Universidade Federal de Minas Gerais
Instituto de Ciências Exatas
Bacharelado em Sistemas de Informação

Trabalho prático para a disciplina de Redes de Computadores:
Controle de Envio de Arquivos

Vitor de Oliveira Mafra - 2018046831

Belo Horizonte
2023

Introdução

O presente trabalho tem como objetivo a implementação de um sistema de transferência de arquivo por meio de sockets em linguagem C, no contexto de redes de computadores. O sistema consiste em um servidor que é capaz de aceitar a conexão de um único cliente, permitindo que este envie um arquivo. O servidor, por sua vez, deve receber o arquivo enviado e armazená-lo em seu sistema de arquivos. Adicionalmente, o cliente deve ser capaz de estabelecer a conexão com o servidor, enviar o arquivo desejado e receber uma confirmação por parte do servidor, atestando o sucesso da transferência. A implementação desse sistema prático oferecerá uma oportunidade para explorar conceitos fundamentais de redes, sockets e programação em C, além de fornecer uma compreensão mais aprofundada dos processos de comunicação entre cliente e servidor.

O código se divide em 3 principais arquivos, que serão melhor descritos a seguir:

server.c

Este arquivo é responsável por criar um servidor TCP/IP que aguarda conexões de clientes. Quando um cliente se conecta, o servidor recebe dados do cliente, armazena-os em um arquivo e envia uma resposta de confirmação ao cliente. O programa utiliza sockets para a comunicação entre o servidor e os clientes. Ele é capaz de lidar com conexões IPv4 e IPv6.

O programa começa recebendo argumentos da linha de comando: o endereço IP e a porta em que o servidor será executado. Em seguida, chama a função para iniciar o servidor, que inicializa a estrutura de endereço storage, que será usada posteriormente.

Imediatamente, o programa cria um socket usando a função padrão para a criação de sockets. O tipo de socket que trataremos aqui é o que aceita o protocolo TCP. Após a criação do socket, o programa faz o **bind**, associando o socket ao endereço e à porta especificados na estrutura storage. Em seguida, chama a função **listen** para colocar o socket em modo de escuta, permitindo que ele aceite conexões dos clientes.

Em seguida, o programa entra em um loop infinito usando a função **accept**. A função accept bloqueia a execução até que uma conexão seja estabelecida com um cliente. Quando uma conexão é estabelecida, um novo socket é criado para se comunicar com o cliente. O programa recebe os dados enviados pelo cliente usando a função recv, que armazena os dados num buffer. Os dados recebidos contêm o nome do arquivo a ser gravado e a mensagem a ser gravada no arquivo.

Por fim, a função **writeStringToFile** é chamada para gravar a mensagem no arquivo especificado. A resposta é enviada de volta para o cliente usando a função send, que transmite os dados através do socket. O programa verifica se todos os dados foram enviados corretamente. Após enviar a resposta, o socket é fechado usando a função close e o programa continua aguardando novas conexões.

client.c

O cliente de rede se conecta a um servidor remoto para enviar e receber arquivos por meio de sockets para estabelecer a comunicação com o servidor. O programa começa obtendo os argumentos da linha de comando, que são o endereço IP e a porta do servidor. Em seguida, ele cria um socket IPv4 ou IPv6 e faz a conexão com o servidor usando a função **connect**. Se houver algum erro durante o processo de conexão, o programa exibe uma mensagem de erro e encerra. Após a conexão ser estabelecida, o programa entra em um loop onde é exibido um menu de opções para o

usuário. O usuário pode digitar comandos para selecionar um arquivo e enviá-lo para o servidor. Os comandos suportados são "**select**" para selecionar um arquivo, "**send**" para enviar o arquivo selecionado e "**exit**" para sair do programa.

Quando o comando "send" é executado, o programa verifica se um arquivo foi selecionado e, em seguida, lê o conteúdo do arquivo em uma string. Essa string é enviada para o servidor usando a função **send**. Se ocorrer algum erro durante o envio, o programa exibe uma mensagem de erro e encerra. O programa continua recebendo dados do servidor usando a função **recv** em um loop até que não haja mais dados para receber. Os dados recebidos são armazenados em um buffer e, no final, são exibidos na saída padrão juntamente com o número total de bytes recebidos. Após a conclusão do processo de recebimento, o socket é fechado e o programa é encerrado.

common.c

O arquivo em questão contém uma biblioteca de funções comuns tanto para o servidor quanto para o cliente do programa. Essas funções são responsáveis por manipular conexões de rede, processar mensagens, validar dados de entrada e gerenciar recursos compartilhados. Além disso, o arquivo também inclui uma variedade de funções úteis que são amplamente utilizadas em todo o código do programa, abrangendo tarefas como manipulação de strings, processamento e leitura de arquivos. Essas funções foram projetadas para promover a reutilização de código, facilitar a comunicação eficiente entre o servidor e o cliente, e melhorar a eficiência e manutenibilidade do programa.

