Prática 03 - Cliente-Servidor e P2P

Introdução

Já praticamos alguns conceitos simples de sockets com python. Além disso, tivemos a oportunidade de testar um Servidor Web com Flask. Recomendo fortemente que você volte aos exemplos anteriores e relembre os conceitos antes de continuar.

Vamos aprofundar mais um pouco esses conhecimentos. Vamos criar uma aplicação P2P muito simples, no começo, e vamos prepará-la para adicionar funções interessantes. Aplicações possuem muitas características e desafios envolvidos, conforme vimos no material. Na prática, para que essas características sejam alcançadas e os desafios sejam resolvidos, precisamos utilizar certos mecanismos e técnicas.

Só para você pensar: até agora precisamos informar o IP e a Porta de forma explícita para que o cliente encontre um servidor. Vimos isso tanto na Prática de Sockets como na Prática de Flask.

Na prática, no seu dia a dia, você utiliza vários aplicativos que se comunicam através da internet e você não tem que ficar digitando o IP de cada um deles. Em seus códigos de programação, cada aplicativo tem seus mecanismos para descoberta. Existem mecanismos que dependem de um servidor, ou seja, todas as aplicações clientes sabem o endereço IP de um servidor e enviam mensagens através dele para funcionar. Geralmente é assim que a internet funciona. Outros mecanismos usam endereçamento direto e descobertas diretas, especialmente em redes locais como empresas, e data centers, além da nuvem.

Neste tutorial, veremos uma forma básica como converter, aos poucos, uma aplicação cliente-servidor uma aplicação P2P. Ao final deste estudo guiado, teremos uma aplicação com a principal característica do modelo P2P: a capacidade de poder atuar como cliente e como servidor dependendo da situação.

Também estudaremos uma das formas mais recomendadas de transmitir dados através da Internet: json.

Inicialmente, faremos isso de forma muito manual, depois criaremos um funcionamento cada vez mais automatizado. Para isso, executaremos uma série de passos nos quais teremos a oportunidade de praticar vários conceitos técnicos de programação.

⚠️ Atenção!

⚠️ Estude devagar. Foque em fazer o código de cada seção funcionar no seu computador de forma independente. Você terá praticamente duas semanas para estudar tudo e enviar o exercício. Enquanto isso, usem muito o chat e o fórum na página da disciplina.

Objetivos da aprendizagem principais:

- Implementação de modelo cliente servidor
- Implementação de modelo P2P
- Implementar técnicas de comunicação se suporte aos modelos cliente-servidor e P2P (unicast, multicast)
- Implementar trocas de mensagens entre clientes, servidores, e entre pares (JSON)

Objetivos de aprendizagem transversais:

- Utilizar técnicas de pesquisa
- Praticar síntese de ideias e técnicas através da integração de partes de sistemas diferentes

Bons estudos!

Seção 0 - Pesquisa

Antes de começar, vale muito a pena ler esse artigo sobre o processo de pesquisa utilizado diariamente por profissionais de TI, especialmente aqueles voltados para programação e análise de sistemas.

https://www.silvestar.codes/articles/developer-s-research-process/?utm_source=tldrwebdev

Seção 1 - Um chat simples em Python usando TCP

Para esta seção, vocês vão precisar de alguns conteúdos. Você pode estudar esse material antes ou depois de ver os vídeos. Fica a seu critério.

Boa parte do material está em inglês. Lembre-se de utilizar a função de tradução do seu navegador. Mas sempre tente traduzir você mesmo para começar a se acostumar com o Inglês. Esse idioma é fundamental para o dia-a-dia de um programador. ⚠️

Funções em python
https://www.w3schools.com/python/python_functions.asp

Funções pré-definidas em python
https://www.w3schools.com/python/python_ref_functions.asp

Python String format() Tutorial
https://www.datacamp.com/tutorial/python-string-format

