Comunicação entre Processos

Comunicação

- memória compartilhada
- troca de mensagens
  - base de comunicação em sistemas distribuídos

PUC RIO

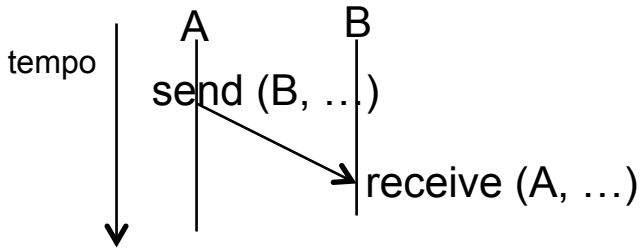
Mensagens básicas

- send (destino, msg)
- receive (origem, mensagem)
  - questões
    - semântica de operações
    - especificação de origem e destino
    - formato de mensagem

PUC RIO

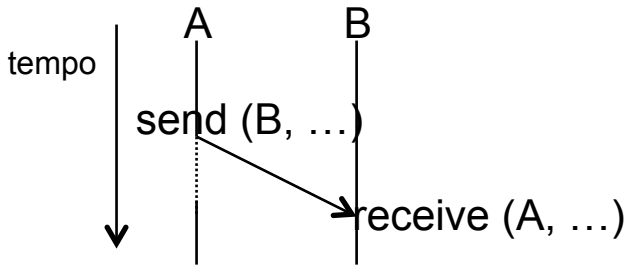
ENVIO SÍNCRONO E ASSÍNCRONO

- envio assíncrono: execução procede imediatamente



- bufferização
- concorrência
- determinismo

- envio síncrono: execução só procede quando destinatário recebe msg



PUC RIO

RECEBIMENTO SÍNCRONO E ASSÍNCRONO

- recebimento síncrono
  - alternativa tradicional
  - execução procede quando há algo a tratar
  - alternativa de recebimento com timeout…

PUC RIO

BLOQUEIOS

- recebimento com bloqueio leva à suspensão de uma linha de execução

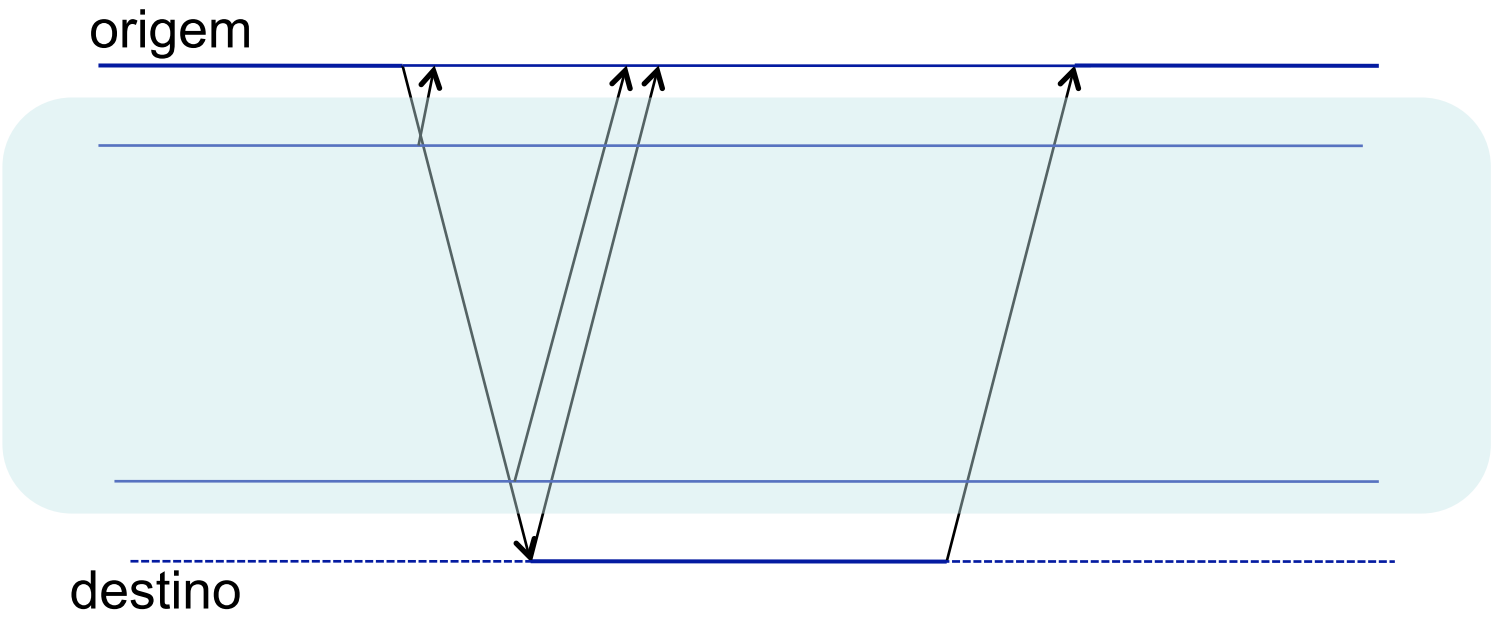
- futuros: sobreposição de computação e comunicação
  - ligado ao conceito de RPC
- orientação a eventos - recebimento implícito

