UFPE

Projeto de Desenvolvimento com Sockets — UDP
Quiz Competitivo (Protocolo de Comunicação)

Equipe: Aslay Clevisson Soares Santos
Marcson Silva dos Santos

Características Gerais do Protocolo UDP

O protocolo UDP é um protocolo de camada 4 (de transporte) no modelo OSI, que se caracteriza por ser mais simples que o TCP ou outro protocolo da camada 4. Enquanto o TCP se preocupa com a conexão e a chegada correta dos dados no destino, o UDP por ser mais simples não tem a mesma preocupação, portanto, ele não verifica o recebimento dos dados pelo destino (também não possui o serviço de reenvio), não ordena as mensagens, ou seja, elas vão sendo agrupadas conforme vão chegando, não controla o fluxo de informações e não verifica a integridade dos dados para o destino. As possibilidades de o destino não receber os dados são várias, como por exemplo: perder os dados, duplicar os dados ou agrupar de forma errada.

Essa simplicidade do UDP pode parecer, à primeira vista, um pouco estranha e provavelmente um leigo diria: se existe um protocolo como o TCP, que garante a chegada correta dos dados no destino, para que usarmos o UDP? A resposta é fácil, porque se o protocolo é simples, ele também é menor, então devemos ver isto como ganho de velocidade na transmissão e recepção de dados, algo que nos dias atuais se torna cada vez mais importante. É claro que em muitas das vezes o pacote pode não chegar ao destino, mas também devemos avaliar que só valerá a pena enviar pacotes utilizando o UDP quando este for pequeno, neste caso menor que 512KB. Ou seja, não será em uma transferência de arquivos, como em um download usual, que será recomendado a utilização do UDP como “meio de transporte”. Uma outra característica importante do UDP e neste ponto, semelhante ao TCP é que ele se baseia em portas para a troca de informações, desta forma, é atribuída uma porta ao destino e uma porta à origem.

Cliente

O código começa com a importação do socket seguido da threading que permite que as diferentes partes do programa funcionem simultaneamente e, por fim, o **inputimeout** e **TimeoutOccurred** que possui a finalidade de determinar o tempo limite para que uma ação seja tomada.

```
from socket import socket, AF_INET, SOCK_DGRAM
from threading import Thread
from inputimeout import inputimeout, TimeoutOccurred
```

Após as bibliotecas, dando início ao código, o socket cliente é criado juntamente com a definição da quantidade de clientes que foi delimitada em 5. A seguir, é possível dar início ao cadastro dos jogadores, sempre que um jogador realiza o cadastro, a mensagem é enviada para o servidor, onde os dados do mesmo **(nome, endereço e porta)** são armazenados para o andamento da partida.

```
iniciar = True
socket_cliente = socket(AF_INET, SOCK_DGRAM)

qtd_clientes = 5

partida_iniciada = []
while iniciar:
    print()
    mensagem = input("Digite o seu login para se conectar: ")
    if mensagem ≠ "":
        mensagem_codificada = mensagem.encode()
        socket_cliente.sendto(mensagem_codificada, ("localhost", 9090))
        resposta_servidor = socket_cliente.recvfrom(1024)

        if partida_iniciada == []:
            cont = 0
            partida_iniciada.append(cont)

        if str(resposta_servidor[0].decode()) == "101":
            print(f"O(A) jogadoror(a) {mensagem} foi Cadastrado(a).")
```

Em seguida a Thread é executada, fazendo com que o jogador receba a informação do servidor que ele foi cadastrado e agora deve aguardar os demais jogadores. Isso ocorre porque dentro da função **esperar(partida_iniciada)** é chamada a função **responder()** que aguarda as respostas do servidor para ser iniciada.

```
iniciar = False

partida_iniciada[0] += qtd_clientes

if partida_iniciada[0] <= qtd_clientes:
    Thread(target=esperar, args=(partida_iniciada)).start()
```

```
def esperar(partida_iniciada):
    print()

    if partida_iniciada == qtd_clientes:
        print("Aguardando Jogadores ... \r\n")
        resposta_servidor = socket_cliente.recvfrom(1024)

        print(str(resposta_servidor[0].decode()))
        print()
        responder()
```

Finalizando a etapa de cadastro dos jogadores, implementamos dois trechos de código, sendo o primeiro, responsável por negar o acesso caso um jogador tente entrar na partida enquanto a mesma se encontra em andamento.

```
else:

    if resposta_servidor[0].decode() == "410":  # negação de acesso
        print("\nPartida em andamento, tente novamente mais tarde.\n")
        socket_cliente.close()
        iniciar = False
```

O segundo trecho tem a função de não permitir que um jogador se cadastre com o nome vazio.

```
else:
    protocolo_erro = "401"
    print("Nome inválido, tente novamente.  \r\n")
    print(f"Negação de acesso {protocolo_erro}.  \r\n")
```

Após todos os jogadores terem se cadastrado, o servidor começa a enviar as perguntas aos jogadores e os mesmos tem **10** segundos para para responder, após isso, o servidor irá enviar uma mensagem relatando o acerto ou o erro do jogador em relação a pergunta da rodada. Quando o servidor enviar a resposta **"500"** para todos os jogadores, indica que a partida acabou de ser finalizada, logo após, a função **ranking(totalJogadores)** é chamada para imprimir o ranking de todos os jogadores que participaram da partida.

```
def responder():

    resposta_servidor = socket_cliente.recvfrom(1024)

    if str(resposta_servidor[0].decode()) != "500":
        print(str(resposta_servidor[0].decode()))
        try:
            mensagem = inputimeout(prompt="Digite sua resposta: ", timeout=10)
        except TimeoutOccurred:
            print("Tempo esgotado.")
            mensagem = "Sem nenhuma resposta"

        mensagem_codificada = mensagem.encode()
        socket_cliente.sendto(mensagem_codificada, ("localhost", 9090))
        print("Resposta enviada ... \r\n")
        resposta_servidor = socket_cliente.recvfrom(1024)
        print(str(resposta_servidor[0].decode()))
        responder()

    else:

        print("\nFim de jogo. \n")
        resposta_servidor = socket_cliente.recvfrom(1024)
        totalJogadores = int(resposta_servidor[0].decode())

        ranking(totalJogadores)


def ranking(totalJogadores):

    print("Ranking")
    for _ in range(totalJogadores):  # testando 2 cliente
        resposta_servidor = socket_cliente.recvfrom(1024)
        print(str(resposta_servidor[0].decode()))
    socket_cliente.close()
```