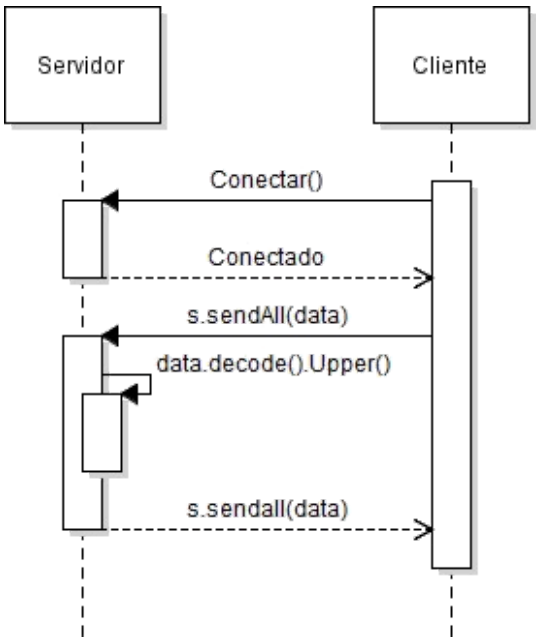
Python f'string
https://realpython.com/python-f-strings/#f-strings-a-new-and-improved-way-to-format-strings-in-python

If __name__ == '__main__'
https://www.alura.com.br/artigos/o-que-significa-if-name-main-no-python

Argumentos de linha de comando em python I
http://www.estruturas.ufpr.br/material/computacaoCientifica/entradaerros/argumentoslinhacomando/

Argumentos de linha de comando em python II
https://acervolima.com/argumentos-de-linha-de-comando-em-python/

O código de sockets que vocês fizeram até agora deve funcionar mais ou menos da seguinte forma.

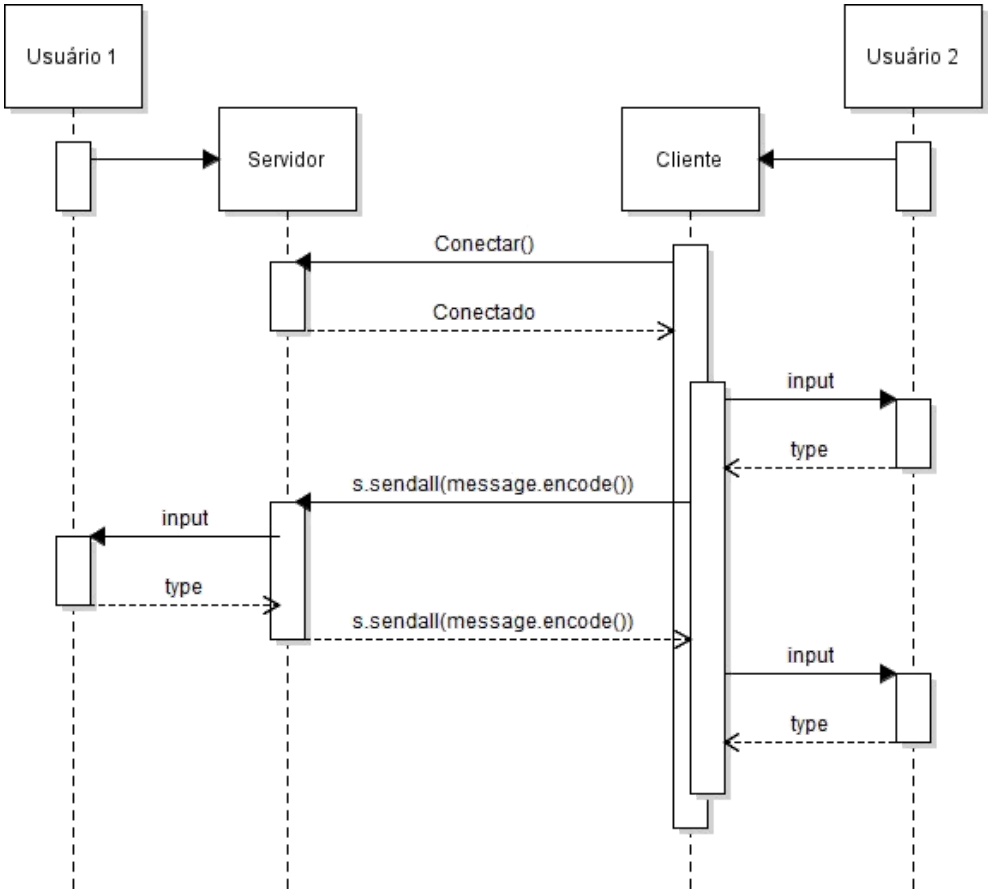


Pesquisem sobre "diagramas de sequência" de UML e validem esse funcionamento. Em breve, talvez até mesmo nesta disciplina, vocês terão que criar alguns diagramas assim. 🔎

Na parte 1, vamos começar com o simples. Existem milhares de tutoriais de como fazer um chat em python. Gostaria que você implementasse o chat de acordo com o seguinte vídeo. Lembre-se de acompanhar o vídeo com VSCode aberto (ou com seu Editor de Código favorito).