PERSISTÊNCIA

- mensagens permanecem disponíveis independentemente do tempo de vida dos processos origem e destino;
  - muitas vezes sistemas com persistência associados a modelos onde não há especificação de origem e destino

ENVIO E RECEBIMENTO

- sincronização em diferentes pontos da interação



FORMATO DE MENSAGENS

- valores com tipos
  - canais declarados
  - chamadas remotas…
- sequências de bytes
  - interpretação por conta do programa
- problemas de conversão de formatos
  - bibliotecas de conversão p/ formato padrão

ESPECIFICAÇÃO DE PARES

- identificação de processos
  - volatilidade e especificidade de endereços locais de processos
  - serviços de nomes
  - caixas de correio ou canais

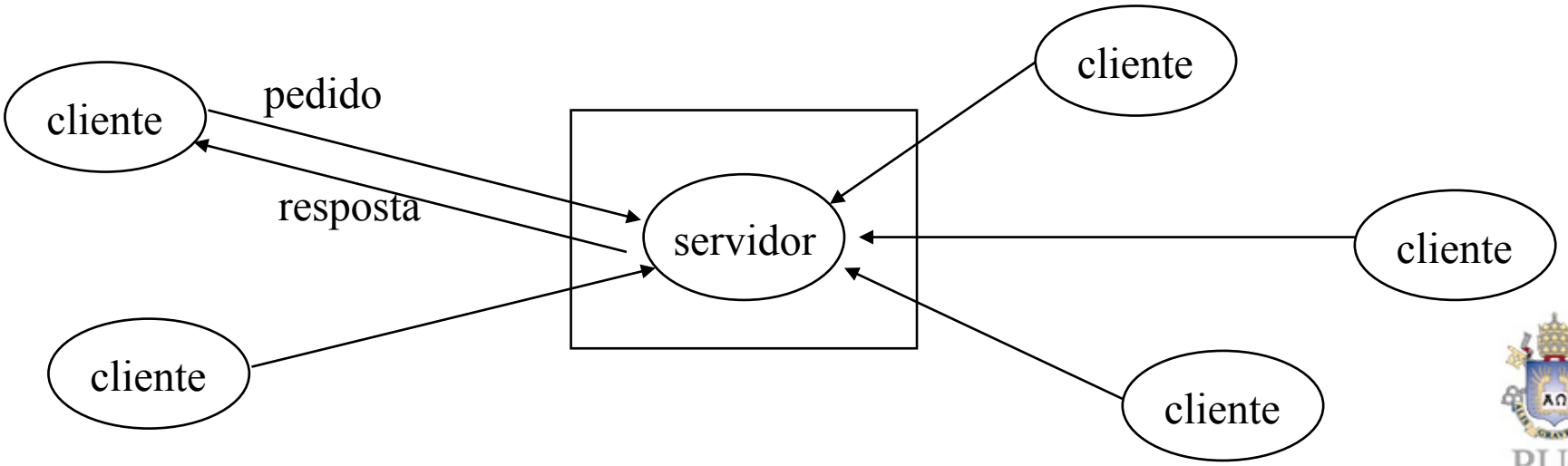
ARQUITETURAS DE SISTEMAS

- (paradigmas, padrões, …)
- necessidade de organizar a comunicação para entender o programa distribuídos
- arquiteturas:
  - centralizadas: cliente-servidor
  - decentralizadas: p2p. filtros, ...

