U.C. Redes de Computadores

1º Trabalho
Sistema de comunicação

Docente: Pedro Patinho
Pedro Salgueiro
Discentes: André Baião 48092
Gonçalo Barradas 48402

Abril 2022

1 Introdução

A forma de conectar duas interfaces numa rede é por meio de um *socket*. Ambas as interfaces se conectam a *sockets* distintos, por meio de um endereço, numa determinada porta. De seguida ambos os *sockets* conectam-se entre si.

Inicialmente é criado um *socket* por parte do servidor, este conecta-se ao *socket* pela porta desejada e aguarda a receção de um pedido de conexão.

O cliente conecta-se a um *socket* por meio do endereço IP e a porta, pedindo uma conexão ao servidor. Assim que um cliente se conecta, o servidor cria uma nova *thread* para poder lidar com esse cliente e assim sucessivamente enquanto houver novos clientes a conectarem-se.

Uma *thread* é uma forma de dividir um processo em duas ou mais tarefas que podem ser executadas simultaneamente. Para este trabalho as *threads* permitem ao servidor lidar com os vários clientes simultaneamente.

Foi-nos proposto elaborar um programa que implementasse um sistema de comunicação de rede para grupos de trabalho, no qual o servidor deve correr o serviço de chat na porta "1234". Este sistema tem ainda as seguintes condições: enviar a mensagem para todos os clientes do canal quando o prefixo da mesma é o sinal de soma (+) e enviar uma mensagem em privado quando o prefixo é o sinal de subtração (-) seguido do nickname do cliente para qual deve ser enviada a mensagem. Quando um cliente sai do sistema todos os clientes ainda conectados devem receber um aviso com o nickname do cliente e a mensagem "Disconnected". Todos os clientes devem ter um nickname.

2 Servidor

O servidor é o responsável por abrir o *socket* na porta desejada e por aceitar novos clientes. Ao ser aceite um cliente, o servidor cria uma *thread* para esse cliente. Após estabelecida a conexão, o servidor guarda o nickname do cliente e analisa as mensagens enviadas pelo mesmo, encaminhando-as para o seu destino correto. Para implementar o servidor usamos três classes:

- `Server`
  Cria a o *socket* e aceita ligações. Também é responsável por enviar mensagens para os clientes. Guarda o *socket* do servidor, a informação da porta à qual está conectado e uma lista de utilizadores conectados ao servidor:
  - `InicializeServer`: esta função cria o *socket* do servidor e chama a `aceptClients`;
  - `aceptClintes`: esta função aceita as conexões efetuadas pelos clientes e cria uma *thread* para cada cliente (`UserClient`);
  - `serverToAll`: recebe como argumento uma mensagem e envia para todos os clientes a mensagem do servidor;
  - `serverToClient`: recebe como argumento uma mensagem e um `UserClient` (destino) e envia a mensagem recebida para esse cliente;
  - `messageForAll`: esta função recebe como argumento uma mensagem e o `UserClient` que pretende enviar a mensagem para todos os clientes, e envia a mensagem para todos os clientes informando quem foi o emissor;
  - `messageToClient`: esta função recebe como argumento uma mensagem, o `UserCliente` emissor e uma String com o nickname do receptor. Após identificar o `UserClient` receptor, adiciona o nickname do emissor e envia a mensagem para esse cliente, caso não o encontre envia um aviso para o cliente emissor a dizer que o utilizador não foi encontrado;
  - `getClient`: recebe como argumento um nickname e devolve o `UserClient` com o mesmo nickname;
  - `removeClient`: recebe como argumento um `UserClient` e retira da lista de clientes conectados esse cliente;
  - `addClient`: recebe como argumento um `UserClient` e adiciona-o á lista de clientes conectados.

- `ProcessUser`:
  Responsável por ler e analisar as mensagens que o cliente envia para o servidor. Guarda informações do servidor e informações do cliente com que está a trabalhar. Esta função é executada em *threads* para que os clientes possam enviar mensagens ao mesmo tempo.

  Possui uma função `HandleRecivedMensages` que é responsável por ler as mensagens enviadas por um cliente e analisar as mesmas.

  Caso a mensagem comece por "+" significa que a mensagem é para todos e então é usado o método `messageForAll`.

  Caso a mensagem comece com "-" significa que a mensagem é privada e nesse caso retiramos da mensagem o nome do destinatário e enviamos a mensagem utilizado o método `messageToClient`.

  Caso a mensagem seja "**Disconect**" significa que um cliente se desconectou e nesse caso efetua-se a remoção do cliente da lista de cliente conectados utilizando a `removeClient` e fechamos o *socket* do cliente, informando os restantes clientes que este cliente se desconectou usando o método `serverToAll`.

- `UserClient`: Esta função cria os *buffers* de leitura e de escrita do cliente, define o nickname do mesmo e cria um `ProcessUser` para processar as mensagens do cliente. Esta Class guarda informações sobre o server, o cliente, o nickname e guarda os *buffers* de leitura e escrita do cliente.
  - `Disconect`: quando é executada o *socket* do cliente em questão é fechado;
  - `getNickname`: devolve o nickname do cliente;
  - `getInput`: devolve o *buffer* de leitura do cliente;
  - `getOutput`: devolve o *buffer* de escrita do cliente;

3 Cliente

O cliente é responsável por se conectar ao servidor e por receber e enviar mensagens. O cliente através de um *socket*, usando um endereço e uma porta, estabelece a conexão com o servidor. Após efetuada a conexão o cliente cria os *buffers* de escrita e leitura, e o cliente envia ao servidor uma mensagem com o seu nickname.

Após definido o nickname, o cliente está pronto para receber e enviar mensagens. Para podermos enviar e recebermos mensagens ao mesmo tempo vamos criar uma *thread* para receber mensagens e assim, enquanto a *thread* vai recebendo mensagens, o cliente poderá enviar mensagens á vontade.

Caso o cliente envie a mensagem para se desconectar, todos os *buffers* e *socket* vão ser fechados.

4 Conclusão

Este tipo de tecnologias pode ser bastante útil dentro de uma empresa ou organização pois possibilita a comunicação entre os vários elementos tanto de forma privada como em grupo permitindo uma organização de trabalho eficiente e de forma mais segura.

O programa desenvolvido foi testado numa rede interna, e funcionou como esperado, tendo sido possível o envio de mensagens entre três utilizadores com sucesso, desta forma é possível concluir que o desenvolvimento de um chat foi cumprido com sucesso.

Referências

[GeeksforGeeks, 2022] GeeksforGeeks (2022). Socket programming in c/c++.
https://www.geeksforgeeks.org/socket-programming-cc/.

[Patinho and Salgueiro, 2022] Patinho, P. and Salgueiro, P. (2022). Aulas de redes de computadores. *Universidade de Évora*.