Implementação de um chat multiusuário em Python com sockets[1]

Implement a multi-user chat in Python with sockets

Jeferson Cevada Girato
Matheus Marques Ferreira
Brendon Henrique Monteiro Esteves

2023

RESUMO

Este artigo apresenta a implementação de um servidor de chat em Python utilizando sockets e threads. O código desenvolvido permite a comunicação entre clientes conectados à rede local, possibilitando o envio de mensagens para todos os participantes do chat. O servidor é capaz de aceitar múltiplas conexões simultaneamente e gerenciar as mensagens recebidas, distribuindo-as para todos os clientes conectados.

Palavras-Chave: Python, sockets, threads, servidor de chat, comunicação em rede

ABSTRACT

This article presents the implementation of a chat server in Python using sockets and threads. The developed code allows communication between clients connected to the local network, enabling messages to be sent to all chat participants. The server is capable of accepting multiple connections simultaneously and managing received messages, distributing them to all connected clients.

Keywords: Python, sockets, threads, chat server, network communication

1 SINCRONIZAÇÃO DE PROCESSOS

Thread é um conceito que permite a execução de múltiplos fluxos de um programa em paralelo, aproveitando os recursos de processamento de máquinas com mais de uma CPU. Em Python, o módulo threading oferece uma interface de alto nível para criar e gerenciar threads, além de sincronizar o acesso a dados compartilhados entre elas. Threads podem ser úteis para realizar tarefas que envolvem entrada e saída de dados, como comunicação em rede ou leitura de arquivos, sem bloquear a execução do programa principal. No entanto, devido ao Global Interpreter Lock (GIL), que impede que mais de uma thread execute código Python ao mesmo tempo, threads não são adequadas para tarefas que demandam alto uso da CPU, pois elas não aumentam o desempenho do programa. Nesses casos, recomenda-se o uso de outros módulos, como multiprocessing ou concurrent.futures, que permitem a criação de processos independentes que podem executar código Python simultaneamente.

[1] Sistemas Distribuidos

"Threads são linhas de execução concorrentes em um mesmo processo (no seu caso, o seu programa em Python). Elas são concorrentes no sentido de que executam simultaneamente, mas cada um com a sua própria linha de execução - quase como se fossem programas diferentes." (VIEIRA, 2016, p. 1)

Alguns dos recursos comuns fornecidos por uso de threads incluem:

- **Queue**: é um recurso que permite armazenar dados em uma estrutura de dados do tipo FIFO (first in, first out), ou seja, o primeiro elemento a entrar na fila é o primeiro a sair. Um aplicativo de chat em LAN pode usar queues para gerenciar as mensagens recebidas e enviadas pelas threads, evitando conflitos de acesso aos dados.
- **Lock**: é um recurso que permite bloquear o acesso a um recurso compartilhado entre as threads, garantindo que apenas uma thread possa usar o recurso por vez. Um aplicativo de chat em LAN pode usar locks para sincronizar as operações de leitura e escrita nos sockets, evitando inconsistências nos dados.
- **Event**: é um recurso que permite sinalizar eventos entre as threads, usando um objeto que pode estar em dois estados: set ou clear. Um aplicativo de chat em LAN pode usar events para notificar as threads sobre a ocorrência de alguma condição, como a conexão ou desconexão de um usuário, ou a recepção de uma mensagem.

A biblioteca threads é muito útil para desenvolvedores que estão trabalhando com programas multithread complexos. Isso ajuda os desenvolvedores a resolver problemas de sincronização e a garantir que os programas multithread sejam executados de forma segura e eficiente.

1.1 Sockets

Sockets são um tipo de ponteiro que permite a comunicação entre processos em sistemas distribuídos. Eles são "um mecanismo de comunicação entre processos que permite a comunicação bidirecional entre dois processos em uma rede. Eles são implementados como objetos de sistema em sistemas operacionais modernos e fornecem uma interface para os processos se comunicarem entre si usando a rede." (Kurose e Ross, 2023)

Existem dois tipos principais de sockets: sockets de fluxo e sockets de datagrama.

Sockets de fluxo são usados para comunicação bidirecional em tempo real. Eles fornecem um fluxo contínuo de dados entre dois processos.

Sockets de datagrama são usados para comunicação assíncrona. Eles permitem que os processos enviem dados um para o outro sem esperar uma resposta.

2 IMPLEMENTAÇÃO DE SOCKETS E THREADS NO SCRIPT DE CHAT

O script de chat apresentado neste artigo usa sockets para estabelecer uma conexão entre o cliente e o servidor. O cliente usa um socket para se conectar ao servidor e enviar mensagens. O servidor usa um socket para aceitar conexões de clientes e distribuir mensagens para todos os clientes conectados.

2.1 Implementação de sockets

A implementação de sockets no script é semelhante à implementação apresentada anteriormente. O cliente e o servidor usam a função socket() para criar um socket. A função socket() aceita dois parâmetros: o tipo de socket e o protocolo.

O tipo de socket determina o tipo de comunicação que será realizada. No caso do script de chat, os sockets são de tipo SOCK_STREAM, que corresponde a uma conexão de fluxo contínuo.