PUC RIO

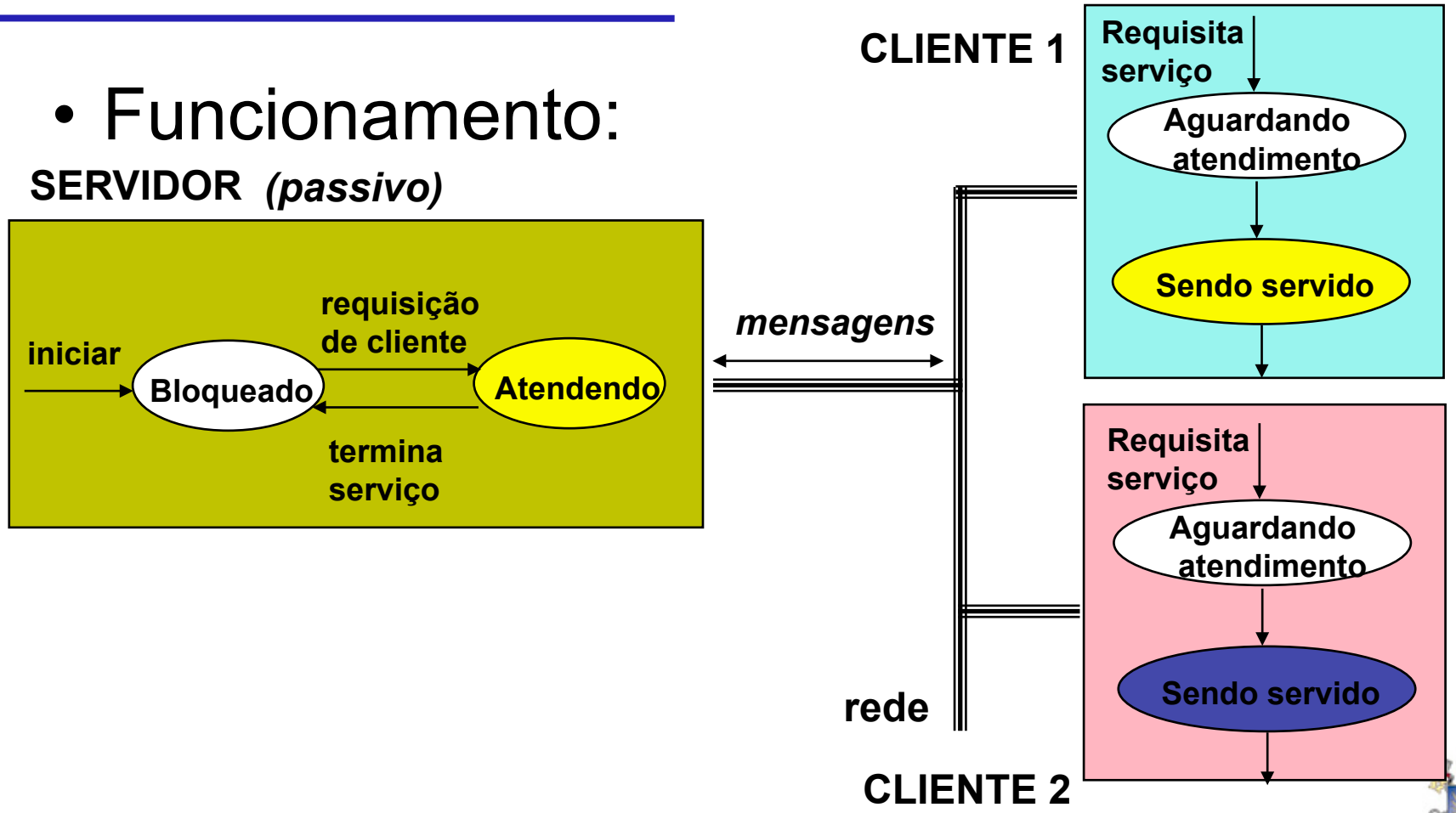
Cliente-Servidor

- Modelo mais usado para aplicações distribuídas não paralelas;
- Um processo *servidor* está sempre a espera de comunicação;
- O processo *cliente* tem a iniciativa de começar a comunicação quando deseja algum serviço.



O Modelo Cliente-Servidor

- Funcionamento:



CLIENTE-SERVIDOR

- servidores clássicos: arquivos, impressão, BD, ...
- servidor como executor de chamadas
  - serviços/funções: criptografa, filtra, autentica, ...
  - outros: paralelismo SPMD
- divisão de papéis (cliente ou servidor) não é rígida

APLICAÇÕES COM N CAMADAS

- cada camada intermediária atua como cliente e servidora
- exemplo clássico: aplicações web
  - interface usuário
  - processamento
  - dados

CLIENTES MAGROS E GORDOS

- maior processamento do lado do cliente ou do servidor?
  - configuração de máquinas
  - estado da rede
- adaptação dinâmica?

MODELOS DE SERVIDORES

- servidores iterativos
- servidores concorrentes
  - multiprocesso
    - (ou multithread)
  - monoprocesso
    - atendimento a vários clientes “misturado” em uma linha de execução

SIST. MSGS P/ CLIENTE/SERV

- exemplo: tcp
- protocolo construído para comunicação entre cliente e servidor

PUC RIO

EXEMPLO: BSD SOCKETS

- comunicação entre processos
  - API para troca de mensagens com vários protocolos
  - histórico: desenvolvimento de interface para TPC/IP no Unix BSD
- idéia é manter interface de entrada e saída
- O socket é identificado por um inteiro chamado *descritor do socket*

descritor_do_socket = socket()

TCP/IP

- transferência de streams de bytes
- suporte a modelos cliente-servidor
- TCP/IP estabelece abstração chamada porta
  - exemplo de caixa de correio!!

PROGRAMANDO COM SOCKETS

- socket para tcp:
  - espera uma conexão
  - inicia uma conexão
- socket passivo: espera por uma conexão (usado por servidores)
- socket ativo: inicia uma conexão (usado pelos clientes)
- complexidade: parâmetros que o programador pode/tem que configurar

