1 2 9 0 FACULDADE DE LETRAS
UNIVERSIDADE D
COIMBRA

Trabalho Prático – Meta Final

Redes de Comunicação

Ana Carolina dos Santos Morais Nº2021222056
Fernanda Margarida Rodrigues Fernandes Nº2021216620

2022/2023

Trabalho Prático – Meta Final

Redes de Comunicação

Dado que na primeira meta foi realizado as configurações necessárias nos dispositivos do GNS3 e ainda implementámos a parte corresponde ao servidor -UDP, agora, para a meta final, foi colocado em prática a execução da parte do servidor -TCP, do cliente – TCP e ainda a colocação do "multicast", dado que é um fator indispensável para o sucesso deste projeto.

Assim, juntamente com este relatório encontra-se um código do servidor, um código do cliente.

Analisando o código do servidor, é de tamanha importância entender quais as estruturas que achámos por bem implementar, desse modo encontra-se uma estrutura com o nome "User", para guardar os dados do utilizador, pois foi criada uma lista de utilizados para facilitar as operações necessárias.

Relativamente ao administrador, dado que é importante guardar o endereço e o porto com intuito de ser mais fácil garantir a saída do mesmo, criámos uma estrutura que vai guardando estes mesmos dados e para sair basta realizar uma comparação para verificar se o porto está na lista de dados e caso esteja é preciso remover, e assim após a saída do administrador caso decida escrever algo no terminal, não será possível realizar essa mesma operação. Só é possível conectar um administrador de cada vez. Uma vez que no exemplo fornecido no enunciado, só está representado um administrador.

Quanto aos tópicos, criámos uma estrutura que guarda a informação necessária relativa a um tópico, o id do tópico, o seu título, o nome do jornalista que o criou, a lista de endereços "multicast", a lista de clientes e a lista de notícias. Assim, é possível ter uma lista de tópicos que vai ser útil para as diversas funções.

Observando a função principal do código do servidor, a "main", podemos analisar que criámos duas "threads", uma para conter a parte do UDP e outra para conter a parte do TCP, assim é possível que ambas as funções correspondentes às "threads" executam ao mesmo tempo e sem ser necessário utilizar uma "shared memory".

Uma das diferenças entre a meta 1 e esta meta é que, anteriormente recorremos a um processo (fork()) em que o processo filho executava a parte

do TCP e o pai, a parte do UDP, esta última parte já se encontrava concluída na primeira meta, daí ser só transformar o processo em duas "threads".

Uma vez que existe uma "thread" TCP, considerámos adequado que, por cada cliente se cria uma nova "thread". Assim a função que esta "thread" desempenha é, digamos, um servidor, pois é aqui que se irá levar a cabo a autenticação de cada cliente (leitor ou jornalista) e, caso seja jornalista, existe um menu, onde cada opção tem uma respetiva função, o mesmo acontece no caso do leitor, mas, claro com funções diferentes, dado que o objetivo para cada cliente é diferente.

Como indica no enunciado, é da funcionalidade do servidor gerar endereços/portas "Multicast", então, com a finalidade de concluir esta tarefa, implementámos duas funções capazes de gerar estes endereços/portas.

Dado que optámos por resolver o exercício utilizando *array* de estruturas, inicializámos uma variável global com tamanho de 100, de forma que uma lista de tópicos pode ter no máximo 100 tópicos e, cada tópico pode ter no máximo 100 clientes e 100 noticias.

Por fim, neste código, temos sempre o cuidado de tratar dos erros de forma a garantir o funcionamento correto do mesmo e até aumentar a robustez, não esquecendo que também podemos afirmar que facilita a depuração.

Dado que já referimos como funciona o servidor - TCP, iremos agora explicar as principais funcionalidades do Cliente – TCP. Como podemos observar no respetivo código, existem algumas estruturas semelhantes ao código do servidor, estas vão nos ser úteis no seguimento do programa.

Basicamente o programa recebe dois argumentos, o endereço do servidor e o número da porta, em seguida, ele cria uma *thread* que tem uma função própria, após isso, o programa aguarda que outras *thread* cliente sejam concluídas. A função que a *thread* executa, mantém a comunicação com o servidor e, tem em atenção que o cliente pode ser do tipo "Leitor" cujas suas funções podem ser: listar tópicos, subscrever um tópico, aqui é criado uma thread para se juntar aos grupos *multicast* outra opção é sair e aqui é removido dos grupos *multicast*. Ou do tipo "Jornalista", cujas funções são: criar um tópico, enviar uma notícia, tal como no tipo "leitor" é aqui que é criado uma *thread* para se juntar aos grupos *multicast* correspondentes aos tópicos de notícias ou sair.

Não podemos esquecer, assim, de fazer a respetiva autenticação e a respetiva verificação de que tipo de cliente é, para depois, apresentar as suas

opções. Cada função tem um comentário acima, que indica a funcionalidade da mesma, de forma a ser de fácil perceção da mesma.

Por último, foi necessário mover cada ficheiro de código, respetivamente, para cada pasta de cada Docker do GNS3, cujas configurações já tinham sido feitas.

Em conclusão, este trabalho permitiu-nos explorar e aplicar diferentes técnicas de comunicação e protocolos da pilha TCP/IP na implementação de um sistema de difusão de notícias. Através das duas fases do trabalho, conseguimos desenvolver funcionalidades tanto para os utilizadores leitores quanto para os jornalistas, demonstrando o conhecimento adquirido e as habilidades de programação em rede.