



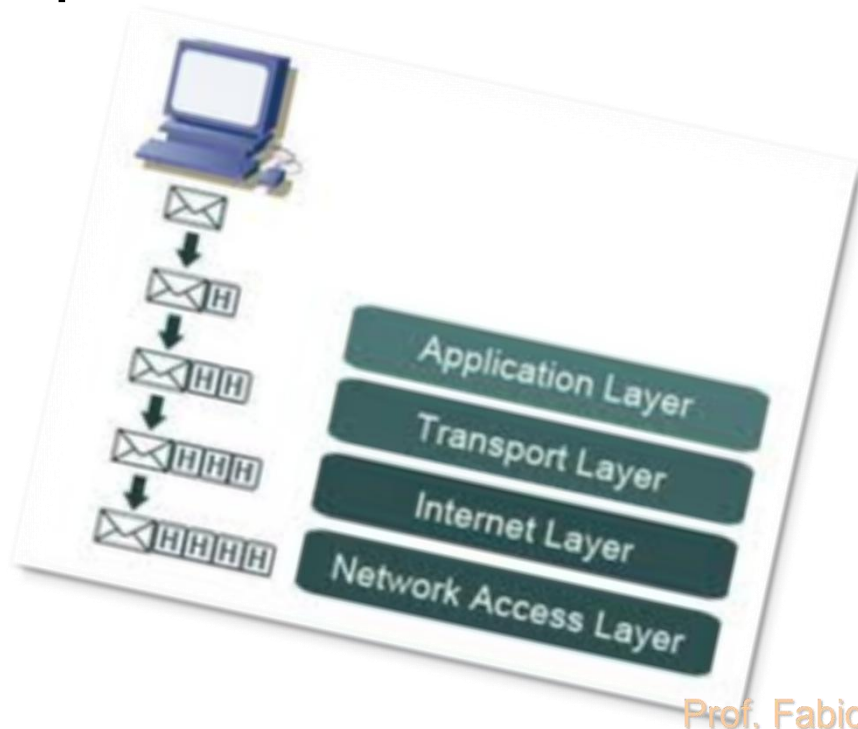
# Sockets

## *Tópicos Introdutórios*

**Prof. Fabio Alexandre SPANHOL, Dr.**  
[faspanhol@utfpr.edu.br](mailto:faspanhol@utfpr.edu.br)

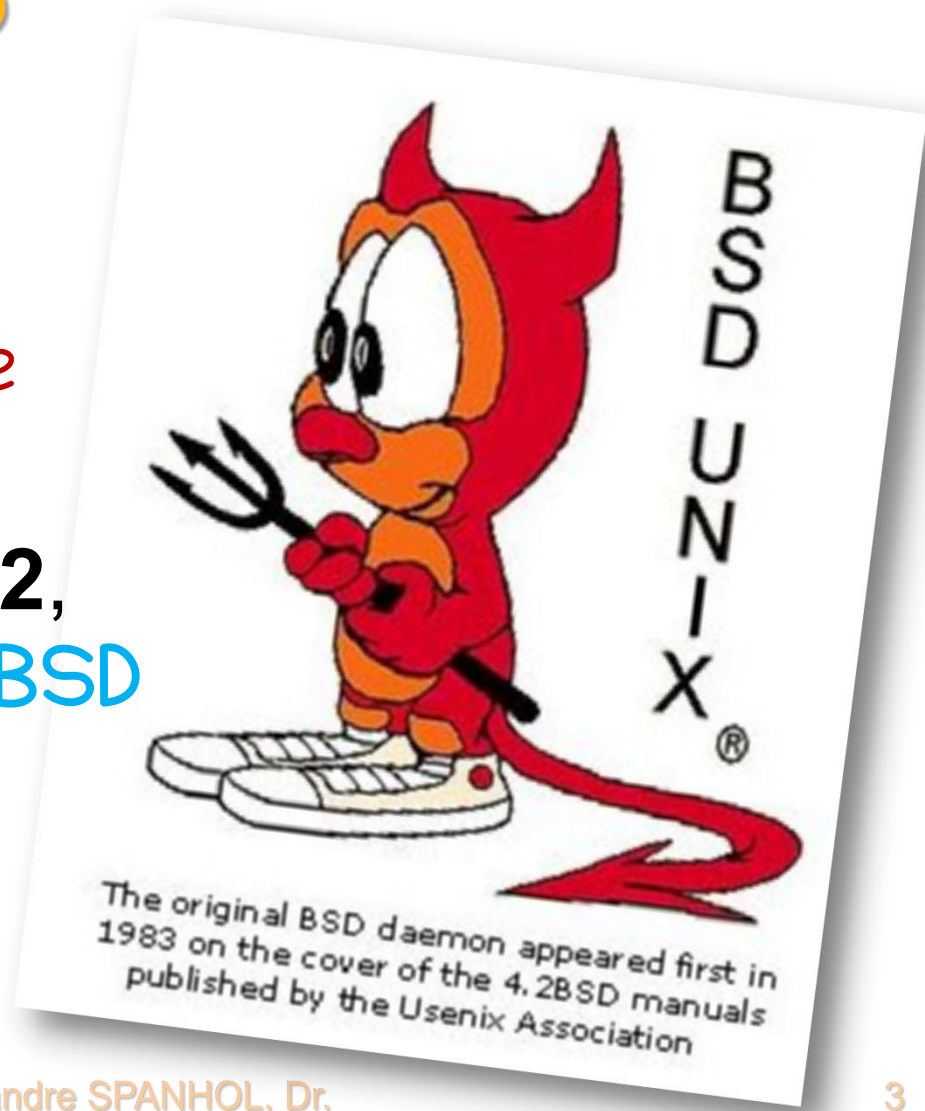
# Sockets: Introdução

O acesso aos serviços providos pela **camada de transporte TCP/IP** pode ser feito através de primitivas denominadas ***sockets***






# Sockets: Introdução

A interface socket e as facilidades de intercomunicação entre processos foram disponibilizadas em **1982**, na versão 4.1c do **Unix BSD** para máquinas VAX



# Sockets: Introdução

-  Em relação ao SO, o *socket* é um **canal de comunicação** com o exterior
-  Cada *socket* criado utiliza endereços para realizar referências entre si
-  O espaço de endereços possíveis é denominado **domínio**
