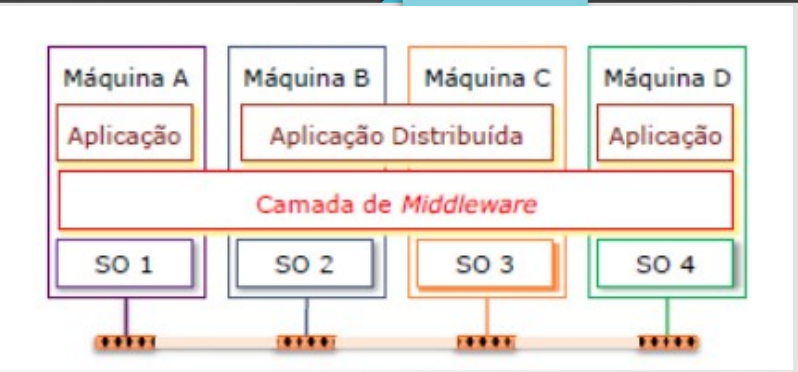
Sistemas Distribuídos



Aula 5
Sockets

Profa. Gisane A. Michelon
Universidade Estadual do Centro Oeste

Sumário

- **Aula anterior:**
  - Modelos de comunicação entre processos:
    - Elementos básicos e desejáveis da comunicação
    - Middleware
- **Nesta aula veremos:**
  - Padrões de comunicação: Sockets

Objetivos de aprendizagem

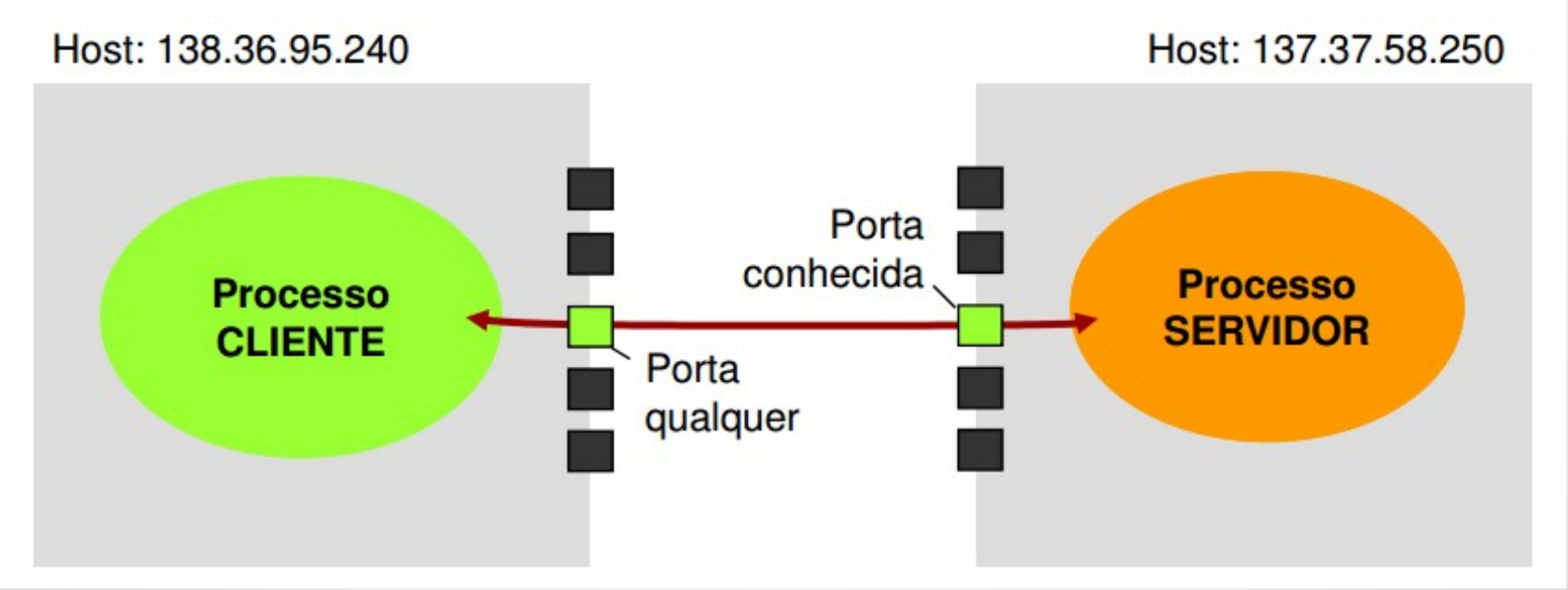
Ao final desta aula, você deve entender o funcionamento de sockets.