[Redes e Sistemas Distribuídos | Aula Prática | Chat P2P - Parte 1]

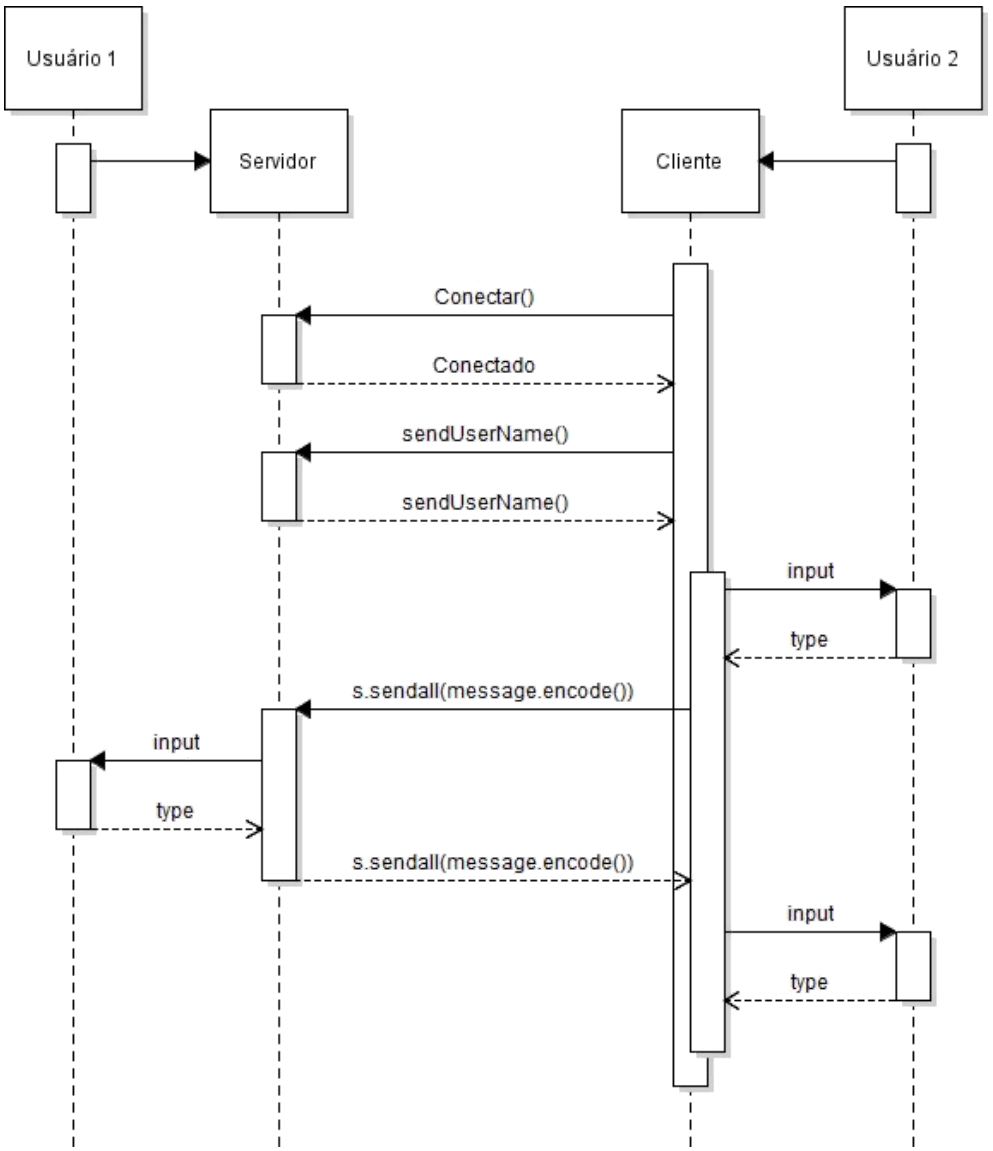
Ao final da Parte 1, seu programa deve se comportar mais ou menos da seguinte forma:



Na parte 2, vamos começar a modularizar o código do cliente e do servidor através de funções. Depois, vamos ver como passar valores para o programa através de linha de comando. Por fim, vamos criar um código um pouco mais complexo para trocar nomes entre o cliente e o servidor antes do chat começar. Lembre-se de acompanhar o vídeo com VSCode aberto (ou com seu Editor de Código favorito).

[Redes e Sistemas Distribuídos | Aula Prática | Chat P2P - Parte 2]

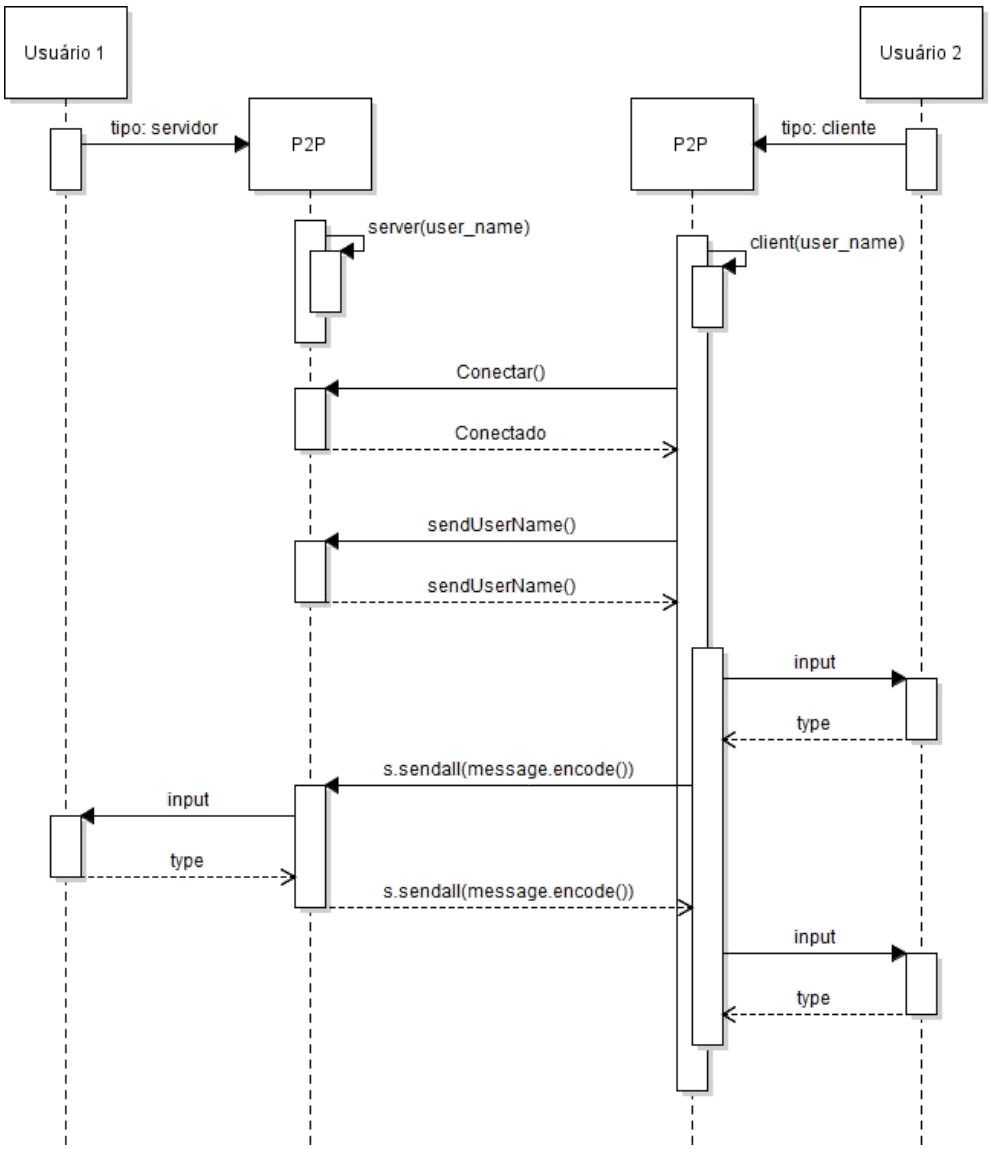
Ao final da Parte 2, seu programa deve se comportar mais ou menos da seguinte forma:



Na parte 3, vamos realmente juntar os códigos do cliente e do servidor no mesmo arquivo. Além disso, adicionar um mecanismo de decisão para informarmos se queremos que o programa atue como cliente ou como servidor. Lembre-se de acompanhar o vídeo com VSCode aberto (ou com seu Editor de Código favorito).

Redes e Sistemas Distribuídos | Aula Prática | Chat P2P - Parte 3

Ao final da Parte 3, seu programa deve se comportar mais ou menos da seguinte forma:



Seção 2 - Multicast em Python

Se quiser ler um pouco mais

Multicast
https://pymotw.com/2/socket/multicast.html

IP multicast

https://lasarojc.github.io/ds_notes/comm/sockets/#ip-multicast

Aqui um exemplo do que poderemos fazer em breve
https://github.com/dpasqualin/pymulticast

Antes de implementar técnicas relacionadas com as características de Sistemas Distribuídos reais, vamos estudar um pouco um dos protocolos que utilizaremos para isso.

Um socket multicast é um mecanismo de comunicação em rede que permite a um emissor enviar dados para um grupo de receptores simultaneamente. Isso é especialmente útil em cenários onde você deseja transmitir dados para múltiplos destinos, como streaming de mídia, jogos online ou atualizações em tempo real.

Características do Socket Multicast:

1. Endereços de Grupo: Em um socket multicast, os receptores são agrupados usando um endereço IP multicast especial. Os endereços multicast estão no intervalo de 224.0.0.0 a 239.255.255.255. Em um grupo de receptores, todos escolhem o mesmo endereço e passam a se comunicar através dele. Na prática, todos podem receber mensagens no mesmo IP de grupo e podem enviar mensagens para o mesmo IP de grupo. Esse modelo é chamado “publisher subscriber”.

   Pesquise sobre esse modelo de comunicação em sistemas distribuídos. Comece por aqui, mas pesquise sempre mais. 🔎

2. Compartilhamento Eficiente: O socket multicast permite o compartilhamento eficiente de dados, uma vez que os pacotes são enviados apenas uma vez e alcançam todos os membros do grupo.

3. Redução de Tráfego: Diferente do broadcast, onde os pacotes são enviados para todos os hosts na rede, o multicast reduz o tráfego, pois os pacotes são entregues apenas aos membros do grupo.

4. Join e Leave: Os hosts podem ingressar (join) ou sair (leave) de grupos multicast dinamicamente. Isso permite uma configuração flexível e adaptável.

Diferenças entre Unicast, Broadcast e Multicast:

- Unicast: É a forma mais comum de comunicação, onde os dados são enviados de um emissor para um receptor específico. A comunicação unicast é ponto a ponto.

- Broadcast: Nesse caso, os dados são enviados para todos os hosts na rede. Isso pode causar tráfego desnecessário, já que todos os hosts recebem os dados, mesmo que não sejam relevantes.

- Multicast: Aqui, os dados são enviados para um grupo específico de receptores que se juntaram ao grupo. A comunicação multicast é mais eficiente em termos de tráfego e recursos.

Exemplo em Python:

Vamos criar um exemplo simples em Python usando a biblioteca socket para ilustrar o funcionamento do socket multicast. Neste exemplo, vamos criar um emissor (sender) e vários receptores (receivers) que compartilham mensagens em um grupo multicast.

Emissor (Sender):

```
import socket

import struct

# Configurações

MULTICAST_GROUP = '224.3.29.71'

MULTICAST_PORT = 5007

# Criar socket

sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM,
socket.IPPROTO_UDP)

ttl = struct.pack('b', 1)

sock.setsockopt(socket.IPPROTO_IP, socket.IP_MULTICAST_TTL, ttl)

# Enviar mensagem

sock.sendto(f"Olá, grupo multicast!".encode('utf-8'),
(MULTICAST_GROUP , MULTICAST_PORT))
```

Inicialmente, importamos dois módulos, socket e struct, que serão usados para configurar a comunicação de socket multicast e manipular estruturas de dados binárias para a configuração do socket.

