Trabalho Prático - Redes de Computadores

Nome: Geiziane Gonçalves e Matheus Sena

Tema: Desenvolvimento de Message Broker

Introdução

Neste relatório, detalhamos o trabalho prático realizado na disciplina de Redes de Computadores no semestre de 2023-2. A tarefa envolve a criação de um Message Broker que adota o padrão publish/subscribe, fundamental para facilitar a comunicação entre aplicativos, sistemas e serviços, independentemente de suas características técnicas. A implementação desse Message Broker, com base em sockets, servirá como base para a simulação de um ambiente de sensoriamento remoto. O projeto está dividido em duas etapas, com a primeira fase concentrando-se no desenvolvimento de módulos de interação via terminal, incluindo Broker, Broker_Sub, Broker_Pub e Broker_Com, fornecendo aos alunos uma compreensão sólida de sistemas de mensagens e redes de computadores para etapas subsequentes.Este relatório abordará somente a primeira parte 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 quatro arquivos que correspondem aos 4 módulos: broker, brokersub,brokerpub e brokercom.

1.1. Aspectos Gerais do Código

Utilização das bibliotecas Socket e Threading foram necessárias, pois a biblioteca socket é usado para criar e gerenciar conexões de rede, enquanto threading é usado para criar threads para lidar com várias conexões de clientes simultâneas.

Ademais, foi utilizado a biblioteca argparse no código mencionado que serve para analisar os argumentos passados na linha de comando e, especificamente, para analisar o argumento que especifica o comando a ser executado pelo programa. O motivo da necessidade da biblioteca argparse é permitir que o programa seja configurado e controlado de maneira flexível a partir da linha de comando.

1.2. Módulo Broker:

O módulo Broker implementa sockets e threads e permite que vários clientes se conectem e realizem as seguintes operações: clienteAssina,clientePublica e listarTopicosAssinantes.

Após a implementação da biblioteca, o dicionário chamado *topicos_e_Assinantes* mapeia os tópicos para os clientes que os assinaram.

A **função comunicacao(servidor)** é o ponto de entrada principal do servidor e é onde a comunicação com os clientes é tratada. Observar-se que o servidor é configurado para ouvir conexões na porta 9000 e no endereço IP local (127.0.0.1).

Dentro da função comunicação há um while,onde código entra em um loop infinito, onde aguarda a conexão de clientes. Quando um cliente se conecta, ele é aceito pelo servidor.O servidor recebe uma mensagem do cliente e com base no conteúdo dessa mensagem, decide qual ação tomar.Ações a serem tomadas são:

1. Se a mensagem do cliente começa com "assinar", o servidor cria uma nova thread para lidar com a solicitação de assinatura. Isso permite que vários clientes assinem tópicos simultaneamente.
2. Se a mensagem começa com "publicar", uma nova thread é criada para tratar da publicação de mensagens em um tópico. A mensagem é enviada para todos os assinantes desse tópico.
3. Se a mensagem começa com "list", o servidor envia uma lista de tópicos e seus assinantes de volta para o cliente.

A **função clienteAssina(cliente, endereco, mensagem)** é responsável por adicionar um cliente à lista de assinantes de um ou mais tópicos. Ela analisa a mensagem do cliente para identificar os tópicos desejados e adiciona o cliente à lista de assinantes correspondente no dicionário topicos_e_Assinantes.

A **função clientePublica(cliente, mensagem)** lida com a publicação de mensagens em um tópico. Ela extrai o nome do tópico e a mensagem da mensagem recebida e, se o tópico existir, envia a mensagem para todos os assinantes daquele tópico.

A **função listarTopicosAssinantes(cliente**) envia ao cliente uma lista de tópicos e seus assinantes. Ela percorre o dicionário topicos_e_Assinantes e gera uma lista de endereços dos assinantes de cada tópico.

Salienta-se que o código implementa um servidor de mensagens que aceita conexões de clientes, permite que eles assinem tópicos, publiquem mensagens nesses tópicos e obtenham uma lista de tópicos com seus assinantes.

