Relatório

Francisco Oliveira Gomes Junior - 12683190

Escola de Artes Ciências e Humanidades, Sistemas de Informação, São Paulo

O presente relatório visa descrever os resultados obtidos no Exercício Programa desenvolvido na disciplina de Desenvolvimento de Sistemas de Infomação Destribuídos.

1. Decisões de projeto

1.1 Paradgma de Programação escolhido

O paradigma POO foi escolhido como base fundamental para o desenvolvimento do sistema peer-to-peer devido às suas características intrínsecas que se alinham perfeitamente com os requisitos do projeto.

1.2 Divisão do programa em threads

As duas principais classes do projeto envolvida na comunicação multiThreads entre os nós é a classe Rede e a classe ThreadComunicacao, utilizando a ferramente de Sockets.

A classe Rede é responsável por gerenciar a rede peer-to-peer do sistema. Ela lida com a criação de conexões, envio e recebimento de mensagens, e gerenciamento de nós vizinhos.

A rede é inicializada através do construtor, informando o endereço IP, porta, e lista de nós vizinhos (opcional). O método iniciarConexao abre um ServerSocket na porta especificada para escutar por novas conexões.

Para se comunicar com outros nós, a Rede utiliza a classe ThreadComunicacao. Threads dessa classe são criadas para lidar com cada conexão individualmente. A thread escuta por novas conexões na rede e cria uma nova ThreadComunicacao para cada conexão aceita.

Quando uma nova conexão é estabelecida, a ThreadComunicacao lê a mensagem recebida do socket e utiliza métodos utilitários para processá-la. Por fim, a thread fecha a conexão com o remetente.

Além disso, a Rede possui métodos para enviar mensagens para outros nós, adicionar novos vizinhos à lista, parar de escutar por conexões, e listar os nós vizinhos atuais.

1.3 Utilizar operações bloqueantes e não bloqueante

A operação serverSocket.accept() garante que a thread threadEscuta() só prossiga após uma nova conexão ser totalmente estabelecida, prevenindo o processamento prematuro de outros dados.

O método enviarMensagem() retorna após enviar a mensagem para o socket, permitindo que a thread continue processando outras operações enquanto a mensagem é transmitida em segundo plano.

A classe ThreadComunicacao utiliza um loop infinito para ler mensagens do socket. A operação de leitura reader.readLine() bloqueia a thread até que uma mensagem seja recebida, mas como o loop é contínuo, a thread processa novas mensagens assim que elas chegam.