  - ◉ Ex.: domínio Unix e **domínio Internet**

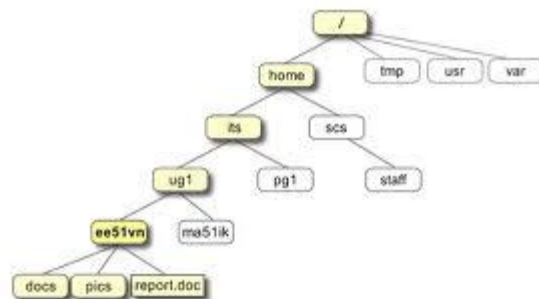


# Sockets: Introdução



## Domínio Unix

- identificado por `AF_UNIX` nas bibliotecas que implementam *sockets*
- endereço é composto por um **caminho dentro da árvore hierárquica do sistema de arquivos** acessível ao processo que criou o *socket*





# Sockets: Introdução



No **domínio Internet**, `AF_INET` nas bibliotecas *sockets*

- ⚡ endereço formado pelo **endereço IP** do *host* e um número de identificação denominado **porta (port number)**
- ⚡ **porta é um valor arbitrário** que referencia vários possíveis *sockets* abertos na mesma máquina
- ⚡ Para o domínio internet e o domínio XNS (protocolo NS da Xerox identificado por `AF_NS`), existem as denominadas **portas reservadas**



# Espera. Porta?

 Sim!

