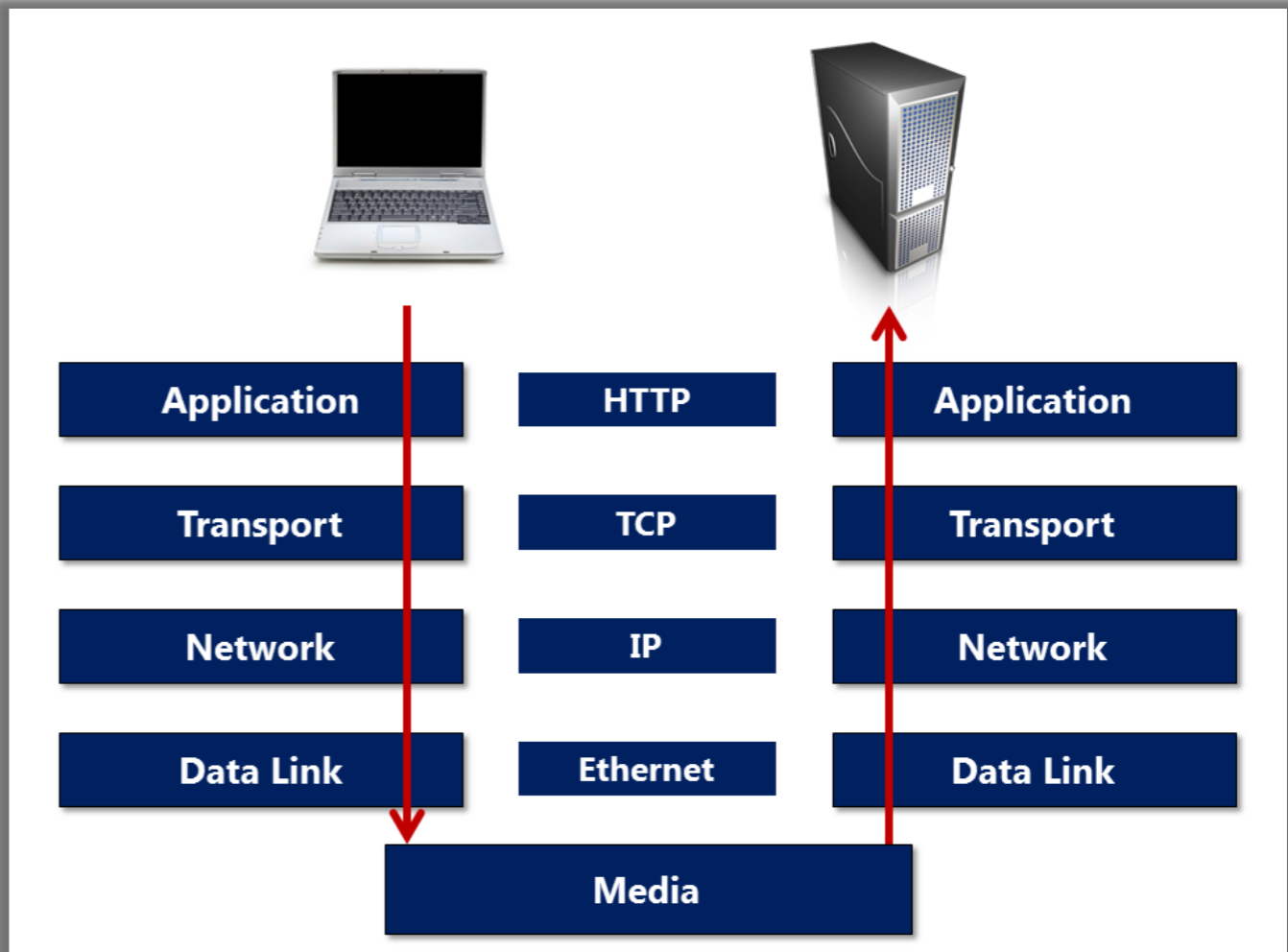
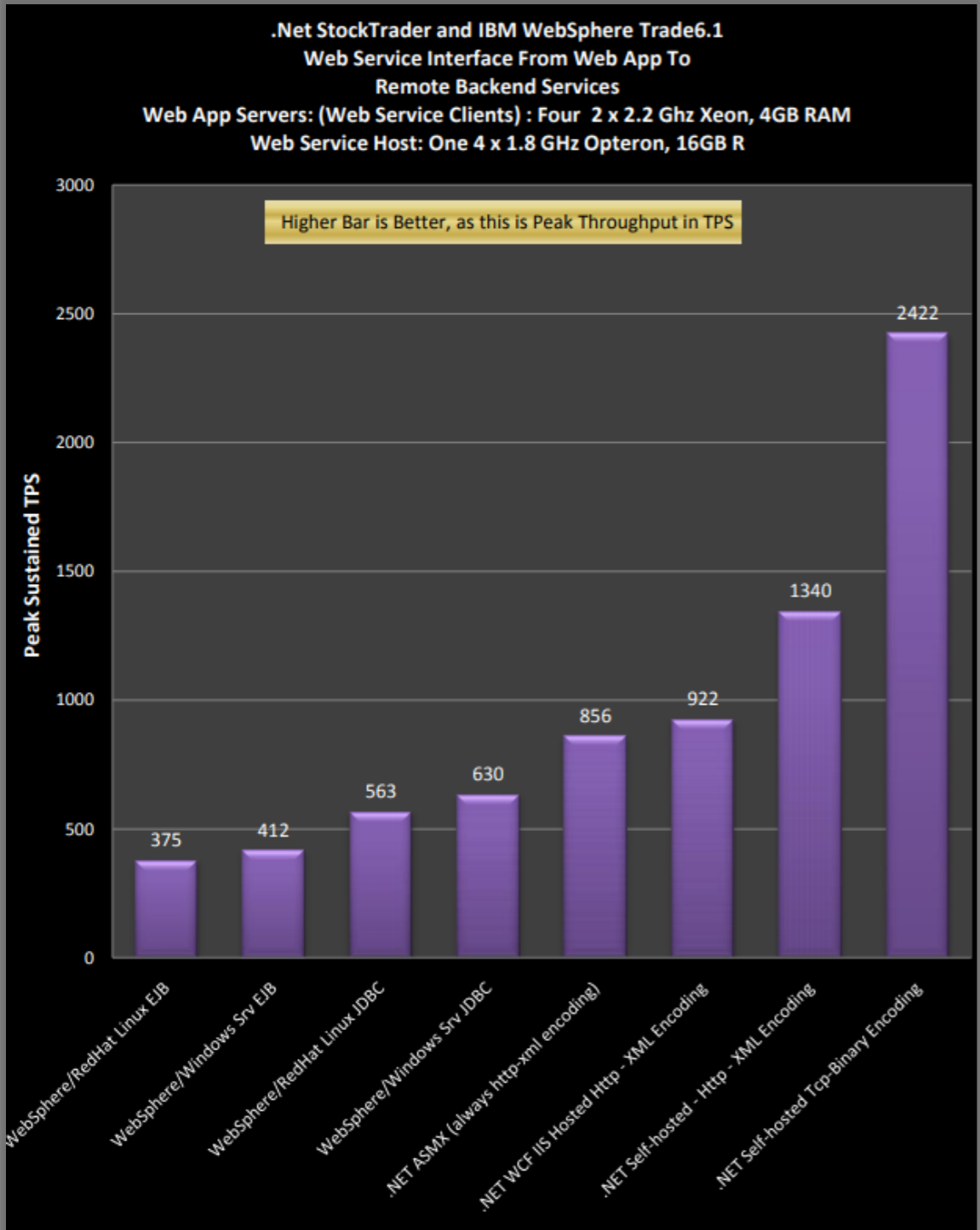
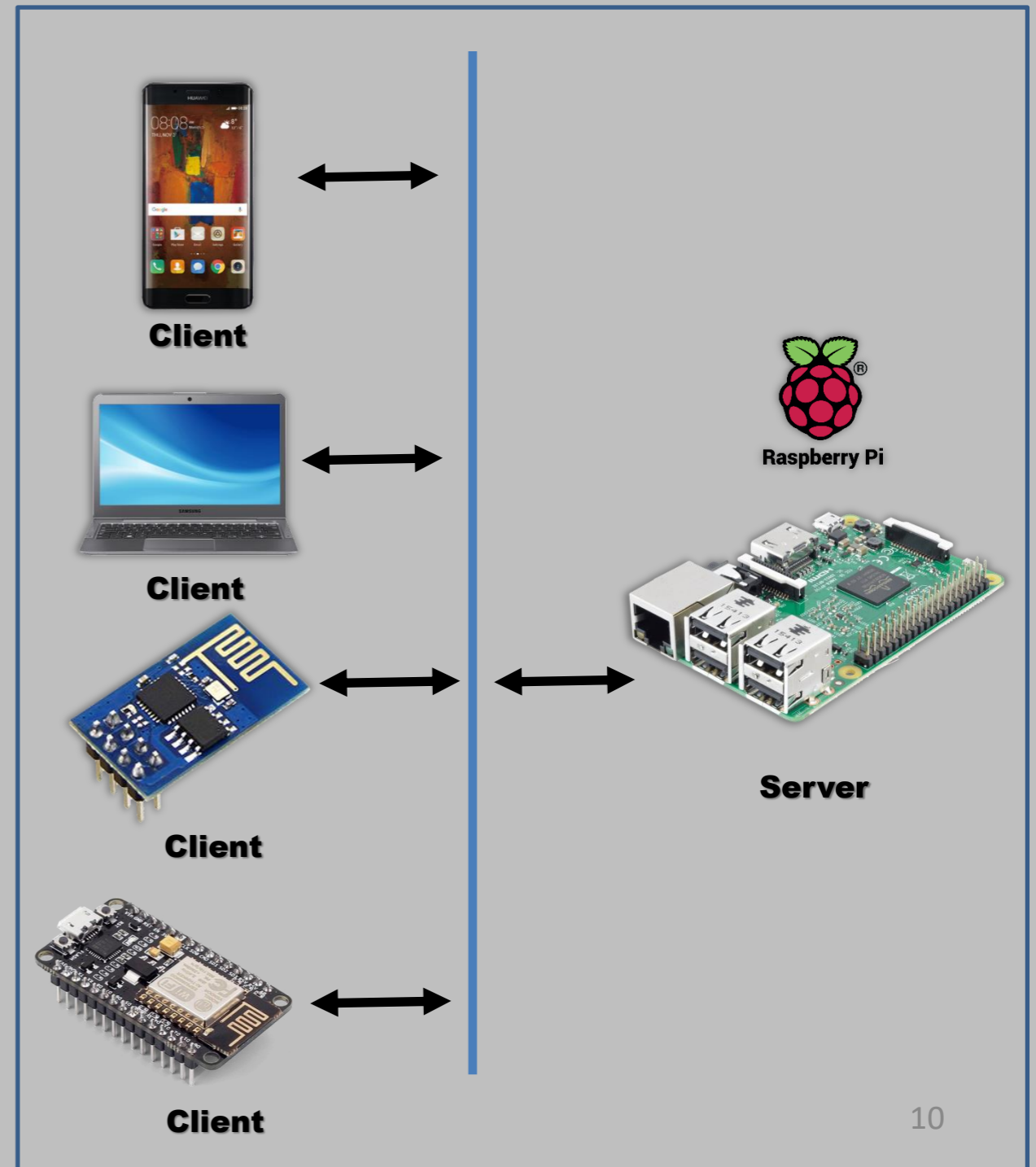