Por fim, é importante esclarecer alguns comandos relacionados a configuração e a comunicação do servidor.

servidor = socket.socket(socket.AF_INET, socket.SOCK_STREAM): Aqui, um objeto de soquete é criado para o servidor. O soquete é configurado para usar a família de endereços IPv6 (AF_INET) e o tipo de soquete TCP (SOCK_STREAM), o que significa que o servidor aceitará conexões TCP.

servidor.bind(("127.0.0.1", 9000)): O servidor é vinculado ao endereço IP 127.0.0.1 (a interface de loopback) e à porta 9000. Isso define o endereço e a porta em que o servidor estará ouvindo por conexões de clientes.

servidor.listen(): Aqui, o servidor é configurado para começar a ouvir por conexões. O argumento padrão (0) especifica o tamanho da fila de conexões pendentes. Um valor de 0 significa que o sistema operacional usará um valor padrão apropriado.

comunicacao(servidor): Essa linha chama a função comunicacao(servidor), que é responsável por gerenciar a comunicação com os clientes. A função comunicacao foi definida anteriormente no código e contém a lógica principal do servidor para aceitar conexões, processar comandos dos clientes e lidar com as operações de assinatura e publicação em tópicos.

1.3. Modulo Broker_Sub:

O código começa importando os módulos socket e argparse. O módulo socket é usado para criar uma conexão de socket com o servidor, enquanto o argparse é usado para analisar argumentos de linha de comando. Inicialmente esse módulo executa a função assinar(topico):

Função assinar(topicos):

A **função assinar(topicos**) é definida para realizar a assinatura nos tópicos, sendo armazenados no broker.Dentro da função assinar ocorre o seguinte:

- Um cliente de soquete é criado e conectado a um servidor na porta 9000.

- Um comando de assinatura é construído com os tópicos e enviado para o servidor.
- O cliente aguarda uma confirmação do servidor.
- Se a confirmação for "assinatura confirmada", o cliente entra em um loop infinito para receber e exibir mensagens do servidor.
- Mensagens recebidas são exibidas na tela, e a confirmação é tratada separadamente.

Para concluir esse módulo, explicaremos os últimos comandos relevantes:

parser = argparse.ArgumentParser(...): Cria um analisador de argumentos que entende os argumentos da linha de comando.

parser.add_argument(...): Define os argumentos da linha de comando, como o argumento -t que aceita uma lista de tópicos.

args = parser.parse_args(): Analisa os argumentos da linha de comando e armazena-os em args.

assinar(args.t): Chama a função assinar com os tópicos especificados na linha de comando, permitindo que o cliente se inscreva nesses tópicos.

1.4. Broker_Pub:

O módulo Broker_Pub também importa os socket e argparse.O argparrse vai configurar como os argumentos da linha de comando que serão tratados.Dois argumentos são definido,-t: Representa o tópico e -m:Representa a mensagem.

A função parse_args() do módulo argparse é usada para analisar os argumentos fornecidos na linha de comando, e os valores especificados para o tópico e mensagem são armazenados nas variáveis topico e mensagem, respectivamente.

Em seguida, o código constrói um comando de publicação no formato publicar tópico mensagem, incorporando as informações do tópico e da mensagem especificadas na linha de comando. O comando é convertido em bytes e enviado para o servidor utilizando cliente.send(comando.encode()).

Após o envio, o cliente aguarda uma mensagem de confirmação do servidor. Se essa mensagem for igual a "publicacao confirmada", isso indica que a mensagem foi publicada com sucesso no tópico especificado. Nesse caso, o código

imprime as informações sobre a mensagem e o tópico na tela. Caso contrário, ele imprime uma mensagem de falha.

Ademais,o código também lida com exceções que podem ocorrer durante a execução, como erros de conexão ou envio de mensagens. Ele garante o encerramento adequado da conexão com o servidor, independentemente do resultado da execução, utilizando o bloco finally e cliente.close().

