Trabalho Prático - Redes de Computadores

Nome: Geiziane Gonçalves e Matheus Sena
Tema: Desenvolvimento de Message Broker

Neste relatório, detalhamos a segunda parte do trabalho prático realizado na disciplina de Redes de Computadores no semestre de 2023-2. O projeto consiste na criação de um ambiente simulado de monitoramento de sensores remotos por meio de um dashboard em Python. O sistema desenvolvido utiliza uma arquitetura de comunicação baseada em sockets, com três sockets clientes (subscribers) conectados a um Broker para assinatura de tópicos específicos. Cada tópico está associado a informações provenientes da leitura de três Publish (simuladores de sensores) que geram dados aleatórios respectivamente de Clima, Temperatura e Umidade mandam para o Broker que envia para os Subscriber(assinantes). O dashboard se conectará apenas com os subscribers, onde cada subscriber vai ser responsável de enviar dados para um dos elementos do dashboard. Com isso, o dashboard terá 3 elementos, que serão atualizados pelos subscribers que recebem dados do broker, que por sua vez recebe dados dos sensores (Publishers).
Ademais, salienta-se que este relatório abordará somente a segunda parte do projeto, explicando conceitos relevantes para o seu entendimento, bem como explicará o código e sua implementação e o uso da aplicação. Em relação ao desenvolvimento, ambos os autores do projeto trabalharam em conjunto utilizando a plataforma Discord e GitHub em todo o trabalho.

1. Implementação

Neste capítulo explicaremos os códigos utilizados no projeto prático, os códigos foram divididos em 8 arquivos que correspondem aos 8 módulos: broker, brokerSub01,brokerSub02, brokerSub03, Clima_sim,Temperatura , Umidade e Dashboard.

1.1 broker

Configuração do Servidor:

O código inicializa um servidor usando sockets TCP/IP (socket.AF_INET, socket.SOCK_STREAM) para facilitar a comunicação entre clientes. Utiliza a função servidor.bind(("127.0.0.1", 9000)) da biblioteca socket para associar o servidor ao endereço IP 127.0.0.1 na porta 9000, permitindo a conexão de clientes neste endereço específico. Em seguida, a função servidor.listen(), também da biblioteca socket, coloca o servidor no estado de escuta, pronto para aceitar conexões vindas

dos clientes. Esse processo de escuta permite que os clientes se conectem ao servidor e iniciem a troca de mensagens ou solicitações.

Função comunicacao(servidor):

A função comunicacao(servidor) gerencia as conexões dos clientes(sub/pub) com o servidor. Utilizando um loop infinito, aguarda e aceita ativamente conexões dos clientes. Quando um cliente é aceito, é criada uma nova thread dedicada a lidar exclusivamente com esse cliente conectado. Isso permite que o servidor gerencie muitas conexões simultaneamente, garantindo um fluxo contínuo de interações.

A função comunicação inicialmente utiliza a função accept() da biblioteca socket para se conectar com um cliente que esteja tentando conexão. Uma vez a conexão aceita pelo broker, através do socket cliente e da função recv o broker irá receber o primeiro dado. O primeiro dado recebido é uma “flag” (mensagem inicial) que irá direcionar a conexão do cliente para a thread apropriada, cada thread irá cuidar de uma função para o funcionamento correto do broker, as funções são: clienteAssina e clientePublica. Essa flag será o primeiro dado compartilhado entre broker e cliente. Ou seja, a função comunicacao tem a capacidade de responder a uma variedade de solicitações.

A função clienteAssina:

A função clienteAssina é acionada quando um cliente envia uma mensagem começando com "assinar". Essa função, ao receber essa mensagem, identifica o tópico específico indicado nesta mensagem. Em seguida, ela adiciona o cliente que enviou essa mensagem à lista de assinantes desse tópico no dicionário topicos_e_Assinantes. Em resumo, ela conecta o cliente ao tópico desejado, permitindo que o cliente receba as mensagens publicadas neste tópico.