Sockets

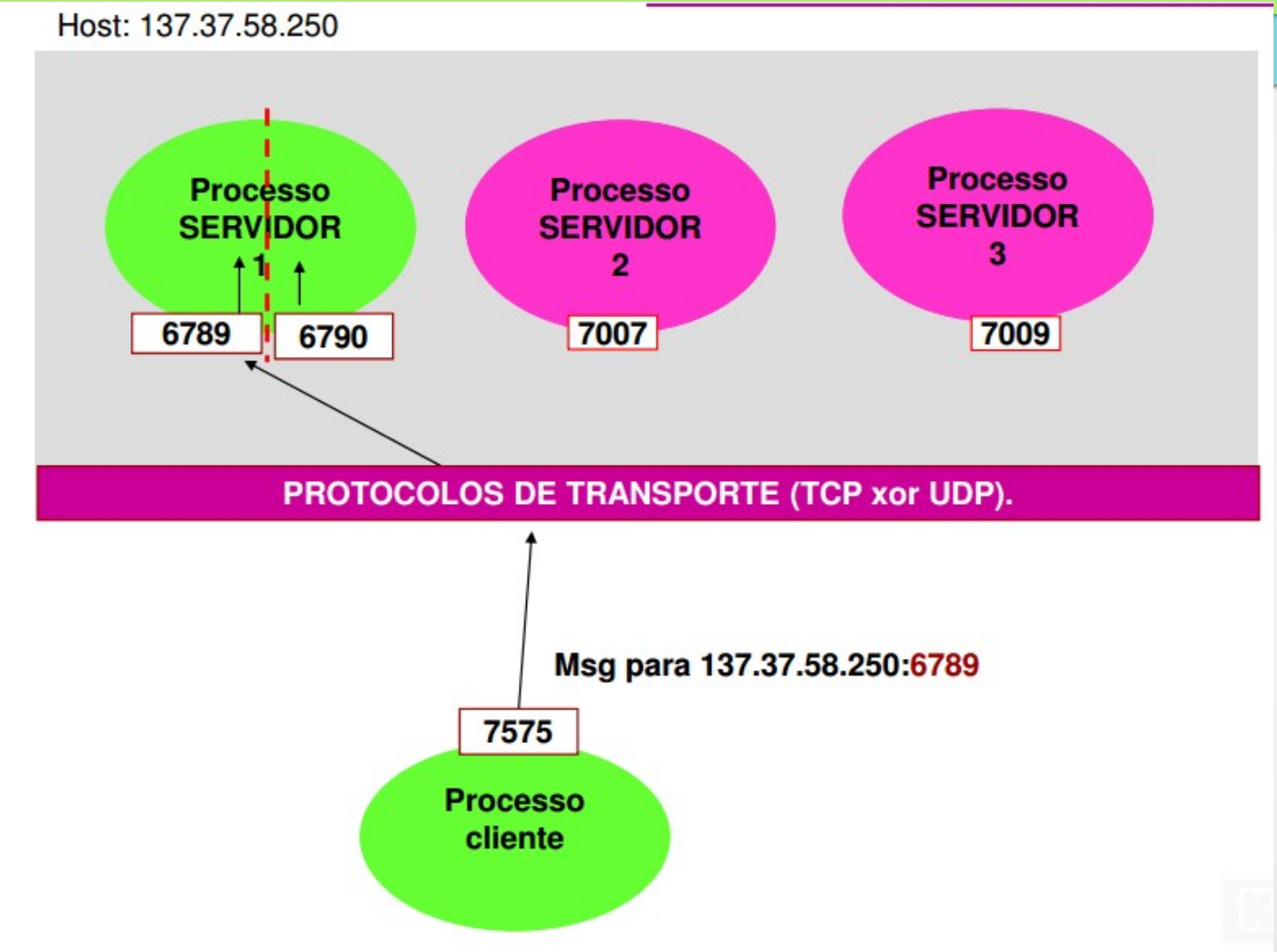
Protocolos UDP/TCP usam soquetes para sua comunicação interprocessos distribuídos nas aplicações cliente-servidor.

