

Sistemas Operacionais Embarcados

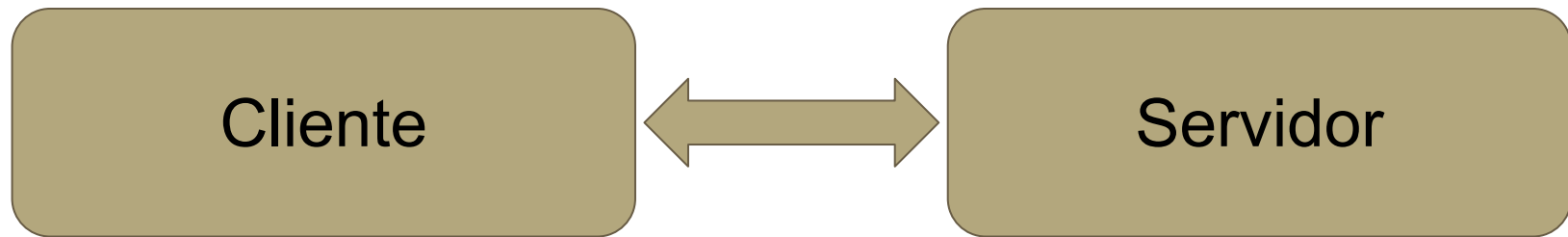
Sockets

Sockets

- Extensão ao pipe, usando o modelo cliente/servidor
- Criada em 1983 na Universidade de Berkeley
- Permite a comunicação entre processos tanto através de rede de computadores quanto internamente em uma mesma máquina
- “Embutido” em sistemas operacionais baseados em UNIX (tornou-se público e livre de licenças em 1989 pela UC Berkeley)
- Em outros sistemas operacionais, são implementados em bibliotecas

Sockets

- Comunicação cliente/servidor



- **Socket ativo:** inicia a comunicação

- Precisa saber o endereço do servidor

- **Socket passivo:** aguarda o início da comunicação

- Ao receber o pedido de um cliente, já toma conhecimento do endereço deste

Tipos de comunicação em rede*

- **Stream:** semelhante a uma ligação telefônica
 1. Um lado começa a comunicação, o outro aceita
 2. Ambos os lados se reconhecem: “olá” na ligação telefônica, SYN/ACK no protocolo TCP
 3. Informações são trocadas
 4. Ambos os lados reconhecem o fim da ligação: “tchau” na ligação, FIN/ACK em TCP
 5. Se um dos lados não reconhecer o fim da ligação, o outro lado pode reiniciar a conexão
 6. A informação chegará na ordem em que foi enviada, e (provavelmente) sem falhas

*<https://stackoverflow.com/questions/4688855/whats-the-difference-between-streams-and-datagrams-in-network-programming>

Tipos de comunicação em rede*

- **Stream:** semelhante a uma ligação telefônica
 1. Um lado começa a comunicação, o outro aceita
 2. Ambos os lados concordam com a comunicação, SYN/ACK no protocolo
 3. Informação é enviada
 4. Ambos os lados concordam com a comunicação, FIN/ACK e a comunicação é encerrada
 5. Se um dos lados não responde, o outro lado pode reiniciar a conexão
 6. A informação chegará na ordem em que foi enviada, e (provavelmente) sem falhas

Stream é usado quando a informação deve chegar intacta e na ordem certa (p.ex., *download* de arquivos)

*<https://stackoverflow.com/questions/468855/whats-the-difference-between-streams-and-datagrams-in-network-programming>

Tipos de comunicação em rede*

- **Datagrama:** semelhante a passar um recado escrito para uma pessoa distante em sala de aula
 1. O recado passa de pessoa em pessoa
 2. O recado pode não chegar
 3. O recado pode ser modificado no meio do caminho
 4. Se dois recados idênticos forem enviados, um pode chegar antes do outro

*<https://stackoverflow.com/questions/4688855/whats-the-difference-between-streams-and-datagrams-in-network-programming>

Tipos de comunicação em rede*

- **Datagrama:** semelhante a passar um recado escrito para uma pessoa distante em sala de aula

1. O recado passa de pessoa em pessoa

2. O recado

3. O recado

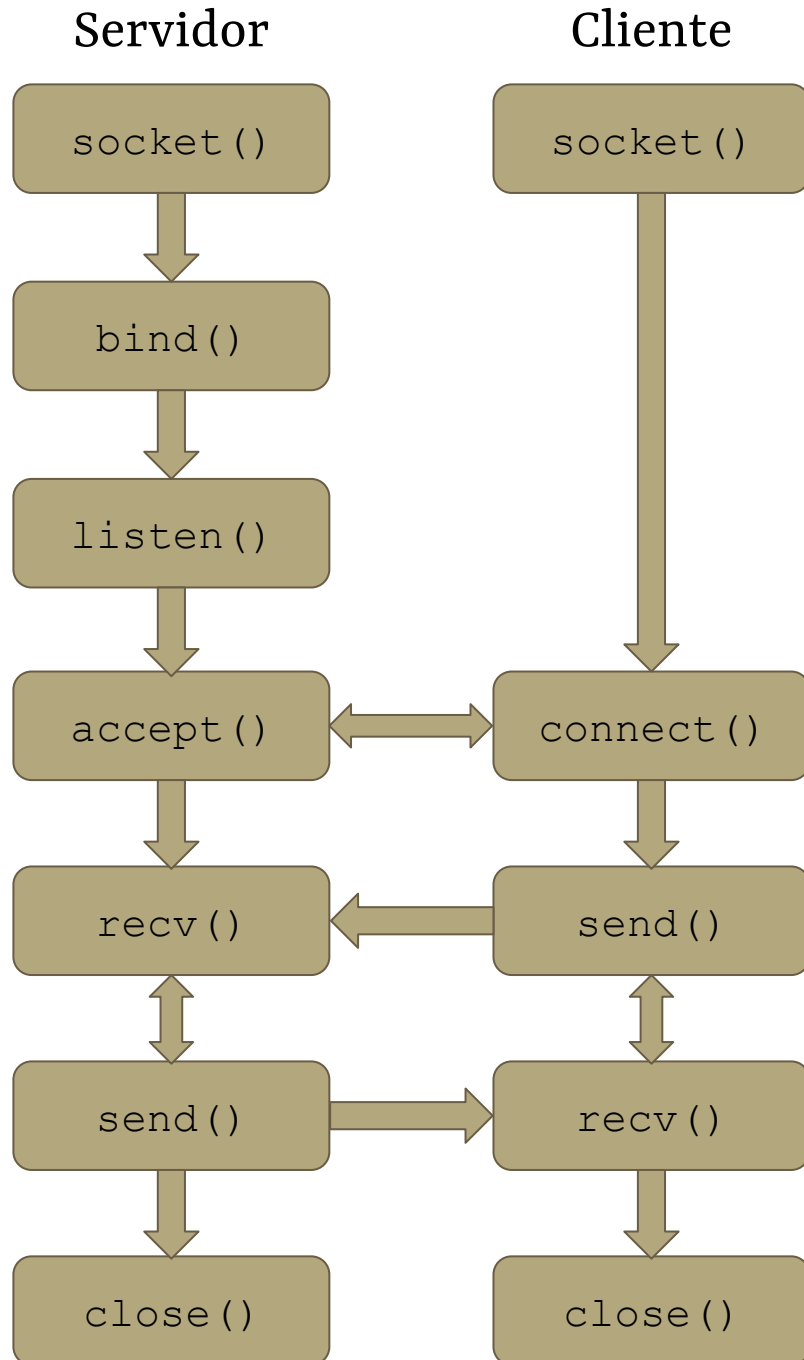
4. Se do
outro

Datagrama é usado quando é preferível reduzir o atraso de comunicação, em detrimento da ordem e da qualidade dos dados (p.ex., VoIP e jogos *online*)

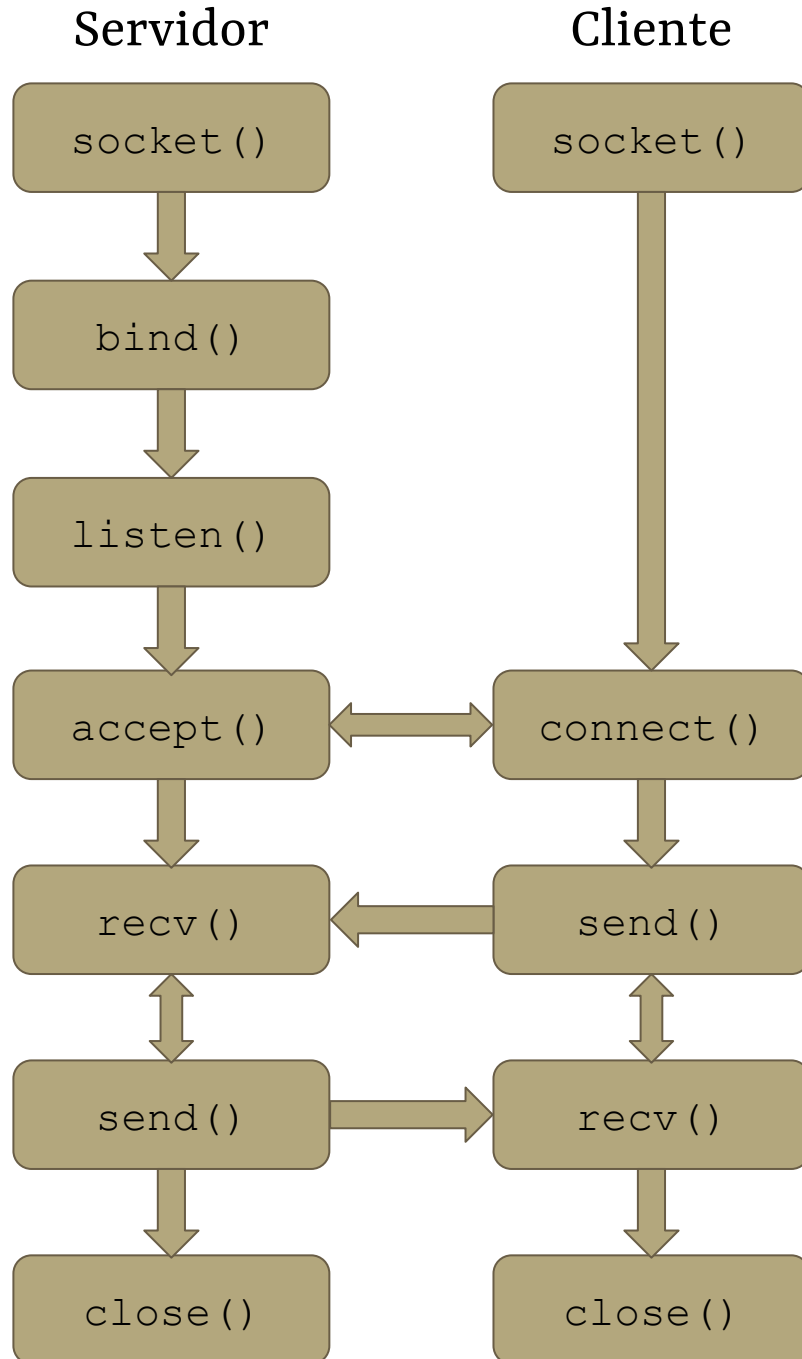
gar antes do

*<https://stackoverflow.com/questions/4688855/whats-the-difference-between-streams-and-datagrams-in-network-programming>

Stream (TCP)