O protocolo determina o protocolo de rede que será usado. No caso do script de chat, o protocolo é IPPROTO_TCP, que corresponde ao protocolo TCP.

Após criar o socket, o cliente e o servidor precisam especificar o endereço e a porta de conexão. O endereço é o endereço IP do computador que está executando o cliente ou o servidor. A porta é o número da porta que está sendo usada para a comunicação.

A especificação do endereço e da porta é feita usando a função bind(). A função bind() aceita três parâmetros: o socket, o endereço IP e a porta.

Depois de especificar o endereço e a porta, o cliente e o servidor podem começar a se comunicar. O cliente usa a função connect() para se conectar ao servidor. A função connect() aceita dois parâmetros: o socket e o endereço IP e a porta do servidor.

O servidor usa a função accept() para aceitar conexões de clientes. A função accept() aceita um parâmetro: o socket.

Implementação de threads

A implementação de threads no script é semelhante à implementação apresentada anteriormente. O servidor usa a função threading.Thread() para criar uma nova thread para cada conexão de cliente. O novo processo é uma thread que é responsável por lidar com a comunicação com o cliente.

A função threading.Thread() aceita dois parâmetros: a função que será executada pela thread e os argumentos que serão passados para a função.

Após criar o novo processo, o servidor retorna à espera de novas conexões. O novo processo continua a lidar com a comunicação com o cliente.

- Exemplo de implementação

O seguinte código abaixo Código 1 mostra um exemplo de implementação de sockets e threads no script de chat:

Código 1 – Exemplo

```
1 import socket
2 import threading
3
4 clientes = []
5
6 def main():
7
8     socketServer = socket.socket(socket.AF_INET, socket.SOCK_STREAM
          )
9
10    try:
11        socketServer.bind(('' , 50000)) #subindo servidor
12        socketServer.listen() #deerminando a quantidade maxima de
              usuarios
13        print("server ativo")
14    except:
15        socketServer.close() #entra no exept de ja houver um server
              na rede
16        return print("Server ja iniciado")
17
18    while True:
19        cliente, endereco = socketServer.accept() #aceitando conexõ
              es
20        clientes.append(cliente) #adicionando clientes a lista de
              clientes do servidor
21        #criando e iniciando a threads que da função menssagem
22        thread = threading.Thread(target=menssagem , args=[cliente
              ])
23        thread.start()
24
25 #função que faz o brodcast recebendo a menssagem de um cliente e
       enviando para dtodos conectados
26 def menssagem(cliente):
27    while True:
28        try: #tentando mandar menssagem
29            msg = cliente.recv(2048)
30            mandarMenssagem(cliente, msg)
31        except: #se der erro remove o cliente que deu o erro por
              estar desconectado
32            clientes.remove(cliente)
33            break
34
35 def mandarMenssagem(cliente, msg):
36    for clienteItem in clientes:
37        if clienteItem != cliente:
38            try:
39                clienteItem.send(msg)
```

```
40              except:
41                  clientes.remove(clienteItem)
42
43
44 def verificarServidor(HOST):
45     for host in HOST:
46         client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
47         client.settimeout(0.1)
48         try:
49             client.connect((host, 50000))
50             print("ja existe um serve na rede")
51             return False
52         except
```

3 CONCLUSÃO

O script de chat apresentado neste artigo é um exemplo de como utilizar sockets e threads para implementar uma comunicação entre cliente e servidor. O script funciona de forma simples e eficaz, permitindo que os usuários enviem mensagens entre si.

Um ponto de melhoria que pode ser implementado no script é a automatização de algumas tarefas. Por exemplo, o script poderia ser modificado para que o servidor inicie automaticamente quando o computador for ligado. Além disso, o script poderia ser modificado para que o usuário possa se conectar ao servidor sem precisar inserir o endereço IP e a porta manualmente.

Outro ponto de melhoria que pode ser implementado no script é a criação de uma interface gráfica. Uma interface gráfica tornaria o uso do script mais fácil e intuitivo para os usuários. A interface gráfica poderia incluir recursos como uma lista de usuários conectados, um campo para inserir mensagens e um botão para enviar mensagens.

Com essas melhorias, o script de chat seria mais fácil de usar e mais funcional. O script poderia ser usado em uma variedade de aplicações, como salas de bate-papo, comunidades online e jogos multiplayer.

Aqui estão alguns passos específicos que podem ser tomados para implementar essas melhorias:

- Para automatizar o início do servidor, o script pode ser modificado para que ele seja executado como um serviço do sistema operacional.
- Para permitir que o usuário se conecte ao servidor sem inserir o endereço IP e a porta manualmente, o script pode ser modificado para que ele use um nome de host para o servidor.
- Para criar uma interface gráfica, o script pode ser modificado para usar uma biblioteca de desenvolvimento de interface gráfica.

REFERÊNCIAS

SOCKET. Socket Python. Disponível em: https://docs.python.org/3/library/socket.html. Acesso em: 25 nov. 2023https://docs.python.org/3/library/threading.html

THREADING. Theading Python. Disponível em: https://docs.python.org/3/library/threading.html. Acesso em: 25 nov. 2023.