

LAB REDES DE COMPUTADORES

PROFESSORA: CAMILA OLIVEIRA

CCT- UFCA







AULA 05

SOCKET





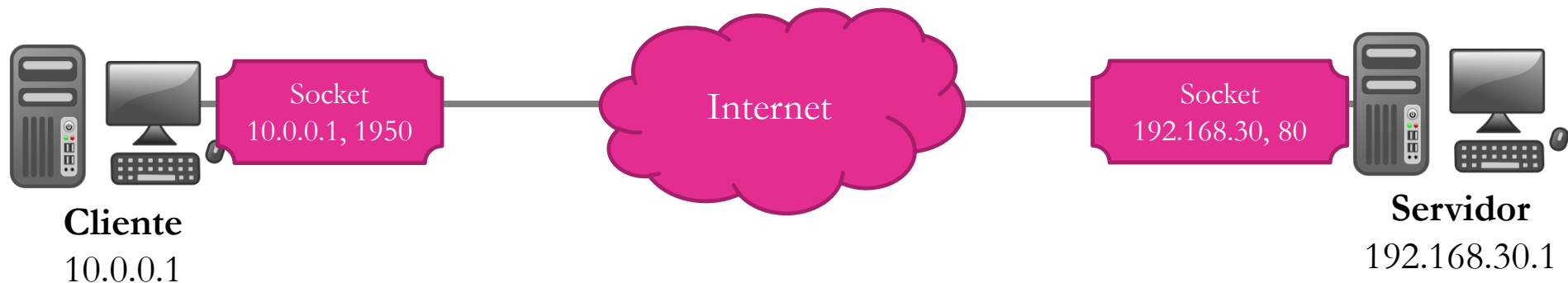
SUMÁRIO

- 
- Definição
-
- 
- Funções
-
- 
- Implementação
-
- 
- Comunicação

SOCKETS

São as extremidades de uma comunicação full-duplex entre dois processos diferentes na mesma máquina ou em máquinas diferentes. Essa extremidade (endpoint) é composta por um endereço IP e um número de porta.

- Sockets são do sistemas operacional e usamos através de bibliotecas oferecidas pelas linguagens de programação.
- Curiosidade: se quiser saber mais sobre socket no linux, basta digitar: **man socket**.
- As aplicações de socket mais comuns são as aplicações cliente/servidor.



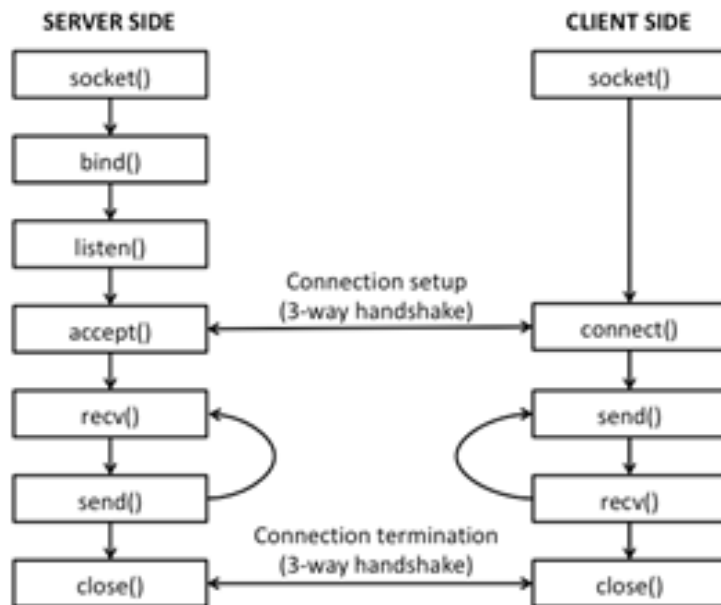
SOCKETS

Exemplo: web

- Abre o navegador e digita o endereço de uma página -> um socket é criado pelo navegador!
- Neste caso, o navegador é o cliente e a máquina onde o site procurado está armazenado é o servidor.
- Para que o request HTTP sai do navegador e chegue no servidor um conjunto de etapas, que são transparentes para o usuário, devem ser executadas internamente pelo sistema operacional. A primeira delas é a criação do socket para comunicação entre cliente e servidor.

FUNÇÕES

Para implementar um software que usa comunicação por socket é necessário entender o fluxo dos comandos que devem ser usados pela linguagem de programação empregada na implementação do software.



Utilizaremos o módulo do Python que oferece uma interface para a API de socket de Berkely. Esta API é o conjunto de funções de comunicação proposto pela a universidade de Berkeley no ano de 1980.