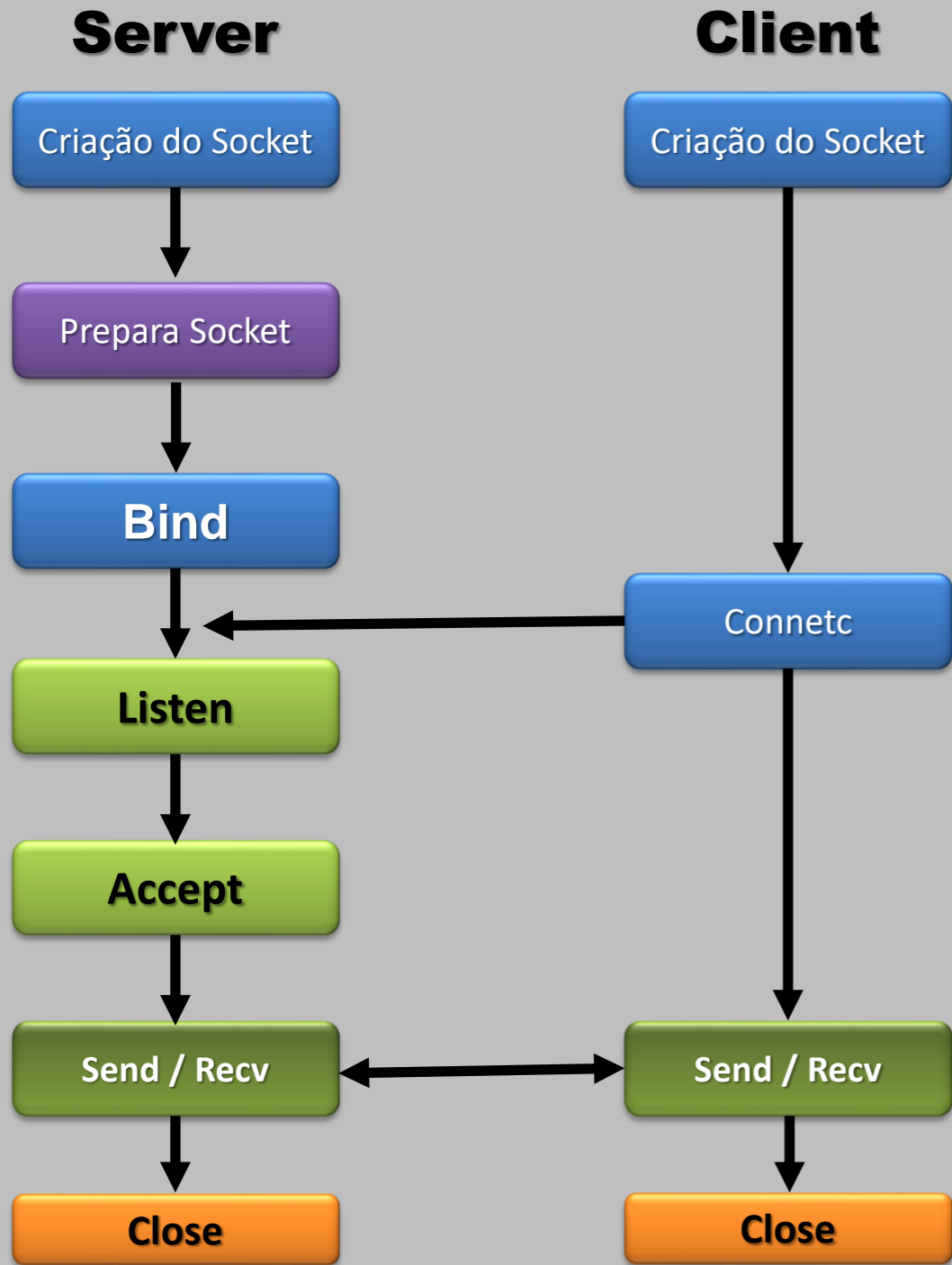
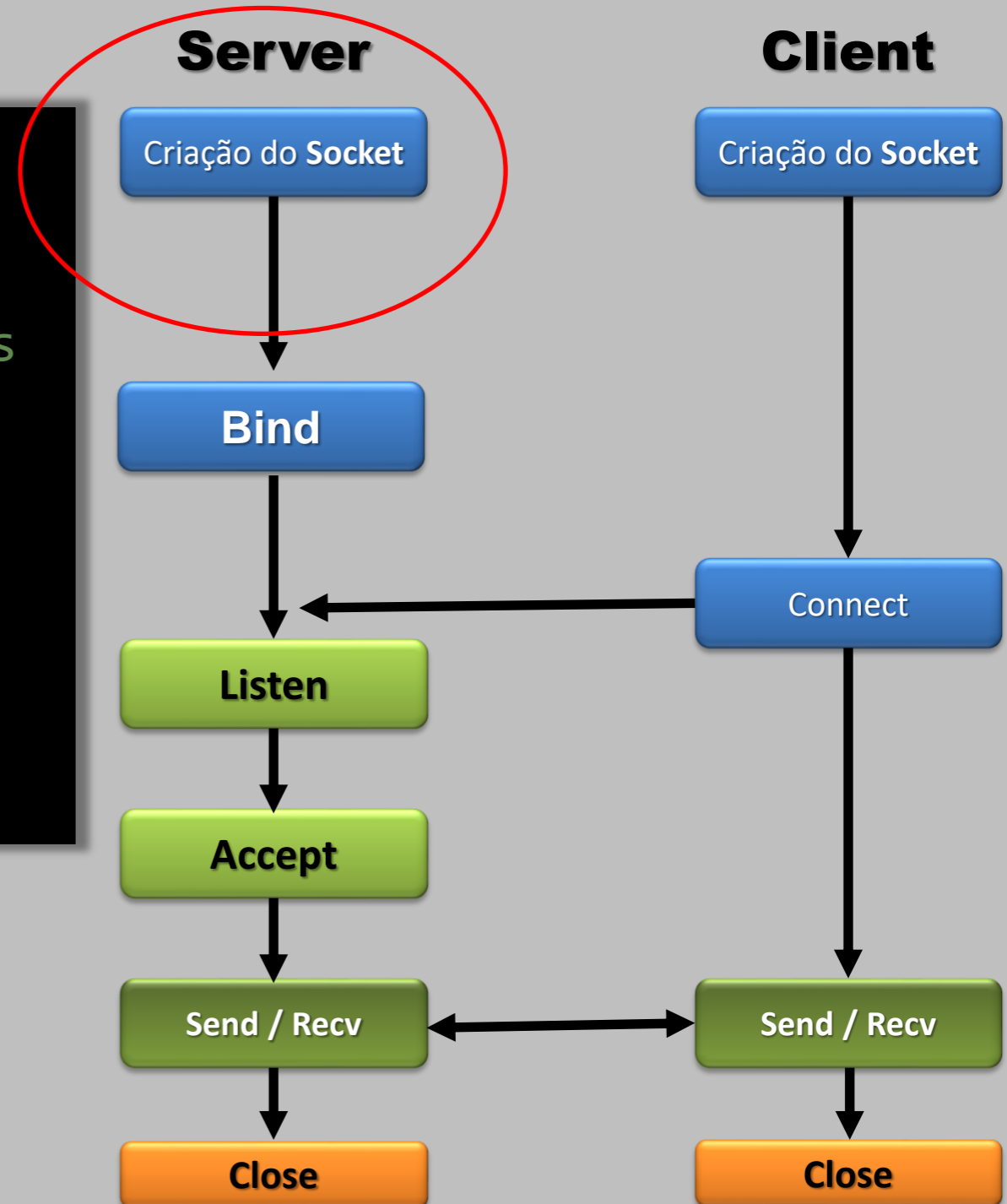