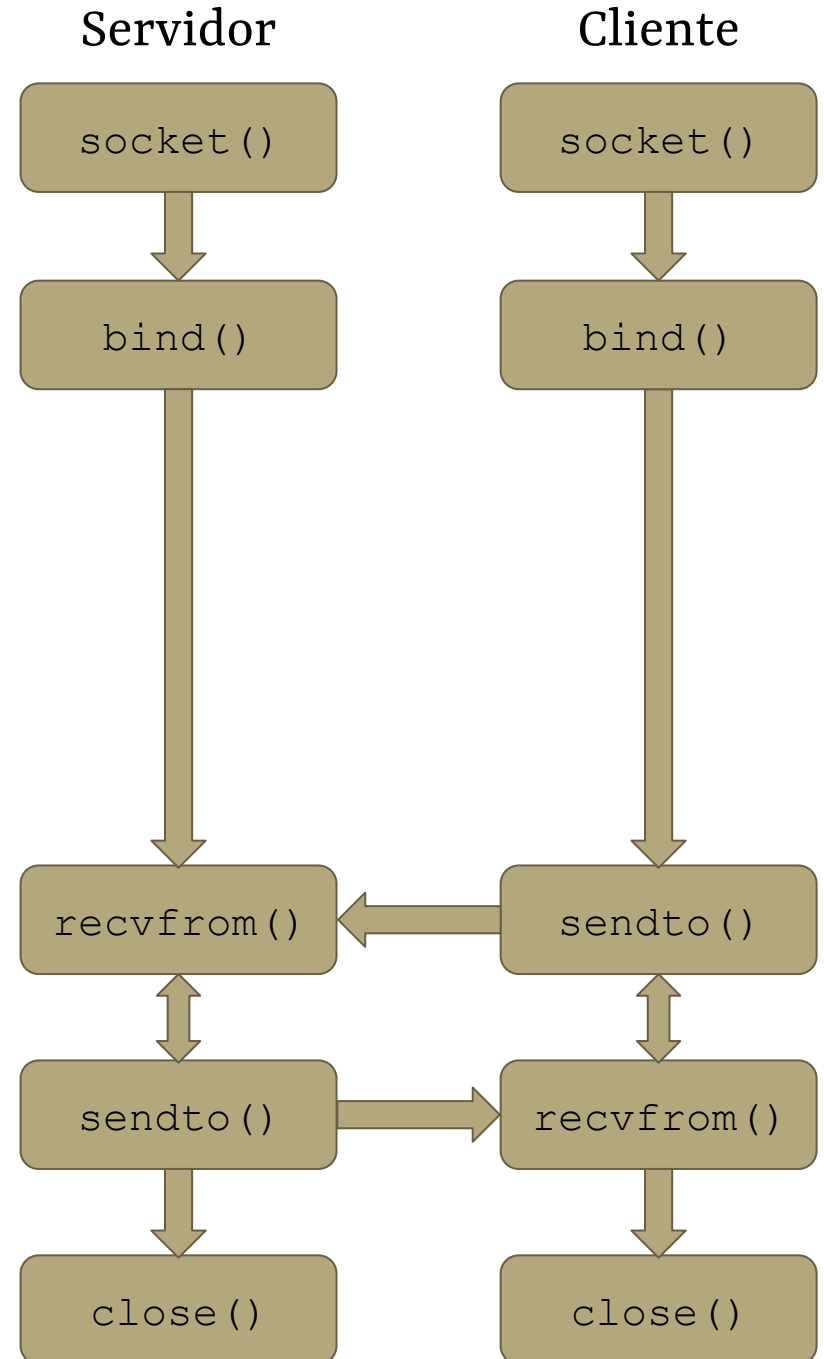


Stream (TCP)



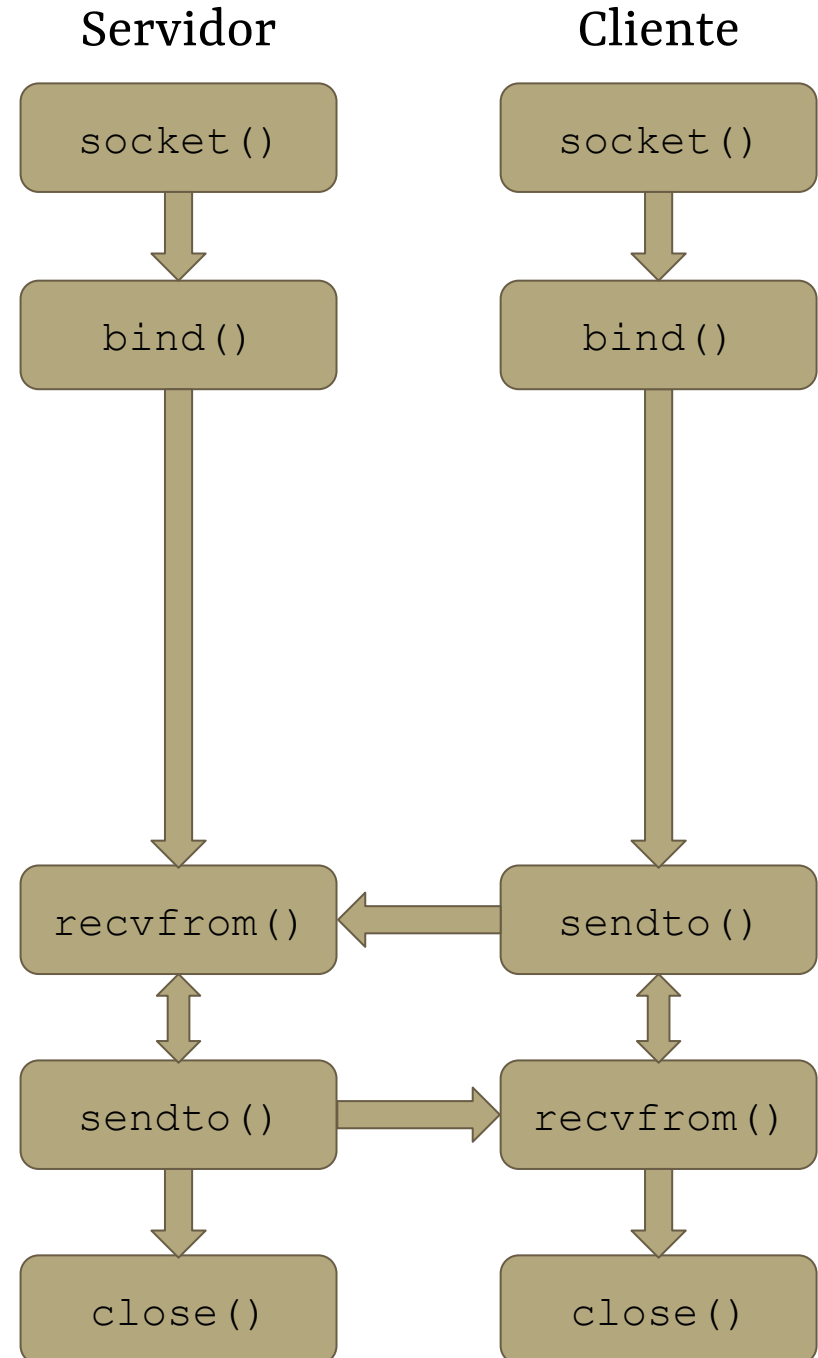
- `socket()` : cria um socket
- `bind()` : associa o socket de um servidor a um endereço
- `listen()` : configura o socket para aceitar conexões
- `accept()` : aceita uma conexão e cria um socket específico para a conexão
- `connect()` : pede uma conexão
- `recv()` : recebe dados pela conexão aceita
- `send()` : envia dados pela conexão aceita
- `close()` : fecha o socket

Datagrama (UDP)

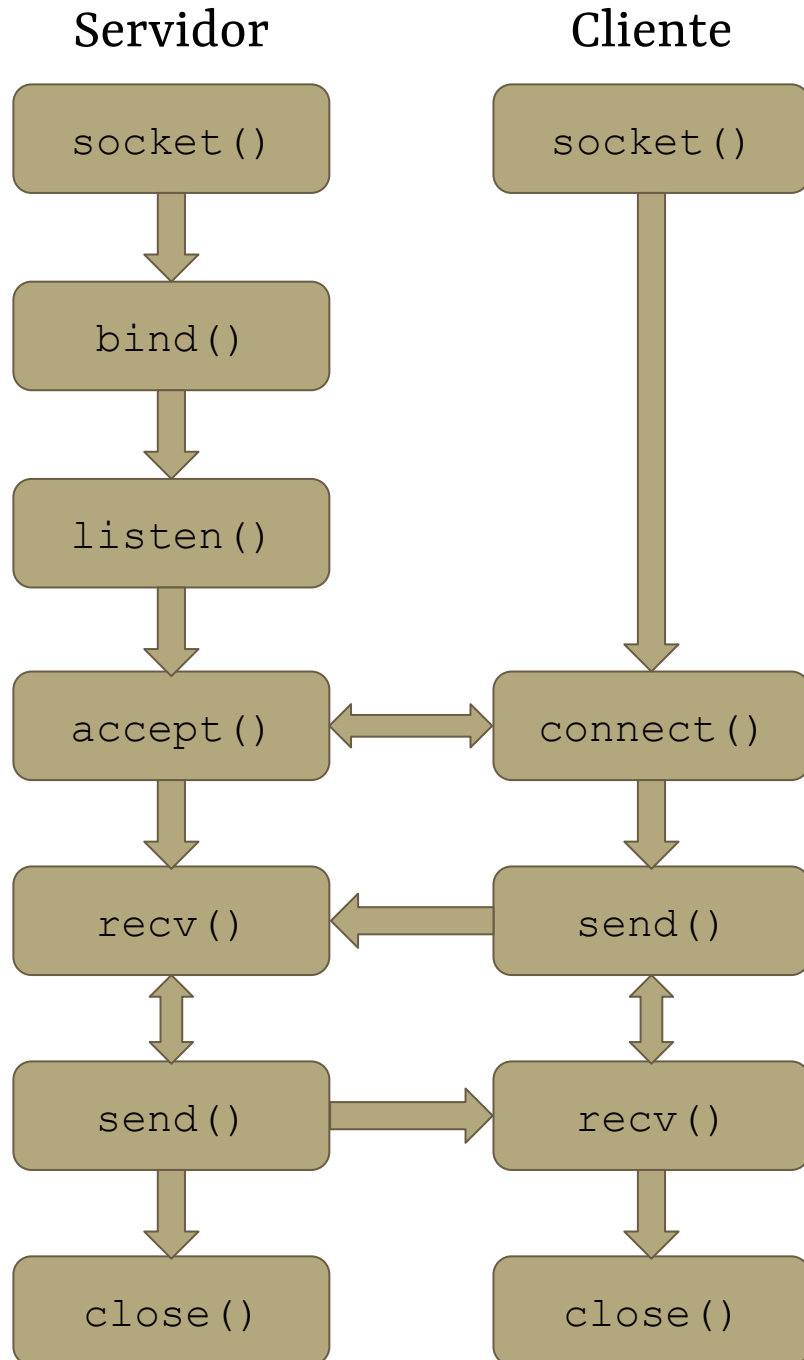


Datagrama (UDP)