Formado por: **endereço_IP_remoto: porta_remota**

Porta: destino da mensagem dentro de um computador.



Sockets



Endereço global x Endereço local.

Exemplo de utilização.

Sockets

Uma porta não pode ser compartilhada por vários processos (usar o mesmo número de porta que outros processos ou serviços do mesmo host).

- Execute no shell dos: *netstat -na*

```
Active Connections

  Proto  Local Address          Foreign Address        State
  TCP    0.0.0.0:135            0.0.0.0:0              LISTENING
  TCP    0.0.0.0:445            0.0.0.0:0              LISTENING
  TCP    0.0.0.0:2804           0.0.0.0:0              LISTENING
  TCP    127.0.0.1:1033         0.0.0.0:0              LISTENING
  TCP    127.0.0.1:1044         127.0.0.1:1045         ESTABLISHED
  TCP    127.0.0.1:1045         127.0.0.1:1044         ESTABLISHED
  TCP    127.0.0.1:1047         127.0.0.1:1048         ESTABLISHED
  TCP    127.0.0.1:1048         127.0.0.1:1047         ESTABLISHED
  TCP    200.17.96.134:139      0.0.0.0:0              LISTENING
  TCP    200.17.96.134:139      200.17.96.175:1209     ESTABLISHED
  TCP    200.17.96.134:2169     200.17.96.235:139      ESTABLISHED
  UDP    0.0.0.0:445            *:*
  UDP    0.0.0.0:500            *:*
  UDP    0.0.0.0:1039           *:*
```

Para haver comunicação é preciso fazer o *binding* (ligação) entre os processos.

Um socket é associado a um determinado protocolo: UDP ou TCP.

Comunicação por datagrama UDP

- Baseada em uma unidade de transmissão: datagrama.
- Transmitido do remetente ao destino sem confirmação ou tentativa de reenvio:
  - Mensagens podem não chegar.
- Aspectos a serem considerados:
  - Tamanho da mensagem;
  - Bloqueio;
  - *Timeout*;
  - Recepção anônima.

Aspectos da comunicação UDP

- **Tamanho da mensagem**:
  - Necessidade de especificar o tamanho *buffer* de recepção:
    - Se buffer menor que o necessário – mensagem truncada.
- **Bloqueio**:
  - Geralmente *Send* é não bloqueante e *Receive* é bloqueante;
  - Recomendável uso de *threads*.

- **Recepção anônima**:
  - *Receive* não especifica uma origem, porém é possível identificar a fonte.

Modelo de falhas UDP

- **Comunicação confiável** implica em **duas propriedades**:
  - **Validade**: qualquer mensagem é entregue a seu destino;
  - **Integridade**: mensagens não podem ser corrompidas nem duplicadas.
- **Datagramas UDP falham** por:
  - **Falhas por omissão**: mensagens podem ser perdidas ou descartadas;
  - **Ordenamento**: mensagens podem ser entregues fora de ordem.
- Responsabilidade dos aplicativos tratarem as falhas.

