INSTITUTO FEDERAL
Norte de Minas Gerais

API Berkeley Sockets

PROGRAMAÇÃO CLIENTE-SERVIDOR COM PYTHON

Rafaela Ramos Farias

SISTEMAS DISTRIBUIDOS

conceito

1. Um Socket é um ponto final de comunicação (um 'endpoint') em uma rede, identificado por uma combinação única de endereço IP e porta.

2. Estrutura: (IP, Porta).

Importância Histórica e Tipos de Sockets

O Padrão Sockets

Importância Histórica:

- Criada na década de 80, forneceu a interface padrão para programar aplicações TCP/IP.
- Permitiu o crescimento da Internet Moderna, abstraindo os detalhes de rede.

O que é um Socket?

- *Um ponto final de comunicação na rede.*
- *Identificado pelo par (Endereço IP, Porta).*

Tipos Principais:

- **SOCK_STREAM (TCP):** Orientado à Conexão, confiável (garante entrega e ordem).
- **SOCK_DGRAM (UDP):** Não orientado à Conexão, rápido, não garante entrega.

Primitivas da API (Funções Essenciais)

- Vocabulário da Comunicação

Primitiva	Função Principal	Quem Usa?
socket()	Cria o objeto socket.	ambos
bind()	Associa o socket a um IP e Porta local.	servidor
listen()	Coloca o socket para aguardar conexões.	servidor
accept()	Bloqueia e aceita a conexão, retorna um novo socket.	servidor
connect()	Inicia a conexão com um servidor remoto.	cliente
send() / recv()	Envia e recebe dados (sempre bytes).	ambos
close()	Encerra a conexão.	ambos

A Biblioteca socket do Python

Implementação em Alto Nível

A Biblioteca socket do Python: Da Teoria à Prática

- **Implementação:** O Python utiliza o módulo padrão socket para expor a API Berkeley Sockets ao programador.
- **A Ponte:** O Python implementa as Primitivas conceituais (como bind e accept) como métodos da classe principal, socket.socket.
- **Vantagem:** Isso oferece uma interface Orientada a Objetos mais limpa e segura do que a API C original.
-