Identifica  
processos no  
mesmo *host*




# Portas



 IANA (*Internet Assigned Numbers Authority*)

 [iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml](http://iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml)

 Globalmente coordena DNS Root, endereçamento IP e outros recursos relacionados aos protocolos da Internet

 Define o espaço de endereços usados pelos protocolos TCP e UDP para endereçar serviços

- porta (**port number**)



# Portas



Três grande faixas de **números 16 bits**

 **Sistema** (*Well Known*) ▶ Portas reservadas ao *root*

- 0–1023

- ex.: **http** ▶ 80/tcp, **dns** ▶ 53/udp, **ssh** ▶ 22/tcp

 **Usuário** (*Registered*)

- 1024–49151

 **Dinâmicas** e/ou Privadas

- 49152–65535

```
spanhol@gandalf: ~
spanhol@gandalf: ~ 80x24
spanhol@gandalf:~$ cat /etc/services
# Network services, Internet style
#
# Note that it is presently the policy of IANA to assign a single well-known
# port number for both TCP and UDP; hence, officially ports have two entries
# even if the protocol doesn't support UDP operations.
#
# Updated from https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml .
#
# New ports will be added on request if they have been officially assigned
# by IANA and used in the real-world or are needed by a debian package.
# If you need a huge list of used numbers please install the nmap package.
tcprmx      1/tcp                                # TCP port service multiplexer
echo        7/tcp
echo        7/udp
discard     9/tcp                                sink null
discard     9/udp                                sink null
systat      11/tcp                                users
daytime     13/tcp
daytime     13/udp
qotd        17/tcp                                quote
```

# Sockets : Primitiva socket



Cria um novo ponto inicial de comunicação e aloca espaço na entidade de transporte local para tratamento da conexão

```
#include <sys/types.h>
```

```
#include <sys/socket.h>
```

```
int socket ( int domain, int type, int protocol );
```

⚡ Retorna, em caso de sucesso, um descritor de arquivo usado para **referenciar o socket** criado

⚡ **não atribui** diretamente um **endereço** ao **socket** criado!

# Sockets : Primitiva bind




 Anexa um endereço local a um *socket*

```
#include <sys/types.h>
#include <sys/socket.h>
int bind (int sockfd, struct sockaddr *my_addr,
int addrlen);
```

 Um processo **servidor** deve executar essa primitiva para **disponibilizar um endereço aos clientes**

 Na arquitetura cliente/servidor após sua execução os clientes podem se conectar ao servidor

# Sockets : Primitiva listen

-  torna o processo **servidor** apto a **aceitar conexões dos clientes**
-  aloca uma fila de espera para conexões pendentes
-  Se a fila já foi alocada mostra o tamanho da fila de conexões daquela entidade de transporte

```
#include <sys/socket.h>  
int listen (int s, int backlop);
```

# Sockets : Primitiva accept



faz com que um processo **servidor** permaneça **bloqueado até** que uma **solicitação de conexão** seja feita

```
#include <sys/types.h>
```


```
#include <sys/socket.h>
```

```
int accept (int s, struct sockaddr  
*addr, int addrlen);
```



# Sockets : Primitiva connect

 cliente solicita uma **conexão ao servidor**

 **bloqueia** o processo até que a **conexão** seja **estabelecida**

 Após estabelecida a conexão pode haver comunicação entre as duas entidades

```
#include <sys/types.h>
```

```
#include <sys/socket.h>
```

```
int connect (int sockfd, struct sockaddr  
*serv_addr, int addrlen);
```

# Sockets : Primitiva send/write



Envia dados pela conexão para a entidade destino

- Ⓢ Parâmetros: o descritor do *socket*, ponteiro para o *buffer* com os dados a serem enviados, o tamanho do *buffer*, códigos de condição especiais (normalmente 0)
- Ⓢ Retorna inteiro negativo em caso de erro

```
#include <sys/types.h>
```

```
#include <sys/socket.h>
```

```
int send (int s, const void *msg, int len,  
unsigned int flags);
```