1.5. Broker_com:

O código define uma função chamada comandos(comando), que é responsável por enviar comandos de controle ao servidor. No início da função, um objeto de soquete chamado cliente é criado e configurado para se conectar ao servidor, cujo endereço IP é '127.0.0.1' (localhost) e a porta é 9000.

Em seguida, o código verifica qual comando de controle foi passado como argumento para a função comandos. Se o comando for "LIST", o código avança para listar os tópicos e seus assinantes no servidor. Caso contrário, se o comando não for "LIST", o código emite uma mensagem de erro indicando que somente o comando "LIST" é aceito.

Se o comando for "LIST", o código envia o comando para o servidor usando cliente.send("list".encode()). O cliente, então, aguarda uma mensagem de confirmação do servidor após o envio do comando.

Após receber a mensagem de confirmação, o código verifica se a mensagem é igual a "confirmado". Se for, isso indica que o comando foi aceito pelo servidor. Nesse caso, o código imprime uma mensagem informando que o comando foi aceito e, em seguida, recebe e imprime a lista de tópicos e seus assinantes enviada pelo servidor.

Para garantir que a conexão com o servidor seja encerrada adequadamente, independentemente do resultado da execução, o código utiliza um bloco finally e a função cliente.close().

Além disso, o código configura os argumentos da linha de comando usando o módulo argparse. O único argumento definido é -c, que representa o comando de controle e é obrigatório.

A função parse_args() do módulo argparse é usada para analisar o argumento fornecido na linha de comando. O valor especificado para -c (comando) é extraído e passado como argumento para a função comandos.

Em resumo, o código permite que o usuário envie um comando de controle para o servidor. Neste exemplo, o comando "LIST" lista os tópicos e seus assinantes. O código estabelece uma conexão com o servidor, envia o comando, aguarda a confirmação e, se bem-sucedido, exibe a lista. Em caso de erro, lida com exceções e fecha a conexão de forma apropriada.

2. Execução do Projeto

Neste capitulo, abordaremos, como executa o projeto.O servidor Broker (broker.py) é o componente central que gerencia tópicos e assinantes. Os clientes Broker Subscriber (brokersub.py) assinam tópicos e recebem mensagens, enquanto os clientes Broker Publisher (brokerpub.py) publicam mensagens em tópicos. O código de comandos (brokercom.py) permite interagir com o servidor Broker para listar tópicos e assinantes.

Para fazer esses códigos funcionarem juntos, nós executamos o servidor Broker em um terminal separado e, em seguida, executamos os clientes Broker Subscriber, Broker Publisher e Broker Commands em terminais separados com os comandos apropriados e argumentos.

Conclusão

No desenvolvimento deste projeto prático de Redes de Computadores, focado na criação de um Message Broker com o padrão publish/subscribe, aprendemos conceitos essenciais relacionados a sistemas de mensagens e redes de computadores. Os módulos do projeto Broker, Broker_Sub, Broker_Pub e Broker_Com foram explicados em detalhes, permitindo entender como eles se encaixam e como cada um desempenha um papel específico na comunicação.

A implementação desses módulos nos proporcionou uma visão prática de como um Message Broker pode ser criado usando a biblioteca de soquetes em Python. Com o Broker, pudemos criar um servidor centralizado que gerencia tópicos e assinantes, facilitando a distribuição de mensagens. Os módulos Broker_Sub e Broker_Pub permitem que os clientes assinem tópicos e publiquem mensagens nesses tópicos, enquanto o Broker_Com facilita a interação com o servidor, permitindo a listagem de tópicos e assinantes.

Além da implementação, também discutimos a estrutura do código, as bibliotecas utilizadas, bem como os comandos e a comunicação entre os módulos. Os códigos foram testados com sucesso, demonstrando a capacidade de estabelecer conexões, lidar com exceções e trocar mensagens entre o servidor e os clientes.

Por fim, conclui-se que no contexto da matéria de redes de computadores, esse projeto prático contribuiu para entender conceitos de sistemas de mensagens, comunicação em tempo real e gerenciamento de conexões.