PUC RIO

PROGRAMANDO COM SOCKETS

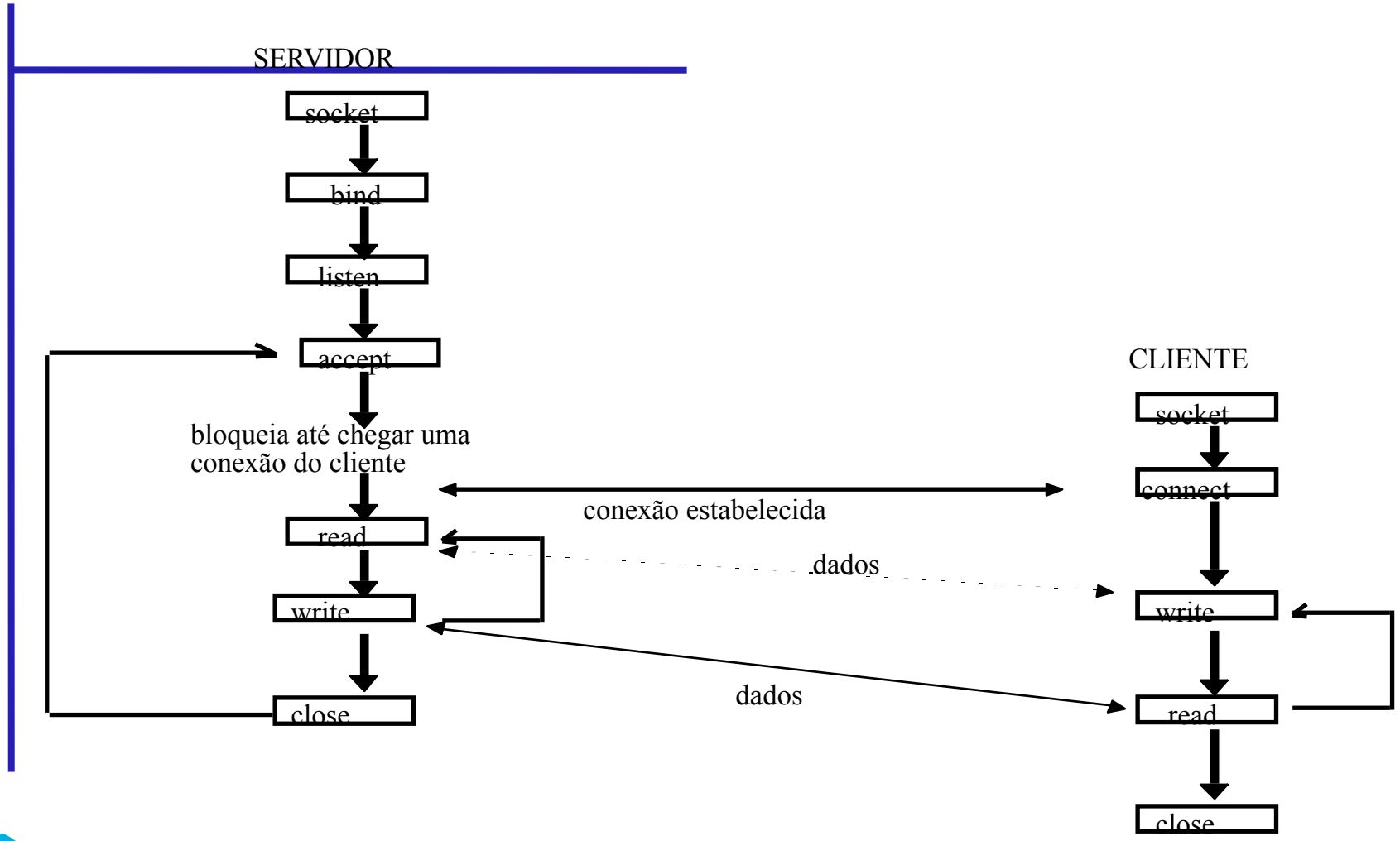
- alguns parâmetros:
  - transferência de:
    - stream (TCP)
    - datagrama (UDP)
  - endereço remoto:
    - específico (geralmente usado pelo cliente)
    - inespecífico (geralmente usado pelo servidor)

Chamadas da API socket

- socket( ) -> cria um socket usado para comunicação e retorna um descritor
- connect( ) -> depois de criar um socket um cliente chama connect para estabelecer uma conexão com um servidor, usando o descritor do socket
- write( ) -> para enviar dados através de uma conexão TCP
- read( ) -> para receber dados através de uma conexão TCP
- close( ) -> para desalocar o socket.
- bind( ) -> usado por servidores para especificar uma porta na qual ele irá esperar conexões.
- listen( ) -> servidores chamam o listen para colocar o socket no modo passivo e torná-lo disponível para aceitar conexões
- accept( ) -> depois que um servidor chama *socket* para criar um socket, *bind* para especificar seu endereço e *listen* para colocá-lo no modo passivo, ele deve chamar o *accept* para pegar a primeira solicitação de conexão na fila.