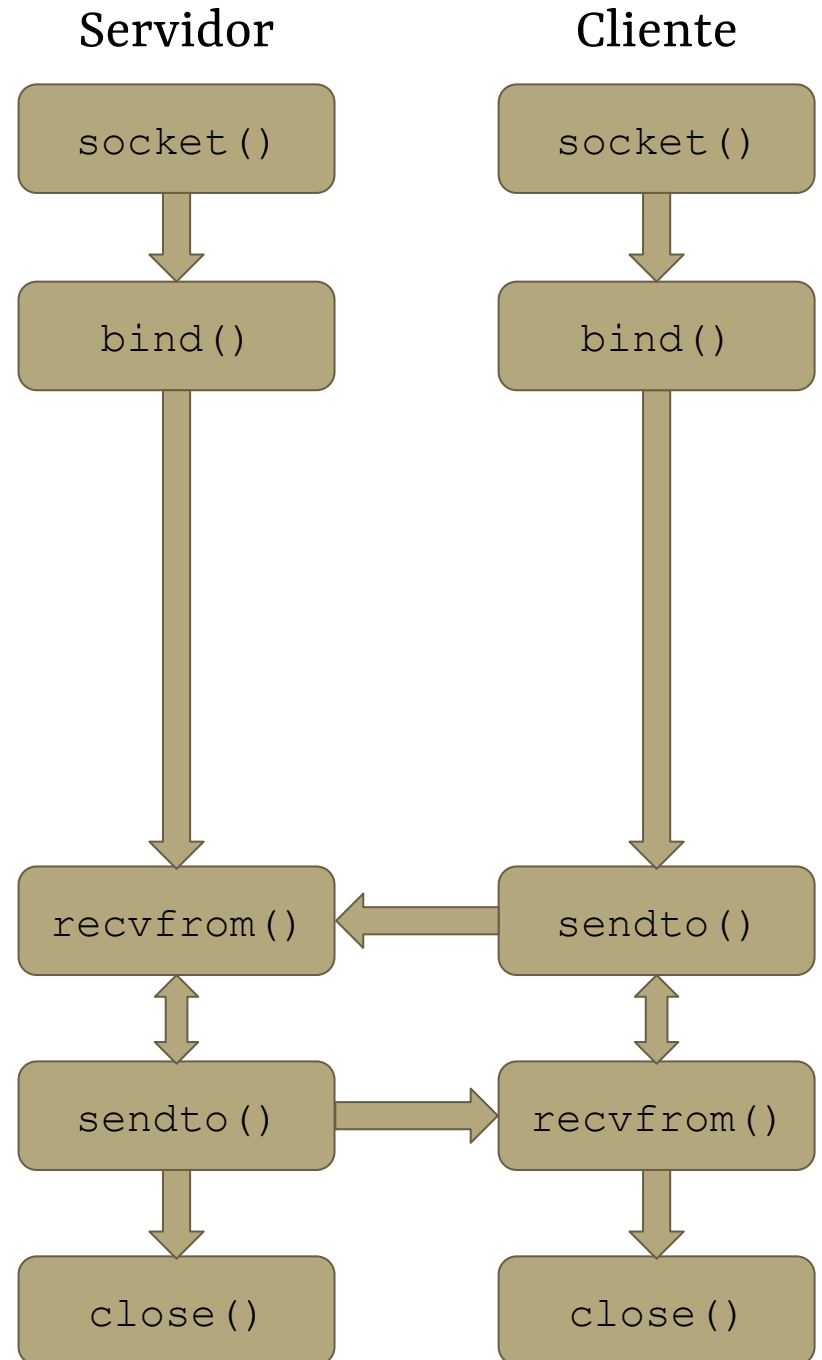
- `socket()` : cria um socket
- `bind()` : associa o socket de um servidor a um endereço
- `recvfrom()` : recebe dados sem conexão estabelecida
- `sendto()` : envia dados sem conexão estabelecida
- `close()` : fecha o socket



Stream (TCP)



Datagrama (UDP)



```

#include "server_funcs.h"

int main (int argc, char* argv[])
{
    if(argc < 2)
    {
        puts("Modo de Uso:");
        printf("%s NOME_SOCKET\n",
            argv[0]);
        exit(1);
    }
    struct sockaddr socket_struct;
    signal(SIGINT, end_server);
    strcpy(socket_name, argv[1]);
    socket_id = socket(PF_LOCAL,
        SOCK_STREAM, 0);
    socket_struct.sa_family =
        AF_LOCAL;
    strcpy(socket_struct.sa_data,
        socket_name);
    bind(socket_id, &socket_struct,
        sizeof(socket_struct));
    listen(socket_id, 10);

```

Code/10_Sockets/01_servidor_local.c

```

while(1)
{
    struct sockaddr cliente;
    int socket_id_cliente,
        end_server_ok;
    socklen_t cliente_len;

    socket_id_cliente =
        accept(socket_id,
            &cliente,
            &cliente_len);
    end_server_ok =
        print_client_message(
            socket_id_cliente);
    close(socket_id_cliente);
    if(end_server_ok)
        end_server(0);
}
return 0;
}

```

(Continuação)

```
#include "server_funcs.h"

int main (int argc, char* argv[])
{
    if(argc < 2)
    {
        puts("Modo de Uso:");
        printf("%s NOME SOCKET\n",
```

```
while(1)
{
    struct sockaddr cliente;
    int socket_id_cliente,
        end_server_ok;
    socklen_t cliente_len;

    socket_id_cliente =
        accept(socket_id,
```

- Este programa cria um servidor local
- O servidor escreve na tela o texto enviado pelo cliente (um programa que veremos adiante)
- Se o cliente transmitir o texto "sair", o servidor se encerra
- Se o usuário pressionar CTRL-C, o servidor também se encerra.
- Modo de Uso:

```
./01_servidor_local.out NOME_SOCKET
```

```
strcpy(socket_struct.sa_data,
        socket_name);
bind(socket_id, &socket_struct,
        sizeof(socket_struct));
listen(socket_id, 10);
```

```
}
```

(Continuação)

Code/10_Sockets/01_servidor_local.c

```
#include "server_funcs.h"

int main (int argc, char* argv[])
{
    if(argc < 2)
    {
        puts("Modo de Uso:");
        printf("%s NOME_SOCKET\n",
            argv[0]);
        exit(1);
    }

    struct so
    signal(SIG
    strcpy(so
    socket_id
    SOCK_S
    socket_st
    AF_LOCAL
    strcpy(socket_struct.sa_data,
        socket_name);
    bind(socket_id, &socket_struct,
        sizeof(socket_struct));
    listen(socket_id, 10);
```

```
while(1)
{
    struct sockaddr cliente;
    int socket_id_cliente,
        end_server_ok;
    socklen_t cliente_len;

    socket_id_cliente =
        accept(socket_id,
            &cliente,
            &cliente_len);

    _message(
        id_cliente);
    _cliente);
    _k);
    _);

    return 0;
}
```

Por exemplo, se executarmos:

```
./01_servidor_local.out /tmp/socket
```

este servidor irá esperar pedidos de conexão de outros processos, feitos através do arquivo /tmp/socket

(Continuação)

Code/10_Sockets/01_servidor_local.c

```
#include "server_funcs.h"

int main (int argc, char* argv[])
{
    if(argc < 2)
    {
        puts("Modo de Uso:");
        printf("%s NOME_SOCKET\n",
            argv[0]);
        exit(1);
    }
    struct sockaddr socket_struct;
    signal(SIGINT, end_server);
    strcpy(socket_name, argv[1]);
    socket_id = socket(PF_LOCAL,
        SOCK_STREAM, 0);
    socket_struct.sa_family =
        AF_LOCAL;
    strcpy(socket_struct.sa_data,
        socket_name);
    bind(socket_id, &socket_struct,
        sizeof(socket_struct));
    listen(socket_id, 10);
```

**Todos os headers
necessários foram incluídos
neste arquivo**

```
        struct sockaddr client_addr cliente;
        int id_cliente,
        server_ok;
        int cliente_len;

        socket_id_cliente =
            accept(socket_id,
                &cliente,
                &cliente_len);
        end_server_ok =
            print_client_message(
                socket_id_cliente);
        close(socket_id_cliente);
        if(end_server_ok)
            end_server(0);
    }
    return 0;
}
```

(Continuação)

Code/10_Sockets/01_servidor_local.c


```

#include "server_funcs.h"

int main (int argc, char* argv[])
{
    if(argc < 2)
    {
        puts("Modo de Uso:");
        printf("%s NOME_SOCKET\n",
            argv[0]);
        exit(1);
    }
    struct sockaddr socket_struct;
    signal(SIGINT, end_server);
    strcpy(socket_name, argv[1]);
    socket_id = socket(PF_LOCAL,
        SOCK_STREAM, 0);
    socket_struct.sa_family =
        AF_LOCAL;
    strcpy(socket_struct.sa_data,
        socket_name);
    bind(socket_id, &socket_struct,
        sizeof(socket_struct));
    listen(socket_id, 10);

```

**Se o usuário não
indicar o arquivo
do socket, não é
possível abri-lo**

```

while(1)
{
    struct sockaddr cliente;
    int socket_id_cliente,
        end_server_ok;
    int cliente_len;

    socket_id_cliente =
        accept(socket_id,
            &cliente,
            &cliente_len);

    end_server_ok =
        print_client_message(
            socket_id_cliente);
    close(socket_id_cliente);
    if(end_server_ok)
        end_server(0);
}
return 0;
}

```

(Continuação)

```

#include "server_funcs.h"

int main (int argc, char* argv[])
{
    if(argc < 2)
    {
        puts("Modo de Uso:");
        printf("%s NOME_SOCKET\n",
            argv[0]);
        exit(1);
    }
    struct sockaddr socket_struct;
    signal(SIGINT, end_server);
    strcpy(socket_name, argv[1]);
    socket_id = socket(PF_LOCAL,
        SOCK_STREAM, 0);
    socket_struct.sa_family =
        AF_LOCAL;
    strcpy(socket_struct.sa_data,
        socket_name);
    bind(socket_id, &socket_struct,
        sizeof(socket_struct));
    listen(socket_id, 10);

```


```

while(1)
{
    struct sockaddr cliente;
    int socket_id_cliente,
        end_server_ok;
    socklen_t cliente_len;

    socket_id_cliente =
        socket_id,
        cliente,
        cliente_len);
    ok =
        client_message(
            socket_id_cliente);
    if(end_server_ok)
        end_server(0);
}
return 0;
}

```

**Esta estrutura
contém campos
para configurarmos
o socket**



(Continuação)

Code/10_Sockets/01_servidor_local.c

```
#include "server_funcs.h"

int main (int argc, char* argv[])
{
    if(argc < 2)
    {
        puts("Modo de Uso:");
        printf("%s NOME_SOCKET\n",
            argv[0]);
        exit(1);
    }
    struct sockaddr socket_struct;
    signal(SIGINT, end_server);
    strcpy(socket_name, argv[1]);
    socket_id = socket(PF_LOCAL,
        SOCK_STREAM, 0);
    socket_struct.sa_family =
        AF_LOCAL;
    strcpy(socket_struct.sa_data,
        socket_name);
    bind(socket_id, &socket_struct,
        sizeof(socket_struct));
    listen(socket_id, 10);
```

```
while(1)
{
    struct sockaddr cliente;
    int socket_id_cliente,
        end_server_ok;
    socklen_t cliente_len;

    socket_id_cliente =
        accept(socket_id,
            &cliente,
```

Se o usuário pressionar CTRL+C, o servidor termina sua execução chamando a função `end_server()`, declarada em `server_funcs.h` e definida em `server_funcs.c`

```
}
```

(Continuação)

Code/10_Sockets/01_servidor_local.c

```
#include "server_funcs.h"

int main (int argc, char* argv[])
{
    if(argc < 2)
    {
        puts("Modo de Uso:");
        printf("%s NOME_SOCKET\n",
            argv[0]);
        exit(1);
    }
    struct sockaddr socket_struct;
    signal(SIGINT, end_server);
    strcpy(socket_name, argv[1]);
    socket_id = socket(PF_LOCAL,
        SOCK_STREAM, 0);
    socket_struct.sa_family =
        AF_LOCAL;
    strcpy(socket_struct.sa_data,
        socket_name);
    bind(socket_id, &socket_struct,
        sizeof(socket_struct));
    listen(socket_id, 10);
```

```
while(1)
{
    struct sockaddr cliente;
    int socket_id_cliente,
        end_server_ok;
    socklen_t cliente_len;

    socket_id_cliente =
        accept(socket_id,
            &cliente,
            &cliente_len);
    if(socket_id_cliente !=
        -1)
        int_message(
            socket_id_cliente);
    if(end_server_ok)
        break;
}

(Continuação)
```