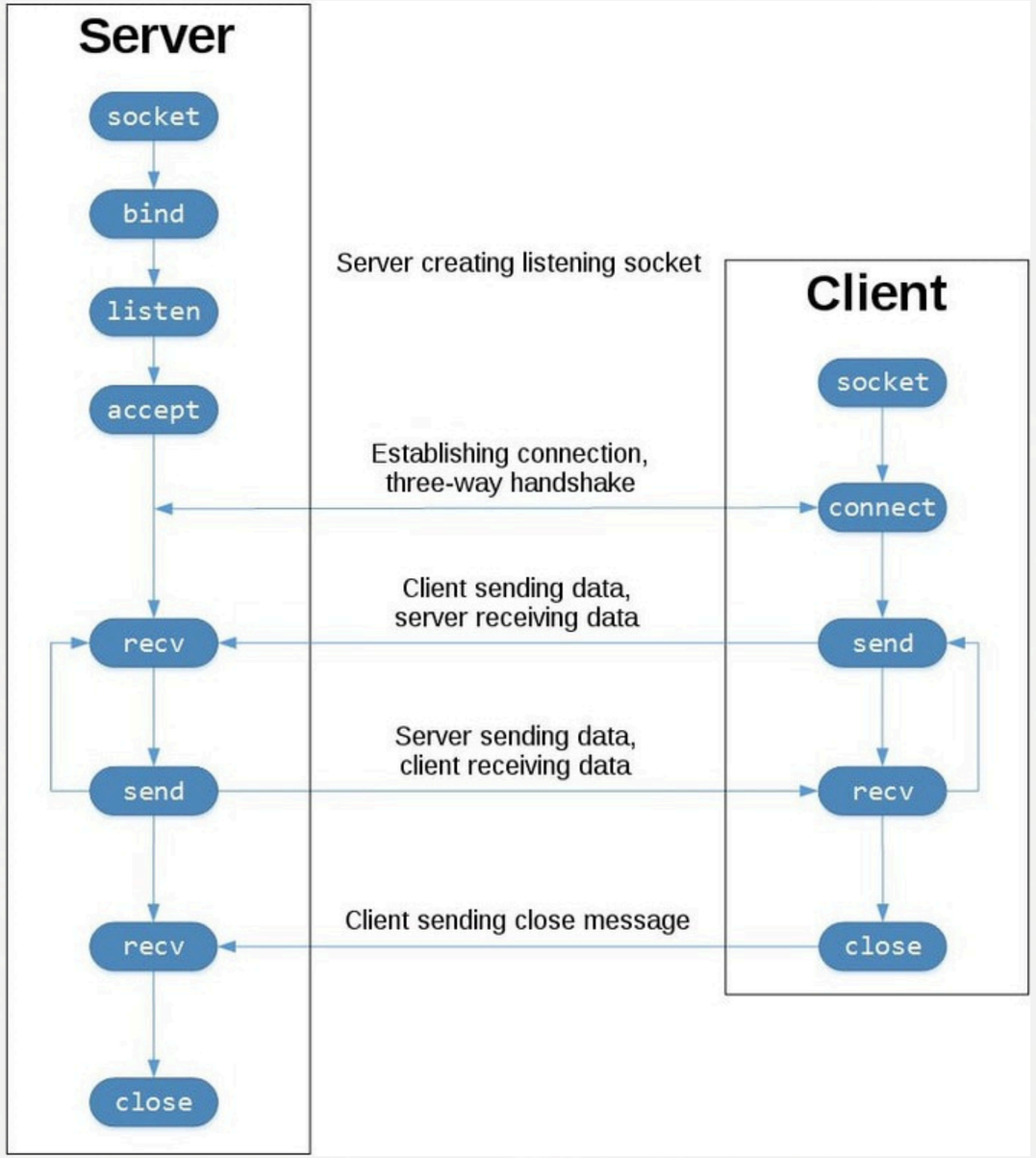
A Biblioteca socket do Python

- Implementação em Alto Nível

Primitiva da API C (Conceito)	Método Correspondente em Python	Exemplo no Código
socket()	Cria o objeto socket.	s = socket.socket(...)
bind()	Método de Associação	s.bind((host, port))
accept()	Método de Aceitação	conn, addr = s.accept()
send() / recv()	Métodos de Transferência	conn.sendall(data)

A Sequência Lógica Cliente-Servidor

Diagrama de Fluxo:



Servidor (server.py): Escuta e Diálogo

- **objetivo:** Abrir o socket na porta 65432, esperar a conexão, identificar o cliente, enviar a saudação, receber a resposta final e encerrar.

- **Operações Chave:**
- **bind() e listen():** O servidor fica pronto em um endereço conhecido.

- **accept():** É a chamada que bloqueia e espera. Retorna o novo socket de comunicação (conn) e o endereço do cliente (addr).

- **Codificação (.encode()):** Os dados textuais devem ser convertidos para bytes antes de serem enviados pela rede.

Servidor (server.py): Escuta e Diálogo

```python
print("Hello, World!")
import socket
HOST, PORT = '127.0.0.1', 65432

# O 'with' garante que o socket principal será fechado
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
    s.bind((HOST, PORT))
    s.listen()

    # accept() retorna o endereço do cliente: IP e Porta
    conn, addr = s.accept()
    with conn:
        print(f"**Cliente conectado!** IP: {addr[0]}, Porta: {addr[1]}")

        # 1. Servidor envia
        mensagem = "bem vindo cliente"
        conn.sendall(mensagem.encode('utf-8'))

        # 2. Servidor recebe a resposta do cliente
        data = conn.recv(1024)
        print(f"  <- Servidor recebeu: {data.decode()}")

    print("**Conexão encerrada pelo Servidor.**")
```

Cliente (client.py): Conexão e Resposta

- **Objetivo:** Conectar-se ao servidor, receber a mensagem de boas-vindas, imprimir, enviar a resposta de agradecimento e fechar a conexão.

- **Operações Chave:**
- **connect():** Inicia o handshake TCP, solicitando a conexão ao endereço do servidor.

- **Decodificação (.decode()):** Os bytes recebidos da rede devem ser convertidos de volta para string para serem impressos ou processados.

- **Tamanho do Buffer:** O argumento em s.recv(1024) define o tamanho máximo de bytes a serem lidos por vez (1024 é um buffer comum).

Cliente (client.py): Conexão e Resposta

```python
import socket
HOST, PORT = '127.0.0.1', 65432

with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
    try:
        # 1. Inicia a conexão com o servidor
        s.connect((HOST, PORT))

        # 2. Cliente recebe a saudação
        data = s.recv(1024)
        print(f"  <- Cliente recebeu: {data.decode('utf-8')}")

        # 3. Cliente envia a resposta de agradecimento
        resposta = "obrigado servidor"
        s.sendall(resposta.encode('utf-8'))

    except ConnectionRefusedError:
        print("Erro: Servidor inativo ou endereço incorreto.")

print("**Conexão encerrada pelo Cliente.**")
```

Demonstração do Ciclo Completo

O que Vimos: A API Sockets estabelece a comunicação por meio de um fluxo sequencial e lógico, seguindo as etapas de Criação, Vinculação/Conexão, Troca de Dados e Encerramento da sessão.

Obrigada

Rafaela Ramos

SISTEMAS DISTRIBUIDOS