FCTUC FACULDADE DE CIÊNCIAS
E TECNOLOGIA
UNIVERSIDADE DE COIMBRA

Relatório

Redes e Comunicação

Saulo José Mendes
Rui Moniz
21 de maio de 2023

Introdução

Neste relatório intermédio, apresentamos o desenvolvimento de uma aplicação que utiliza os protocolos UDP e TCP para criar um serviço de notícias, como parte das atividades da disciplina de Redes e Comunicação. Deste modo, foram implementadas todas as funções necessárias para o funcionamento da consola do administrador, do cliente do consumidor, além das estruturas internas e a ferramenta de *multithreading,* que permite o funcionamento simultâneo de ambos protocolos. Além disso, é apresentado o circuito no GNS3 e a sua configuração também será exposta.

Servidor TCP (TCPserver.c)

1. void listTopics(int client, TopicList *topicList): Esta função é responsável por mostrar todos os tópicos disponíveis na topicList e enviá-los ao cliente. Vale ressaltar que recorre à função topicToString() para converter-los em strings usando e enviar-los ao cliente com a função send().

2. void subscribeToTopic(int client, int topicID, TopicList *topicList, User *user): Esta função trata da subscrição de um utilizador a um tópico específico. Primeiro verifica se o usuário já está inscrito no tópico chamando a função findTopicByID() com a topicList do utilizador. Se o utilizador não estiver inscrito, tenta encontrar o tópico solicitado através da findTopicByID() com a topicList principal. Se o tópico existir, este é inserido na topicList do usuário usando a função insertTopic().

3. void leitorMenu(int client, TopicList *topicList, User *myUser): Esta função executa o menu de um leitor. Esta recebe continuamente comandos do cliente e executa as ações correspondentes.

Os comandos disponíveis são:

- "LIST_TOPICS": para listar todos os tópicos existentes utilizando a função listTopics();
- "SUBSCRIBE_TOPIC <topicID>": para se inscrever em um tópico , cujo ID é passado como parâmetro, para isto é utilizada a função subscribeToTopic();
- "SAIR”: este comando é utilizado para encerrar a sessão do utilizador.

4. int sendMulticastMessage(Topic *myTopic, char *article): Esta função é responsável por enviar uma mensagem de multicast para todos os utilizadores que se subscreveram num tópico específico, utilizando a função sendto() para enviar a mensagem para o socket de cada inscrito. A função retorna 1 se a mensagem for enviada com sucesso, caso contrário, retorna 0.

5. void jornalistaMenu(int client, User *myUser, TopicList *topicList): Esta função executa o menu de um jornalista. Semelhante ao leitorMenu(), recebe continuamente comandos do cliente e executa as ações correspondentes.

Os comandos disponíveis são:

- "CREATE_TOPIC <topicID> <topicName>": para criar um novo tópico utilizando a função createTopic();
- "SEND_NEWS <topicID> <article>": para enviar notícias para um tópico específico, utilizando a função sendMulticastMessage().
- "SAIR": este comando é utilizado para encerrar a sessão do utilizador.

6. void * myTCPServer(void *args): Esta função é executada em uma thread separada para cada conexão de cliente TCP. Ela lida com a autenticação do cliente, caso a autenticação for bem-sucedida, o cliente entra no leitorMenu() ou jornalistaMenu() dependendo do tipo de utilizador. Após o encerramento do menu, ela termina a conexão.

7. void * bootTCP(void *args): Esta função é executada por uma thread independente, esta aguarda conexões de clientes, aceita-as e cria uma nova thread para lidar com cada utilizador usando a função myTCPServer(). Ela também gere a quantidade de threads ativas para garantir que apenas um certo número de clientes possa estar conectado simultaneamente.

Servidor UDP (UDPserver.c)

1. **void * bootUDP(void *args):** À semelhança de função **bootTCP()**, esta função é executada em uma thread separada. Ela cria um socket para receber pacotes UDP e associa-o a um determinado endereço e porta. Esta aguarda a conexão de novos clientes e aceita-as, após a realização da autenticação de um administrador a função espera por pela receção de comandos do administrador.