**Copiamos o nome do
arquivo indicado pelo
usuário para a variável
global**

**socket_name[],
declarada em
server_funcs.h**

```

#include "server_funcs.h"

int main (int argc, char* argv[])
{
    if(argc < 2)
    {
        puts("Modo de Uso:");
        printf("%s NOME_SOCKET\n",
            argv[0]);
        exit(1);
    }
    struct sockaddr socket_struct;
    signal(SIGINT, end_server);
    strcpy(socket_name, argv[1]);
    socket_id = socket(PF_LOCAL,
        SOCK_STREAM, 0);
    socket_struct.sa_family =
        AF_LOCAL;
    strcpy(socket_struct.sa_data,
        socket_name);
    bind(socket_id, &socket_struct,
        sizeof(socket_struct));
    listen(socket_id, 10);

```

```

while(1)
{
    struct sockaddr cliente;
    int socket_id_cliente,
        end_server_ok;
    socklen_t cliente_len;

    socket_id_cliente =
        accept(socket_id,
            &cliente,
            &cliente_len);
    end_server_ok =
        print_client_message(
            cliente);
    return 0;
}

```

Criamos um socket de *stream* do tipo local, e recebemos o descritor de arquivo `socket_id` do socket

(Continuação)

Code/10_Sockets/01_servidor_local.c

```
#include "server_funcs.h"

int main (int argc, char* argv[])
{
    if(argc < 2)
    {
        puts("Modo de Uso:");
        printf("%s NOME_SOCKET\n",
            argv[0]);
        exit(1);
    }
    struct sockaddr socket_struct;
    signal(SIGINT, end_server);
    strcpy(socket_name, argv[1]);
    socket_id = socket(PF_LOCAL,
        SOCK_STREAM, 0);
    socket_struct.sa_family =
        AF_LOCAL;
    strcpy(socket_struct.sa_data,
        socket_name);
    bind(socket_id, &socket_struct,
        sizeof(socket_struct));
    listen(socket_id, 10);
```

```
while(1)
{
    struct sockaddr cliente;
    int socket_id_cliente,
        end_server_ok;
    socklen_t cliente_len;

    socket_id_cliente =
        accept(socket_id,
            &cliente,
            &cliente_len);
    end_server_ok =
        print_client_message(
            socket_id_cliente);
    close(socket_id_cliente);
    if(end_server_ok)
```

**Aqui preenchemos a estrutura
do tipo `sock_addr` com o
tipo de socket (local)...**

Code/10_Sockets/01_servidor_local.c

```
#include "server_funcs.h"

int main (int argc, char* argv[])
{
    if(argc < 2)
    {
        puts("Modo de Uso:");
        printf("%s NOME_SOCKET\n",
            argv[0]);
        exit(1);
    }
    struct sockaddr socket_struct;
    signal(SIGINT, end_server);
    strcpy(socket_name, argv[1]);
    socket_id = socket(PF_LOCAL,
        SOCK_STREAM, 0);
    socket_struct.sa_family =
        AF_LOCAL;
    strcpy(socket_struct.sa_data,
        socket_name);
    bind(socket_id, &socket_struct,
        sizeof(socket_struct));
    listen(socket_id, 10);
```

```
while(1)
{
    struct sockaddr cliente;
    int socket_id_cliente,
        end_server_ok;
    socklen_t cliente_len;

    socket_id_cliente =
        accept(socket_id,
            &cliente,
            &cliente_len);
    end_server_ok =
        print_client_message(
            socket_id_cliente);
    close(socket_id_cliente);
    if(end_server_ok)
```

**Aqui preenchemos a estrutura
do tipo `sock_addr` com o
tipo de socket (local)...**

o seu endereço...

Code/10_Sockets/01_servidor_local.c

```
#include "server_funcs.h"

int main (int argc, char* argv[])
{
    if(argc < 2)
    {
        puts("Modo de Uso:");
        printf("%s NOME_SOCKET\n",
            argv[0]);
        exit(1);
    }
    struct sockaddr socket_struct;
    signal(SIGINT, end_server);
    strcpy(socket_name, argv[1]);
    socket_id = socket(PF_LOCAL,
        SOCK_STREAM, 0);
    socket_struct.sa_family =
        AF_LOCAL;
    strcpy(socket_struct.sa_data,
        socket_name);
    bind(socket_id, &socket_struct,
        sizeof(socket_struct));
    listen(socket_id, 10);
```

Code/10_Sockets/01_servidor_local.c

```
while(1)
{
    struct sockaddr cliente;
    int socket_id_cliente,
        end_server_ok;
    socklen_t cliente_len;

    socket_id_cliente =
        accept(socket_id,
            &cliente,
            &cliente_len);
    end_server_ok =
        print_client_message(
            socket_id_cliente);
    close(socket_id_cliente);
    if(end_server_ok)
```

**Aqui preenchemos a estrutura
do tipo `sock_addr` com o
tipo de socket (local)...**

o seu endereço...

**... e o conectamos ao socket
aberto.**


```

#include "server_funcs.h"

int main (int argc, char* argv[])
{
    if(argc < 2)
    {
        puts("Modo de Uso:");
        printf("%s NOME_SOCKET\n",
            argv[0]);
        exit(1);
    }
    struct sockaddr socket_struct;
    signal(SIGINT, end_server);
    strcpy(socket_name, argv[1]);
    socket_id = socket(PF_LOCAL,
        SOCK_STREAM, 0);
    socket_struct.sa_family =
        AF_LOCAL;
    strcpy(socket_struct.sa_data,
        socket_name);
    bind(socket_id, &socket_struct,
        sizeof(socket_struct));
    listen(socket_id, 10);

```

```

while(1)
{
    struct sockaddr cliente;
    int socket_id_cliente,
        end_server_ok;
    socklen_t cliente_len;

    socket_id_cliente =
        accept(socket_id,
            &cliente,
            &cliente_len);
    end_server_ok =
        print_client_message(
            socket_id_cliente);
    close(socket_id_cliente);
    if(end_server_ok)
        end_server(0);
}
return 0;
}

```

Em seguida, colocamos o
socket em modo passivo,
aguardando o contato de até
10 clientes

```

#include "server_funcs.h"

int main (int argc, char* argv[])
{
    if(argc < 2)
    {
        puts("
        printf("
        ar
        exit(1)
    }
    struct sockaddr socket_struct;
    signal(SIGINT, end_server);
    strcpy(socket_name, argv[1]);
    socket_id = socket(PF_LOCAL,
        SOCK_STREAM, 0);
    socket_struct.sa_family =
        AF_LOCAL;
    strcpy(socket_struct.sa_data,
        socket_name);
    bind(socket_id, &socket_struct,
        sizeof(socket_struct));
    listen(socket_id, 10);

```

**Enquanto não chegarem
pedidos, o servidor fica preso
na função `accept()`**

```

while(1)
{
    struct sockaddr cliente;
    int socket_id_cliente,
        end_server_ok;
    socklen_t cliente_len;

    socket_id_cliente =
        accept(socket_id,
            &cliente,
            &cliente_len);
    end_server_ok =
        print_client_message(
            socket_id_cliente);
    close(socket_id_cliente);
    if(end_server_ok)
        end_server(0);
}
return 0;
}

```

(Continuação)

Code/10_Sockets/01_servidor_local.c

```

#include "server_funcs.h"

int main (int argc, char* argv[])
{
    if(argc < 2)
    {
        puts("Usage: ./server_local <ip> <port>");
        printf("Error: %s\n", argv[0]);
        exit(1);
    }

    struct sockaddr socket_struct;
    signal(SIGINT, end_server);
    strcpy(socket_name, argv[1]);
    socket_id = socket(PF_LOCAL,
        SOCK_STREAM, 0);
    socket_struct.sa_family =
        AF_LOCAL;
    strcpy(socket_struct.sa_data,
        socket_name);
    bind(socket_id, &socket_struct,
        sizeof(socket_struct));
    listen(socket_id, 10);

```

Quando chegar um pedido, é aberto um socket novo para essa conexão

```

while(1)
{
    struct sockaddr cliente;
    int socket_id_cliente,
        end_server_ok;
    socklen_t cliente_len;

    socket_id_cliente =
        accept(socket_id,
            &cliente,
            &cliente_len);
    end_server_ok =
        print_client_message(
            socket_id_cliente);
    close(socket_id_cliente);
    if(end_server_ok)
        end_server(0);
}
return 0;
}

```

(Continuação)

Code/10_Sockets/01_servidor_local.c

```
#include "server_funcs.h"

int main (int argc, char* argv[])
{
    if(argc < 2)
    {
        puts("Modo de Uso:");
        printf("%s NOME_SOCKET\n",
            argv[0]);
        exit(1);
    }
    struct
    signal (
    strcpy (
    socket_
        SOC
    socket_
        AF_L
    strcpy(socket_struct.sa_data,
        socket_name);
    bind(socket_id, &socket_struct,
        sizeof(socket_struct));
    listen(socket_id, 10);
```

**A função
print_client_message(),
declarada em server_funcs.h
e definida em server_funcs.c,
imprime a mensagem enviada
pelo cliente**

```
while(1)
{
    struct sockaddr cliente;
    int socket_id_cliente,
        end_server_ok;
    socklen_t cliente_len;

    socket_id_cliente =
        accept(socket_id,
            &cliente,
            &cliente_len);

    end_server_ok =
        print_client_message(
            socket_id_cliente);
    close(socket_id_cliente);
    if(end_server_ok)
        end_server(0);
}
return 0;
}
```

(Continuação)

```
#include "server_funcs.h"

int main (int argc, char* argv[])
{
    if(argc < 2)
    {
        puts("Modo de Uso:");
        printf("%s NOME SOCKET\n",
            argv[1]);
        exit(1);
    }
    struct
    signal(
    strcpy(socket_name, argv[1]);
    socket_id = socket(PF_LOCAL,
        SOCK_STREAM, 0);
    socket_struct.sa_family =
        AF_LOCAL;
    strcpy(socket_struct.sa_data,
        socket_name);
    bind(socket_id, &socket_struct,
        sizeof(socket_struct));
    listen(socket_id, 10);
```

Ela retorna o valor 1 se a mensagem for igual a “sair”, e 0 em caso contrário

```
while(1)
{
    struct sockaddr cliente;
    int socket_id_cliente,
        end_server_ok;
    socklen_t cliente_len;

    socket_id_cliente =
        accept(socket_id,
            &cliente,
            &cliente_len);

    end_server_ok =
        print_client_message(
            socket_id_cliente);
    close(socket_id_cliente);
    if(end_server_ok)
        end_server(0);
}
return 0;
}
```

(Continuação)

Code/10_Sockets/01_servidor_local.c

```
#include "server_funcs.h"

int main (int argc, char* argv[])
{
    if(argc < 2)
    {
        puts("Modo de Uso:");
        printf("%s NOME_SOCKET\n",
            argv[0]);
        exit(1);
    }
    struct
    signal
    strcpy
    socket
        SO
    socket
        AF_LOCAL;
    strcpy(socket_struct.sa_data,
        socket_name);
    bind(socket_id, &socket_struct,
        sizeof(socket_struct));
    listen(socket_id, 10);
```

Neste exemplo, o servidor não responde nada ao cliente. Logo, podemos fechar o socket de conexão ao cliente

```
while(1)
{
    struct sockaddr cliente;
    int socket_id_cliente,
        end_server_ok;
    socklen_t cliente_len;

    socket_id_cliente =
        accept(socket_id,
            &cliente,
            &cliente_len);
    end_server_ok =
        print_client_message(
            socket_id_cliente);
    close(socket_id_cliente);
    if(end_server_ok)
        end_server(0);
}
return 0;
}
```

(Continuação)

