Controle de Envio de Arquivos

Documentação

Samuel Lipovetsky

May 21, 2023

1 Client

O client implementado é responsável por entender os comando select file , send file e exit. Assim como, separar os comandos entre nome do arquivo e o tipo de comando. Ainda, é necessário imprimir na tela as respostas do server e também as respostas do próprio cliente em caso de erros no select file ou send file.

1.1 Analisando o comando recebido

Para saber qual rotina o client deve executar, inicialmente foi necessário entender o comando recebido. Dessa forma, existe uma função capaz de ler o input do usuário e retornar o tipo do comando. Isso foi feito analisando o prefixo do comando escrito pelo usuário, ainda, é preciso identificar o nome do arquivo no caso do comando select file e dar o devido tratamento em casos de arquivos não existentes ou inválidos.

1.2 select file

Uma vez que temos já separados o tipo de comando e o nome do arquivo, podemos começar a tratar os possíveis erros com o select file. Inicialmente, é conferido se o arquivo é valido, logo, foi feita uma função que recebe o nome do arquivo e compara o sufixo desse nome com as extensões válidas ".txt", ".c", ".cpp", ".py", ".tex", ".java".

Caso a extensão seja Válida , também é necessário conferir se o arquivo existe, isso foi feito usando a função acesss da biblioteca unistd.h.

1.3 send file

Ao receber o comando send file, é conferido se um select file valido já foi executado, caso contrario , o client imprime " no file selected ". Após o usuário selecionar um arquivo válido , é possível utilizar o comando send file. Assim, é necessário que o client monte a mensagem a ser enviada via socket de acordo com o protocolo proposto. Dessa forma, é feita a concatenação do cabeçalho, corpo da mensagem e o fim. O cabeçalho é simplesmente o nome do arquivo a ser criado pelo servidor, o corpo da mensagem é de fato o conteúdo do arquivo selecionado e o fim é simplesmente a string /end.

Após montar a mensagem a ser enviada, é preciso enviar a mensagem e esperar a resposta do servidor. Isso foi feito utilizado as funções send() e recv(), sendo que a mensagem recebida do servidor é impressa na tela do cliente.

1.4 exit

Ainda, o client pode ler o comando exit da entrada padrão. No caso desse comando, simplesmente é enviada a mensagem "exit/end" para o server, além disso, também é fechada a conexão e a execução do client é interrompida.

2 Server

O server só pode receber dois tipos de mensagens , uma com o formato cabeçalho, corpo da mensagem e fim , e a outra a mensagem "exit". Antes do server responder o client, é necessário separar as mensagem recebidas entre mensagens de criação de arquivos ou mensagens do tipo exit. Para isso, é feito uma simples comparação de string com o a mensagem recebida e a string "exit/end" , caso as string sejam diferentes, é assumido que foi recebido uma mensagem de criação de arquivos.

2.1 Criando arquivos

É importante notar que o server não checa a extensão do arquivo, sendo que isso é responsabilidade do client. Ao receber uma mensagem de criação de arquivos , inicialmente é necessário checar se o arquivo já existe. Assim, foi feito um sistema de flags para o tipo de resposta a ser retornada para o client. Portanto, quando é conferido que um arquivo já existe no diretório do server , é setado uma flag "already_exists" internamente no server e a resposta para o client é "file [x] overwriten" . Destaca-se que essa flag é totalmente interna ao código do server e não são enviadas para o client, o client recebe de fato a mensagem "file [x] overwriten". Caso o arquivo não exista , essa flag é setada como falsa, e o server responde ao cliet "file [x] received".

Entretanto, antes de criar o arquivo e enviar a confirmção é necessário separar as partes da mensagem recebida, assim, foi criada uma função que separa a mensagem em cabeçalho e corpo. Após , essa separação , simplesmente é chamada uma função que cria um arquivo de nome "cabeçalho" e é escrito nesse arquivo "corpo da mensagem". Ainda, é importante notar que o "/end" que sinaliza o fim da mensagem é retirado do conteúdo da mensagem.

2.2 Exit

Ao receber o comando exit, o server fecha as conexões com o client, e uma flag chamada "exit" é setada como verdadeira, assim , a parte que envia respostas ao cliente sabe que deve ser retornada a mensagem "connection closed " além de fechar as conexões e encerar a execução do server.

3 Criação das conexões

Tanto o server quanto client recebem argumentos que definem a conexão a ser feita em tempo de execução. Para isso , existem funções em common.c que são responsáveis por analisar esses argumentos e inicializar os structs necessários para a conexão.

3.1 client

Para a criação da conexão do socket no client, inicialmente é utilizado o struct sockaddr_storage como placeholder de sockdaddr, uma vez que esse struct consegue armazenar informações independentement de ipv4 ou ipv6. Ao executar o programa client, são passados como argumento o endereço e porta do server, esses argumentos são utilizados para popular sockaddr_storage que em sequencia é utilizado para criar o socket por meio da função socket(). Em seguida, é feito o cast de sockaddr_storage para sockaddr que logo em seguida é usado na função connect(). A partir desse ponto, é possível utilizar send() e recv() , passando como argumento o socket criado.

3.2 server

Similarmente ao client, struct sockaddr_storage é utilizado inicialmente e nesse struct são inseridos os argumentos recebidos pelo client que são versão do IP e a porta. Dependendo do argumento tipo de versão, sockaddr_storage é preenchido com a versão descrita. Em seguida, é criado o socket e é utilizada a função setsockop para configurar esse socket. Após isso, é feito o cast de sockaddr_storage para sockaddr e então bind() é executada para ligar o server à porta especificada. Em sequencia, é executada a função listen(). Por fim, existe um loop dos comandos accept(), recv() e send() que ficam respectivamente aceitando conexões, recebendo mensagens e enviando respostas.

4 Emprevistos e desafios

Apesar do trabalho ser sobre o uso e criação de sockets, essa parte foi relativamente simples e a documentação e vídeos disponibilizados foram uma base muita boa para a criação dessa parte do código e do meu aprendizado. Porém, a parte mais difícil foi testar e depurar as funcionalidades tanto do server quanto do client. Como os programas foram feitos todos em C , a parte de lidar com texto, tanto durante a alocação de memória quanto a parte de escrita requer um cuidado especial para evitar falhas de segmentação.