Diagrama de estado do modelo Client / Server



Código Servidor main()

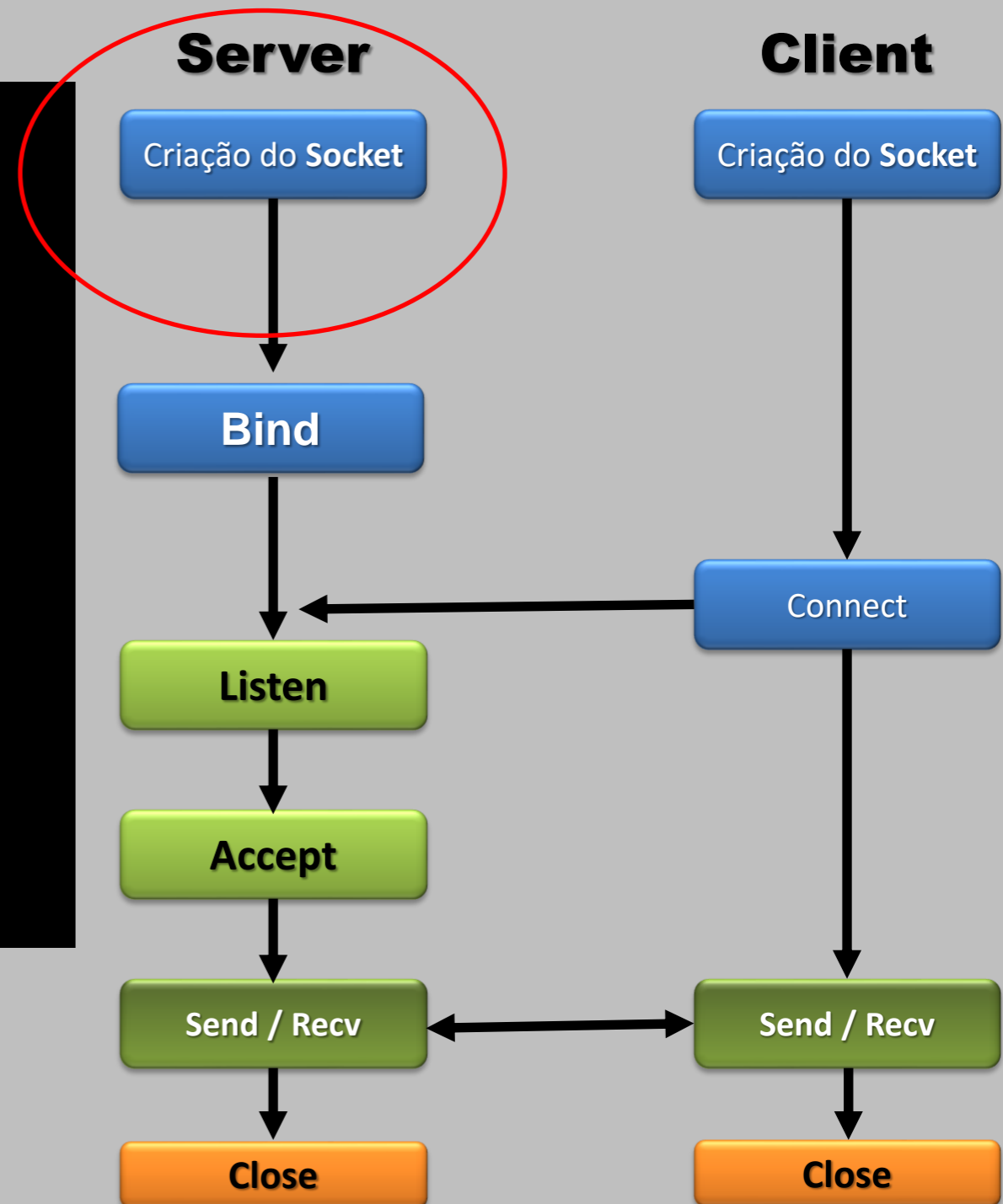
```
int main(int argc , char *argv[])
{
    int socket_servidor;
    setaPinos(); //seta os pinos dos reles
    //cria e prepara o socket
    if(!preparaSocket(&socket_servidor))
        return 0;
    iniciaServidor(socket_servidor);
    //inicia o servidor e executa
    um loop de recepção de mensagens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



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);
}
```

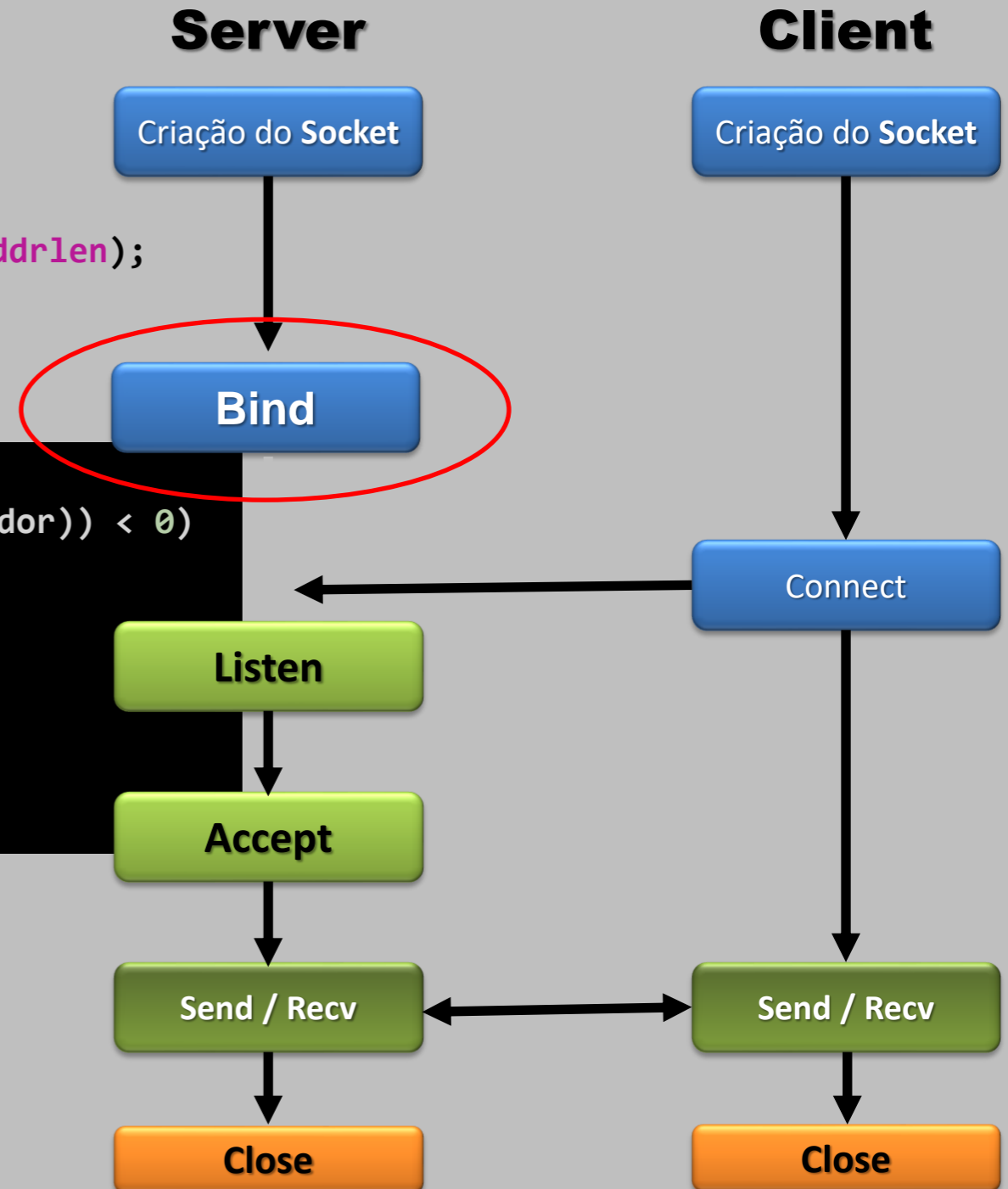
domínio: inteiro, domínio de comunicação, por exemplo, protocolo AF_INET (protocolo IPv4), AF_INET6 (protocolo IPv6) Quando **INADDR_ANY** é especificado na chamada de ligação, o soquete será vinculado a *todas* as interfaces locais



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 ligação");  
    return false;  
}  
puts("Ligação efetuada");  
return true;
```



Código Servidor

void iniciaServidor(int socket_servidor)