Code/10_Sockets/01_servidor_local.c

```

#include "server_funcs.h"

int main (int argc, char* argv[])
{
    if(argc < 2)
    {
        puts("Modo de Uso:");
        printf("%s NOME_SOCKET\n",
            argv[0]);
        exit(1);
    }
    struct
    signal
    strcpy
    socket
    SO
    socket
    AF_
    strcpy(socket_struct.sa_data,
        socket_name);
    bind(socket_id, &socket_struct,
        sizeof(socket_struct));
    listen(socket_id, 10);

```

Se o usuário mandar a mensagem "sair", terminamos a execução chamando a função `end_server()`, a mesma chamada quando o usuário pressiona CTRL+C

```

while(1)
{
    struct sockaddr cliente;
    int socket_id_cliente,
        end_server_ok;
    socklen_t cliente_len;

    socket_id_cliente =
        accept(socket_id,
            &cliente,
            &cliente_len);
    end_server_ok =
        print_client_message(
            socket_id_cliente);
    close(socket_id_cliente);
    if(end_server_ok)
        end_server(0);
}
return 0;
}

```

(Continuação)

Code/10_Sockets/01_servidor_local.c

```
#ifndef SERVER_FUNCS_H
#define SERVER_FUNCS_H

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <signal.h>
#include <sys/socket.h>
#include <sys/un.h>
#include <arpa/inet.h>

char socket_name[100];
int  socket_id;
int  print_client_message(int client_socket);
void end_server(int signum);

#endif // SERVER_FUNCS_H
```

Code/10_Sockets/server_funcs.h


```
#include "server_funcs.h"

int print_client_message(int client_socket)
{
    int length, end_server_ok;
    char* text;
    read(client_socket, &length, sizeof(int));
    text = (char*) malloc(length);
    read(client_socket, text, length);
    fprintf(stderr, "Mensagem = %s\n", text);
    end_server_ok = !strcmp(text, "sair");
    free(text);
    return end_server_ok;
}

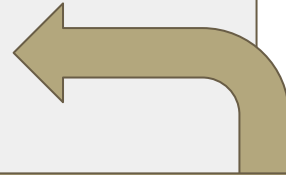
void end_server(int signum)
{
    fprintf(stderr, "Fechando servidor local\n");
    unlink(socket_name);
    close(socket_id);
    exit(0);
}
```

Code/10_Sockets/server_funcs.c

```
#include "server_funcs.h"

int print_client_message(int client_socket)
{
    int length, end_server_ok;
    char* text;
    read(client_socket, &length, sizeof(int));
    text = (char*) malloc(length);
    read(client_socket, text, length);
    fprintf(stderr, "Mensagem = %s", text);
    end_server_ok = !strcmp(text, "quit");
    free(text);
    return end_server_ok;
}

void end_server(int signum)
{
    fprintf(stderr, "Fechando servidor\n");
    unlink(socket_name);
    close(socket_id);
    exit(0);
}
```



Não sabemos de antemão o tamanho da mensagem enviada pelo cliente

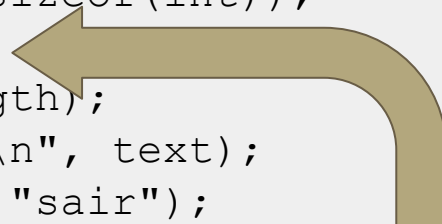
Veremos mais à frente que o cliente manda 4 bytes para indicar o tamanho da mensagem, e depois manda a mensagem

Code/10_Sockets/server_funcs.c

```
#include "server_funcs.h"

int print_client_message(int client_socket)
{
    int length, end_server_ok;
    char* text;
    read(client_socket, &length, sizeof(int));
    text = (char*) malloc(length);
    read(client_socket, text, length);
    fprintf(stderr, "Mensagem = %s\n", text);
    end_server_ok = !strcmp(text, "sair");
    free(text);
    return end_server_ok;
}

void end_server(int signum)
{
    fprintf(stderr, "Fechando servidor\n");
    unlink(socket_name);
    close(socket_id);
    exit(0);
}
```




Depois de lermos o tamanho da mensagem, alocamos um vetor deste tamanho...

Code/10_Sockets/server_funcs.c

```
#include "server_funcs.h"

int print_client_message(int client_socket)
{
    int length, end_server_ok;
    char* text;
    read(client_socket, &length, sizeof(int));
    text = (char*) malloc(length);
    read(client_socket, text, length);
    fprintf(stderr, "Mensagem = %s\n", text);
    end_server_ok = !strcmp(text, "sair");
    free(text);
    return end_server_ok;
}

void end_server(int signum)
{
    fprintf(stderr, "Fechando servidor\n");
    unlink(socket_name);
    close(socket_id);
    exit(0);
}
```



Depois de lermos o tamanho da mensagem, alocamos um vetor deste tamanho...


lamos a mensagem...

Code/10_Sockets/server_funcs.c

```
#include "server_funcs.h"

int print_client_message(int client_socket)
{
    int length, end_server_ok;
    char* text;
    read(client_socket, &length, sizeof(int));
    text = (char*) malloc(length);
    read(client_socket, text, length);
    fprintf(stderr, "Mensagem = %s\n", text);
    end_server_ok = !strcmp(text, "sair");
    free(text);
    return end_server_ok;
}

void end_server(int signum)
{
    fprintf(stderr, "Fechando servidor\n");
    unlink(socket_name);
    close(socket_id);
    exit(0);
}
```



Depois de lermos o tamanho da mensagem, alocamos um vetor deste tamanho...

lemos a mensagem...


e a escrevemos na tela.

Code/10_Sockets/server_funcs.c

```
#include "server_funcs.h"

int print_client_message(int client_socket)
{
    int length, end_server_ok;
    char* text;
    read(client_socket, &length, sizeof(int));
    text = (char*) malloc(length);
    read(client_socket, text, length);
    fprintf(stderr, "Mensagem = %s\n", text);
    end_server_ok = !strcmp(text, "sair");
    free(text);
    return end_server_ok;
}

void end_server(int signum)
{
    fprintf(stderr, "Fechando servidor\n");
    unlink(socket_name);
    close(socket_id);
    exit(0);
}
```



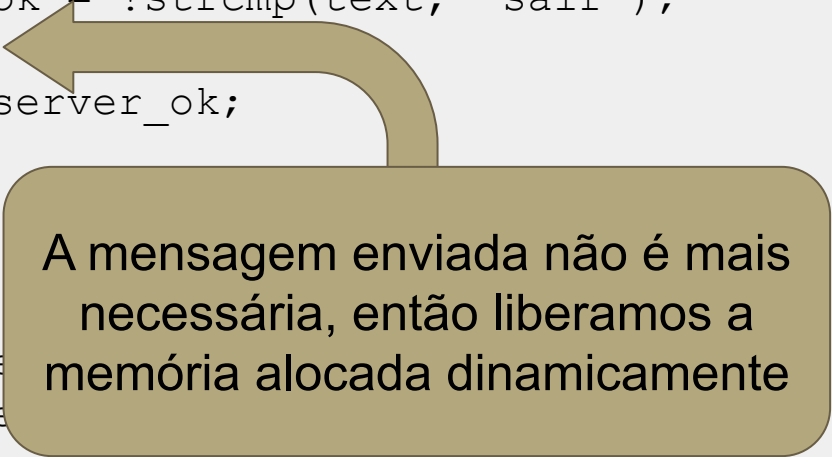
Depois de lida a mensagem,
conferimos se o usuário mandou
a palavra "sair"

Code/10_Sockets/server_funcs.c

```
#include "server_funcs.h"

int print_client_message(int client_socket)
{
    int length, end_server_ok;
    char* text;
    read(client_socket, &length, sizeof(int));
    text = (char*) malloc(length);
    read(client_socket, text, length);
    fprintf(stderr, "Mensagem = %s\n", text);
    end_server_ok = !strcmp(text, "sair");
    free(text);
    return end_server_ok;
}

void end_server
{
    fprintf(stderr, "Encerrando servidor\n");
    unlink(socket_path);
    close(socket_id);
    exit(0);
}
```



A mensagem enviada não é mais necessária, então liberamos a memória alocada dinamicamente

Code/10_Sockets/server_funcs.c

```
#include "server_funcs.h"

int print_client_message(int client_socket)
{
    int length, end_server_ok;
    char* text;
    read(client_socket, &length, sizeof(int));
    text = (char*) malloc(length);
    read(client_socket, text, length);
    fprintf(stderr, "Mensagem = %s\n", text);
    end_server_ok = !strcmp(text, "sair");
    free(text);
    return end_server_ok;
}

void end_server(int signum)
{
    fprintf(stderr, "Fechando\n");
    unlink(socket_name);
    close(socket_id);
    exit(0);
}
```

A função `end_server()` libera o arquivo indicado pelo usuário para servir de socket...


```
#include "server_funcs.h"

int print_client_message(int client_socket)
{
    int length, end_server_ok;
    char* text;
    read(client_socket, &length, sizeof(int));
    text = (char*) malloc(length);
    read(client_socket, text, length);
    fprintf(stderr, "Mensagem = %s\n", text);
    end_server_ok = !strcmp(text, "sair");
    free(text);
    return end_server_ok;
}

void end_server(int signum)
{
    fprintf(stderr, "Fechando\n");
    unlink(socket_name);
    close(socket_id);
    exit(0);
}
```

A função `end_server()` libera o arquivo indicado pelo usuário para servir de socket...

... libera o descritor de arquivo do socket...

Code/10_Sockets/

```
#include "server_funcs.h"

int print_client_message(int client_socket)
{
    int length, end_server_ok;
    char* text;
    read(client_socket, &length, sizeof(int));
    text = (char*) malloc(length);
    read(client_socket, text, length);
    fprintf(stderr, "Mensagem = %s\n", text);
    end_server_ok = !strcmp(text, "sair");
    free(text);
    return end_server_ok;
}

void end_server(int signum)
{
    fprintf(stderr, "Fechando\n");
    unlink(socket_name);
    close(socket_id);
    exit(0);
}
```

Code/10_Sockets/

A função `end_server()` libera o arquivo indicado pelo usuário para servir de socket...

... libera o descritor de arquivo do socket...

... e encerra o processo.