Comunicação por fluxo TCP

- Define a abstração de fluxo (*stream*)
- Características de um fluxo TCP:
  - Não define limite de tamanho para mensagens;
  - Efetua controle de erro;
  - Regula controle de fluxo (regula a taxa de leitura e escrita no fluxo para prevenir *overflow*);
  - Garante entrega ordenada e não duplicação dos dados;
  - Define a abstração de conexão como identificadores de processo origem e destino.

Aspectos na comunicação TCP

- **Correspondência entre itens de dados**:
  - Processos devem concordar quanto ao conteúdo dos dados transmitidos:
    - Se um enviar n *bytes* como sendo inteiro o outro deve concordar.
- **Bloqueio**:
  - Dados enviados em um fluxo são mantidos até serem lidos:
    - Processo destino é bloqueado se tentar ler dados não disponíveis;
    - Processo remetente é bloqueado pelo fluxo TCP se não há espaço disponível no destinatário para recepção.
- ***Timeout***:
  - Limitar tempo de espera do *receive*;
  - Problema: dimensionar o *timeout*.
- ***Threads***:
  - Recomendável para simplificar programação sem bloqueio (várias *threads* = vários fluxos = maiores *buffers*).

Modelo de falhas do TCP

- **Funcionamento do TCP**:
  - **Validade**: usa *timeout* e retransmissões para tratar perda de mensagens;
  - **Integridade**: uso de *checksum* e número de sequência para garantir mensagens não corrompidas nem duplicadas.
- **Não totalmente confiáveis**:
  - Conexões TCP podem ser desfeitas:
    - Processo não distinguem entre falha de rede e falha de processo.

Comunicação entre Cliente e Servidor

Servidor: criação do socket, associando-o a um endereço local (onde fica aguardando a solicitação de conexão pelo cliente). Após aceitar a conexão de um cliente, lê as requisições, envia reposta e ao final fecha o socket.

Cliente: criação do socket, tenta estabelecer conexão com o servidor. Uma vez estabelecida, envia a requisição e aguarda resposta. Ao final fecha o socket.

API's Sockets

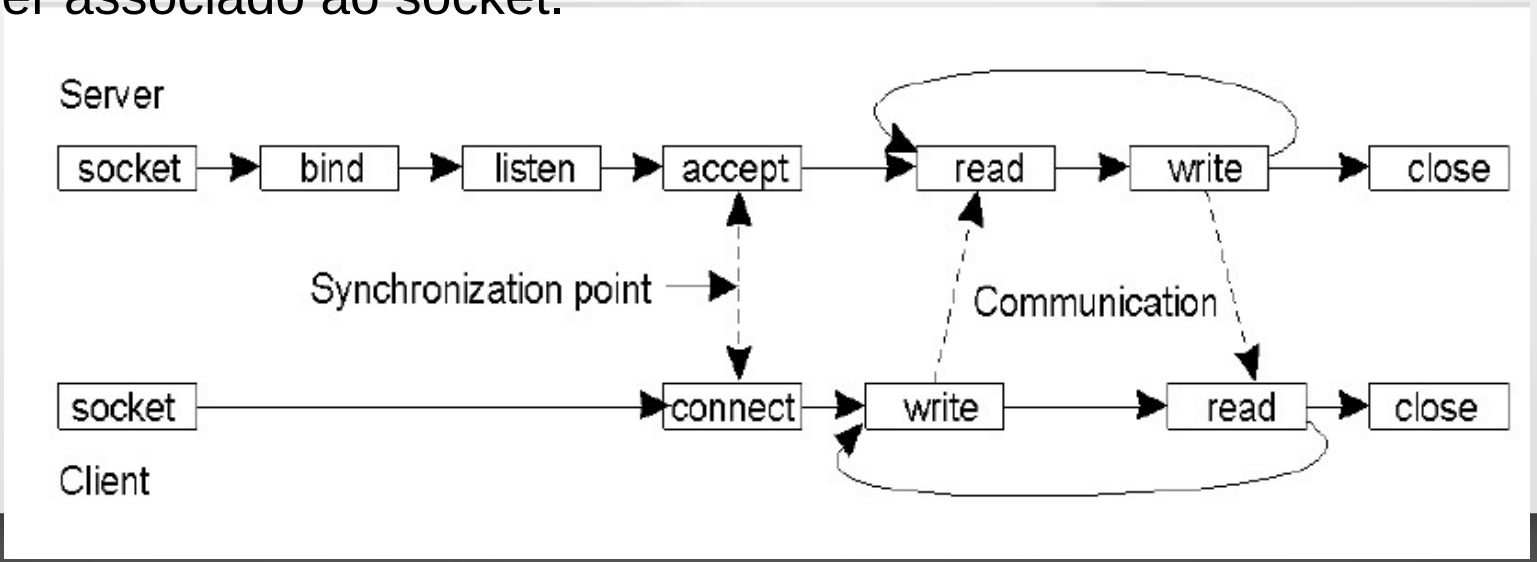
Bind: utilizada apenas pelo servidor (associa um IP e porta TCP/UDP para o processo servidor).