PUC RIO

EXEMPLO DO USO DE SOCKETS



Código do Servidor usando o Protocolo TCP/IP

```
main(int argc, char *argv[])
{       int                             sockfd, newsockfd, tam_cli, filho_pid;
        struct sockaddr_in      cli_addr, serv_addr;
        if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0)   /* abre um socket TCP */
                erro;
        /* Liga o processo servidor ao seu endereço local */
        bzero((char *)&serv_addr, sizeof(serv_addr));
        serv_addr.sin_family = AF_INET;
        serv_addr.sin_addr.s_addr = htonl(INADDR_ANY);
        serv_addr.sin_port = htons(SERV_TCP_PORT);
        if  (bind(sockfd, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) < 0)  erro;
        listen(sockfd, 5);
        for (;;) {
                newsockfd = accept(sockfd, (struct sockaddr *)&cli_addr, &tam_cli);
                if (newsockfd <0)   erro;
                if (( filho_pid = fork()) < 0)  erro;
                else if (filho_pid == 0)   {          /* processo filho */
                        close(sockfd);              /* fecha o socket original */
                        str_echo(newsockfd); /* processa a solicitação */
                        exit();
                }
                close(newsockfd);             /* processo pai */
        }
}
```

⇐ servidor concorrente

Código do Cliente usando o Protocolo TCP/IP

```
main(int argc, char *argv[])
{     int                                sockfd;
      struct sockaddr_in       serv_addr;
      if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0)   /* abre um socket TCP */
            erro;
      /* Preenche a estrutura “serv_addr” com o endereço do servidor com o qual ele deseja se conectar*/
      bzero((char *)&serv_addr, sizeof(serv_addr));
      serv_addr.sin_family = AF_INET;
      serv_addr.sin_addr.s_addr = inet_addr(SERV_HOST_ADDR);
      serv_addr.sin_port = htons(SERV_TCP_PORT);
      if (connect(sockfd, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) < 0)  erro;
      str_cli(stdin, sockfd);    /* envia a solicitação */
      close(sockfd);
      exit();
      }
}
```

Endereçamento

- O socket oferece uma estrutura diferente para cada protocolo onde o programador especifica os detalhes de endereçamento
  - Para o protocolo TCP/IP a estrutura *sockaddr_in* especifica o formato de endereço

```
struct sockaddr_in {
        u_short sin_family;  /* tipo de endereço */
        u_short sin_port;    /* número da porta */
        u_long sin_addr;     /* endereço IP */
}
```

Definindo o endereço de um servidor

```
struct sockaddr_in {
        u_short sin_family;  /* tipo de endereço */
        u_short sin_port;    /* número da porta */
        u_long sin_addr;     /* endereço IP */
   }
```

```
struct sockaddr_in   serv_addr;
bzero((char *)&serv_addr, sizeof(serv_addr));
serv_addr.sin_family =  AF_INET;
serv_addr.sin_addr.s_addr = htonl (INADDR_ANY)
serv_addr.sin_port = htons("6644")
```

PUC RIO

Portas

- mecanismo de associação a processos específicos:
  - o processo solicita uma porta específica (tipicamente usado por servidores)
  - o sistema automaticamente atribui uma porta para o processo (tipicamente usado por clientes)
    - o processo especifica porta = 0 antes de fazer a chamada ao bind. O bind atribui uma porta automaticamente

sockets em Lua

- biblioteca LuaSocket
  - uso de facilidades Lua para criação de interface simplificada

  - `socket.tcp()`

  - `master:bind(address, port)`
  - `master:listen(backlog)`

  - `master:connect(address, port)`
  - `client:receive([pattern [, prefix]])`
  - `client:send(data [, i [, j]])`

Trabalho 1