Os comandos disponíveis são:

- **"ADD_USER <username> <password> <administrador/cliente/jornalista>":** adiciona um novo utilizador à lista de utilizadores, utilizando a função **createUser()** e a função **insertNode().**
- **"DEL <username>":** Remove um utlizador existente da lista de utilizadores, utilizando a função **deleteUser().**
- **"LIST":** A função mostra uma lista de todos os utilizadores.
- "QUIT ": O comando encerra a sessão do administrador, encerrando assim a consola.
- **"QUIT_SERVER":** O comando encerra o servidor. Antes de encerrar,guarda as informações em um ficheiro de configuração usando **writeTextFileConfig().**

Funções Auxiliares

1. **readTextFileConfig:** Lê informações dos utilizadores de um ficheiro de texto e cria os utilizadores na base nos dados.
2. **createUser:** Cria um novo utilizador.
3. **insertNode**: Insere um novo utilizador na lista de utilizadores.
4. **createUserList:** Inicializa a lista de utilizadores vazia.
5. **deleteUserList:** Elimina a lista de utilizadores e liberta a memória associada.
6. **deleteUser:** Elimina um utilizador da lista de utilizadores.
7. **findUserByUsername:** Procura um utilizador na lista de utilizadores com base no username.
8. **authenticateUser:** Autentica um utilizador.
9. **authenticateAdmin:** Autentica um administrador.
10. **userToString:** Representa um utilizador em string.
11. **writeTextFileConfig:** Escreve a lista de utilizadores em um ficheiro de texto quando o programa é encerrado.
12. **getUserFuncao:** Retorna a função de um utilizador com base no username (administrador/leitor/jornalista).
13. **createTopicList:** Inicializa a lista de tópicos vazia.
14. **isValidID:** Verifica se uma determinada string é um ID válido.
15. **isUniqueID:** Verifica se um ID dado é único dentro de uma lista de tópicos.
16. **createTopicSocket:** Cria um socket multicast para um tópico.
17. **createTopic**: Cria um novo tópico com a informação fornecida.
18. **insertTopic:** Insere um tópico na lista de tópicos.
19. **ipAddressToString:** Representa um endereço IP em string.
20. **findTopicByID** Procura um tópico na lista de tópicos com base em um ID.
21. **topicToString:** Representa um tópico em string.
22. **deleteTopicList:** Elimina a lista de tópicos e liberta a memória associada.

Cliente TCP (clienteTCP.c)

1. void killChildProcesses(): função é utilizada para encerrar todos os processos filhos ao enviar-lhes um sinal SIGINT, após encerrarem fecha o socket e liberta a memória reservada com o malloc.

2. void timeToDie(): esta função é usada para encerrar o processo quando recebe um sinal SIGINT, removendo a associação ao socket multicast e libertando a memória reservada com o malloc.

3. void workerListenToMulticast(char *s): Esta função é chamada quando um caractere '@' é recebido no buffer, esta é executada em um processo filho para lidar com a escuta do multicast. O argumento s contém o endereço IP e o porto multicast separados por ':'. A função fica à escuta de mensagens na socket multicast , imprimindo-as no ecrã.

4. int main(int argc, char *argv[]): Esta é a função principal do programa,é responsável por conectar-se ao servidor especificado nos argumentos da linha de comando e ficar à escuta de mensagens do servidor, imprimindo-as no ecrã. Caso uma mensagem começar com o caracter '@', indica que é necessário criar um processo filho para lidar com o multicast. Esta função termina quando o utilizador insere o comando “SAIR”, chamando a função killChildProcesses() para encerrar os processos filhos e liberar a memória reservada com o malloc.

Conclusão

No decorrer deste projeto, concluímos o desenvolvimento do protótipo de uma aplicação que utiliza os protocolos UDP e TCP para criar um serviço de notícias. Durante o processo, implementamos todas as funcionalidades necessárias para o funcionamento da consola do administrador, da consola do cliente, das estruturas internas com recurso à ferramenta de multithreading, que permite o funcionamento simultâneo de ambos os protocolos, e múltiplos utilizadores. Além disso, configuramos e aprimoramos o circuito no GNS3.

Durante o desenvolvimento do projeto, enfrentamos desafios significativos, como a integração dos protocolos UDP e TCP e o gerenciamento adequado da concorrência por meio do multithreading. Adicionalmente, a configuração do cenário do GNS3 também constituiu um desafio na execução do projeto.

Em termos de contribuição, este projeto nos permitiu aprofundar nosso conhecimento em redes e comunicação, bem como adquirir habilidades práticas no desenvolvimento de aplicações utilizando protocolos de transporte como o UDP e o TCP. Além disso, a experiência de trabalhar em um ambiente simulado no GNS3 nos proporcionou uma compreensão mais profunda das configurações de rede e dos desafios envolvidos na implementação de sistemas distribuídos.