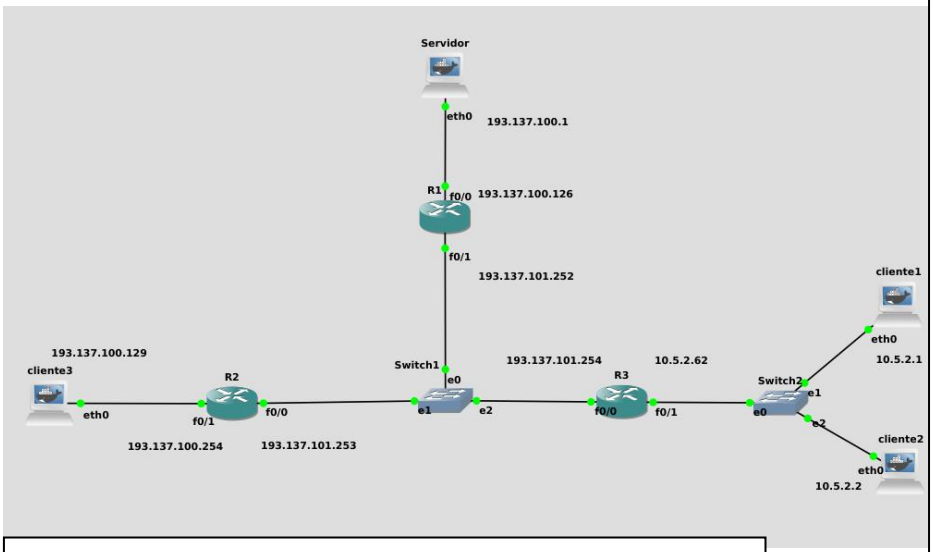
Relatório Projeto RC

Autores:

Alexandre Ferreira 2021236702

João Tinoco 2021223708

Cenário de comunicações:



R3:

ip route 193.137.100.128 255.255.255.128 193.137.101.253

ip route 193.137.100.0 255.255.255.128 193.137.101.252

R3: config terminal

access-list 30 permit 10.5.2.0 0.0.0.63

ip nat inside source list 30 interface FastEthernet0/0 overload

interface FastEthernet0/1

ip nat inside

no shutdown

exit

interface FastEthernet0/0

ip nat outside

no shutdown

exit

end

Figura 1: Print screen do cenário criado para a aplicação no GNS3

Modo de Funcionamento:

Primeiramente, o programa do servidor (“**server.c”**) deve ser iniciado, para que as comunicações sejam efetuadas com sucesso. Ao iniciar este programa, devem ser dados como argumentos, o Porto de Notícias ao qual os clientes e servidor irão esperar por mensagens TCP, o Porto de Configurações, este que será o Porto destinado para a receção e envio de mensagens UDP, e por fim, o ficheiro de configurações.

De seguida, é lido do ficheiro todas as contas de utilizadores existentes e armazenada essa informação num array situado na memória partilhada ed maneira a que todas as entidades constituintes do programa servidor sejam capazes de usufruir e alterar os mesmos dados.

Posteriormente, dá-se início à criação de processos filhos responsáveis por tratarem de cada tipo de comunicação. O programa não termina enquanto estes processos não terminarem a sua tarefa.

De forma a ser possível existir a consola do administrador, é criada a função “comunicação_udp()”. Esta recebe como argumento o ponteiro para o ficheiro de configurações, possibilitando a atualização de informação contida no mesmo.

É criado um socket que é configurado para utilizar o protocolo UDP (SOCK_DGRAM) e o endereço de internet (AF_INET). É preenchida uma estrutura si_minha do tipo struct sockaddr_in

com as informações do endereço de IP e porta em que o servidor vai receber os pacotes. A porta é definida como PORT_CONFIG, e o endereço de IP é configurado para qualquer endereço disponível na máquina local (INADDR_ANY). O socket é associado ao endereço especificado na estrutura si_minha usando a função bind().

A partir deste momento, entra-se em ciclos que tratam tanto da autenticação como dos pedidos, na sua correta forma.

Para o administrador poder enviar os seus pedidos e receber as informações pretendidas, concebeu-se um programa próprio ("**udp_client.c**"), onde é criado um socket UDP que é configurado para utilizar o protocolo UDP (SOCK_DGRAM) e o endereço de internet (AF_INET).

A estrutura si_servidor é configurada com o endereço IP e porta do servidor, obtidos através dos argumentos recebidos do arranque do programa. São utilizados ciclos novamente, para tratar do envio de comandos e receção de respostas do servidor. Os ciclos referidos anteriormente apenas são terminados quando o administrador digita "QUIT" ou "QUIT_SERVER". Caso seja "QUIT_SERVER", todo o servidor é desligado e fechado todos os sockets criados até ao momento. Este utilizador, tem permissão para listar todas as contas existentes na aplicação ("LIST"), como também, remover e adicionar um utilizador ("ADD_USER" E "DEL").

