Desenvolvimento de um Sistema Client-Servidor com Tkinter e Socket em Python

Pedro Capelari

December 16, 2024

Contents

1 Introdução

Este documento aborda a criação de um sistema de comunicação **client-servidor** em Python utilizando a biblioteca `socket` para comunicação em rede e `Tkinter` para a interface gráfica do usuário (GUI). Um sistema client-servidor é um modelo de arquitetura onde um servidor centralizado gerencia e responde às solicitações de vários clientes, permitindo a troca de informações em tempo real.

Além disso, demonstra-se como integrar essa arquitetura com uma interface gráfica para tornar a experiência do usuário mais intuitiva e interativa. Esta implementação é ideal para aplicações como sistemas de chat, gerenciamento de dados e aplicações colaborativas.

2 Arquitetura Client-Servidor

A arquitetura **client-servidor** organiza os processos em duas partes principais:

- **Servidor**: Um programa central que aguarda conexões de múltiplos clientes, processa suas requisições e envia respostas.
- **Cliente**: Um programa que se conecta ao servidor, envia requisições e recebe respostas.

2.1 Funcionamento Básico

- **Servidor**: Fica em execução contínua, aguardando conexões dos clientes. Ele processa as mensagens recebidas e, em um sistema de chat, retransmite essas mensagens para os outros clientes conectados.
- **Cliente**: Conecta-se ao servidor, envia mensagens e aguarda respostas. Após a comunicação, pode optar por se desconectar.

3 Configurando o Servidor de Chat

O servidor de chat utiliza a biblioteca `socket` para gerenciar conexões e `threading` para lidar com múltiplos clientes simultaneamente. Essa abordagem permite que o servidor continue funcionando enquanto atende a vários clientes ao mesmo tempo.

3.1 Código do Servidor (`server.py`)

```
1 import socket
2 import threading
3
4 # Configura  es do servidor
5 HOST = '127.0.0.1'
6 PORT = 12345
7
8 # Lista para armazenar clientes conectados
9 clients = []
```

```python

# Fun   o para transmitir mensagens para todos os clientes
def broadcast(message, _client):
    for client in clients:
        if client != _client:
            try:
                client.send(message)
            except:
                clients.remove(client)

# Fun   o para lidar com cada cliente
def handle_client(client):
    while True:
        try:
            message = client.recv(1024)
            if message:
                print(f"Mensagem recebida: {message.decode()}")
                broadcast(message, client)
        except:
            clients.remove(client)
            client.close()
            break

# Fun   o principal para aceitar conex es
def main():
    server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server.bind((HOST, PORT))
    server.listen(5)
    print(f"Servidor rodando em {HOST}:{PORT}...")

    while True:
        client, addr = server.accept()
        print(f"Nova conex o: {addr}")
        clients.append(client)
        thread = threading.Thread(target=handle_client, args=(
           client,))
        thread.start()

if __name__ == "__main__":
    main()
```

Listing 1: Servidor em Python

3.2 Explicação do Código

- **Configurações do Servidor**:
  - `HOST = '127.0.0.1'`: Define o endereço IP onde o servidor estará escutando (localhost).
  - `PORT = 12345`: Define a porta de comunicação.

- **Lista de Clientes Conectados**: `clients = []`: Armazena os clientes conectados para facilitar o envio de mensagens a todos.
- **Função** `broadcast`: Envia mensagens para todos os clientes conectados, exceto o remetente.
- **Função** `handle_client`: Recebe mensagens de um cliente específico e retransmite para os outros clientes usando `broadcast`. É executada em uma thread separada para cada cliente.
- **Função** `main`: Configura o servidor, aceita conexões e inicia uma thread para cada cliente.

4 Cliente Integrado ao Tkinter

Vamos integrar uma interface gráfica com Tkinter para o cliente. O cliente permitirá enviar e receber mensagens através de uma interface intuitiva.

4.1 Código do Cliente (`client.py`)

```
import tkinter as tk
from tkinter import scrolledtext
import socket
import threading

# Configura  es do cliente
HOST = '127.0.0.1'
PORT = 12345

class ChatClient(tk.Toplevel):
    def __init__(self, master):
        super().__init__(master)
        self.title("Chat Client")
        self.geometry("400x400")

        #  rea  de mensagens
        self.chat_area = scrolledtext.ScrolledText(self, state='
            disabled', wrap='word')
        self.chat_area.pack(padx=10, pady=10, fill='both', expand
            =True)

        # Entrada para mensagem
        self.msg_entry = tk.Entry(self)
        self.msg_entry.pack(padx=10, pady=10, fill='x')
        self.msg_entry.bind("<Return>", self.send_message)

        # Bot o de enviar
        self.send_button = tk.Button(self, text="Enviar", command
            =self.send_message)
        self.send_button.pack(pady=5)
```

```
28
29          # Conectar ao servidor
30          self.client = socket.socket(socket.AF_INET, socket.
               SOCK_STREAM)
31          self.client.connect((HOST, PORT))
32
33          # Thread para receber mensagens
34          threading.Thread(target=self.receive_messages, daemon=
               True).start()
35
36      def send_message(self, event=None):
37          message = self.msg_entry.get()
38          if message:
39              self.client.send(message.encode())
40              self.msg_entry.delete(0, tk.END)
41
42      def receive_messages(self):
43          while True:
44              try:
45                  message = self.client.recv(1024).decode()
46                  if message:
47                      self.chat_area.configure(state='normal')
48                      self.chat_area.insert(tk.END, message + '\n')
49                      self.chat_area.configure(state='disabled')
50                      self.chat_area.yview(tk.END)
51              except:
52                  break
53
54  # Interface principal
55  class App(tk.Tk):
56      def __init__(self):
57          super().__init__()
58          self.title("Gest o Industrial com Chat")
59          self.geometry("300x200")
60
61          tk.Button(self, text="Abrir Chat", command=self.open_chat
               ).pack(pady=20)
62
63      def open_chat(self):
64          ChatClient(self)
65
66  if __name__ == "__main__":
67      app = App()
68      app.mainloop()
```

Listing 2: Cliente em Python com Tkinter

4.2 Explicação do Código

- **Configurações do Cliente**:

  - `HOST = '127.0.0.1'`: Endereço do servidor.
  - `PORT = 12345`: Porta de conexão.

- **Classe** `ChatClient`:
  - Interface gráfica com área de mensagens (`ScrolledText`), campo de entrada (`Entry`) e botão de envio.
  - Conecta ao servidor ao ser inicializada.
  - Envia mensagens com a função `send_message`.
  - Recebe mensagens em uma thread separada com a função `receive_messages`.
- **Classe** `App`: Interface principal com um botão para abrir a janela de chat.

5 Como Executar

5.1 Iniciar o Servidor

Execute o servidor em um terminal:

```
python server.py
```

5.2 Iniciar o Cliente

Abra múltiplos terminais e execute o cliente em cada um:

```
python client.py
```

5.3 Teste a Comunicação

Envie mensagens entre os clientes conectados para verificar o funcionamento.

6 Conclusão

Este documento apresentou um sistema client-servidor com uma interface gráfica em Python. Esse modelo pode ser expandido para aplicações mais complexas, como sistemas de gerenciamento industrial, chats multiusuário e muito mais.