```
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(1);
    *novo_socket = socket_cliente;
  }
}
```

Server

Criação do Socket

Bind

Listen

Accept

Send / Recv

Close

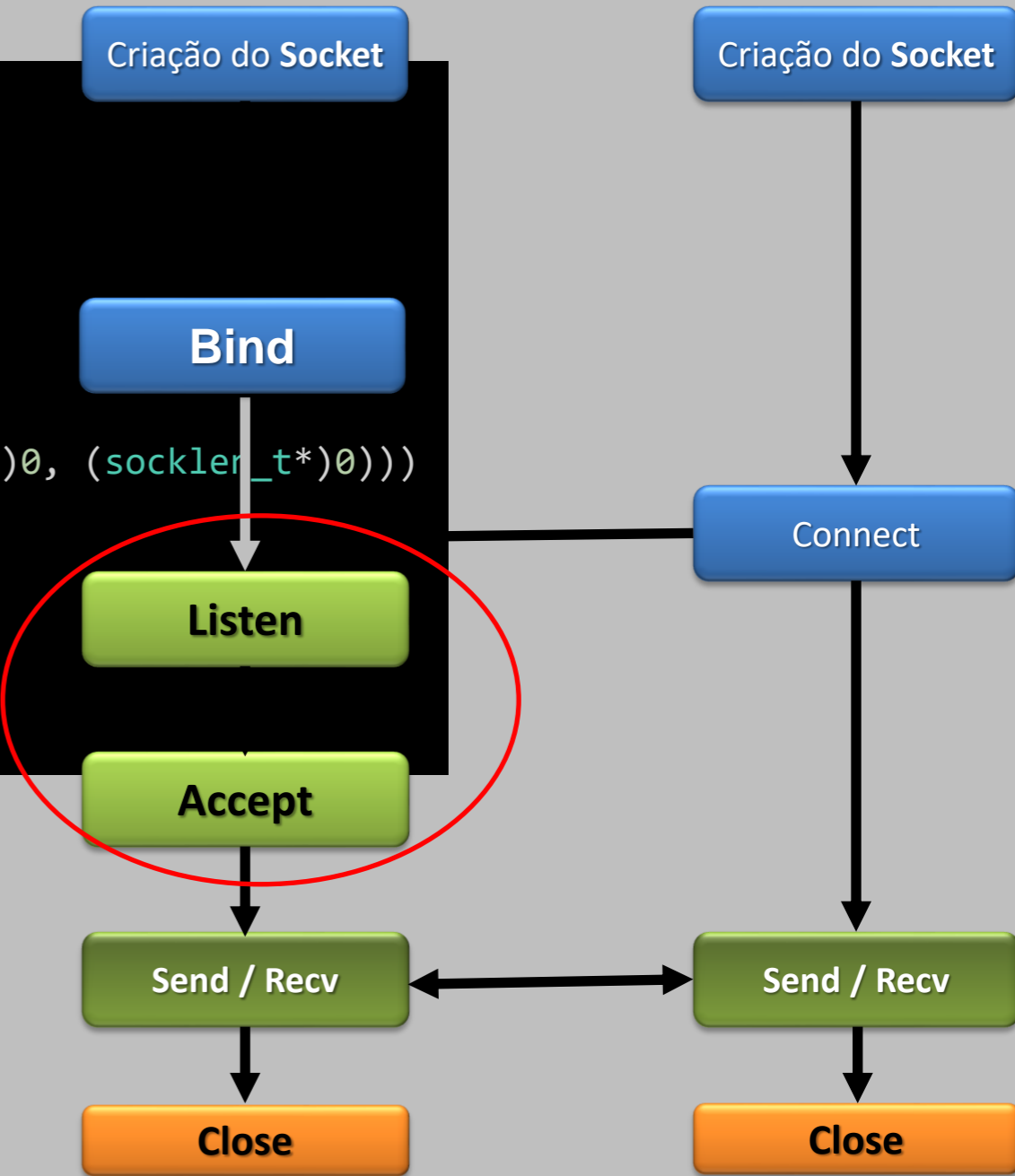
Client

Criação do Socket

Connect

Send / Recv

Close



Código Servidor

void iniciaServidor(int socket_servidor)

```
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");
    }
}
```

Server

Criação do Socket

Bind

Listen

Accept

Send / Recv

Close

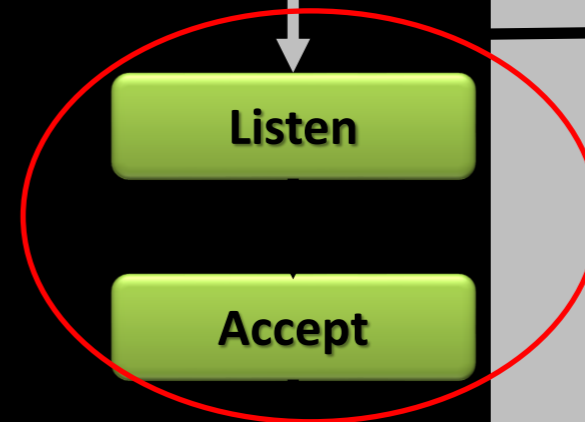
Client

Criação do Socket

Connect

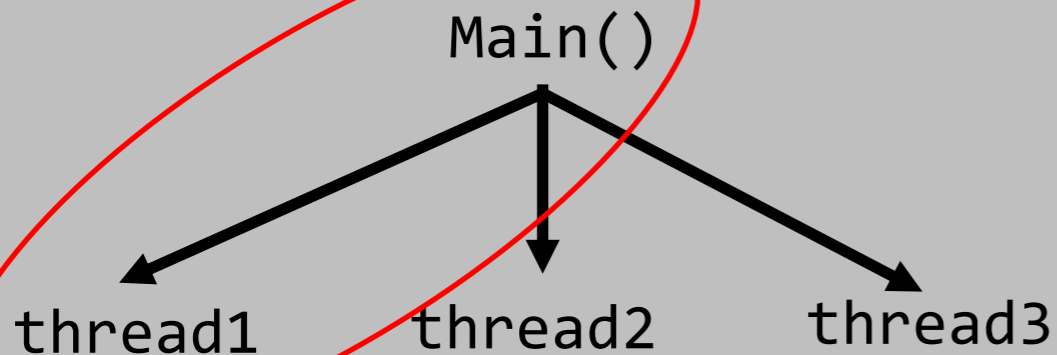
Send / Recv

Close



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)
```