Todavia, de modo a fornecer capacidade para responder a clientes que se comunicam por TCP, foi criado uma função comunicação_tcp(). Esta aceita conexões de clientes e cria um processo filho para lidar com cada conexão. São utilizadas as variáveis fd (descritor de arquivo do socket do servidor) e client (descritor de arquivo do socket do cliente), bem como as estruturas addr (endereço do servidor) e client_addr (endereço do cliente). A estrutura addr é inicializada com as informações do endereço do servidor, incluindo o IP (INADDR_ANY) e a porta (PORT_NOTICIAS).

Em seguida, é criado um socket TCP usando a função socket() e é associado ao endereço do servidor usando a função bind(). O servidor aguarda por uma nova conexão usando a função accept(). Quando uma nova conexão é estabelecida, é criado um novo socket para essa conexão e o descritor de arquivo é armazenado em client. O endereço do cliente é armazenado em client_addr. Se a conexão foi estabelecida com sucesso, é criado um processo filho usando a função fork(), sendo responsável por processar a conexão chamando a função process_client().

No programa do cliente ("**cliente_tcp.c**"), sendo este usado pelos clientes da aplicação, é criado um socket TCP em que o seu descritor de arquivo é armazenado em fd. A família de endereços é especificada como AF_INET, o tipo de socket é definido como SOCK_STREAM (TCP).

A estrutura server_addr é preenchida com as informações do endereço do servidor e o cliente conecta-se ao servidor usando a função connect().

Neste momento, estando os dois a funcionar corretamente, é possível existir a comunicação TCP entre o servidor e cliente. Iniciam-se ciclos para que a autenticação e tratamentos de pedidos sejam realizados com sucesso. Se o cliente for leitor, caso tenha permissão, irá ter a capacidade de listar os tópicos existentes ("LIST_TOPICS"), listar todas a notícias recebidas dos tópicos que está associado ("LIST_NEWS") e subscrever qualquer tópico ("SUBSCRIBE_TOPIC {id do tópico}"), de forma a que este receba do servidor o endereço

multicast através do qual poderá passar a receber notícias. Se for jornalista, tem permissão para listar os tópicos, criar um tópico (“CREATE_TOPIC {id} {titulo}”) como também enviar para o grupo multicast correspondente ao tópico, notícias para todos os clientes subscritos nesse tópico (SEND_NEWS {id} {mensagem}”).

Os clientes podem escolher sair do seu programa através do comando “QUIT”. Contudo o seu programa pode ser terminado através do pedido do administrador para desligar o servidor.

Opções Tomadas:

Ao longo da construção do trabalho, foram-se construindo várias estratégias para o correto funcionamento da aplicação.

De modo a que o servidor seja capaz de lidar com ambas as comunicações, foram criados dois processos filhos encarregues de utilizar funções que tratam de cada comunicação. Consequentemente, para ser possível partilha de dados entre processos, criou-se memória partilhada, que contém informação das contas dos utilizadores, de todos os tópicos existentes, dos portos e endereços já utilizados e algumas variáveis auxiliares.

Cada utilizador tem um nome, password, tipo e um array de inteiros com todos os id de tópicos correspondentes aos grupos que pertence do multicast. Cada tópico tem um id, endereço único, porto único e uma matriz que armazena todas as notícias criadas naquele tópico, dando a possibilidade do cliente listá-las sempre que desejar. Os endereços e portos únicos de cada tópico são gerados através de funções auxiliares (“gerarPortoUnico()” e “generate_multicast_ip()”), sendo que o porto gerado pode ser de 1024 a 5000, evitando as portas especiais localizadas abaixo de 1024, e o endereço gerado pode abranger endereços entre 224.0.0.0 e 239.255.255.255, estes que são exclusivos para grupos multicast.

De forma a que os leitores pudessem receber no momento exato as notícias que são publicadas dos tópicos que estão subscritos, e para além disso continuarem a ter a capacidade de digitar comandos, assim que o leitor digita o comando de subscrever um tópico, é criado um processo filho que trata da entrada do utilizador no grupo multicast pretendido e a constante receção de novas mensagens do mesmo. O código da função utilizada pelo processo filho configura um socket UDP para se juntar a um grupo multicast, definindo o endereço e a porta do socket, e utilizando a função setsockopt() para associá-lo ao grupo multicast especificado.

Como implementação extra, para que o utilizador possa ter a possibilidade de ter uma password que contém o caracter “;”, criou-se uma função auxiliar que trata dos ficheiros de configuração corretamente (“pos_ocorrencias”). Por fim, uma vez que o netcat não estava a funcionar corretamente, optou-se por criar um programa próprio para o administrador, oferecendo assim uma melhor estruturação e viabilidade da aplicação.