**Relatório do Sistema de Transferência de Arquivos via Sockets**

**1. Introdução**

O Sistema de Transferência de Arquivos via Sockets é uma aplicação desenvolvida em Python para permitir a transferência de arquivos entre um cliente e um servidor por meio de sockets TCP. A comunicação entre os sistemas ocorre de maneira direta e eficiente, usando um protocolo confiável e orientado à conexão, garantindo que os arquivos sejam enviados sem perdas. Este relatório descreve a implementação do sistema, suas funcionalidades principais, protocolos utilizados, desafios enfrentados durante o desenvolvimento e melhorias possíveis para o projeto.

**2. Descrição do Projeto e Suas Funcionalidades**

**2.1 Funcionalidades Principais**

O sistema é composto por um servidor que escuta em uma porta específica para aceitar conexões de clientes. Quando um cliente se conecta, o servidor permite que ele realize operações de envio e recepção de arquivos.

**1. Funcionalidades do Servidor:**

• O servidor aceita múltiplos clientes simultaneamente utilizando threads.

• O servidor escuta em uma porta específica, esperando que o cliente envie o nome do arquivo e, em seguida, o arquivo propriamente dito, em pacotes.

• O servidor pode enviar um arquivo solicitado pelo cliente.

**2. Funcionalidades do Cliente:**

• O cliente pode listar os arquivos disponíveis para envio (localizados na pasta uploads).

• O cliente pode enviar arquivos ao servidor, escolhendo um arquivo local.

• O cliente pode baixar arquivos disponíveis no servidor.

**2.2 Como Funciona a Transferência de Arquivos**

• Upload de Arquivos:

O cliente escolhe um arquivo de seu sistema e o envia ao servidor. O arquivo é enviado em partes (ou pacotes), com cada pacote enviado até que o arquivo seja completamente transferido.

• Download de Arquivos:

O cliente pode solicitar o download de um arquivo disponível no servidor. O servidor envia o arquivo solicitado em pacotes, e o cliente o armazena localmente.

• Listagem de Arquivos:

O cliente pode listar os arquivos disponíveis no diretório uploads do servidor, facilitando a escolha de arquivos para transferência.

• Gerenciamento de Erros:

O sistema conta com tratamento de exceções para lidar com erros de rede e casos onde o arquivo solicitado não existe no servidor.

**2.3 Exemplo de Fluxo do Sistema**

1. O servidor é iniciado e começa a escutar por conexões.

2. O cliente se conecta e escolhe entre listar, enviar ou baixar arquivos.

3. O cliente envia o arquivo ou solicita um arquivo do servidor.

4. O servidor envia o arquivo de volta, ou confirma o recebimento do arquivo, com sucesso ou erro.

5. O cliente recebe feedback de sucesso ou falha na transferência.

**3. Protocolos Utilizados**

**3.1 Protocolo TCP**

O sistema utiliza o Protocolo de Controle de Transmissão (TCP) para garantir uma comunicação confiável entre o cliente e o servidor. O TCP é orientado à conexão e assegura que os dados sejam transmitidos corretamente, garantindo a integridade e a ordem dos pacotes de dados durante a transferência de arquivos.

**3.2 Funções e Técnicas Utilizadas**

• Sockets (socket.socket): A biblioteca socket é utilizada para criar o servidor e os clientes, permitindo a comunicação direta por TCP.

• Threads (threading): A utilização de threads permite que o servidor atenda múltiplos clientes simultaneamente, sem que o processo de um bloqueie os outros.

• Envio de Arquivos em Pacotes (BUFFER\_SIZE): Para evitar que o sistema trave ao transferir arquivos grandes, o arquivo é dividido em pacotes de 1024 bytes, sendo enviados um por vez.

**4. Desafios Enfrentados e Soluções Implementadas**

**4.1 Gerenciamento de Conexões Múltiplas**

• Desafio: O servidor precisa lidar com múltiplos clientes tentando se conectar simultaneamente, o que poderia bloquear a aplicação.

• Solução: A implementação de threads para cada cliente garante que cada conexão seja tratada de forma independente, sem bloquear as outras conexões.

**4.2 Transferência de Arquivos Grandes**

• Desafio: Transferir arquivos grandes sem sobrecarregar a memória ou causar interrupções.

• Solução: Os arquivos são enviados em pacotes de tamanho fixo (1024 bytes). Isso evita problemas de memória e assegura uma transferência controlada e sem interrupções.

**4.3 Tratamento de Erros e Conexões Interrompidas**

• Desafio: Gerenciar erros de rede e garantir que o cliente seja informado corretamente sobre falhas, como arquivos não encontrados ou desconexões.

• Solução: O sistema implementa tratamento de exceções robusto, como a verificação da existência do arquivo antes de iniciar a transferência e o envio de mensagens adequadas ao cliente.

**5. Melhorias que Poderiam Ser Feitas no Projeto**

**5.1 Segurança na Transferência de Arquivos**

• Proposta: Implementar criptografia de dados durante a transferência para proteger informações sensíveis. O uso de bibliotecas como ssl para envolver a comunicação TCP em uma camada segura pode ser uma adição importante.

**5.2 Interface Gráfica do Usuário (GUI)**

• Proposta: A criação de uma interface gráfica (GUI) usando bibliotecas como Tkinter ou PyQt pode tornar o sistema mais intuitivo para usuários que não estão familiarizados com o terminal.

**5.3 Logs e Monitoramento**

• Proposta: Adicionar um sistema de logs que registre todas as transferências realizadas, erros, e status das conexões. Isso facilitará a monitoração e depuração do sistema.

**5.4 Suporte a Transferência de Diretórios**

• Proposta: Implementar uma funcionalidade para enviar e receber diretórios inteiros, mantendo a estrutura de pastas. Isso proporcionaria maior flexibilidade e organização na transferência de arquivos.

**5.5 Eficiência na Transferência de Grandes Quantidades de Dados**

• Proposta: Otimizar a divisão dos pacotes para arquivos muito grandes, com a utilização de técnicas como compressão de arquivos, para minimizar o tempo de transferência e o consumo de banda.

**6. Conclusão**

O Sistema de Transferência de Arquivos via Sockets desenvolvido em Python oferece uma solução eficiente e robusta para a transferência de arquivos entre clientes e servidores. A utilização de sockets TCP e threads para múltiplos clientes garante a confiabilidade e escalabilidade do sistema. As melhorias sugeridas podem tornar o sistema ainda mais seguro, eficiente e amigável, ampliando suas funcionalidades para atender a uma variedade maior de necessidades. O projeto proporciona uma boa base para sistemas de comunicação mais avançados e de maior escala.