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/un.h>
#include <unistd.h>

int main(int argc, char* argv[])
{
    if(argc < 3)
    {
        puts("Modo de Uso:");
        printf("%s NOME_SOCKET MSG\n",
            argv[0]);
        exit(1);
    }
    char *socket_name;
    char *mensagem;
    int socket_id;
    struct sockaddr name;
    int length;

    socket_name = argv[1];
    mensagem = argv[2];
    socket_id = socket(PF_LOCAL,
        SOCK_STREAM, 0);

```

```

    name.sa_family = AF_LOCAL;
    strcpy(name.sa_data,
        socket_name);
    connect(socket_id, &name,
        sizeof(name));
    length = strlen(mensagem) + 1;
    write(socket_id, &length,
        sizeof(length));
    write(socket_id, mensagem,
        length);
    close(socket_id);
    return 0;
}

```

(Continuação)

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/un.h>
#include <unistd.h>
```

```
int main(int
```

```
{
```

```
    if (argc
```

```
    {
```

```
        pu
```

```
        pr
```

```
        ex
```

```
    }
```

```
    char *
```

```
    char *
```

```
    int soc
```

```
    struct sockaddr name;
```

```
    int length;
```

```
    socket_name = argv[1];
```

```
    mensagem = argv[2];
```

```
    socket_id = socket(PF_LOCAL,
                       SOCK_STREAM, 0);
```

```
    name.sa_family = AF_LOCAL;
    strcpy(name.sa_data,
           socket_name);
    connect(socket_id, &name,
            sizeof(name));
    length = strlen(mensagem) + 1;
    write(socket_id, &length,
```

```
    mensagem,
```

- Este programa cria um cliente que se comunica ao servidor local do exemplo anterior
- O servidor deve ser executado em outro terminal
- O servidor escreve na tela todo texto enviado pelo cliente
- Se o cliente transmitir o texto "sair", o servidor se encerra.
- Se o usuário pressionar CTRL-C para o servidor, ele também se encerra
- **Modo de Uso:**
`./01_cliente_local.out NOME_SOCKET MSG`

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/un.h>
#include <unistd.h>

int main(int argc, char* argv[])
{
    if(argc < 3)
    {
        puts("Modo de Uso:");
        printf("%s NOME_SOCKET MSG\n",
            argv[0]);
        exit(1);
    }
    char *socket_name;
    char *mensagem;
    int socket_id;
    struct sockaddr name;
    int length;

    socket_name = argv[1];
    mensagem = argv[2];
    socket_id = socket(PF_LOCAL,
        SOCK_STREAM, 0);

```

```

    name.sa_family = AF_LOCAL;
    strcpy(name.sa_data,
        socket_name);
    connect(socket_id, &name,
        sizeof(name));
    length = strlen(mensagem) + 1;
    write(socket_id, &length,
        sizeof(length));
    write(socket_id, mensagem,
        length);

```

Se o usuário não indicar o arquivo local do socket e a mensagem, o processo é terminado

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/un.h>
#include <unistd.h>

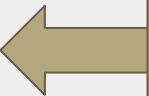
int main(int argc, char* argv[])
{
    if(argc < 3)
    {
        puts("Modo de Uso:");
        printf("%s NOME_SOCKET MSG\n",
            argv[0]);
        exit(1);
    }
    char *socket_name;
    char *mensagem;
    int socket_id;
    struct sockaddr name;
    int length;

    socket_name = argv[1];
    mensagem = argv[2];
    socket_id = socket(PF_LOCAL,
        SOCK_STREAM, 0);
```

```
    name.sa_family = AF_LOCAL;
    strcpy(name.sa_data,
        socket_name);
    connect(socket_id, &name,
        sizeof(name));
    length = strlen(mensagem) + 1;
    write(socket_id, &length,
        sizeof(length));
    write(socket_id, mensagem,
        length);
    close(socket_id);
    return 0;
}
```

(Continuação)

**Criamos um socket de *stream*
do tipo local, e recebemos o
descritor de arquivo
socket_id do socket**



```
#include <stdio.h>
#include <
#include <
#include <
#include <
#include <
```

A função `connect()` pede a abertura de uma conexão com o servidor local

```
int main(int argc, char* argv[])
{
    if(argc < 3)
    {
        puts("Modo de Uso:");
        printf("%s NOME_SOCKET MSG\n",
            argv[0]);
        exit(1);
    }
    char *socket_name;
    char *mensagem;
    int socket_id;
    struct sockaddr name;
    int length;

    socket_name = argv[1];
    mensagem = argv[2];
    socket_id = socket(PF_LOCAL,
        SOCK_STREAM, 0);
```

```
name.sa_family = AF_LOCAL;
strcpy(name.sa_data,
    socket_name);
connect(socket_id, &name,
    sizeof(name));
length = strlen(mensagem) + 1;
write(socket_id, &length,
    sizeof(length));
write(socket_id, mensagem,
    length);
close(socket_id);
return 0;
}
```

(Continuação)

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <unistd.h>
#include <sys/socket.h>
```

```
int main(int argc, char *argv[])
{
```

```
    if (argc < 3)
    {
```

```
        printf("Uso: %s <endereco> <mensagem>\n", argv[0]);
        exit(1);
    }
```

```
    char *socket_name;
```

```
    char *mensagem;
```

```
    int socket_id;
```

```
    struct sockaddr name;
```

```
    int length;
```

```
    socket_name = argv[1];
```

```
    mensagem = argv[2];
```

```
    socket_id = socket(PF_LOCAL,
                      SOCK_STREAM, 0);
```

Como explicado no código do servidor, o cliente deve mandar inicialmente 4 bytes para indicar o tamanho da mensagem

Somamos 1 para contar também o fim da string ('\\0')

```
    name.sa_family = AF_LOCAL;
    strcpy(name.sa_data,
           socket_name);
    connect(socket_id, &name,
           sizeof(name));
    length = strlen(mensagem) + 1;
    write(socket_id, &length,
           sizeof(length));
    write(socket_id, mensagem,
           length);
    close(socket_id);
    return 0;
}
```


(Continuação)


```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/un.h>
#include <unistd.h>
```

```
int main(int argc, char *argv[])
{
    if(argc < 3)
    {
        puts("Modo de uso: ./cliente <nome do socket> <mensagem>");
        printf("%s\n", argv[0]);
        exit(1);
    }
    char *socket_name;
    char *mensagem;
    int socket_id;
    struct sockaddr name;
    int length;
```

```
    socket_name = argv[1];
    mensagem = argv[2];
    socket_id = socket(PF_LOCAL,
                      SOCK_STREAM, 0);
```

Em seguida, o cliente manda a mensagem...



```
    name.sa_family = AF_LOCAL;
    strcpy(name.sa_data,
           socket_name);
    connect(socket_id, &name,
           sizeof(name));
    length = strlen(mensagem) + 1;
    write(socket_id, &length,
           sizeof(length));
    write(socket_id, mensagem,
           length);
    close(socket_id);
    return 0;
}
```

(Continuação)

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/un.h>
#include <unistd.h>
```

```
int main(int argc, char *
```

```
{
    if(argc < 3)
```

```
{
```

```
        puts("Modo de
```

```
        printf("%s N
```

```
        argv[0]);
```

```
        exit(1);
```

```
}
```

```
char *socket_name;
```

```
char *mensagem;
```

```
int socket_id;
```

```
struct sockaddr name;
```

```
int length;
```

```
socket_name = argv[1];
```

```
mensagem = argv[2];
```

```
socket_id = socket(PF_LOCAL,
    SOCK_STREAM, 0);
```

**Em seguida, o
cliente manda a
mensagem...
... e fecha o socket.**

```
name.sa_family = AF_LOCAL;
strcpy(name.sa_data,
    socket_name);
connect(socket_id, &name,
    sizeof(name));
length = strlen(mensagem) + 1;
write(socket_id, &length,
    sizeof(length));
write(socket_id, mensagem,
    length);
close(socket_id);
return 0;
}
```

(Continuação)

Execução em paralelo do servidor local e do cliente

```
~/Code/10_Sockets $ ./01_servidor_local.out /tmp/socket1
```

```
~/Code/10_Sockets $
```

Execução em paralelo do servidor local e do cliente

```
~/Code/10_Sockets $ ./01_servidor_local.out /tmp/socket1  
Mensagem = Ola socket!
```

```
~/Code/10_Sockets $ ./01_cliente_local.out /tmp/socket1 "Ola  
socket!"  
~/Code/10_Sockets $
```

Execução em paralelo do servidor local e do cliente

```
~/Code/10_Sockets $ ./01_servidor_local.out /tmp/socket1  
Mensagem = Ola socket!  
Mensagem = 12345
```

```
~/Code/10_Sockets $ ./01_cliente_local.out /tmp/socket1 "Ola  
socket!"  
~/Code/10_Sockets $ ./01_cliente_local.out /tmp/socket1 12345  
~/Code/10_Sockets $
```

Execução em paralelo do servidor local e do cliente

```
~/Code/10_Sockets $ ./01_servidor_local.out /tmp/socket1  
Mensagem = Ola socket!  
Mensagem = 12345  
Mensagem = sair12345
```

```
~/Code/10_Sockets $ ./01_cliente_local.out /tmp/socket1 "Ola  
socket!"  
~/Code/10_Sockets $ ./01_cliente_local.out /tmp/socket1 12345  
~/Code/10_Sockets $ ./01_cliente_local.out /tmp/socket1  
sair12345  
~/Code/10_Sockets $
```

Execução em paralelo do servidor local e do cliente

```
~/Code/10_Sockets $ ./01_servidor_local.out /tmp/socket1
Mensagem = Ola socket!
Mensagem = 12345
Mensagem = sair12345
Mensagem = sair
Fechando servidor local
~/Code/10_Sockets $
```

```
~/Code/10_Sockets $ ./01_cliente_local.out /tmp/socket1 "Ola
socket!"
~/Code/10_Sockets $ ./01_cliente_local.out /tmp/socket1 12345
~/Code/10_Sockets $ ./01_cliente_local.out /tmp/socket1
sair12345
~/Code/10_Sockets $ ./01_cliente_local.out /tmp/socket1 sair
~/Code/10_Sockets $
```

```

#include "server_funcs.h"

int main (int argc, char* argv[])
{
    if(argc < 2)
    {
        puts("Modo de Uso:");
        printf("%s PORT_NUM\n",
            argv[0]);
        exit(1);
    }
    unsigned short servidorPorta;
    struct sockaddr_in servidorAddr;
    servidorPorta = atoi(argv[1]);
    signal(SIGINT, end_server);
    socket_id = socket(PF_INET,
        SOCK_STREAM, IPPROTO_TCP);
    memset(&servidorAddr, 0,
        sizeof(servidorAddr));
    servidorAddr.sin_family = AF_INET;
    servidorAddr.sin_addr.s_addr =
        htonl(INADDR_ANY);
    servidorAddr.sin_port =
        htons(servidorPorta);
    bind(socket_id,
        (struct sockaddr*) &servidorAddr,
        sizeof(servidorAddr));
    listen(socket_id, 10);

```

```

while(1)
{
    int socketCliente,
        end_server_ok;
    struct sockaddr_in
        clienteAddr;
    unsigned int clienteLength;

    socketCliente = accept(
        socket_id,
        (struct sockaddr*)
            &clienteAddr,
            &clienteLength);
    fprintf(stderr, "Conexão do
Cliente %s\n",
        inet_ntoa(
            clienteAddr.sin_addr));
    end_server_ok =
        print_client_message(
            socketCliente);
    close(socketCliente);
    if(end_server_ok)
        end_server(0);
}
return 0;
}

```

(Continuação)

Code/10_Sockets/02_servidor_tcp.c


```
#include "server_funcs.h"

int main (int argc, char* argv[])
{
    if(argc < 2)
    {
        puts("Modo de Uso:");
```

```
while(1)
{
    int socketCliente,
        end_server_ok;
    struct sockaddr_in
        clienteAddr;
    unsigned int clienteLength;
```

- Este programa cria um servidor TCP/IP, podendo receber mensagens de outros computadores ligados à mesma rede de internet
- O servidor escreve na tela o texto enviado pelo cliente (um programa que veremos adiante)
- Se o cliente transmitir o texto "sair", o servidor se encerra
- Se o usuário pressionar CTRL-C, o servidor também se encerra.
- Modo de Uso:

```
./02_servidor_tcp.out PORT_NUM
```

```
htonl(INADDR_ANY);
servidorAddr.sin_port =
    htons(servidorPorta);
bind(socket_id,
    (struct sockaddr*) &servidorAddr,
    sizeof(servidorAddr));
listen(socket_id, 10);
```

```
if(end_server_ok)
    end_server(0);
}
return 0;
}
```

(Continuação)