Segue, abaixo, uma breve explicação de cada uma delas:

exitLog: exibe uma mensagem de erro fornecida como argumento e termina o programa com um código de erro. Ela usa a função perror() para imprimir a mensagem de erro associada ao valor da variável de erro.

addrParse: recebe uma representação textual de um endereço IP e uma string contendo um número de porta e converte essas informações em uma estrutura de dados de endereço de soquete. Ela suporta tanto endereços IPv4 quanto IPv6. A função verifica se os argumentos são válidos, converte o número de porta para o formato correto (host to network short) e preenche a estrutura de acordo com o tipo de endereço (IPv4 ou IPv6).

addrToStr: recebe uma estrutura de dados de endereço de soquete e converte essa estrutura em uma representação textual do endereço IP e da porta correspondentes. Ela suporta tanto endereços IPv4 quanto IPv6. A função verifica o tipo de endereço (IPv4 ou IPv6), converte o endereço IP e a porta para o formato correto (network to host short) e gera a representação textual correspondente. A representação resultante é armazenada na string fornecida como argumento.

serverInit: inicializa a estrutura de dados de endereço de soquete com base no protocolo (IPv4 ou IPv6) e no número de porta fornecidos. Ela suporta a inicialização de servidores tanto para IPv4 quanto para IPv6. A função converte o número de porta para o formato correto (host to network short), limpa a memória da estrutura de dados e preenche os campos apropriados com as informações do protocolo e da porta.

fileExists: verifica se um arquivo com o nome fornecido existe. Ela tenta abrir o arquivo em modo de leitura e fecha-o imediatamente. Se o arquivo for aberto com sucesso, significa que ele existe, e a função retorna 1. Caso contrário, retorna 0 para indicar que o arquivo não existe.

hasValidExtension: verifica se um arquivo com o nome fornecido possui uma extensão válida. Ela compara a extensão do arquivo (extraída a partir do último ponto) com uma lista de extensões válidas. Se a extensão do arquivo corresponder a uma das extensões válidas, a função retorna 1. Caso contrário, retorna 0 para indicar que o arquivo não possui uma extensão válida.

readFileToString: lê o conteúdo de um arquivo com o nome fornecido e o retorna como uma string. Ela tenta abrir o arquivo em modo de leitura, determina o tamanho do arquivo, aloca memória para a string que será retornada (considerando também o tamanho do nome do arquivo), adiciona o nome do arquivo como prefixo da string, lê o conteúdo do arquivo para a string e adiciona "\end" como sufixo da string. Em caso de sucesso, a função retorna a string com o conteúdo do arquivo. Em caso de falha, retorna NULL.

writeStringToFile: escreve uma string fornecida em um arquivo com o nome fornecido. Ela cria um diretório chamado "server_files" (se ainda não existir), gera o caminho completo para o arquivo (com base no diretório e no nome do arquivo), verifica se o arquivo já existe e sobrescreve-o se necessário. Em seguida, ela abre o arquivo em modo de escrita, escreve a string no arquivo (excluindo os últimos 4 caracteres, que são "\end") e fecha o arquivo. A função também imprime uma mensagem indicando se o arquivo foi sobrescrito ou recebido com sucesso.

Conclusão

O controle de envio de arquivos por meio da comunicação entre sockets de um cliente e um servidor apresenta desafios significativos ao programador, especialmente ao trabalhar com a linguagem C. A implementação eficiente e segura de sockets requer um profundo entendimento da comunicação entre cliente e servidor, bem como do funcionamento e manipulação dos próprios sockets. Ao enfrentar esses desafios, pode-se adquirir uma valiosa experiência que contribui para a consolidação do conhecimento em redes de computadores. O domínio dos conceitos e técnicas relacionados aos sockets é de extrema importância para a melhor compreensão de aplicações robustas e confiáveis, e proporciona uma base sólida para o desenvolvimento de soluções avançadas no campo das redes de computadores.