A seguir, definimos duas constantes: MULTICAST_GROUP e MULTICAST_PORT. MULTICAST_GROUP é o endereço IP multicast para o qual você deseja enviar a mensagem, e MULTICAST_PORT é a porta associada a essa comunicação.

Logo depois, criamos um socket UDP usando a família de endereços AF_INET (IPv4) e o tipo de socket SOCK_DGRAM (datagrama, para comunicação sem conexão). Você também está especificando que o protocolo usado é IPPROTO_UDP para comunicação usando o protocolo UDP. Você já ouviu falar de Datagramas antes. Lembra onde? 🤔

A variável ttl serve para configurar o TTL (Time to Live) para os pacotes multicast que serão enviados. TTL é um número que limita a vida útil de um pacote na rede. Você está usando a função struct.pack() para empacotar o valor 1 como um único byte ('b') para definir o TTL como 1. Isso significa que os pacotes multicast serão entregues apenas à rede local. O uso de TTL controla até que ponto a mensagem será entregue na rede e limita sua propagação.

Para entender mais, pesquisa sobre TTL e roteamento 🔍

Por fim, enviamos uma mensagem para o grupo multicast especificado. Utilizamos o método sendto() no socket para enviar a mensagem codificada como bytes. A função encode('utf-8') é usada para transformar a string "Olá, grupo multicast!" em uma sequência de bytes codificados em UTF-8. A mensagem é enviada para o endereço IP e a porta multicast definidos.

O que é codificação de caracteres em programação? Porque é necessário fazer isso? 🔍

Receptor (Receiver):

```
import socket

import struct

# Configurações

MULTICAST_GROUP = '224.3.29.71'

MULTICAST_PORT = 5007

# Criar socket

sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM,
socket.IPPROTO_UDP)

sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)

sock.bind(('', MULTICAST_PORT))

# Grupo multicast

group = socket.inet_aton(MULTICAST_GROUP)

mreq = struct.pack('4sL', group, socket.INADDR_ANY)

sock.setsockopt(socket.IPPROTO_IP, socket.IP_ADD_MEMBERSHIP, mreq)

# Receber e exibir mensagens

while True:

    print("Receptor aguardando mensagens...")

    data, addr = sock.recvfrom(1024)

  print(f'Recebido de {addr}: {data.decode('utf-8')}")
```

Esse código representa a parte receptora de uma aplicação multicast. Ele configura o socket para receber mensagens de um grupo multicast específico e exibir as mensagens recebidas juntamente com os endereços dos remetentes.

Inicialmente, importamos dois módulos, socket e struct, que serão usados para configurar a comunicação de socket multicast e manipular estruturas de dados binárias para a configuração do socket.

Logo depois, definimos duas constantes: MULTICAST_GROUP e MULTICAST_PORT. MULTICAST_GROUP é o endereço IP do grupo multicast que utilizamos para receber mensagens. MULTICAST_PORT é a porta associada a essa comunicação.

A seguir, criamos um socket UDP para comunicação usando o protocolo IP. Configuramos o socket para reutilizar endereços através do comando setsockopt() e o vinculamos ao

endereço IP e porta definidos. O endereço IP vazio ('') indica que estamos vinculando o socket a todas as interfaces de rede disponíveis no host.

Depois disso, convertermos o endereço IP multicast em formato binário usando socket.inet_aton(). Isso é necessário para configurar o endereço no comando seguinte. Empacotamos o endereço IP multicast e o endereço IP local (INADDR_ANY) em uma estrutura binária usando struct.pack(). Isso é necessário para configurar o socket para se juntar ao grupo multicast usando setsockopt() com IP_ADD_MEMBERSHIP.

Por fim, criamos um loop infinito para receber e exibir mensagens multicast. O método recvfrom() recebe até 1024 bytes de dados e o endereço do remetente. Utilizamos o print() para exibir uma mensagem indicando que o receptor está aguardando mensagens e, em seguida, exibimos as mensagens recebidas juntamente com os endereços dos remetentes.