# Sockets : Primitiva receive/read



Recebe dados da conexão

Ⓜ Parâmetros: o descritor do *socket*, ponteiro para uma estrutura onde os dados recebidos podem ser colocados, o tamanho da estrutura, códigos de condição especiais (normalmente 0)

Ⓜ Retorna o número de bytes recebidos

```
#include <sys/types.h>
```

```
#include <sys/socket.h>
```

```
int recv (int s, void *buf, int len, int flags);
```

# Sockets : Primitiva close



Termina a conexão

- ⚡ Como o encerramento da conexão é simétrico apenas quando ambos os processos tiverem executado essa primitiva a conexão será terminada
- ⚡ Parâmetros: o descritor do *socket* e códigos de término (normalmente 0)

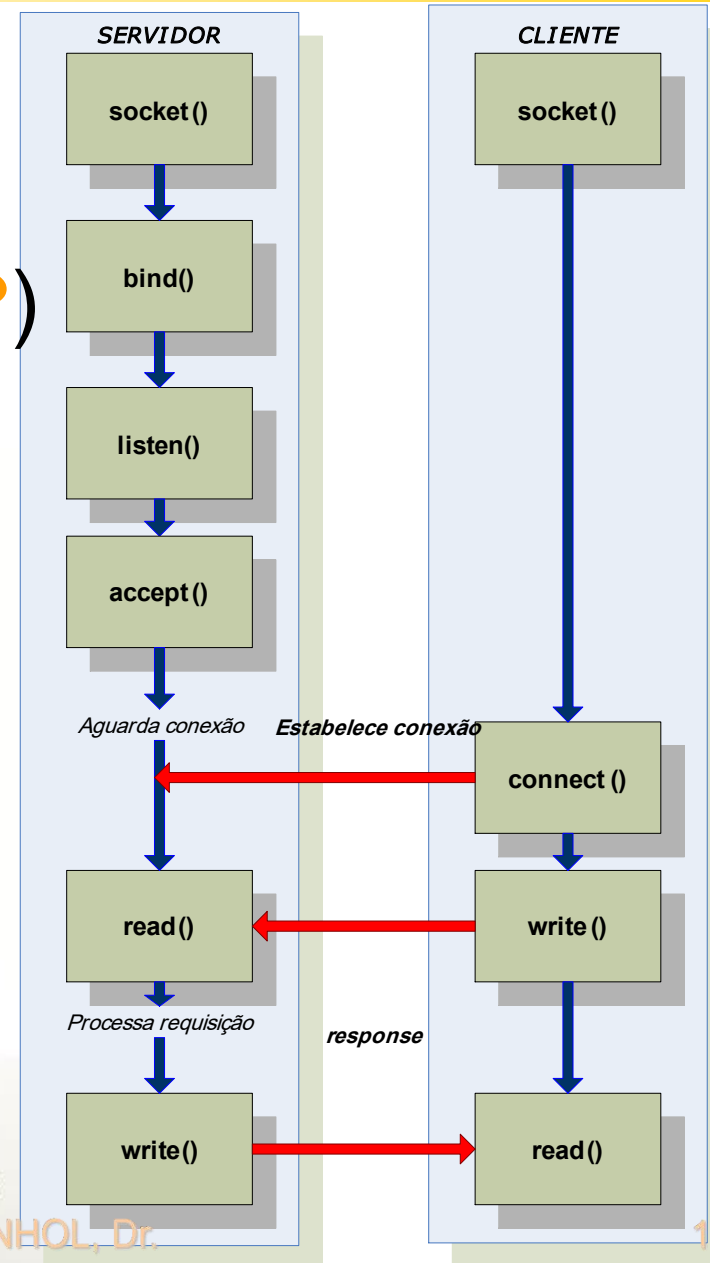
```
#include <sys/socket.h>
```

```
int shutdown (int s, int how);
```