Code/10_Sockets/02_servidor_tcp.c

```
#include "server_funcs.h"

int main (int argc, char* argv[])
{
    if(argc < 2)
    {
        puts("Modo de Uso:");
        printf("%s PORT_NUM\n",
            argv[0]);
        exit(1);
    }
    unsigned short servidorPorta;
    struct sockaddr_in servidorAddr;
    signal(SIGINT, end_server);
    socket_id = socket(AF_INET, SOCK_STREAM, 0);
    memset(&servidorAddr, 0, sizeof(servidorAddr));
    servidorAddr.sin_family = AF_INET;
    servidorAddr.sin_addr.s_addr = htonl(INADDR_ANY);
    servidorAddr.sin_port = htons(servidorPorta);
    bind(socket_id, (struct sockaddr*)&servidorAddr, sizeof(servidorAddr));
    listen(socket_id, 10);
```

```
while(1)
{
    int socketCliente,
        end_server_ok;
    struct sockaddr_in clienteAddr;
    unsigned int clienteLength;

    socketCliente = accept(
        socket_id,
        (struct sockaddr*)&clienteAddr,
        &clienteLength);

    if(socketCliente < 0)
    {
        perror("Falha na conexão do cliente");
        continue;
    }

    if(recv(socketCliente, message, sizeof(message), 0) < 0)
    {
        perror("Falha ao receber mensagem do cliente");
        continue;
    }

    if(end_server_ok)
        end_server(0);

    return 0;
}
```

Por exemplo, se executarmos:

```
./02_servidor_tcp.out 8080
```

este servidor irá esperar pedidos de conexão de outros processos, feitos através da porta TCP/IP 8080

(Continuação)

Code/10_Sockets/02_servidor_tcp.c

```
#include "server_funcs.h"
```

```
int main (int argc, char* argv[])
{
    if(argc < 2)
    {
        puts("Modo de Uso:");
        printf("%s PORT_NUM\n",
            argv[0]);
        exit(1);
    }
    unsigned short servidorPorta;
    struct sockaddr_in servidorAddr;
    servidorPorta = atoi(argv[1]);
    signal(SIGINT, end_server);
    socket_id = socket(PF_INET,
        SOCK_STREAM, IPPROTO_TCP);
    memset(&servidorAddr, 0,
        sizeof(servidorAddr));
    servidorAddr.sin_family = AF_INET;
    servidorAddr.sin_addr.s_addr =
        htonl(INADDR_ANY);
    servidorAddr.sin_port =
        htons(servidorPorta);
    bind(socket_id,
        (struct sockaddr*) &servidorAddr,
        sizeof(servidorAddr));
    listen(socket_id, 10);
```

**Reaproveitamos as
mesmas funções do
servidor local**

```
        Cliente,
        server_ok;
        struct sockaddr_in
        clienteAddr;
        unsigned int clienteLength;

        socketCliente = accept(
            socket_id,
            (struct sockaddr*)
                &clienteAddr,
                &clienteLength);
        fprintf(stderr, "Conexão do
        Cliente %s\n",
            inet_ntoa(
                clienteAddr.sin_addr));
        end_server_ok =
            print_client_message(
                socketCliente);
        close(socketCliente);
        if(end_server_ok)
            end_server(0);
    }
    return 0;
}
```

(Continuação)

Code/10_Sockets/02_servidor_tcp.c

```

#include "server_funcs.h"

int main (int argc, char* argv[])
{
    if(argc < 2)
    {
        puts("Modo de Uso:");
        printf("%s PORT_NUM\n",
            argv[0]);
        exit(1);
    }
    unsigned short servidorPorta;
    struct sockaddr_in servidorAddr;
    servidorPorta = atoi(argv[1]);
    signal(SIGINT, end_server);
    socket_id = socket(PF_INET,
        SOCK_STREAM, IPPROTO_TCP);
    memset(&servidorAddr, 0,
        sizeof(servidorAddr));
    servidorAddr.sin_family = AF_INET;
    servidorAddr.sin_addr.s_addr =
        htonl(INADDR_ANY);
    servidorAddr.sin_port =
        htons(servidorPorta);
    bind(socket_id,
        (struct sockaddr*) &servidorAddr,
        sizeof(servidorAddr));
    listen(socket_id, 10);

```

```

while(1)
{
    int socketCliente,
        end_server_ok;
    struct sockaddr_in
        clienteAddr;
    unsigned int clienteLength;

    socketCliente = accept(
        socket_id,
        (struct sockaddr*)
            &clienteAddr,
            &clienteLength);

    do do
        (servidorAddr));
    if (socketCliente)
        close(socketCliente);
    if(end_server_ok)
        end_server(0);
}
return 0;
}

```

A função `atoi()` recebe uma string e retorna o número inteiro correspondente

(Continuação)

```

#include "server_funcs.h"

int main (int argc, char* argv[])
{
    if(argc < 2)
    {
        puts("Modo de Uso:");
        printf("%s PORT_NUM\n",
            argv[0]);
        exit(1);
    }
    unsigned short servidorPorta;
    struct sockaddr_in servidorAddr;
    servidorPorta = atoi(argv[1]);
    signal(SIGINT, end_server);
    socket_id = socket(PF_INET,
        SOCK_STREAM, IPPROTO_TCP);
    memset(&servidorAddr, 0,
        sizeof(servidorAddr));
    servidorAddr.sin_family = AF_INET;
    servidorAddr.sin_addr.s_addr =
        htonl(INADDR_ANY);
    servidorAddr.sin_port =
        htons(servidorPorta);
    bind(socket_id,
        (struct sockaddr*) &servidorAddr,
        sizeof(servidorAddr));
    listen(socket_id, 10);

```

```

while(1)
{
    int socketCliente,
        end_server_ok;
    struct sockaddr_in
        clienteAddr;
    unsigned int clienteLength;

    socketCliente = accept(
        socket_id,
        (struct sockaddr*)
            &clienteAddr,
            &clienteLength);
    fprintf(stderr, "Conexão do
        .sin_addr));

    print_client_message(
        socketCliente);
    close(socketCliente);
    if(end_server_ok)
        end_server(0);
}
return 0;
}

```

Se o usuário apertar CTRL+C, o servidor é fechado

(Continuação)

```

#include "server_funcs.h"

int main (int argc, char* argv[])
{
    if(argc < 2)
    {
        puts("Modo de Uso:");
        printf("%s PORT_NUM\n",
            argv[0]);
        exit(1);
    }
    unsigned short servidorPorta;
    struct sockaddr_in servidorAddr;
    servidorPorta = atoi(argv[1]);
    signal(SIGINT, end_server);
    socket_id = socket(PF_INET,
        SOCK_STREAM, IPPROTO_TCP);
    memset(&servidorAddr, 0,
        sizeof(servidorAddr));
    servidorAddr.sin_family = AF_INET;
    servidorAddr.sin_addr.s_addr =
        htonl(INADDR_ANY);
    servidorAddr.sin_port =
        htons(servidorPorta);
    bind(socket_id,
        (struct sockaddr*) &servidorAddr,
        sizeof(servidorAddr));
    listen(socket_id, 10);

```

Criamos um socket de *stream* pela internet via protocolo TCP, e recebemos o descritor de arquivo `socket_id` do socket

```

while(1)
{
    int socketCliente,
        end_server_ok;
    struct sockaddr_in
        clienteAddr;
    unsigned int clienteLength;

    socketCliente = accept(
        socket_id,
        (struct sockaddr*)
            &clienteAddr,
            &clienteLength);
    // ... (continuação do código) ...
    close(socketCliente);
    if(end_server_ok)
        end_server(0);
}
return 0;
}

```

(Continuação)

```

#include "server_funcs.h"

int main (int argc, char* argv[])
{
    if(argc < 2)
    {
        puts("Modo de Uso:");
        printf("%s PORT_NUM\n",
            argv[0]);
        exit(1);
    }
    unsigned short servidorPorta;
    struct sockaddr_in servidorAddr;
    servidorPorta = atoi(argv[1]);
    signal(SIGINT, end_server);
    socket_id = socket(PF_INET,
        SOCK_STREAM, IPPROTO_TCP);
    memset(&servidorAddr, 0,
        sizeof(servidorAddr));
    servidorAddr.sin_family = AF_INET;
    servidorAddr.sin_addr.s_addr =
        htonl(INADDR_ANY);
    servidorAddr.sin_port =
        htons(servidorPorta);
    bind(socket_id,
        (struct sockaddr*) &servidorAddr,
        sizeof(servidorAddr));
    listen(socket_id, 10);

```

```

while(1)
{
    int socketCliente,
        end_server_ok;
    struct sockaddr_in
        clienteAddr;
    unsigned int clienteLength;

    socketCliente = accept(
        socket_id,
        (struct sockaddr*)
        &clienteAddr,
        &clienteLength);

    if(socketCliente < 0)
    {
        printf("Falha na criação do socket\n");
        continue;
    }
    memcpy(&servidorAddr, &clienteAddr,
        sizeof(servidorAddr));
    memcpy(&servidorAddr.sin_addr, &clienteAddr.sin_addr);
    memcpy(&servidorPorta, &clientePorta,
        sizeof(servidorPorta));
    memcpy(&servidorAddr.sin_port, &clienteAddr.sin_port);
    memcpy(&servidorAddr.sin_family, &clienteAddr.sin_family);
    memcpy(&servidorAddr.sin_addr.s_addr, &clienteAddr.sin_addr.s_addr);
    memcpy(&servidorAddr.sin_port, &clienteAddr.sin_port);
    memcpy(&servidorAddr.sin_family, &clienteAddr.sin_family);
    close(socketCliente);
    if(end_server_ok)
        end_server(0);
}
return 0;
}

```

A função memset () aqui zera todos os bytes dessa estrutura

(Continuação)

```

#include "server_funcs.h"

int main (int argc, char* argv[])
{
    if(argc < 2)
    {
        puts("Modo de Uso:");
        printf("%s PORT_NUM\n",
            argv[0]);
        exit(1);
    }
    unsigned short servidorPorta;
    struct sockaddr_in servidorAddr;
    servidorPorta = atoi(argv[1]);
    signal(SIGINT, end_server);
    socket_id = socket(PF_INET,
        SOCK_STREAM, IPPROTO_TCP);
    memset(&servidorAddr, 0,
        sizeof(servidorAddr));
    servidorAddr.sin_family = AF_INET;
    servidorAddr.sin_addr.s_addr =
        htonl(INADDR_ANY);
    servidorAddr.sin_port =
        htons(servidorPorta);
    bind(socket_id,
        (struct sockaddr*) &servidorAddr,
        sizeof(servidorAddr));
    listen(socket_id, 10);

```

```

while(1)
{
    int socketCliente,
        end_server_ok;
    struct sockaddr_in
        clienteAddr;
    unsigned int clienteLength;

    socketCliente = accept(
        socket_id,
        (struct sockaddr*)
            &clienteAddr,
            &clienteLength);
    fprintf(stderr, "Conexão do
Cliente %s\n",
        inet_ntoa(
            clienteAddr.sin_addr));
    end_server_ok =
        print_client_message(

```

**Ligamos o socket aberto à
porta TCP indicada pelo
usuário**

Code/10_Sockets/02_servidor_tcp.c


```

#include "server_funcs.h"

int main (int argc, char* argv[])
{
    if(argc < 2)
    {
        puts("Modo de Uso:");
        printf("%s PORT_NUM\n",
            argv[0]);
        exit(1);
    }
    unsigned short servidorPorta;
    struct sockaddr_in servidorAddr;
    servidorPorta = atoi(argv[1]);
    signal(SIGINT, end_server);
    socket_id = socket(PF_INET,
        SOCK_STREAM, IPPROTO_TCP);
    memset(&servidorAddr, 0,
        sizeof(servidorAddr));
    servidorAddr.sin_family = AF_INET;
    servidorAddr.sin_addr.s_addr =
        htonl(INADDR_ANY);
    servidorAddr.sin_port =
        htons(servidorPorta);
    bind(socket_id,
        (struct sockaddr*) &servidorAddr,
        sizeof(servidorAddr));
    listen(socket_id, 10);

```

```

while(1)
{
    int socketCliente,
        end_server_ok;
    struct sockaddr_in
        clienteAddr;
    unsigned int clienteLength;

    socketCliente = accept(
        socket_id,
        (struct sockaddr*)
            &clienteAddr,
            &clienteLength);
    fprintf(stderr, "Conexão do
Cliente %s\n",
        inet_ntoa(
            clienteAddr.sin_addr));
    end_server_ok =
        print_client_message(
            socketCliente);
    close(socketCliente);
    if(end_server_ok)
        end_server(0);
}