O funcionamento do multicast permite que vários hosts recebam a mesma mensagem. Observe, nesta imagem, que eu uso três terminais. Inicialmente, eu executei dois receptores (receiver) usando o mesmo arquivo de código. Depois, executo o comando do emissor (sender). Utilizamos o mesmo endereço ip e a mesma porta, eu consegui enviar dados para duas outras aplicações, que poderiam ser, por exemplo, dois outros computadores da mesma rede.

Multicast funciona bem em redes locais, mas não funciona bem na Internet. Pesquise o motivo. 🔍

Seção 3 - JSON

Se quiser ler um pouco mais

Python Dictionaries
https://www.w3schools.com/python/python_dictionaries.asp

Python JSON
https://www.w3schools.com/python/python_json.asp

JSON
https://www.json.org/json-pt.html

Por fim, vamos estudar um pouco sobre o formato mais comum de troca de dados através de redes de computadores e da internet.

JSON (JavaScript Object Notation) é um formato de troca de dados leve e legível. Ele se torna compreensível tanto por computadores quanto por humanos. Ele é usado para

representar informações estruturadas de maneira semelhante a um objeto ou dicionário em Python. O JSON é amplamente utilizado para transmitir dados entre um servidor e um cliente, armazenar configurações, trocar informações em APIs, entre outros.

Características:

Legibilidade: O JSON é fácil de ler e entender, tanto por humanos quanto por máquinas. Isso facilita a depuração e a compreensão dos dados.

Estrutura Hierárquica: O JSON permite representar dados de forma hierárquica, com objetos e arrays aninhados, o que reflete muitos tipos de informações.

Compatibilidade: Como o nome sugere, o JSON foi originalmente baseado na sintaxe de objetos JavaScript, mas hoje é amplamente aceito em várias linguagens de programação.

Troca de Dados: É amplamente usado em APIs, comunicação cliente-servidor e outras formas de compartilhamento de informações.

Enviando Mensagem JSON com Sockets:

Vamos supor que você queira enviar um objeto JSON contendo informações de um usuário (nome, idade, e-mail) de um cliente para um servidor usando sockets em Python.

Este seria seu cliente:

```
import socket
import json

# Dados do usuário em formato de dicionário
usuario = {
    "nome": "Alice",
    "idade": 25,
    "email": "alice@example.com"
}

# Converter dicionário para JSON

usuario_json = json.dumps(usuario)

# Configurações do servidor

SERVER_IP = '127.0.0.1'

SERVER_PORT = 12345

# Criar socket

cliente_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# Conectar ao servidor

cliente_socket.connect((SERVER_IP, SERVER_PORT))

# Enviar JSON

cliente_socket.send(usuario_json.encode('utf-8'))

# Fechar o socket do cliente

cliente_socket.close()
```

Para começar, importamos dois módulos: socket e json. O módulo socket será usado para comunicação, enquanto o módulo json é usado para manipulação de dados.

Logo em seguida, criamos um dicionário chamado usuario que contém informações sobre um usuário, como nome, idade e e-mail. Esse dicionário será convertido em formato JSON através do comando json.dumps(). Isso transforma os dados estruturados em uma string no formato JSON.

Logo em seguida, definimos as configurações do servidor: o endereço IP em que o servidor está executando ('127.0.0.1'), e a porta que o servidor está ouvindo (12345). Além disso, criamos um objeto de socket usando a família de endereços AF_INET (IPv4) e o tipo de socket SOCK_STREAM (para comunicação orientada a conexão, como TCP). Em seguida utilizamos o método connect() do socket para estabelecer uma conexão com o servidor.

Observe que passamos o endereço IP e a porta do servidor como uma tupla para este método.

Depois disso, enviamos dados JSON para o servidor usando o método send() do socket. Antes de enviar, codificamos string JSON em bytes usando encode(), da mesma forma que fizemos nos vídeos.

Por fim, fechamos a conexão.