# Chamadas socket



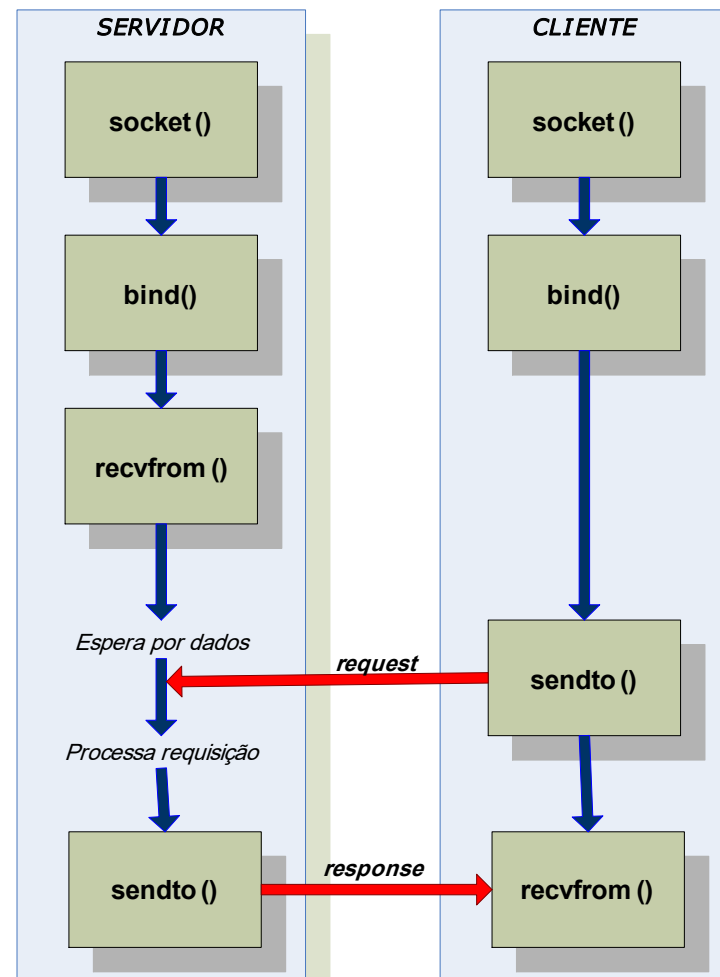
## Orientado a Conexão (TCP)





# Chamadas socket

 Não-orientado a Conexão (**UDP**)