Server

Criação do Socket

Bind

Listen

Accept

Recv

Close

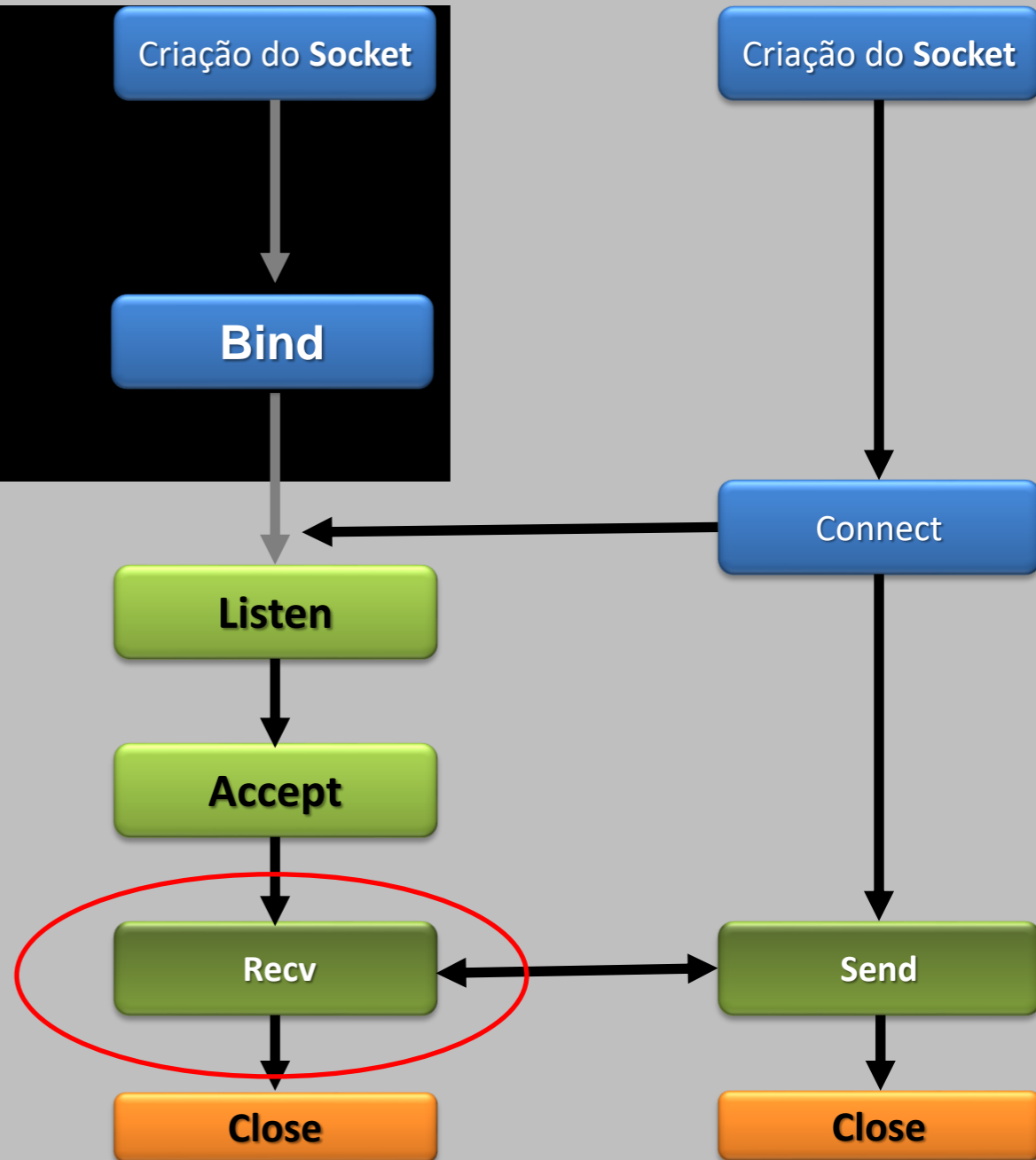
Client

Criação do Socket

Connect

Send

Close



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

Server

Criação do Socket

Bind

Listen

Accept

Recv

Close

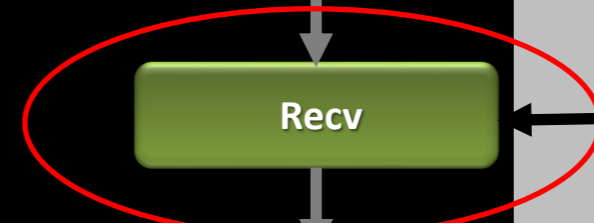
Client

Criação do Socket

Connect

Send

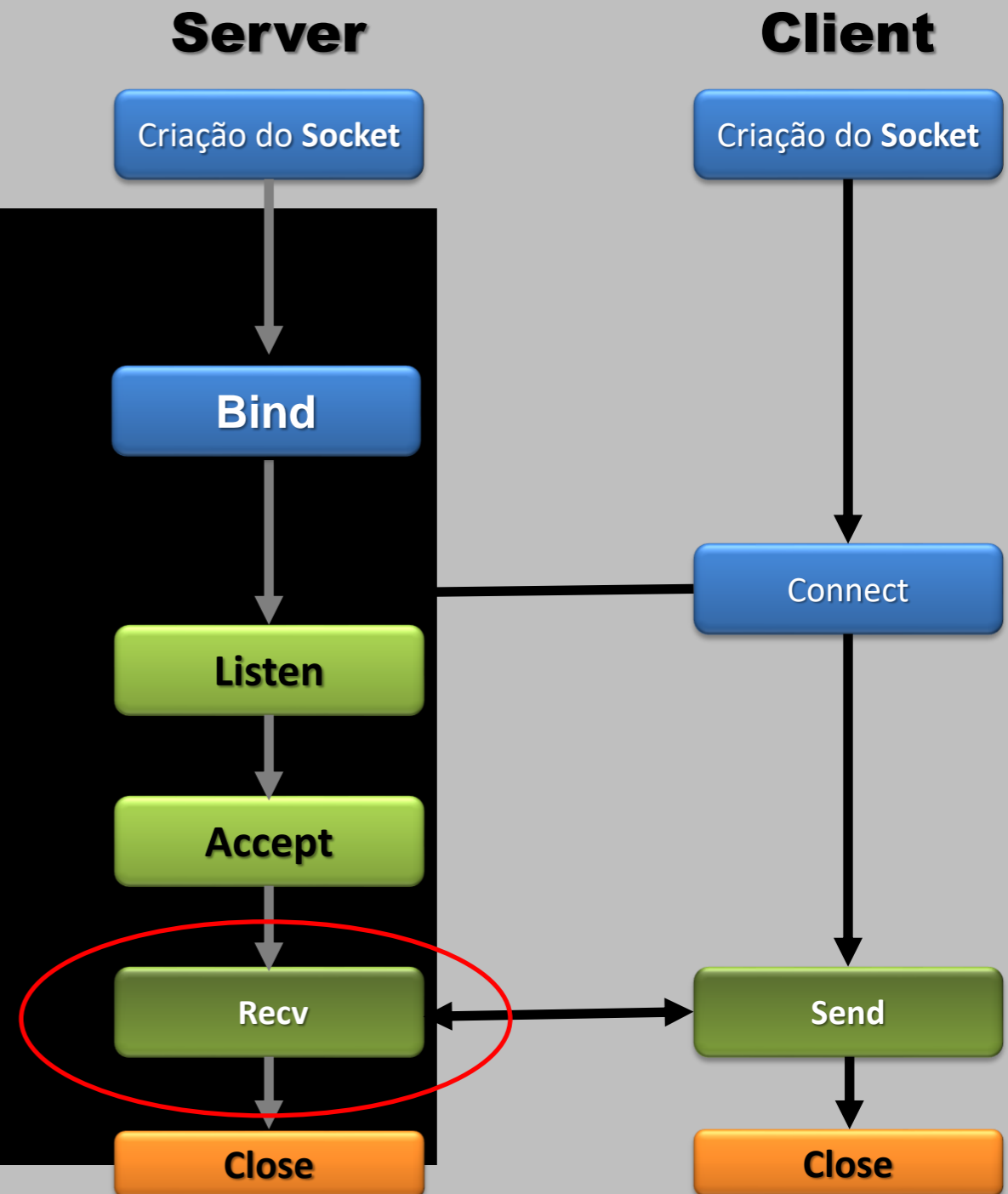
Close



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;
}
```

Server

Criação do Socket

Bind

Listen

Accept

Recv

Close

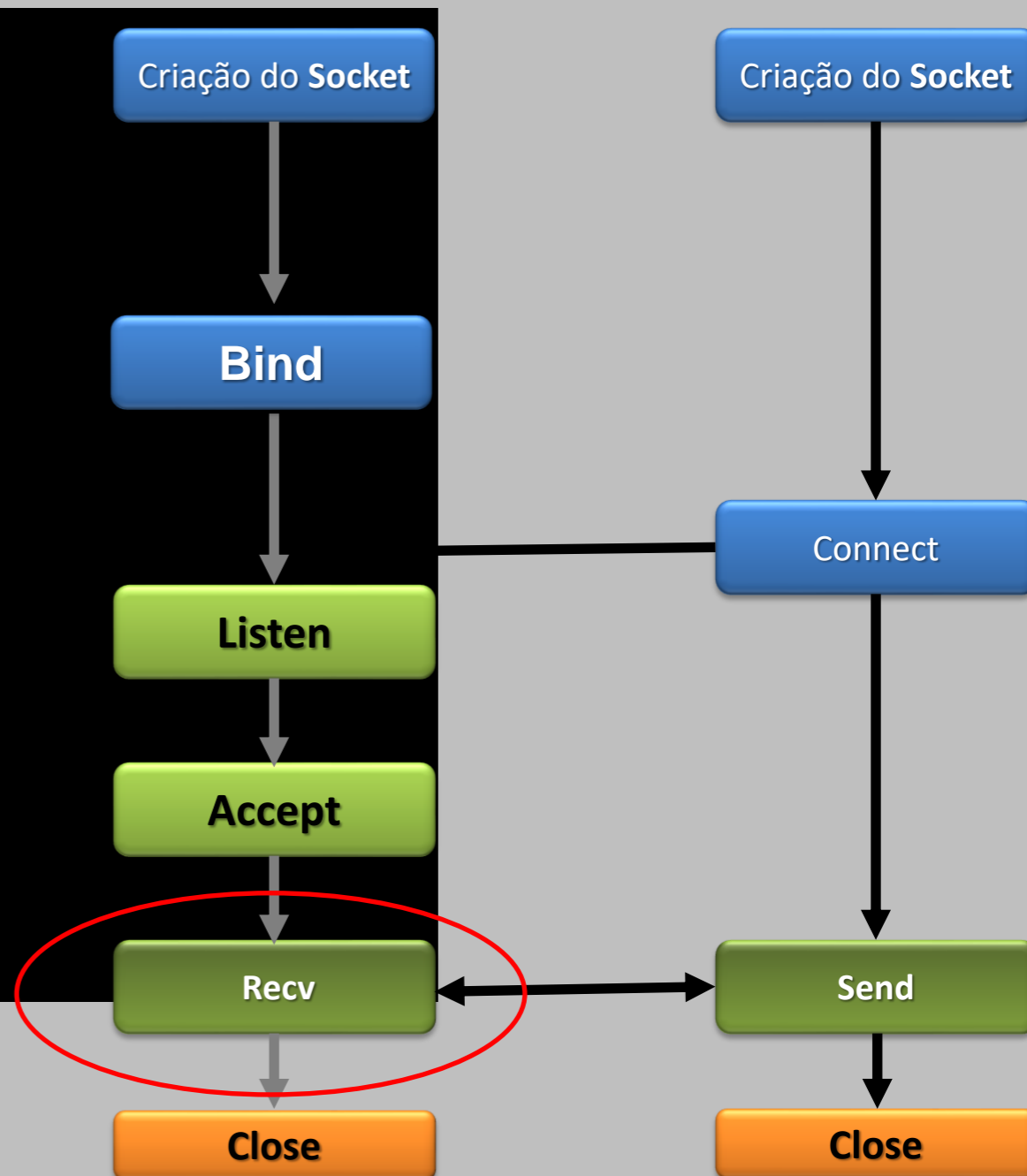
Client

Criação do Socket

Connect

Send

Close



Código Servidor

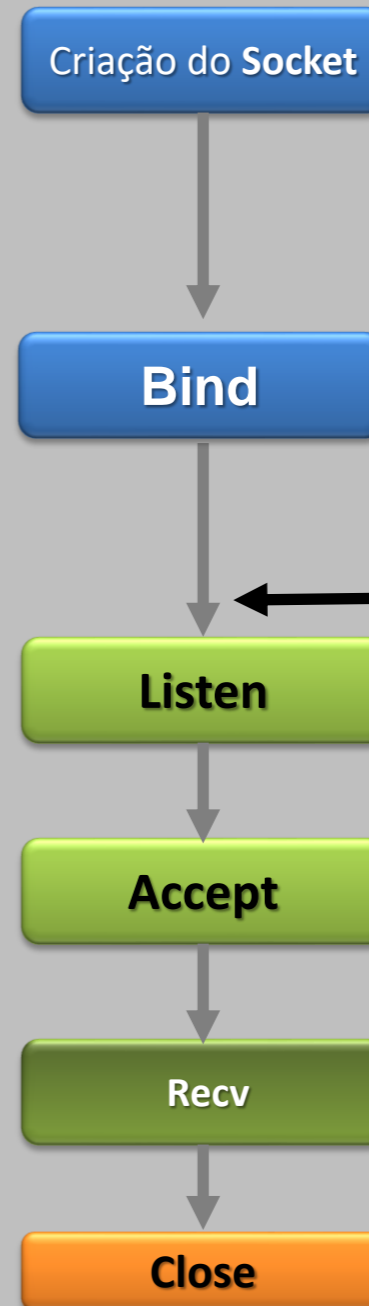
```
void ativaEmSequencia()
```