Listen: indica ao sistema para colocar o socket em modo de espera (aguarda conexões de clientes).

Connect: tenta estabelecer uma conexão com um socket.

Accept: cria um novo socket depois do estabelecimento de uma conexão para iniciar a comunicação.

Read / Write: lê o conteúdo do buffer associado ao socket e escreve os dados em um buffer associado ao socket.



Exemplo Cliente e Servidor Socket

```
class Server:
  def run(self):
    s = socket(AF_INET, SOCK_STREAM)
    s.bind((HOST, PORT))
    s.listen(1)
    (conn, addr) = s.accept()  # returns new socket and addr. client
    while True:                # forever
      data = conn.recv(1024)   # receive data from client
      if not data: break       # stop if client stopped
      conn.send(data+b"*")     # return sent data plus an "*"
    conn.close()               # close the connection



class Client:
  def run(self):
    s = socket(AF_INET, SOCK_STREAM)
    s.connect((HOST, PORT)) # connect to server (block until accepted)
    s.send(b"Hello, world") # send same data
    data = s.recv(1024)     # receive the response
    print(data)             # print what you received
    s.send(b"")             # tell the server to close
    s.close()               # close the connection
```

Retirado de VAN STEEN and TANENBAUM, 2023.

Referências

- COULOURIS, George; DOLLIMORE, Jean; KINDBERG, Tim; Blair, Gordon. Sistemas distribuídos: conceitos e projeto. 5. ed. Porto Alegre: Bookman, 2013.
- TANENBAUM, Andrew S. and VAN STEEN, Maarten. Sistemas distribuídos: princípios e paradigmas. 2. ed. São Paulo: Pearson / Prentice Hall, 2007.
- VAN STEEN, Maarten and TANENBAUM, Andrew S., Distributed Systems, 4th ed., distributed-systems.net, 2023. Disponível em https://www.distributed-systems.net/index.php/books/ds4/.
- LAINE, Jean M. Aulas da disciplina de Sistemas Distribuídos, 2015.
- MONTEIRO, Gustavo. Slides de P2P - Sistemas Distribuídos Monitor at DGTI – UFLA. Disponível em "http://pt.slideshare.net/gmsilva41/p2p-sistemas-distribudos".
- FERRAZ, Carlos. Material de aula - Modelos Arquiteturais. Universidade Federal de Pernambuco.
- TACLA, Cesar Augusto. Material de aula Curso de especialização em Teleinformática – Disciplina Sistemas Distribuídos http://www.dainf.ct.utfpr.edu.br/~tacla/EspSD/Aula3/0050-Sockets-comentados.pdf. 2003.
- NICOLAY, DANIEL AZEVEDO Sistemas Distribuidos https://www.overleaf.com/articles/trabalho-sistemas-distribuidos/yfnhxtgqbmkr/viewer.pdf. 2015.