IMPLEMENTAÇÃO

Servidor

1. `socket()`: demanda ao SO a criação de um socket.
 - Parâmetros – família de protocolos usadas e o modo de transporte de dados que será utilizado.
 - Exemplo – `AF_INET` para endereços IPv4 e `SOCKET_STREAM` para o protocolo TCP na camada de transporte.
2. `bind()`: faz a relação entre a estrutura socket e o par endereço IP, porta do servidor.
3. `listen()`: usado para ativar o socket servidor que fica no estado aguardando solicitações dos clientes. Isso acontece quando o cliente chama a função `connect()`.
 - Recebe como parâmetro a quantidade de conexões que serão enfileiradas pelo TCP até que o servidor execute o `accept`.

IMPLEMENTAÇÃO

Servidor

4. `accept()`: aceita pedido de conexão dos clientes.
 - Retorna tupla (`conn`, `address`), onde o *conn* é um novo objeto socket através do qual os dados serão enviados e recebidos, e o *address* é o endereço ligado ao socket no lado cliente da comunicação.
5. `send()` e `recv()`: usados juntos e intercalados.
 - Se o cliente executa o comando `send()` então o servidor executa o comando `recv()` e vice-versa.
6. `close()`: encerra conexão entre o cliente e o servidor.

IMPLEMENTAÇÃO

Cliente

1. `socket()`: demanda ao SO a criação de um socket.
 - Parâmetros – família de protocolos usadas e o modo de transporte de dados que será utilizado.
 - Exemplo – `AF_INET` para endereços IPv4 e `SOCKET_STREAM` para o protocolo TCP na camada de transporte.
2. `connect()`: usado para dizer ao socket do cliente que ele deve se conectar ao socket do servidor.
 - Parâmetros: IP e porta do servidor
3. `send()` e `recv()`: usados juntos e intercalados.
 - Se o cliente executa o comando `send()` então o servidor executa o comando `recv()` e vice-versa.
4. `close()`: encerra conexão entre o cliente e o servidor.

IMPLEMENTAÇÃO

Sockets TCP

1. Criar o socket.

Especificar o tipo de socket e a família de protocolo utilizados.

Como estamos implementando o socket TCP o tipo de socket é `SOCK_STREAM`.

Lembrando que o TCP é uma boa escolha porque é confiável, ele entrega todos os pacotes; e entrega na ordem correta!

OBS: Se fosse usar um socket UDP deveria passar como parâmetro de tipo de socket o `SOCK_DGRAM`.

IMPLEMENTAÇÃO

Servidor e cliente echo

Aplicação básica para mostrar os conceitos de comunicação entre um cliente e servidor através de um socket.

servidor.py

```
host = "127.0.0.1" # Standard loopback interface address (localhost)
port = 65432 # Port to listen on (non-privileged ports are > 1023)
data_payload = 1024 #The maximum amount of data to be received at once

# Create a TCP socket
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# Bind the socket to the port
server_address = (host, port)
print ("Starting up echo server on %s port %s" % server_address)
sock.bind(server_address)

# Listen to clients, argument specifies the max no. of queued connections
sock.listen(5)
client, address = sock.accept()

while True:
    print ("Waiting to receive message from client")
    data = client.recv(data_payload)
    message = data.decode()
    if message:
        print ("Data: %s" % message)
        client.sendall(message.encode())
    else:
        break
print ('Closing client connection', client)
client.close()
```

IMPLEMENTAÇÃO

Servidor echo

Importante!

O novo socket **client** é diferente do socket **sock** que fica escutando o pedido de novas conexões.

Bloqueia a execução e espera o `connect()` do cliente. Quando cliente conecta, ele cria um novo socket (client) e retorna uma tupla com a conexão client e o endereço do cliente.

Cria o objeto socket.

Os valores passados aqui depende da família de endereço definida no `socket()`. Neste caso, deve ser a tupla host, porta. OBS: se host estiver vazio, o servidor aceita conexão em todas as interfaces IPV4.