```
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;

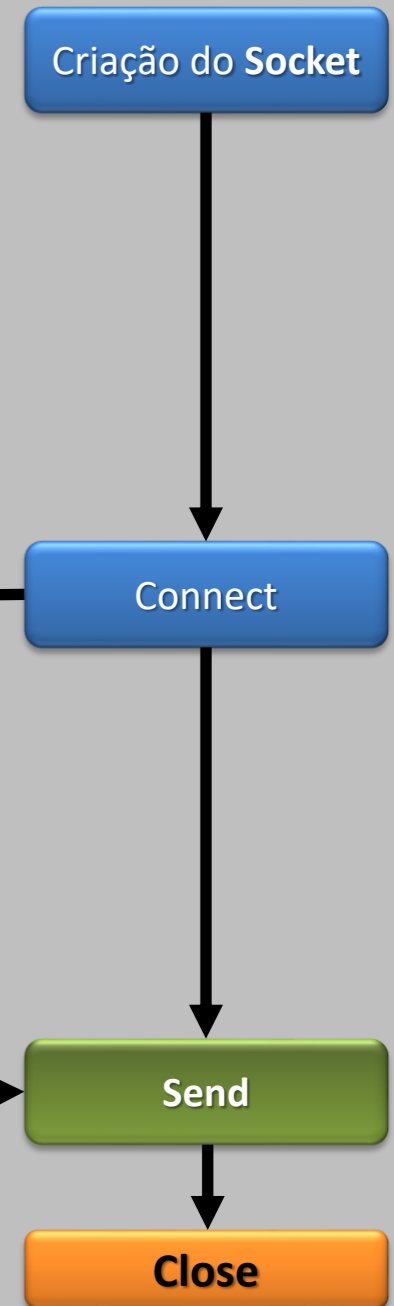
  for(int i=0; i<16; i++)
  {
    digitalWrite(vet[i],LOW);
    delay(200);
    digitalWrite(vet[i],HIGH);
    delay(200);
  }
}
```

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

Server



Client



Código Servidor

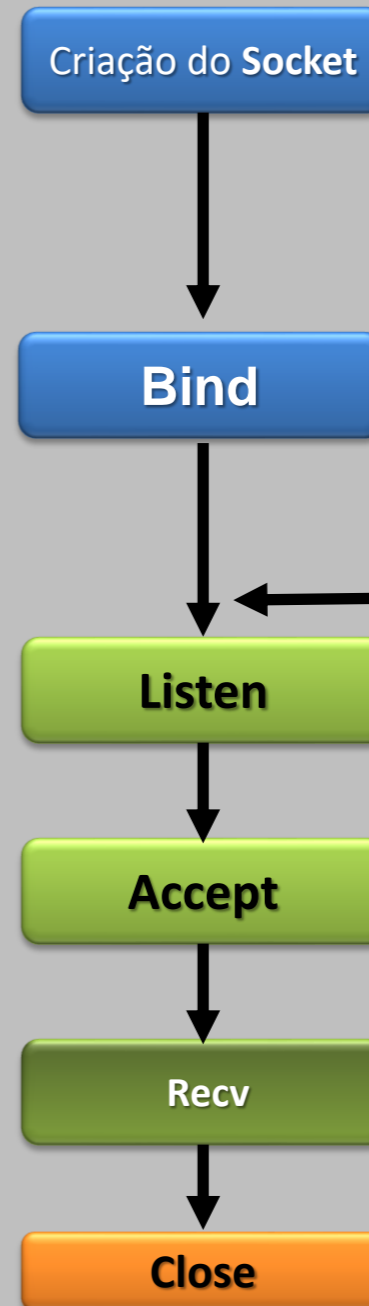
```
void ativaEmSequencia()
```

```
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;

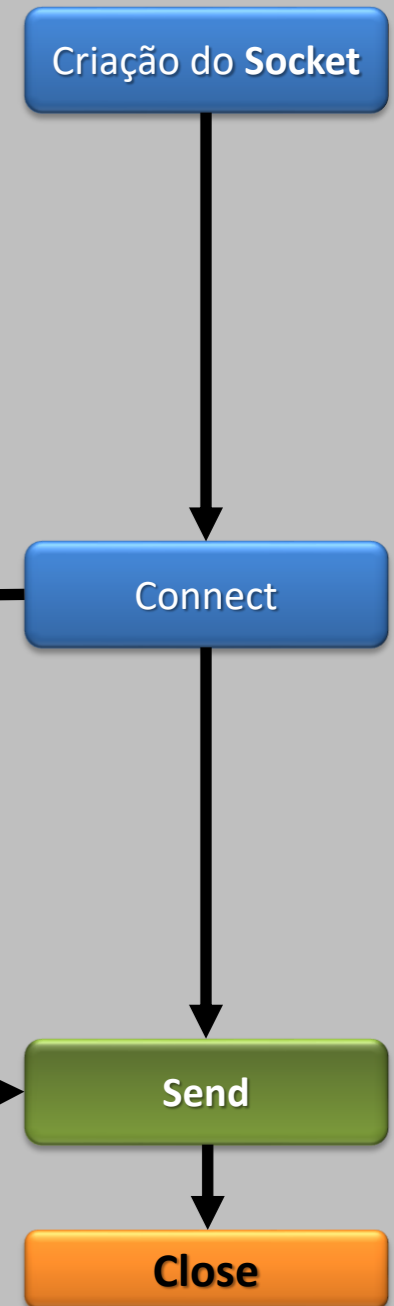
  for(int i=0; i<16; i++)
  {
    digitalWrite(vet[i],LOW);
    delay(200);
    digitalWrite(vet[i],HIGH);
    delay(200);
  }
}
```

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

Server



Client



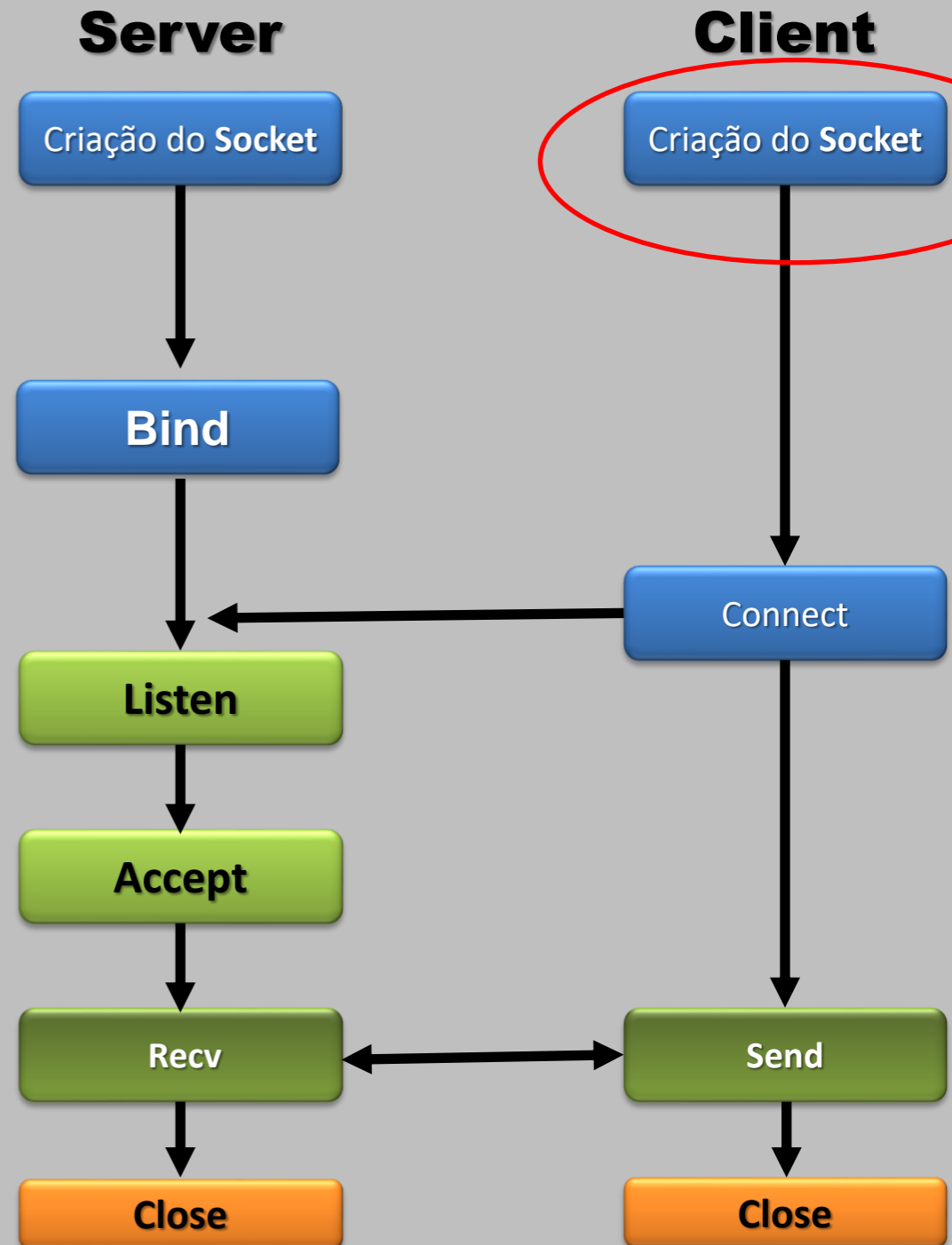
Código do Client

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;
}
```

