# Trabalho Prático - Redes de Computadores

Nome: Geiziane Silva 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 as etapas do trabalho em conjunto utilizando a plataforma Discord e GitHub.

#### **Conceitos Relevantes**

**Broker de Tópicos (Publish-Subscribe Broker):** Um broker de tópicos é um tipo de intermediário usado em sistemas de publicação e assinatura (pub-sub) para gerenciar a distribuição de mensagens. Os clientes podem publicar mensagens em tópicos específicos e assinar tópicos de seu interesse, recebendo apenas as mensagens relevantes.

#### **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.

##### **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.

##### **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, do dicionário chamado topicos\_e\_Assinantes, que serve para mapear 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 8080 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.

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 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 IPv4 (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", 8080))**: O servidor é vinculado ao endereço IP 127.0.0.1 (a interface de loopback) e à porta 8080. 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.

##### **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 em tópicos no servidor de tópicos (broker).Dentro da função assinar ocorre o seguinte:

* Um cliente de soquete é criado e conectado a um servidor na porta 8889.
* 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 par
* Se a confirmação for "assinatura confirmada", o cliente entra em um loop para receber mensagens.
* Mensagens recebidas são exibidas na tela, e a confirmação é tratada separadamente.

Concluindo esse modo explicaremos os últimos comandos relavantes:

**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.

##### **Broker\_Pub:**

O modulo Broker\_Pub também importa os módulos socket e argparse.

1. **Importação de Módulos**:
   * O código começa importando dois módulos: socket e argparse.
2. **Configuração dos Argumentos da Linha de Comando**:
   * Através do módulo argparse, o código configura como os argumentos da linha de comando serão analisados. Dois argumentos são definidos:
     + -t: Representa o tópico, que é obrigatório (required=True).
     + -m: Representa a mensagem, que é uma lista de palavras (nargs='+') e também é obrigatória.
3. **Análise dos Argumentos da Linha de Comando**:
   * A função parse\_args() do módulo argparse é usada para analisar os argumentos fornecidos na linha de comando.
   * Os valores especificados para -t (tópico) e -m (mensagem) são armazenados nas variáveis topico e mensagem, respectivamente.
4. **Bloco try...except para Tratamento de Exceções**:
   * O código utiliza um bloco try...except para capturar exceções que possam ocorrer durante a execução.
5. **Criação do Cliente e Conexão com o Servidor**:
   * Um objeto de soquete chamado cliente é criado.
   * O cliente é conectado a um servidor que está escutando no endereço IP '127.0.0.1' (localhost) e na porta 8889.
6. **Envio de Mensagem ao Servidor**:
   * O código constrói um comando de publicação no formato publicar tópico mensagem. Isso é feito utilizando as informações do tópico e da mensagem especificados na linha de comando.
   * O comando é codificado em bytes com cliente.send(comando.encode()) e enviado para o servidor.
7. **Recebimento da Confirmação do Servidor**:
   * O cliente aguarda uma mensagem de confirmação do servidor após o envio do comando.
8. **Verificação da Confirmação**:
   * A mensagem de confirmação recebida é verificada. Se for igual a "publicacao confirmada", significa que a mensagem foi publicada com sucesso no tópico especificado. Nesse caso, a mensagem é impressa na tela com informações sobre o tópico e a mensagem.
   * Se a mensagem de confirmação não for "publicacao confirmada", é exibida uma mensagem de falha.
9. **Tratamento de Exceções**:
   * Dentro do bloco except, o código lida com exceções que podem ocorrer durante a execução, como erros de conexão ou envio de mensagens. Qualquer exceção é capturada e uma mensagem de erro é impressa na tela.
10. **Encerramento da Conexão**:
    * No bloco finally, é feito o encerramento da conexão com o servidor utilizando cliente.close(). Independentemente do resultado da execução (bem-sucedida ou com erro), a conexão é fechada de forma adequada.

Em resumo, o código permite que o usuário envie uma mensagem para ser publicada em um tópico no servidor. Ele estabelece uma conexão com o servidor, envia a mensagem, verifica a confirmação e lida com exceções, encerrando a conexão de forma apropriada no final.

##### **Broker\_com:**

Importação do M**ódulos**:

* + O código começa importando os módulos socket e argparse. socket é usado para estabelecer conexões de rede, e argparse é usado para analisar os argumentos da linha de comando.

1. **Definição da Função comandos(comando)**:
   * Uma função chamada comandos é definida para executar comandos de controle no servidor.
2. **Criação do Cliente e Conexão com o Servidor**:
   * A função comandos é chamada com um argumento comando.
   * Dentro da função, é criado um objeto de soquete chamado cliente.
   * O cliente é conectado ao servidor que está escutando no endereço IP '127.0.0.1' (localhost) e na porta 8889.
3. **Verificação do Comando**:
   * A função verifica se o comando passado como argumento é igual a "LIST". Se for, ele executa o código relacionado à listagem de tópicos e assinantes.
4. **Envio do Comando ao Servidor**:
   * Se o comando for "LIST", o cliente envia o comando "list" para o servidor usando cliente.send("list".encode()).
5. **Recebimento da Confirmação do Servidor**:
   * O cliente aguarda uma mensagem de confirmação do servidor após o envio do comando.
6. **Verificação da Confirmação**:
   * A mensagem de confirmação recebida é verificada. Se for igual a "confirmado", o código continua a listar os tópicos e seus assinantes.
7. **Listagem de Tópicos e Assinantes**:
   * O cliente recebe uma lista de tópicos e seus assinantes do servidor, que é armazenada na variável lista.
   * A lista é impressa na tela, exibindo os tópicos e seus assinantes.
8. **Tratamento de Erros**:
   * Se o comando passado não for igual a "LIST", é exibida uma mensagem de erro.
9. **Tratamento de Exceções**:
   * O código inclui um bloco try...except para lidar com exceções que podem ocorrer durante a execução, como erros de conexão ou envio de mensagens. Qualquer exceção é capturada e uma mensagem de erro é impressa na tela.
10. **Encerramento da Conexão**:
    * No bloco finally, a conexão com o servidor é encerrada usando cliente.close(). Independentemente do resultado da execução (bem-sucedida ou com erro), a conexão é fechada.

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.

##### 

#### **Execução do 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 (Socket e Threading), 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.

A execução prática do projeto envolveu a inicialização do servidor Broker em um terminal separado e a interação com os clientes Broker Subscriber, Broker Publisher e Broker Commands, permitindo que os usuários assinem tópicos, publiquem mensagens e listem tópicos e assinantes.

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