Disciplina: Redes de Computadores (UAB00078)
Curso: Bacharelado em Engenharia da Computação
Departamento: Unidade Acadêmica de Belo Jardim (UABJ)
Professor: Ygor Amaral Barbosa Leite de Sena
E-mail: ygor.amaral@ufrpe.br

Projeto 1: Implementação de Chat Cliente-Servidor

Para o projeto de chat cliente-servidor, você pode escolher entre implementar em Python ou C++. A estrutura é dividida em três versões, cada uma com funcionalidade adicional sobre a anterior, sempre com o uso de *sockets* TCP e *threads* para garantir a comunicação simultânea entre cliente e servidor.

Chat – Versão 1

A versão 1 deve começar com um cliente que conversa diretamente com o servidor, onde ambos podem enviar e receber mensagens de forma simultânea. Para isso, o uso de *threads* é essencial, permitindo que o cliente e o servidor operem de maneira independente para ler e enviar mensagens ao mesmo tempo, sem bloqueio. Em Python, você pode utilizar o módulo *socket* para a conexão TCP e o módulo *threading* para criar uma *thread* que escuta mensagens recebidas e outra para enviar. Em C++, o mesmo conceito pode ser implementado com as bibliotecas <*thread*>, <*sys/socket.h*>, e <*arpa/inet.h*>, criando threads distintas para lidar com o envio e a recepção de mensagens de forma assíncrona. Dessa maneira, o cliente e o servidor conseguem manter uma conversa ininterrupta.

Chat – Versão 2

A versão 2 deve expandir essa comunicação para suportar múltiplos clientes, permitindo que o servidor gerencie conexões independentes com cada cliente conectado. Nesta versão, o servidor precisa abrir uma nova *thread* para cada cliente que se conecta, garantindo que as mensagens enviadas ou recebidas de um cliente não interfiram nas comunicações de outros. Em Python, isso é feito novamente com *socket* para os *sockets* TCP e *threading* para criar *threads* para cada cliente. Em C++, a funcionalidade é parecida, utilizando <*thread*> para criar novas threads e *accept()* para estabelecer múltiplas conexões no servidor (um *accept()* para cada nova conexão/requisição aceita). Com essa estrutura, o servidor pode se comunicar simultaneamente com vários clientes, mantendo sessões independentes e possibilitando que cada cliente tenha uma conversa contínua e exclusiva com o servidor.

Chat – Versão 3

Versão 3 leva o chat a um nível mais complexo ao permitir que, além da comunicação com o servidor, os clientes também possam conversar diretamente uns com os outros. Nesta versão, o servidor age como um intermediário que roteia mensagens entre os clientes. Para isso, o servidor deve manter uma lista de clientes conectados e incluir um sistema de roteamento para encaminhar as mensagens para o destinatário correto. Em Python, você continuará usando *socket* para a conexão TCP e *threading* para as *threads*. Em C++, a estrutura é semelhante, para controlar a comunicação. O servidor precisa

interpretar quando uma mensagem deve ser redirecionada para outro cliente específico, garantindo que todos os clientes possam trocar mensagens entre si.

Considerações finais

Sugere-se criar o projeto com uma interface gráfica para facilitar a implementação da abordagem de escrita e recepção de mensagens simultâneas. Em Python, o *Tkinter* (pronuncia-se "T-K-inter") é uma ótima escolha pela sua simplicidade e integração com a linguagem. Para C++, a *Qt* é uma alternativa recomendada, oferecendo recursos completos para construção de interfaces gráficas e um bom suporte para manipulação de *eventos* e *threads*. Isso tornará o chat mais interativo e intuitivo para os usuários.

Além disso, solicita-se que todos os projetos da disciplina de redes de computadores sejam *commitados* no GitHub, permitindo o acompanhamento das versões e facilitando a colaboração e o *feedback* ao longo do desenvolvimento.

Boa diversão! =)