Server

Criação do Socket

Bind

Listen

Accept

Recv

Close

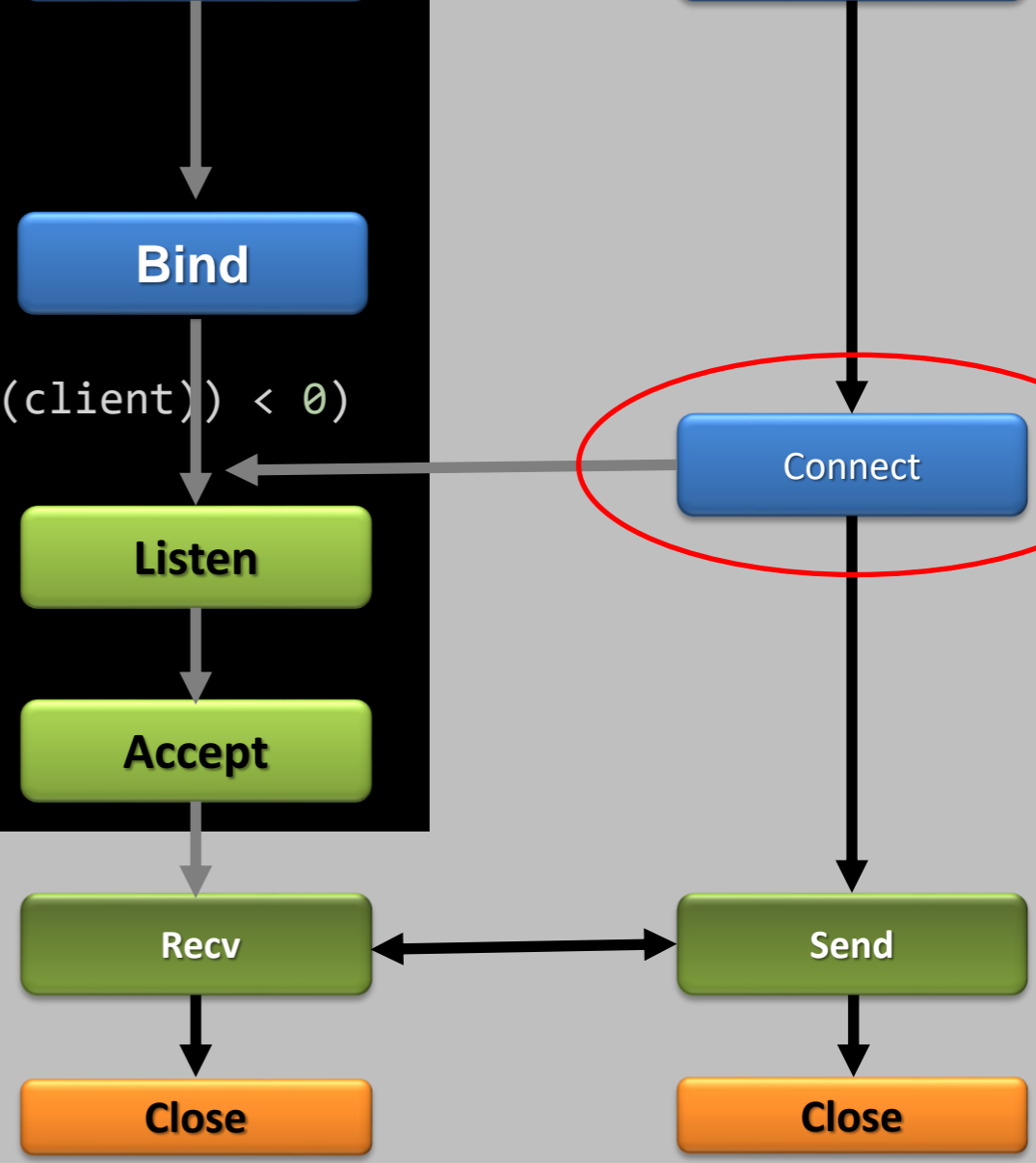
Client

Criação do Socket

Connect

Send

Close



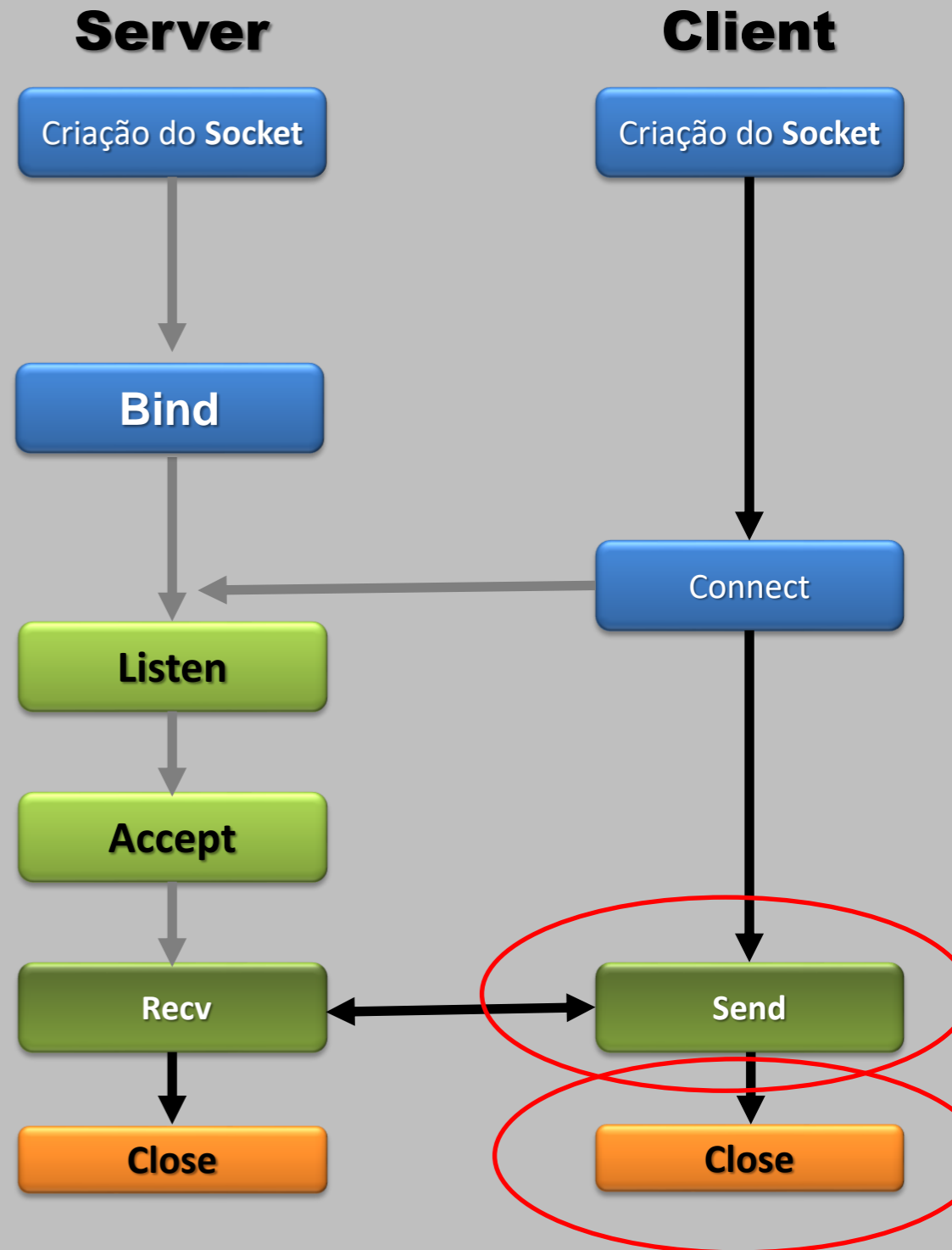
Código Cliente main()

```
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
```