A função clientePublica:

A função clientePublica é acionada quando a flag vem com valor “publica”, informando que o cliente se trata de um Publisher. Essa função verifica se o tópico fornecido pelo cliente já existe no dicionário topicos_e_Assinantes. Se o tópico não existe, a função cria um novo, preparando-o para receber mensagens.Uma vez com o tópico definido. Em seguida, o broker recebe uma mensagem do Publisher e a manda para todos os clientes que estão inscritos nesse tópico em particular. Ou seja, isso permite que qualquer mensagem enviada por um cliente Publisher seja distribuída para todos os outros clientes Subscribers que estejam armazenados no dicionário topicos_Assinantes e tenham assinado aquele tópico específico.

Aspectos Gerais do Código:

O while True é usado para manter a thread do cliente, seja ele subscriber ou publisher, em execução contínua, garantindo que permaneçam ativos e prontos para receber ou enviar dados indefinidamente. Por outro lado, o for é empregado para percorrer elementos de um dicionário (tópicos), permitindo realizar ações específicas, como verificar a existência de subscribers inscritos ou adicionar novos subs.

1.2 Pub(Clima_sim,Temperatura e Umidade)

Este projeto optou por criar três arquivos publish e nomeá-los de Clima_sim,Temperatura e Umidade.

Os três códigos compartilham uma estrutura básica em comum.

Nota-se logo no início que todos utilizam as mesmas bibliotecas.A biblioteca random é usada para gerar números aleatórios, representando as condições meteorológicas (ensolarado, chuvoso, nublado) e também para simular valores aleatórios de temperatura e umidade. A socket é utilizada para estabelecer conexões entre os "publishers" (simuladores de dados de clima) e um servidor remoto (broker), possibilitando o envio de dados por meio de sockets. A biblioteca sys acessa argumentos passados pela linha de comando para determinar o intervalo de tempo entre as atualizações dos dados climáticos. Por fim, a biblioteca time é usada para controlar a frequência com que os dados são enviados para o servidor, introduzindo atrasos específicos entre as atualizações dos dados de clima, temperatura e umidade. Em resumo, essas bibliotecas são essenciais para gerar dados simulados de clima, temperatura e umidade e estabelecer a comunicação desses dados com um servidor remoto, controlando a frequência e o modo como esses dados são enviados.

Em sequência constata-se que todos eles também têm uma função principal chamada Publica que gera valores aleatórios (representando dados de Clima, Temperatura ou Umidade) em um loop infinito. Isso é realizado usando a função random.randint da biblioteca random, para gerar valores dentro de intervalos específicos.

Após a geração desses valores, cada código cria uma mensagem ou comando associado a esses valores. Por exemplo, o Clima pode ser "nublado", "ensolarado" ou "chuvoso", enquanto para Temperatura ou Umidade, são valores numéricos.

Ademais, todos os códigos estabelecem uma conexão com um servidor usando sockets (socket.socket(socket.AF_INET, socket.SOCK_STREAM)) e se conectam a um endereço IP e porta específicos (cliente.connect(('127.0.0.1', 9000))).

Depois da conexão, os dados gerados pelo simulador são enviados para o servidor usando cliente.send(comando.encode()). O formato da mensagem varia conforme o tipo de dado simulado.
Todos os códigos têm um loop while True que executa continuamente, enviando os dados simulados para o servidor com intervalos de tempo definidos por time.sleep(numero).

1.3 Sub(brokerSub01,brokerSub02,brokerSub03)

Este projeto optou por criar três arquivos Subs e nomeá-los de brokerSub01,brokerSub02 e brokerSub03. Estes três arquivos irão funcionar como intermediários entre o Broker e o Dashboard.

Todos os códigos utilizam as mesmas bibliotecas, como socket e argparse. Cada arquivo subscriber possui uma função chamada assinar, recebendo um cliente (para comunicação com o broker), um tópico (passado via terminal) e um clienteDash (para comunicação com o dashboard).

Esses códigos enviam comandos para o broker por meio de um socket do tipo cliente. Os comandos enviados são strings que indicam a ação a ser executada no broker, baseada no tópico fornecido. Além disso, enviam comandos para o dashboard utilizando um socket clienteDash para terem acesso aos elementos do Dashboard, diferenciando o comando de acordo com o tipo de informação que está sendo enviada ("Clima", "Temperatura", "Umidade").

Todos os códigos apresentam um loop infinito que recebe mensagens do broker e as envia para o dashboard, usando os sockets dos clientes para essa troca de mensagens.