5 é o parâmetro que especifica o número de conexões ainda não aceitas pelo servidor que o sistema permite na fila antes de recusar uma nova conexão.

```
host = "127.0.0.1" # Standard loopback interface address (localhost)
port = 65432 # Port to listen on (non-privileged ports are > 1023)
data_payload = 1024 #The maximum amount of data to be received at once

# Create a TCP socket
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# Bind the socket to the port
server_address = (host, port)
print ("Starting up echo server on %s port %s" % server_address)
sock.bind(server_address)

# Listen to clients, argument specifies the max no. of queued connections
sock.listen(5)
client, address = sock.accept()

while True:
    print ("Waiting to receive message from client")
    data = client.recv(data_payload)
    message = data.decode()
    if message:
        print ("Data: %s" % message)
        client.sendall(message.encode())
    else:
        break
print ('Closing client connection', client)
client.close()
```

IMPLEMENTAÇÃO

Servidor echo

Quando o `recv()` retorna um byte vazio significa que o cliente fechou a conexão, logo o loop é finalizado.

```
host = "127.0.0.1" # Standard loopback interface address (localhost)
port = 65432 # Port to listen on (non-privileged ports are > 1023)
data_payload = 1024 #The maximum amount of data to be received at once

# Create a TCP socket
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# Bind the socket to the port
server_address = (host, port)
print ("Starting up echo server on %s port %s" % server_address)
sock.bind(server_address)

# Listen to clients, argument specifies the max no. of queued connections
sock.listen(5)
client, address = sock.accept()

while True:
    print ("Waiting to receive message from client")
    data = client.recv(data_payload)
    message = data.decode()
    if message:
        print ("Data: %s" % message)
        client.sendall(message.encode())
    else:
        break
print ('Closing client connection', client)
client.close()
```

Loop que ler tudo que o cliente envia (`recv()`) e envia de volta (`sendall()`).

IMPLEMENTAÇÃO

Cliente echo

```
import socket
host = '127.0.0.1'
port=65432
# Create a TCP/IP socket
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
# Connect the socket to the server
client_address = (host, port)
print ("Connecting to %s port %s" % client_address)
sock.connect(client_address)
# Send data
message = "Hello, world"
sock.sendall(message.encode())
data = sock.recv(1024)
print ("Received: %s" % data.decode())

print ("Closing connection to the server")
sock.close()
```

← Cria o objeto socket.

← Conecta com o servidor enviando seu endereço

← Chama o `recv()` para receber o que vai ser enviado pelo servidor.

IMPLEMENTAÇÃO

Execução

Agora você vai executar os dois lados da aplicação e observar o que acontece.

- Abra um terminal execute o servidor.py.
- O seu servidor vai ficar esperando o pedido de connect() do cliente.
- Neste momento, o servidor vai aceitar a conexão. O que você observa no lado servidor e cliente?

IMPLEMENTAÇÃO

Verificando o estado

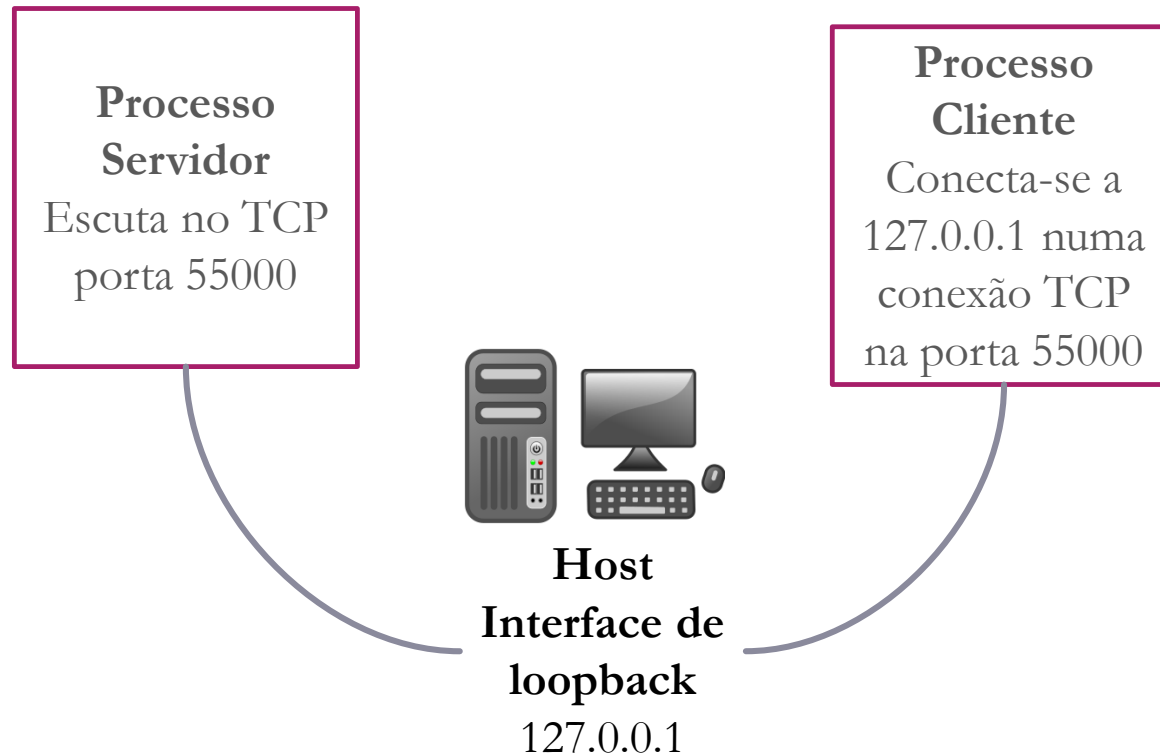
Vamos usar o netstat para verificar o estado do socket.

