Aplicação: **Sistema de Transferência de Arquivos Simples (Client-Server)**

Alunos:

Arthur Mendes

Júlio César

Kaio Henrique

Lucas Cabral

Pedro Henrique Gaioso

Rafael Pereira Vilefort

A aplicação em rede a ser desenvolvida é um Sistema de Transferência de Arquivos Simples (Client-Server) com interface gráfica Java Swing. O servidor será capaz de atender múltiplos clientes simultaneamente através de *multithreading*. A aplicação terá duas funcionalidades de rede distintas: a transferência de dados do arquivo em si, que demandará o protocolo TCP, e o envio periódico de mensagens de *status* e monitoramento para o cliente, que será realizado via UDP. O cliente possuirá uma interface Swing que permitirá selecionar o arquivo a ser solicitado ao servidor, iniciar a transferência e exibir uma barra de progresso, além de uma área para mensagens de *status* em tempo real. O servidor, ao receber uma solicitação de arquivo via TCP, criará uma *thread* dedicada para gerenciar a transferência, enquanto utiliza outra *thread* (ou a principal) para enviar datagramas UDP contendo informações como o percentual de conclusão da transferência.

O protocolo TCP (Transmission Control Protocol) será utilizado para a funcionalidade principal da aplicação, que é a transferência dos dados do arquivo. A escolha se justifica pela natureza orientada à conexão e confiável do TCP. A transferência de arquivos exige que os dados cheguem ao destino sem perdas, na ordem correta e sem corrupção. O TCP oferece mecanismos intrínsecos de confirmação (ACKs), retransmissão e controle de fluxo, garantindo a integridade e a completude do arquivo. Se um pacote de dados do arquivo for perdido ou danificado durante a transmissão, o TCP irá detectá-lo e solicitar sua retransmissão, assegurando que o arquivo transferido seja idêntico ao original.

O protocolo UDP (User Datagram Protocol) será empregado para a funcionalidade de monitoramento de status da transferência. Especificamente, o servidor enviará periodicamente datagramas UDP para o cliente, contendo a porcentagem de progresso da transferência (por exemplo, "25% concluído"). A escolha do UDP é ideal, pois esta funcionalidade não requer confiabilidade ou garantia de entrega. O objetivo é fornecer

uma atualização rápida e em tempo real do status. A perda ocasional de um datagrama UDP, que apenas indicaria um percentual de progresso, não é crítica para o funcionamento da aplicação, pois o cliente receberá uma atualização mais recente pouco tempo depois. O UDP, sendo mais leve e sem o *overhead* de estabelecimento de conexão e confirmação do TCP, garante a baixa latência necessária para uma sensação de "tempo real" no monitoramento.

Para que os códigos Java TCP/UDP básicos (apresentados em sala de aula) se tornem multithreading no lado do servidor, o principal ajuste é no *loop* de escuta de novas conexões/mensagens. No código TCP, após o `ServerSocket.accept()` retornar um novo `Socket` para um cliente, o servidor não deve processar a comunicação imediatamente na *thread* principal. Em vez disso, ele deve instanciar uma nova *thread*, passando-lhe o novo `Socket` do cliente, e em seguida chamar o método `start()` dessa nova *thread*. Esta nova *thread* será responsável por toda a comunicação TCP com aquele cliente específico. Para o UDP, a multithreading pode ser aplicada para processar mensagens recebidas (se o processamento for demorado) ou para as operações de envio de status. Por exemplo, uma thread de envio de status UDP separada pode ser criada para gerenciar o envio periódico de datagramas de progresso para todos os clientes ativos. Ambas as implementações de *thread* podem ser feitas estendendo a classe `Thread` ou, preferencialmente, implementando a interface `Runnable`.