Há também uma estrutura de configuração similar para estabelecer a conexão com o servidor broker e o servidor dashboard, incluindo a criação de um socket e a conexão com um endereço IP e uma porta específicos.

1.4 Dash

Este código foi desenvolvido para criar um dashboard em Python com o objetivo de exibir informações sobre clima, temperatura e umidade. Ele utiliza uma abordagem cliente-servidor, onde o código atua como servidor, aguardando por três conexões distintas, cada uma responsável por um tópico específico: Clima, Temperatura e Umidade.

Para a comunicação entre o Dashboard e os Subscribers, utiliza-se a biblioteca socket, possibilitando a troca de dados através de conexões via sockets. A

utilização da biblioteca Streamlit cria a interface do dashboard, permitindo a exibição de títulos, barras de progresso e frases relacionadas aos dados. Além disso, o Streamlit torna a criação de interfaces web interativas mais simples e intuitivas, possibilitando a apresentação clara e visualmente compreensível das informações para o usuário.

Primeiramente, o código define as informações de IP e porta para o Dashboard. Em seguida é utilizada a função listen para receber conexões. A execução de dash ocorre assim, o servidor aguarda conexões de clientes em um socket específico no ip e porta definidos. Quando um cliente se conecta, a função comunicacao é acionada, aguardando por três clientes diferentes, um para cada tópico.

A função “comunicacao” é responsável por iniciar threads para cada subscriber que se conectar ao Dashboard. Essas threads vão ser responsáveis por atualizar uma lista global, onde para cada índice temos os tópicos Clima, Temperatura e Umidade. Esses índices são responsáveis por guardar os dados nessa lista global que será utilizada pelo Dashboard.

O dashboard trava na função comunica, e só irá iniciar o while True responsável pela execução contínua do While quando, e não antes disso, os três subscribers responsáveis pelos dados de Clima, Temperatura e Umidade estarem conectados com o Dashboard. Uma vez a conexão estabelecida com os três subscribers o Dashboard irá iniciar o whille True responsável pela atualização de dados localizado no final do código. Este loop principal do código é responsável por atualizar a interface gráfica do dashboard a cada dois segundos.

Em resumo, teremos quatro linhas de execução, uma no programa principal que será o Dashboard consultando a lista global para pegar os valores de seus elementos situados nos três índices da lista, as outras três threads serão as funções AtualizaClima, AtualizaTemperatura e AtualizaUmidade. Essas três funções nessas três threads irão atualizar os índices da lista global para serem utilizados pelo Dashboard. A criação de threads é fundamental e é feita através da biblioteca threading. Além disso, a utilização da biblioteca time garante atualizações constantes na interface do dashboard.

A função markdown do Streamlit é aplicada para estilizar o texto, definindo cor, tamanho da fonte e alinhamento. É nessa parte do código que encontramos as instruções para mudar a cor do texto (color), definir o tamanho da fonte (font-size) e alinhar o texto ao centro (text-align).Quanto a criação de Barras de Progresso são criadas com o comando st.progress(valor). Elas representam visualmente o progresso de um valor específico, normalmente entre 0 e 100. A atualização dessas barras é feita através de comandos como linha2.progress(v2 / 100.0), onde v2 representa o valor atual relacionado à temperatura, por exemplo. Esse valor é dividido por 100 para adequar-se à escala da barra de progresso do Streamlit, que opera normalmente entre 0 e 1.

1.5 Regras para inicializar o projeto

O passo a passo a seguir deve ser feito para a execução correta dos códigos:

1- O broker.py ou Dashboard.py devem ser executados antes dos demais.

2- Com o item 1 concluído, devemos executar os Publishers Clima_sim.py, Temperatura.py e Umidade.py, em todos no momento da execução do terminal, devemos acrescentar um número inteiro que será responsável pelo intervalo de envio dos dados.

3- Após o item 1 e 2, seguimos para a inicialização dos subscribers brokerSub01, brokerSub02 e brokerSub03, que devem ser executados com o comando no terminal “-t nome do tópico”.

Observação: Para o funcionamento correto do código, é preciso colocar subscribers brokerSub01 -t Clima, brokerSub02 -t Temperatura e brokerSub03 -t Umidade.

Observação 02: Comando para executar o Dashboard é “streamlit run .\DashBoard.py”