- Digite: netstat -an
- O que seria o resultado desse comando se o servidor estivesse usando como endereço o vazio ("") ? Faça o teste.
- E se você executar o cliente sem que o servidor esteja executando, o que acontece?

COMUNICAÇÃO

Interface de Loopback

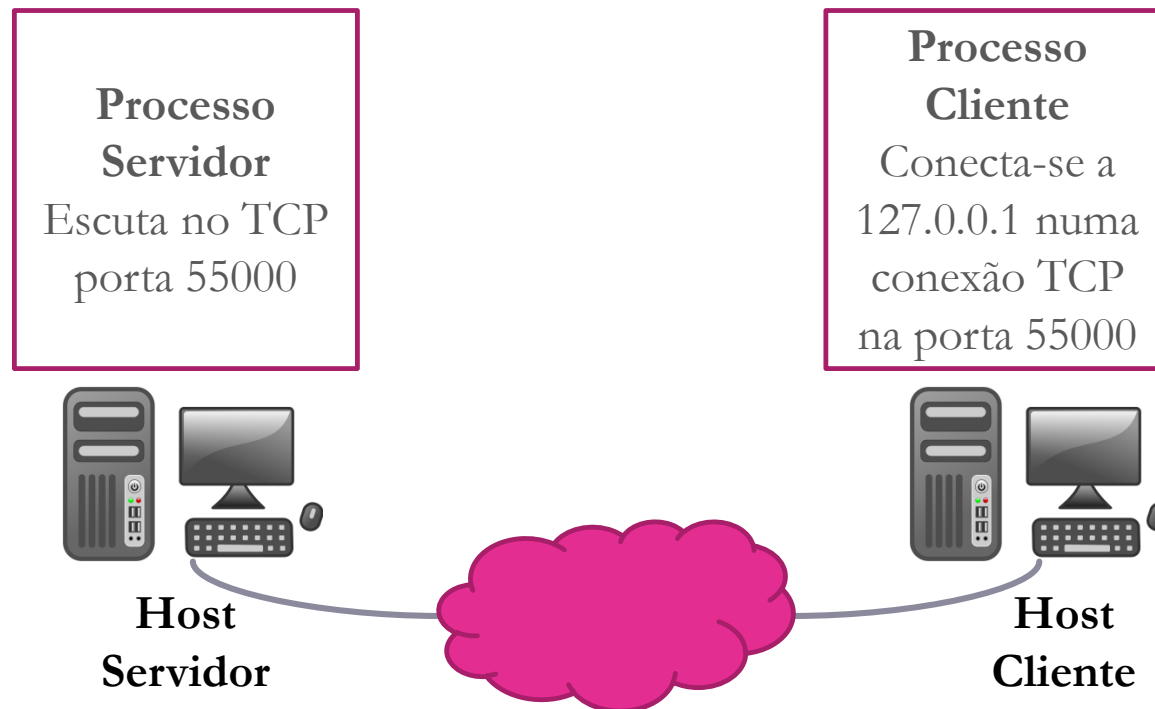
Usando esta interface os dados enviados não saem do host, ou seja, a comunicação estabelecida é entre processos locais. Por isso, a interface loopback é referenciada como **localhost**.



COMUNICAÇÃO

Interface ethernet

Quando o servidor usa um outro endereço ele está ligado a interface Ethernet. Esta interface conecta o localhost com o mundo exterior que pode ser uma rede local ou a Internet.





REFERÊNCIAS

- Redes de computadores e a Internet, Kurose, J.
- <https://medium.com/@urapython.community/introdu%C3%A7%C3%A3o-a-sockets-em-python-44d3d55c60d0>.
- <https://realpython.com/python-sockets/>