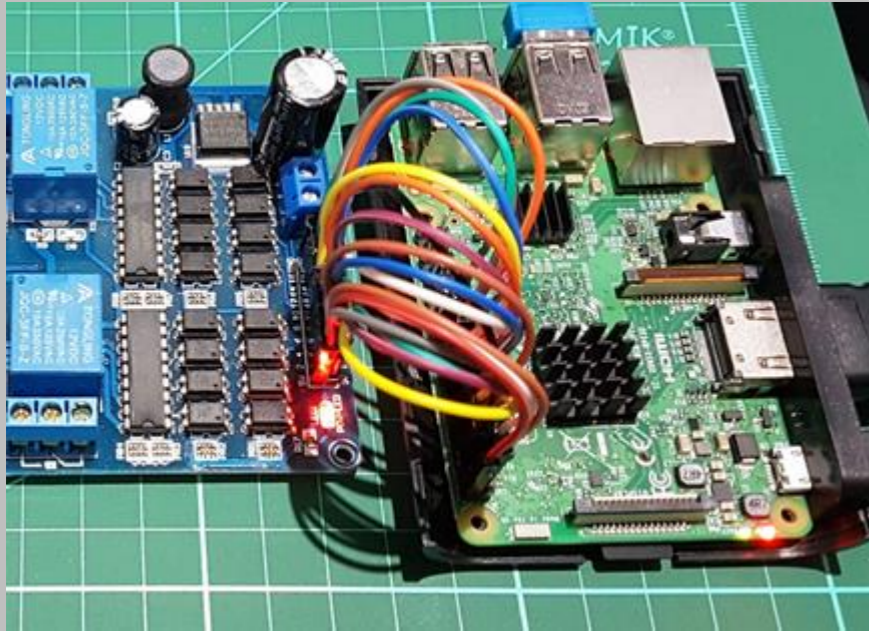
| Raspberry Pi Model B+ (J8 Header) | | | | | |
|-----------------------------------|----------------------|----|---|------|-------------------------|
| GPIO# | NAME | | | NAME | GPIO# |
| | 3.3 VDC Power | 1 |  | 2 | 5.0 VDC Power |
| 8 | GPIO 8 SDA1 (I2C) | 3 |  | 4 | 5.0 VDC Power |
| 9 | GPIO 9 SCL1 (I2C) | 5 |  | 6 | Ground |
| 7 | GPIO 7 GPCLK0 | 7 |  | 8 | GPIO 15 TxD (UART) 15 |
| | Ground | 9 |  | 10 | GPIO 16 RxD (UART) 16 |
| 0 | GPIO 0 | 11 |  | 12 | GPIO 1 PCM_CLK/PWM0 1 |
| 2 | GPIO 2 | 13 |  | 14 | Ground |
| 3 | GPIO 3 | 15 |  | 16 | GPIO 4 4 |
| | 3.3 VDC Power | 17 |  | 18 | GPIO 5 5 |
| 12 | GPIO 12 MOSI (SPI) | 19 |  | 20 | Ground |
| 13 | GPIO 13 MISO (SPI) | 21 |  | 22 | GPIO 6 6 |
| 14 | GPIO 14 SCLK (SPI) | 23 |  | 24 | GPIO 10 CE0 (SPI) 10 |
| | Ground | 25 |  | 26 | GPIO 11 CE1 (SPI) 11 |
| 30 | SDA0 (I2C ID EEPROM) | 27 |  | 28 | SCL0 (I2C ID EEPROM) 31 |
| 21 | GPIO 21 GPCLK1 | 29 |  | 30 | Ground |
| 22 | GPIO 22 GPCLK2 | 31 |  | 32 | GPIO 26 PWM0 26 |
| 23 | GPIO 23 PWM1 | 33 |  | 34 | Ground |
| 24 | GPIO 24 PCM_FS/PWM1 | 35 |  | 36 | GPIO 27 27 |
| 25 | GPIO 25 | 37 |  | 38 | GPIO 28 PCM_DIN 28 |
| | Ground | 39 |  | 40 | GPIO 29 PCM_DOUT 29 |

Attention! The GPIO pin numbering used in this diagram is intended for use with WiringPi / Pi4J. This pin numbering is not the raw Broadcom GPIO pin numbers.

<http://www.pi4j.com>

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

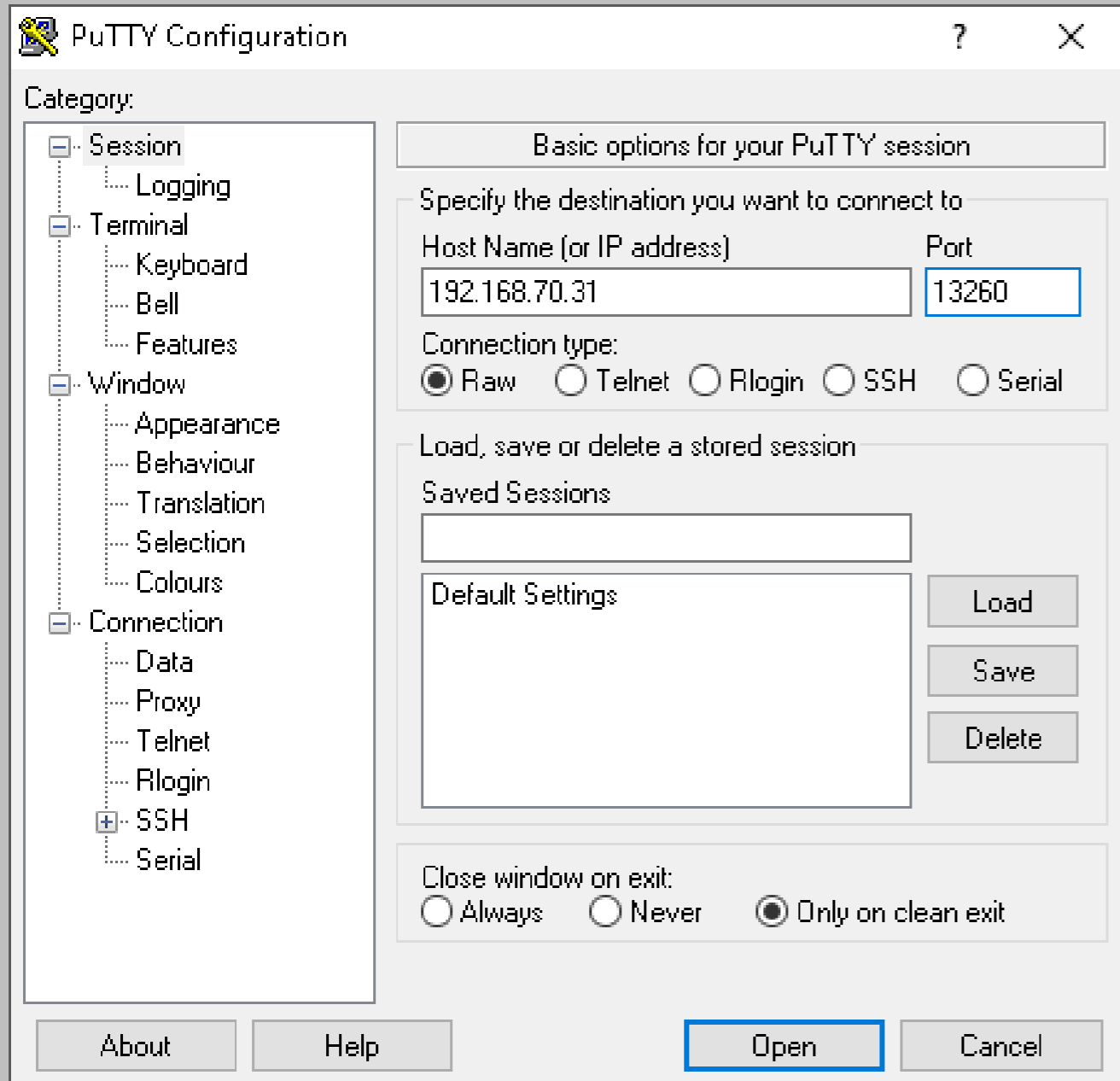
Após compilar deve-se executar o programa com o comando “./servidor”, assim o servidor já será iniciado e todas as mensagens recebidas serão informadas no console

A terminal window titled "pi@Rasp_Marcelo_2: ~/Desktop/Gabriel/SocketServer" with a menu bar containing "Arquivo", "Editar", "Abas", and "Ajuda". The terminal shows the command `./servidor` being executed, followed by several status messages: "Socket criado", "Ligação efetuada", "Aguardando novas conexões...", "Conexão aceita", "Thread criada", and "Thread de conexão iniciada". After a short delay, it displays "Recebido: A1" and "Thread finalizada e socket fechado.".

```
pi@Rasp_Marcelo_2: ~/Desktop/Gabriel/SocketServer
Arquivo  Editar  Abas  Ajuda
pi@Rasp_Marcelo_2:~/Desktop/Gabriel/SocketServer $ ./servidor
Socket criado
Ligação efetuada
Aguardando novas conexões...
Conexão aceita
Thread criada
Thread de conexão iniciada

Recebido: A1
Thread finalizada e socket fechado.
```

Enviando comandos pelo putty...



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

Em www.fernandok.com

Download arquivos PDF e do código fonte