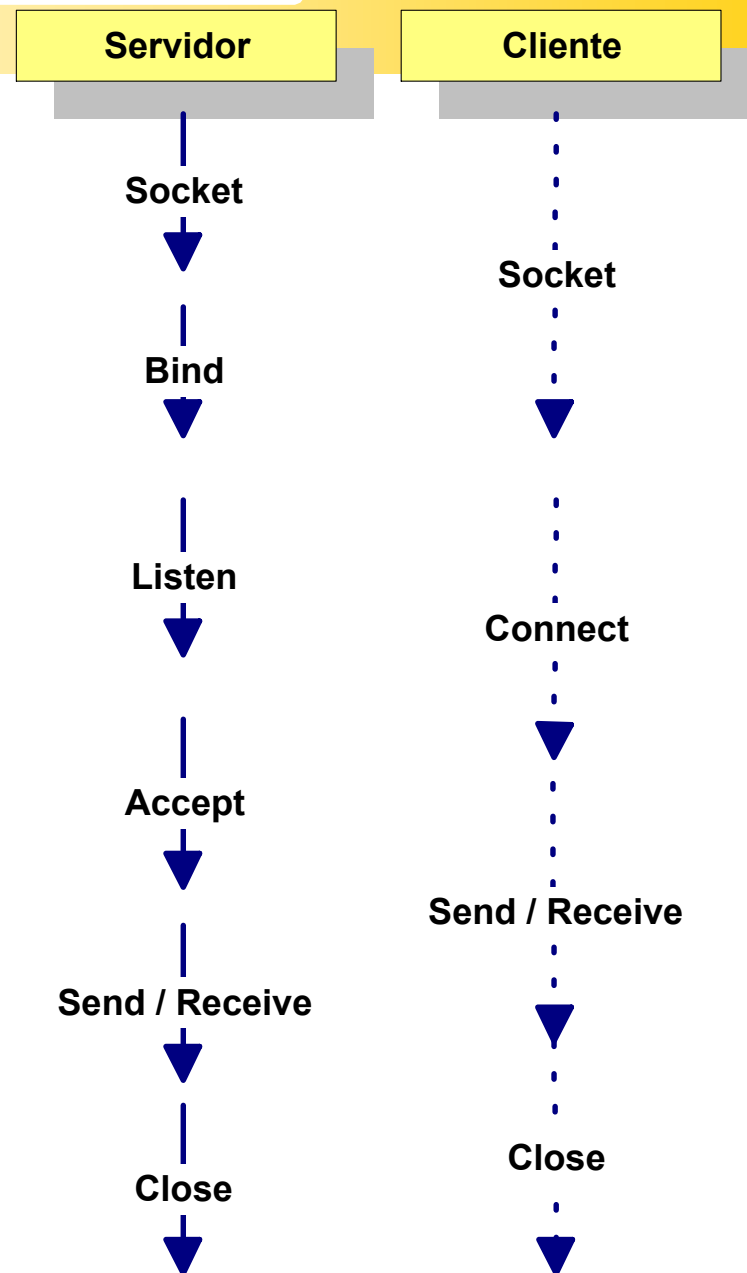


# Chamadas socket



## Ordem de execução

primitivas mais abaixo  
somente podem ser  
executadas após a  
execução das primitivas  
mais acima na precedência



# Servidor em C (1-3)

```

1  /* Servidor socket TCP
2     A porta é passada como argumento */
3  #include <stdio.h>
4  #include <stdlib.h>
5  #include <string.h>
6  #include <unistd.h>
7  #include <sys/types.h>
8  #include <sys/socket.h>
9  #include <netinet/in.h>
10 void error(const char *msg){
11     perror(msg);
12     exit(1);
13 }
14 const char response[]="Sou o Servidor. Recebi sua mensagem.";
15 int main(int argc, char *argv[]){
16     int sockfd, newsockfd, portno;
17     socklen_t clilen;
18     char buffer[256];
19     struct sockaddr_in serv_addr, cli_addr;
20     int n;
21     if (argc < 2) {
22         fprintf(stderr,"ERRO: porta não fornecida\n");
23         exit(1);
24     }

```

# Servidor em C (2-3)

```

25     sockfd = socket(AF_INET, SOCK_STREAM, 0);
26     if (sockfd < 0)
27         error("ERRO: abertura do socket");
28     bzero((char *) &serv_addr, sizeof(serv_addr));
29     portno = atoi(argv[1]);
30     serv_addr.sin_family = AF_INET;
31     serv_addr.sin_addr.s_addr = INADDR_ANY;
32     serv_addr.sin_port = htons(portno);
33     if (bind(sockfd, (struct sockaddr *) &serv_addr,
34             sizeof(serv_addr)) < 0)
35         error("ERRO: ligação");
36     printf("Servidor ouvindo na porta [%s]...\n", argv[1]);
37     listen(sockfd, 5);
38     clilen = sizeof(cli_addr);
39     newsockfd = accept(sockfd,
40             (struct sockaddr *) &cli_addr,
41             &clilen);
42     if (newsockfd < 0)
43         error("ERRO: aceite");


```



# Servidor em C (3-3)

```

44     bzero(buffer,256);
45     n = read(newsockfd,buffer,255);
46     if (n < 0) error("ERRO: leitura do socket");
47     printf("Mensagem recebida do cliente: %s\n",buffer);
48     n = write(newsockfd,response,strlen(response));
49     if (n < 0) error("ERRO: escrita no socket");
50     close(newsockfd);
51     close(sockfd);
52     return 0;
53 }
```



```

spanhol@aragorn: ~/aula

spanhol@aragorn:~/aula$ ./socket_server1 5488
Servidor ouvindo na porta [5488]...
```