Agora, vamos ver o Servidor:

```
import socket

import json

# Configurações do servidor

SERVER_IP = ''

SERVER_PORT = 12345

# Criar socket

servidor_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# Vincular o socket ao endereço e porta

servidor_socket.bind((SERVER_IP, SERVER_PORT))


# Outra forma de exibir dados recebidos (forma 2)

print("Dados recebidos:")

print(dados['nome'])

print(dados['idade'])

print(dados['email'])


# Mais uma forma de exibir dados recebidos (forma 3)

print(f"Dados recebidos. Nome: {dados['nome']}. Idade:
{dados['idade']}. Email: {dados['email']}  " )

# Fechar conexão e socket do servidor

conexao.close()

servidor_socket.close()
```

```
# Aguardar conexões

servidor_socket.listen()

print("Aguardando conexões...")

# Aceitar conexão

conexao, endereco = servidor_socket.accept()

print(f"Conectado a {endereco}")

# Receber dados JSON

dados_json = conexao.recv(1024).decode('utf-8')

dados = json.loads(dados_json)

# Exibir dados recebidos (forma 1)

print("Dados recebidos:")

print(dados)
```

Da mesma forma que o cliente, importamos os dois módulos: socket e json e realizamos toda a configuração necessária para o servidor reservar uma porta e aguardar conexões.

Após o comando accept(), que é executado depois que o cliente realiza a conexão, o servidor recebe a string JSON. Recebemos os dados JSON da mesma forma que recebemos qualquer mensagem de acordo com os vídeos. A diferença é que podemos, agora, manipular esses dados.

Na forma 1 exibimos os dados na forma de um dicionário. Isso mostra o que o servidor recebeu do cliente.

Na forma 2 estamos acessando elementos específicos do dicionário dados (nome, idade e email) e os exibimos separadamente.

Na forma 3 utilizamos uma f-string para formatar a exibição dos dados recebidos de forma mais detalhada.

Veja o vídeo a seguir e tente fazer as alterações conforme eu fiz. Lembre-se de acompanhar o vídeo com VSCode aberto (ou com seu Editor de Código favorito).

Redes e Sistemas Distribuídos | Aula Prática | JSON

Seção 4 - Entrega - Prática 03

Data de entrega: 28 de Junho de 2024, até 23:59

Forma de Entrega: Vídeo de, no máximo, 5 minutos demonstrando o funcionamento do código multicast.py de acordo com as instruções a seguir

Descrição

Neste exercício, vamos juntar os conteúdos das três seções.

Altere o código multicast para fazer um chat. O “chat multicast” não funcionará da mesma forma que o “chat tcp” que fizemos anteriormente. Isso acontece devido às próprias características do multicast.

O requisito é que você altere o código do receptor e do emissor da seguinte forma:

1. Encapsule o código do emissor em uma função.
2. Encapsule o código do receptor em uma função.
3. Junte as duas funções em apenas um arquivo.
4. Adicione a lógica para realizar a seleção do tipo de código através de argumentos de linha de comando, de forma semelhante ao vídeo do chat TCP.
5. Também adicione a lógica para passar o nome de usuário por linha de comando para o código, da mesma forma que foi feito nos vídeos.
6. Altere a função emissor para que ela entre num loop em que o usuário digita várias mensagens, com uma lógica semelhante ao vídeo do chat TCP. O loop deve acabar quando o usuário digita a palavra “sair”.
7. Altere, também, a função do receptor. Entretanto, nessa função não deve ter o comando input pois ele apenas recebe dados através da rede por multicast. O receptor não digita dados.
8. Adicione a lógica para que o emissor envie um json com a seguinte estrutura message = {"user": None, "data": None}, na qual “None” pode ser substituído por uma string apropriada. (Que tal pesquisar o que é ‘None’ em Python? 🔎)
9. Não enviaremos o nome pelo socket. Antes, colocaremos o nome e a mensagem no dicionário e enviaremos o dicionário pelo socket usando JSON.
10. Adicione a lógica para que o receptor receba o json e utilize o valor da chave “user” para mostrar em cada mensagem, assim como fizemos no video TCP, e como você pode ver no GIF.

Resultado esperado no vídeo

Observem, neste gif, que executei o receptor nos dois terminais da esquerda, e o emissor nos dois terminais da direita. Envio uma mensagem de cada emissor separadamente, mas os dois receptores recebem a mesma mensagem. Observe que os emissores não recebem a mensagem um do outro. Se eu enviar a mensagem “sair” por qualquer emissor, os dois receptores serão desligados.