Trabalho Prático 3

Sistema de Compartilhamento de Mensagens e Conteúdos multimídias Orientado a Eventos

Guilherme Mendes de Oliveira
Departamento de Ciência da Computação

Universidade Federal de Minas Gerais (UFMG) – Belo Horizonte, MG – Brasil

guilhermemendes@ufmg.br

1. Introdução

Este trabalho tem como objetivo a implementação de um sistema de envio de mensagem orientado a eventos. O sistema é baseado em três agentes básicos:

Emissor: Um tipo de cliente que envia mensagens para um outro cliente;

Exibidor: Um tipo de cliente que recebe mensagens de outro cliente;

Servidor: Um servidor que fornece uma interface recebendo as mensagens de um Emissor e direcionado a um Exibidor específico* ou a todos os Exibidores instanciados..

Ressaltando que o range de identificador para um Emissor é de 1 a 2^{12}-1 e para um Exibidor é de 2^{12} a 2^{13}-1

*As mensagens direcionadas de um emissor a um exibidor específico devem ter obrigatoriamente o ID do exibidor.

As mensagens possuem um *header* que é responsável por "classificar" o tipo da mensagem, definir o destinatário e o remetente e o contador de mensagens, alguns tipos de mensagem possui apenas este *header* e outras possuem o campo de dados.

Tipo da mensagem | Identificador de origem | Identificador de destino | Número de sequência

As mensagens são classificadas em OI (conexão entre um cliente e servidor) , ERRO (identificação de erros no processamento da mensagem anterior) , FLW (desconexão entre cliente e servidor), CLIST (requisição que lista clientes conectados), OK (confirmação de processamento correto da mensagem anterior), CREQ (requisição que lista de clientes conectados ao servidor) e MSG (mensagem enviada do emissor e mostrada no exibidor)

2. Implementação

2.1 Instruções de compilação e execução

O programa foi desenvolvido na linguagem Python versão 3.9 e executado no ambiente Linux distribuição Ubuntu versão 18.04.02, utilizando bibliotecas padrão da linguagem. Para executar temos os seguintes comandos:

Servidor: python3 servidor.py <porta>

Exibidor: python3 exibidor.py <host> <porta>

Emissor: python3 emissor.py <host> <porta> [idExibidor]

Obs: Os testes foram realizados utilizando localhost (127.0.0.1) e porta 50511, para o caso do Emissor o parâmetro idExibidor é opcional caso não queira relacionar com um exibidor específico por default as

mensagens serão visíveis para todos os exibidores

2.2 Estrutura de Dados

Para a implementação dos clientes foi instanciada uma classe para cada um dos tipos Emissor e Exibidor, ambas são heranças de uma classe pai e especializam seus métodos específicos.

Para a implementação do servidor também foi atribuída uma classe Server que realiza a interface de troca de mensagens entre os clientes e gerencia a instância de conexões, além disso temos a classe Connection uma classe "auxiliar" que é responsável por gerenciar as conexões com os *sockets*, o identificador de cada conexão e a que tipo de cliente ela se refere.

2.3 Código

Para a implementação dos agentes de comunicação foram utilizadas bibliotecas padrão de sockets baseado no protocolo TCP e para a utilização do paradigma baseado em eventos foi utilizada a biblioteca select. As demais bibliotecas são apoios para a montagem dos cabeçalhos e pacotes de envios e para a codificação e decodificação dos elementos da mensagem (*strings* e binários)

2.3.1 Servidor

O Servidor recebe como parâmetro para inicializar a porta de comunicação a ser utilizada uma vez que está setado para executar no *localhost* (127.0.0.1) de acordo com o paradigma orientado a eventos e as funções da biblioteca select ele aguarda algum "gatilho" para que realize alguma ação, estes gatilhos podem ser um comando de *status* para ver a lista de clientes conectados ou *sair* para encerrar o servidor,, uma mensagem contendo ou não dados , ou uma conexão de um cliente ao servidor

2.3.2 Exibidor

O Exibidor recebe como parâmetro o host e a porta de conexão, nos testes foram utilizados o localhost (127.0.0.1) e a porta 50511, o exibidor aguarda algum evento encaminhado do servidor para exibição de mensagem ou desconexão.

2.3.2 Emissor

O Emissor recebe como parâmetro o host e a porta de conexão, nos testes foram utilizados o localhost (127.0.0.1) e a porta 50511, além disso permite a conexão direta com um Exibidor específico caso seja passado via linha de comando é opcional, caso não seja fornecido não haverá associação e as mensagens serão disponibilizadas para todos os exibidores, após a conexão no terminal o Emissor aguardará texto para ser enviado ao servidor que por sua vez direciona aos exibidores conectados naquele momento.

O emissor pode enviar uma mensagem diretamente para um exibidor de identificador específico com o comando /msgTO idExibidor mensagem, se esse comando /msgTO não for utilizado o servidor irá repassar a mensagem para todos os exibidores conectados conforme padrão definido na implementação.

3. Desafios, dificuldades e imprevistos

O trabalho proposto teve dificuldades mais elevadas que os dois primeiros a começar por um novo paradigma de programação proposto e até então desconhecido para a implementação a biblioteca select do python foi fundamental no entanto poucas referências foram encontradas na internet apenas nos portais padrões como o *GeeksForGeeks* com utilizações mais básicas. Na comunicação direta entre um Emissor e um Exibidor específico também foi complexo uma vez que gerenciar os clientes em seus tipos e os identificadores demandou a implementação de dicionários e classes auxiliares.Além disso não foi possível implementar a comunicação atráves dos tipos de mensagem FILE, ao tratar os dados provenientes de arquivos houve erro no envio dos bytes pelo comportamento observado nas tentativas foi algo relacionado ao tratamento do tamanho dos arquivos no envio e recebimento das funções *send()* e *recv()* da biblioteca *socket.*

4.Conclusão

Lidar com um paradigma diferente causou estranheza e apreensão no começo da implementação, mas foi interessante pois trouxe uma motivação extra que resultou em mais estudo, busca por referências aliado com

mais tempo devido tanto a extensão do prazo de entrega quanto a diminuição de outros compromissos do calendário acadêmico. Foi possível integrar grande parte do conteúdo visto durante o curso, no entanto mesmo com dedicação não foi possível concluir com 100% as especificações iniciais a transmissão de dados de arquivos não foram implementadas para a entrega, no desenvolvimento estava acarretando em quebra do funcionamento do programa e por isso não foi disponibilizado na entrega.

5.Referências

<https://steelkiwi.com/blog/working-tcp-sockets/>. Acesso em: 24 de Agosto de 2021

<https://www.geeksforgeeks.org/simple-chat-room-using-python/>. Acesso em: 21 de Agosto de 2021

<https://docs.python.org/3/library/select.html>. Acesso em: 21 de Agosto de 2021