# Cliente em C (1-3)

```

1  /* Cliente socket TCP.
2     Nome do servidor e porta passados por argumento.
3  */
4  #include <stdio.h>
5  #include <stdlib.h>
6  #include <unistd.h>
7  #include <string.h>
8  #include <sys/types.h>
9  #include <sys/socket.h>
10 #include <netinet/in.h>
11 #include <netdb.h>
12 void error(const char *msg){
13     perror(msg);
14     exit(0);
15 }
16 int main(int argc, char *argv[]){
17     int sockfd, portno, n;
18     struct sockaddr_in serv_addr;
19     struct hostent *server;
20
21     char buffer[256];
22     if (argc < 3) {
23         fprintf(stderr, "Use %s <host> <porta>\n", argv[0]);
24         exit(0);
25     }

```

# Cliente em C (2-3)

```

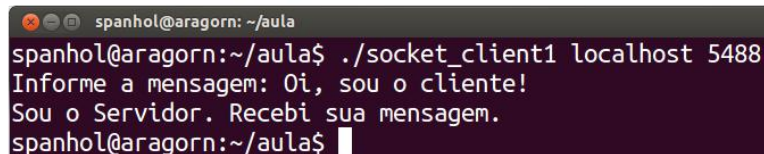
26     portno = atoi(argv[2]); /* porta */
27     sockfd = socket(AF_INET, SOCK_STREAM, 0);
28     if (sockfd < 0)
29         error("ERRO: abertura do socket");
30     server = gethostbyname(argv[1]);
31     if (!server) {
32         fprintf(stderr, "ERRO: host desconhecido\n");
33         exit(0);
34     }
35     bzero((char *) &serv_addr, sizeof(serv_addr));
36     serv_addr.sin_family = AF_INET;
37     bcopy((char *) server->h_addr,
38         (char *)&serv_addr.sin_addr.s_addr,
39         server->h_length);
40     serv_addr.sin_port = htons(portno);
41     if (connect(sockfd,
42         (struct sockaddr *) &serv_addr,
43         sizeof(serv_addr)) < 0)
44         error("ERRO: conexão");

```

# Cliente em C (3-3)

```

45     printf("Informe a mensagem: ");
46     bzero(buffer,256);
47     fgets(buffer,255,stdin);
48     n = write(sockfd,buffer,strlen(buffer));
49     if (n < 0)
50         error("ERRO: escrita no socket");
51     bzero(buffer,256);
52     n = read(sockfd,buffer,255);
53     if (n < 0)
54         error("ERRO: leitura do socket");
55     printf("%s\n",buffer);
56     close(sockfd);
57     return 0;
58 }
```



```

spanhol@aragorn: ~/aula
spanhol@aragorn:~/aula$ ./socket_client1 localhost 5488
Informe a mensagem: Oi, sou o cliente!
Sou o Servidor. Recebi sua mensagem.
spanhol@aragorn:~/aula$
```



# Sockets: Servidor em Python

```

1  # !/usr/bin/python3
2  # coding: utf-8
3
4  import time
5  from socket import *
6
7  HOST = '' # indica qualquer interface IPv4
8  PORT = 8888 # porta para ouvir
9
10 s = socket(AF_INET, SOCK_STREAM) # cria o socket TCP
11 s.bind((HOST, PORT)) # bind para a porta
12 s.listen(5) # ouvindo, até 5 conexões pendentes
13
14 while True:
15     conn, addr = s.accept() # obtém a conexão
16     print(f"Accepted connection from [{addr}]")
17     curr_time = time.ctime(time.time())
18
19     conn.send(str.encode(curr_time)) # envia
20     conn.close() # fecha a conexão
    
```



