Capítulo 4: Teoria - Sockets

Sockets:

- Abstração que permite que um aplicativo envie e receba dados
- Permitem que programas em diferentes computadores se comuniquem, enviando e recebendo dados.
    - Usam tanto TCP como UDP:
        - **Sockets de fluxo (Stream sockets):,**
        - Usam o protocolo TCP
        - Eles oferecem serviço confiável, onde os dados são transmitidos sem perda e na ordem correta.
        - **Sockets de datagrama (Datagram sockets):**
            - Usam o protocolo UDP.
            - Eles oferecem um serviço de datagrama com melhor esforço, onde os dados são transmitidos em pacotes independentes chamados datagramas, sem garantia de entrega ou ordem.

- **Tipos de Sockets:**
  - **Sockets de Cliente**: São usados pelo programa que inicia a comunicação. Eles se conectam a um servidor para enviar e receber dados.
  - **Sockets de Servidor**: São usados pelo programa que está esperando por conexões. Eles "ouvem" por conexões de clientes e respondem a elas.

PROCEDIMENTOS DE SOCKETS

- Funções para lidar com Sockets, para criar, configurar, conectar, enviar e receber dados através de Sockets

Função	Conceito
socket():	criar um novo Socket.Ele especifica o tipo de protocolo usado (TCP ou UDP)
bind():	Associa um endereço IP e um número de porta a um socket
listen():	Se o Socket for um servidor, esta função é usada para colocá-lo no modo de escuta
accept():	aceitar uma conexão de um cliente entrante
connect():	Conectar a um servidor remoto.Ela especifica o endereço IP e o número de porta do servidor ao qual o cliente deseja se conectar.
send() e recv():	Enviar e receber dados através do Socket
close():	fechar um Socket quando ele não é mais necessário

OBS: Para sockets de datagrama (como `SOCK_DGRAM`), se o objetivo é apenas enviar dados, não é necessário chamar `bind()`.
O sistema operacional escolherá automaticamente uma porta disponível toda

vez que o socket enviar um pacote. No entanto, se estivermos recebendo dados, ainda será necessário chamar
`bind()` *para associar o socket a uma porta específica.*

CHAT BLOQUEANTE E NÃO BLOQUEANTE

1.Chat Bloqueante

- A execução do programa é **suspensa** enquanto espera por entrada ou saída.
    - Quando uma operação de entrada ou saída é chamada, o programa não continua sua execução até que essa operação seja concluída.

```
import socket

# Criação do socket TCP
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STR

# Associa o socket a um endereço e porta
server_socket.bind(('localhost', 12345))

# Espera por conexões
server_socket.listen(5)

while True:
    # Aceita conexões de clientes
    client_socket, client_address = server_socket.accept()

    # Recebe mensagem do cliente (bloqueante)
    message = client_socket.recv(1024)

    # Processa a mensagem
    # (neste exemplo, apenas envia a mensagem de volta)
    client_socket.sendall(message)
```

```
    # Fecha o socket do cliente
    client_socket.close()
```

2. Chat Não-Bloqueante

- O programa continua sua execução mesmo durante operações de entrada ou saída.

```
import socket  # Importa o módulo de soquete para comunicação
import threading  # Importa o módulo de threading para execut

class TCPClient:
    def __init__(self, server_name, server_port):
        self.server_name = server_name  # Endereço IP do serv
        self.server_port = server_port  # Porta do servidor
        self.client_socket = socket.socket(socket.AF_INET, so
        self.client_socket.connect((server_name, server_port)
        self.receive_thread = threading.Thread(target=self.re
        self.receive_thread.start()  # Inicia a thread de rec

    def send_message(self, message):
        self.client_socket.send(message.encode('utf-8'))  # E

    def receive_messages(self):
        while True:
            try:
                modified_sentence = self.client_socket.recv(1
                if not modified_sentence:  # Verifica se a me
                    break  # Se estiver vazia, encerra o loop
                print("\n Servidor:", modified_sentence.decod
            except ConnectionAbortedError:
                break  # Se ocorrer um erro de conexão, encer

    def start_chat(self):
        while True:
            sentence = input('Mensagem: ')  # Solicita ao usu
            self.send_message(sentence)  # Envia a mensagem p
```

```
    def close_connection(self):
        self.client_socket.close()  # Fecha o socket do clien

# Inicia cliente com o endereço IP 'localhost' e porta '12000
client = TCPClient('localhost', 12000)

# Inicia a thread para receber mensagens do servidor
client.receive_thread.start()

# Inicia o loop para enviar mensagens para o servidor
client.start_chat()

# Aguarda até que a thread de recebimento de mensagens seja c
client.receive_thread.join()

# Fecha a conexão do cliente
client.close_connection()
```

Single Thread ≠ Multithread

1. SingleThread:

- Processa uma conexão de cada vez, esperando a conclusão total antes de aceitar uma nova.

  Passo:

  - O servidor espera por uma nova conexão (`accept()`).
  - Uma vez que a conexão é aceita, o servidor lida com a solicitação do cliente (`request()`) e envia a resposta (`response()`).
  - Só depois de terminar todos esses passos é que o servidor volta a aceitar uma nova conexão.
    - Se qualquer etapa (por exemplo, o `request()` ou `response()`) demorar, *nenhuma nova conexão será aceita até que a etapa seja concluída.*

```
while True:
    connection = accept()  # Espera por uma conexão
    request = handle_request(connection)  # Processa a soli
    response = send_response(request)  # Envia a resposta
    close_connection(connection)  # Fecha a conexão
```

2. Multithread

- Cria novas threads para cada conexão, permitindo o processamento simultâneo de múltiplas conexões.

Passo:

- O servidor principal aceita novas conexões (`accept()`).
- Para cada nova conexão, o servidor cria uma nova thread para lidar com `request()` e `response()`.
- O *servidor principal continua aceitando novas conexões enquanto as threads lidam com as solicitações.*

```
def handle_client(connection):
    request = handle_request(connection)  # Processa a solici
    response = send_response(request)  # Envia a resposta
    close_connection(connection)  # Fecha a conexão

while True:
    connection = accept()  # Espera por uma conexão
    create_new_thread(handle_client, connection)  # Cria uma
```