```

Em seguida, colocamos o socket em modo passivo, aguardando o contato de até 10 clientes

Code/10_Sockets/02_servidor_tcp.c


```

#include "server_funcs.h"

int main (int argc, char* argv[])
{
    if(argc < 2)
    {
        puts("Modo de Uso:");
        printf("%s PORT_NUM\n",
            argv[0]);
        exit(1);
    }
    unsigned short servidorPorta;
    struct
    servi
    signa
    socke
    S

    memset(&servidorAddr, 0,
        sizeof(servidorAddr));
    servidorAddr.sin_family = AF_INET;
    servidorAddr.sin_addr.s_addr =
        htonl(INADDR_ANY);
    servidorAddr.sin_port =
        htons(servidorPorta);
    bind(socket_id,
        (struct sockaddr*) &servidorAddr,
        sizeof(servidorAddr));
    listen(socket_id, 10);

```

**Aqui o servidor escreve na tela
o endereço IP do cliente**

```

while(1)
{
    int socketCliente,
        end_server_ok;
    struct sockaddr_in
        clienteAddr;
    unsigned int clienteLength;

    socketCliente = accept(
        socket_id,
        (struct sockaddr*)
            &clienteAddr,
            &clienteLength);
    fprintf(stderr, "Conexão do
    Cliente %s\n",
        inet_ntoa(
            clienteAddr.sin_addr));
    end_server_ok =
        print_client_message(
            socketCliente);
    close(socketCliente);
    if(end_server_ok)
        end_server(0);
}
return 0;
}

```

(Continuação)

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <sys/un.h>
#include <unistd.h>

int main (int argc, char* argv[])
{
    if(argc < 4)
    {
        puts("Modo de Uso:");
        printf("%s SERVER_IP SERVER_PORT\n",
            argv[0]);
        exit(1);
    }
    int socket_id;
    struct sockaddr_in servidorAddr;
    int length;
    unsigned short servidorPorta;
    char *IP_Servidor;
    char *mensagem;
    IP_Servidor = argv[1];
    servidorPorta = atoi(argv[2]);
    mensagem = argv[3];
    socket_id = socket(AF_INET,
        SOCK_STREAM, IPPROTO_TCP);

```

```

memset(&servidorAddr, 0,
    sizeof(servidorAddr));
servidorAddr.sin_family = AF_INET;
servidorAddr.sin_addr.s_addr =
    inet_addr(IP_Servidor);
servidorAddr.sin_port =
    htons(servidorPorta);
connect(socket_id,
    (struct sockaddr *)
    &servidorAddr,
    sizeof(servidorAddr));
length = strlen(mensagem) + 1;
write(socket_id, &length,
    sizeof(length));
write(socket_id, mensagem,
    length);
close(socket_id);
return 0;
}

```

(Continuação)

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <sys/un.h>
#include <unistd.h>
```

```
int main (int argc, char* argv[])
{
```

```
    if
    {
MSG\n"
```

```
    }
    in
    st
    in
    un
    char
```

```
    char *mensagem;
    IP_Servidor = argv[1];
    servidorPorta = atoi(argv[2]);
    mensagem = argv[3];
    socket_id = socket(AF_INET,
        SOCK_STREAM, IPPROTO_TCP);
```

```
    memset(&servidorAddr, 0,
        sizeof(servidorAddr));
    servidorAddr.sin_family = AF_INET;
    servidorAddr.sin_addr.s_addr =
        inet_addr(IP_Servidor);
    servidorAddr.sin_port =
        htons(servidorPorta);
    connect(socket_id,
        (struct sockaddr *)
```

- Este programa cria um cliente que se comunica ao servidor TCP do exemplo anterior
- O servidor deve ser executado em outro terminal
- O servidor escreve na tela todo texto enviado pelo cliente
- Se o cliente transmitir o texto "sair", o servidor se encerra.
- Se o usuario pressionar CTRL-C para o servidor, ele tambem se encerra
- **Modo de Uso:**

```
./02_cliente_tcp.out SERVER_IP SERVER_PORT MSG
```

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <sys/un.h>
#include <unistd.h>

int main (int argc, char* argv[])
{
    if(argc < 4)
    {
        puts("Modo de Uso:");
        printf("%s SERVER_IP SERVER_PORT\n",
            argv[0]);
        exit(1);
    }
    int socket_id;
    struct sockaddr_in servidorAddr;
    int length;
    unsigned short servidorPorta;
    char *IP_Servidor;
    char *mensagem;
    IP_Servidor = argv[1];
    servidorPorta = atoi(argv[2]);
    mensagem = argv[3];
    socket_id = socket(AF_INET,
        SOCK_STREAM, IPPROTO_TCP);

```

```

memset(&servidorAddr, 0,
    sizeof(servidorAddr));
servidorAddr.sin_family = AF_INET;
servidorAddr.sin_addr.s_addr =
    inet_addr(IP_Servidor);
servidorAddr.sin_port =
    htons(servidorPorta);
connect(socket_id,
    (struct sockaddr *)
    &servidorAddr,
    sizeof(servidorAddr));
length = strlen(mensagem) + 1;
write(socket_id, &length,
    sizeof(length));
write(socket_id, mensagem,
    length);
close(socket_id);
return 0;
}

```

(Continuação)

Criamos um socket de *stream* pela internet via protocolo TCP, e recebemos o descritor de arquivo `socket_id` do socket

Code/10_Sockets/02_cliente_tcp.c

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <arpa/inet.h>
#include <sys/socket.h>
#include <unistd.h>

int main (int
{
    if(argc < 4)
    {
        puts("Modo de Uso:");
        printf("%s SERVER_IP SERVER_PORT
MSG\n",
            argv[0]);
        exit(1);
    }
    int socket_id;
    struct sockaddr_in servidorAddr;
    int length;
    unsigned short servidorPorta;
    char *IP_Servidor;
    char *mensagem;
    IP_Servidor = argv[1];
    servidorPorta = atoi(argv[2]);
    mensagem = argv[3];
    socket_id = socket(AF_INET,
        SOCK_STREAM, IPPROTO_TCP);

```

Ligamos o socket aberto ao endereço IP e à porta TCP do servidor, indicados pelo usuário

```

memset(&servidorAddr, 0,
        sizeof(servidorAddr));
servidorAddr.sin_family = AF_INET;
servidorAddr.sin_addr.s_addr =
        inet_addr(IP_Servidor);
servidorAddr.sin_port =
        htons(servidorPorta);
connect(socket_id,
        (struct sockaddr *)
        &servidorAddr,
        sizeof(servidorAddr));
length = strlen(mensagem) + 1;
write(socket_id, &length,
        sizeof(length));
write(socket_id, mensagem,
        length);
close(socket_id);
return 0;
}

```

(Continuação)

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <sys/un.h>
#include <unistd.h>

int main (int argc, char* argv[])
{
    if(argc < 4)
    {
        puts("
        printf("
MSG\n",
        ar
        exit(1
    }
    int socket_id;
    struct sockaddr_in servidorAddr;
    int length;
    unsigned short servidorPorta;
    char *IP_Servidor;
    char *mensagem;
    IP_Servidor = argv[1];
    servidorPorta = atoi(argv[2]);
    mensagem = argv[3];
    socket_id = socket(AF_INET,
        SOCK_STREAM, IPPROTO_TCP);

```

**Assim como no cliente local,
o cliente TCP envia o
tamanho da mensagem em 4
bytes, depois manda a
mensagem e fecha o socket**

```

memset(&servidorAddr, 0,
        sizeof(servidorAddr));
servidorAddr.sin_family = AF_INET;
servidorAddr.sin_addr.s_addr =
        inet_addr(IP_Servidor);
servidorAddr.sin_port =
        htons(servidorPorta);
connect(socket_id,
        (struct sockaddr *)
        &servidorAddr,
        sizeof(servidorAddr));
length = strlen(mensagem) + 1;
write(socket_id, &length,
        sizeof(length));
write(socket_id, mensagem,
        length);
close(socket_id);
return 0;
}

```

(Continuação)

Execução em paralelo do servidor TCP e do cliente

```
~/Code/10_Sockets $ ./02_servidor_tcp.out 8080
```

```
~/Code/10_Sockets $
```

Execução em paralelo do servidor TCP e do cliente

```
~/Code/10_Sockets $ ./02_servidor_tcp.out 8080  
Conexão do Cliente 127.0.0.1  
Mensagem = Ola socket!
```

```
~/Code/10_Sockets $ ./02_cliente_tcp.out 127.0.0.1 8080 "Ola  
socket!"  
~/Code/10_Sockets $
```

Execução em paralelo do servidor TCP e do cliente

```
~/Code/10_Sockets $ ./02_servidor_tcp.out 8080  
Conexão do Cliente 127.0.0.1  
Mensagem = Ola socket!  
Conexão do Cliente 127.0.0.1  
Mensagem = 12345
```

```
~/Code/10_Sockets $ ./02_cliente_tcp.out 127.0.0.1 8080 "Ola  
socket!"  
~/Code/10_Sockets $ ./02_cliente_tcp.out 127.0.0.1 8080 12345  
~/Code/10_Sockets $
```

Execução em paralelo do servidor TCP e do cliente

```
~/Code/10_Sockets $ ./02_servidor_tcp.out 8080
Conexão do Cliente 127.0.0.1
Mensagem = Ola socket!
Conexão do Cliente 127.0.0.1
Mensagem = 12345
Conexão do Cliente 192.168.25.197
Mensagem = Oi!
```

```
~/Code/10_Sockets $ ./02_cliente_tcp.out 127.0.0.1 8080 "Ola
socket!"
~/Code/10_Sockets $ ./02_cliente_tcp.out 127.0.0.1 8080 12345
~/Code/10_Sockets $ ./02_cliente_tcp.out 192.168.25.197 8080
Oi!
~/Code/10_Sockets $
```

Execução em paralelo do servidor TCP e do cliente

```
~/Code/10_Sockets $ ./02_servidor_tcp.out 8080
Conexão do Cliente 127.0.0.1
Mensagem = Ola socket!
Conexão do Cliente 127.0.0.1
Mensagem = 12345
Conexão do Cliente 192.168.25.197
Mensagem = Oi!
```

Considere que o IP da
máquina neste exemplo
é 192.168.25.197

```
~/Code/10_Sockets $ ./02_cliente_tcp.out 127.0.0.1 8080 "Ola
socket!"
~/Code/10_Sockets $ ./02_cliente_tcp.out 127.0.0.1 8080 12345
~/Code/10_Sockets $ ./02_cliente_tcp.out 192.168.25.197 8080
Oi!
~/Code/10_Sockets $
```

Execução em paralelo do servidor TCP e do cliente

```
~/Code/10_Sockets $ ./02_servidor_tcp.out 8080
Conexão do Cliente 127.0.0.1
Mensagem = Ola socket!
Conexão do Cliente 127.0.0.1
Mensagem = 12345
Conexão do Cliente 192.168.25.197
Mensagem = Oi!
Conexão do Cliente 192.168.25.197
Mensagem = sair
Fechando servidor local
~/Code/10_Sockets $
```

```
~/Code/10_Sockets $ ./02_cliente_tcp.out 127.0.0.1 8080 "Ola
socket!"
~/Code/10_Sockets $ ./02_cliente_tcp.out 127.0.0.1 8080 12345
~/Code/10_Sockets $ ./02_cliente_tcp.out 192.168.25.197 8080
Oi!
~/Code/10_Sockets $ ./02_cliente_tcp.out 192.168.25.197 8080
sair
~/Code/10_Sockets $
```