# Sockets: Cliente em Python



Cliente

🔌 que horas são?



```

1 # !/usr/bin/python3
2 # coding: utf-8
3
4 import sys
5 from socket import *
6
7
8 def get_srv_time(srv_addr='localhost', port=8888):
9     s = socket(AF_INET, SOCK_STREAM) # cria o socket
10
11     s.connect((srv_addr, int(port))) # conecta-se ao server
12
13     tm = s.recv(1024) # recebe até 1024 B
14
15     s.close()
16
17     print(f"Received from server:{tm}")
18
19
20 if __name__ == "__main__":
21     if len(sys.argv) < 3:
22         get_srv_time()
23     else:
24         get_srv_time(sys.argv[1], sys.argv[2])

```



# Sockets: Servidor em Java (1-2)

```

1  import java.net.*;
2  import java.io.*;
3  //Thread para atender as conexões
4  class Connection extends Thread {
5      public Connection (Socket s) {
6          outputLine = s;
7      }
8      public void run(){
9          try{
10             PrintWriter pout = new PrintWriter (outputLine.getOutputStream(), true);
11             //endereço da outra ponta
12             System.out.println ("#Cliente atendido no socket:" +
13                 outputLine.getRemoteSocketAddress());
14             //envia resposta para o cliente
15             pout.println("Agora é:" + new java.util.Date().toString());
16             //fecha o socket
17             outputLine.close();
18         }catch (java.io.IOException err) {
19             System.out.println(err);
20         }
21     }
22     private Socket outputLine;
23 }

```



# Sockets: Servidor em Java (2-2)

```

24 public class Server {
25     public Server() {
26         try {
27             S = new ServerSocket (5155);
28         } catch (java.io.IOException err){
29             System.out.println (err);
30             System.exit(1);
31         }
32         System.out.println ("Servidor ouvindo na porta 5155...");
33         try {
34             while (true) {
35                 client = S.accept();
36                 c = new Connection (client); //thread para atender
37                 c.start();
38             }
39         } catch (java.io.IOException err) {
40             System.out.println(err);
41         }
42     }
43     public static void main (String args[]) {
44         Server timeOfDayServer = new Server();
45     }
46     private ServerSocket S;
47     private Socket client;
48     private Connection c;
49 }

```

# Sockets: Cliente em Java

```

1  import java.net.*;
2  import java.io.*;
3  public class Client {
4      public Client () {
5          try{
6              Socket s = new Socket ("localhost",5155);
7              InputStream in = s.getInputStream();
8              BufferedReader bin = new BufferedReader
9                  (new InputStreamReader(in));
10             System.out.println (bin.readLine());
11             s.close();
12         }catch (java.io.IOException err) {
13             System.out.println (err);
14             System.exit(1);
15         }
16     }
17     public static void main (String args[]) {
18         Client clt = new Client();
19     }
20 }
    
```





# Sockets: Servidor 2 em Python

```

1  from socket import *
2
3  my_host = '' # localhost
4  my_port = 5050
5
6  sock = socket(AF_INET, SOCK_STREAM)
7  sock.bind((my_host, my_port))
8  sock.listen(5)
9
10 while 1:
11     conn, addr = sock.accept()
12     print(f'Servidor conectado por {addr}')
13     while 1:
14         data = conn.recv(1024)
15         if not data:
16             break
17         conn.send(str.encode(f'Echo ==>{data}'))
18     conn.close()
19

```

# Sockets: Cliente 2 em Python

```

1  import sys
2  from socket import *
3  server_host = 'localhost'
4  server_port = 5050
5
6  message = ('Hello world!', 'Again!')
7  if len(sys.argv) > 1:
8      server_host = sys.argv[1]
9      if len(sys.argv) > 2:
10         message = sys.argv[2]
11
12  sock = socket(AF_INET, SOCK_STREAM)
13  sock.connect ((server_host, server_port))
14
15  for line in message:
16      sock.send(str.encode(line))
17      data = sock.recv(1024).decode()
18      print(f'Received: {data}')
19  sock.close()
20

```