Servidor

Neste trecho de código, o servidor é iniciado e após a sua inicialização é feita a leitura do arquivo que contém as perguntas e repostas do quiz e em seguida uma lista (**sub_lista_n_pergunta = []**) recebe de forma aleatória **20** valores, essa lista será usada dentro da função **pergutaResposta(participantes, perguntas_e_respostas, valor, contador, indicesP)**.

```
socket_servidor = socket(AF_INET, SOCK_DGRAM)
socket_servidor.bind(("localhost", 9090))
perguntas_e_respostas = ler_arquivo()

# implementação randomificação
sub_lista_n_pergunta = []
while len(sub_lista_n_pergunta) < 20:
    indice_aleatorio = random.randint(0, 19)

    if indice_aleatorio not in sub_lista_n_pergunta:
        sub_lista_n_pergunta.append(indice_aleatorio)
```

Após a inicialização do servidor e a leitura do arquivo, o programa checa se a quantidade de jogadores é menor que **5,** se essa afirmação for verdadeira, o servidor aguardará a entrada de até no máximo **5** jogadores por **20** segundos, caso nenhum jogador entre nesse período de tempo, o servidor é fechado, porém, se pelo menos **1** jogador estiver cadastrado (quando um jogador se cadastrar o servidor envia uma mensagem informando que o mesmo foi cadastrado), o servidor iniciará a partida após o término dos **20** segundos chamando a **Thread(target=iniciarPerguntas, args=(mensagem_start, participantes, valor)).start()**, assim, iniciando a partida com os jogadores que estão na sala.

```python
contador_indice_pergunta = 0
conexao_start = True
valor = 0
participantes = {}
qtd_clientes = 0
max_jogadores = 5
min_jogadores = 1
aceitar_mais_jogadores = True

while conexao_start and len(participantes) < max_jogadores and aceitar_mais_jogadores == True:

    print()
    try:
        print("Aguardando requisições ... \r\n")
        socket_servidor.settimeout(20) #tempo para incrição de participantes

        mensagem_cliente, endereco_cliente = socket_servidor.recvfrom(1024)
        participantes[endereco_cliente] = [mensagem_cliente, 0]
        print(f"O/A participante {mensagem_cliente.decode()} entrou")
        resposta = "101"
        resposta_cliente = str.encode(resposta)
        socket_servidor.sendto(resposta_cliente, endereco_cliente)
        print("Resposta enviada para o/a participante \r\n")
    except:
        qtd_clientes = len(participantes)
        aceitar_mais_jogadores = False

    qtd_clientes = len(participantes)


if len(participantes) == qtd_clientes and len(participantes) >= min_jogadores:

    mensagem_start = "Quiz de conhecimentos gerais comecou!"

    Thread(target=iniciarPerguntas, args=(mensagem_start, participantes, valor)).start()

else:
    print('Não há jogadores suficientes para começar a partida.')
    socket_servidor.close()
```

Após o chamado da **Thread,** a função **iniciarPerguntas(mensagem_cliente, participantes, valor)** é iniciada e o servidor enviar a mensagem para todos participantes informando que a partida foi iniciada e após o envio da mensagem, a função **pergutaResposta(participantes,perguntas_e_respostas,valor, contador_indice_pergunta, sub_lista_n_pergunta)** é iniciada.

```
def iniciarPerguntas(mensagem_cliente, participantes, valor):

    for endereco in participantes.keys():
        socket_servidor.sendto(str.encode(mensagem_cliente), (endereco))
        print("A pergunta foi enviada aos jogadores \n")

    pergutaResposta(participantes, perguntas_e_respostas,
                valor, contador_indice_pergunta, sub_lista_n_pergunta)
```

Com a execução da função, as perguntas são enviadas ao cliente de forma aleatória de acordo com com os índices da **sub_lista_n_pergunta** que agora seu nome é **IndicesP**.

```
def pergutaResposta(participantes, perguntas_e_respostas, valor, contador, indicesP):

    for endereco in participantes:
        pergunta = str.encode(
            perguntas_e_respostas[indicesP[contador]][0])
        socket_servidor.sendto(pergunta, (endereco))
```

Após o envio das perguntas o servidor aguarda a resposta dos jogadores, quando um jogador responde, o servidor imprime em sua tela, a mensagem que o jogador enviou. Além disso, se algum jogador tentar se conectar ao servidor durante a partida, o servidor não deixará o mesmo se conectar e enviará uma resposta de negação **“410”** para o cliente.

```
qtd_msg = 0
dic_resposta = {}
while qtd_msg < qtd_clientes:

    mensagem_cliente, endereco_cliente = socket_servidor.recvfrom(1024)
    if endereco_cliente in participantes.keys():
        dic_resposta[endereco_cliente] = mensagem_cliente.decode()
        print(f"MSG: {mensagem_cliente.decode()} do(a) jogador(a) {participantes[endereco_cliente][0].decode()}")
        qtd_msg += 1

    else:
        resposta_negação = "410"
        print(">>>Jogador tentando se conetar, mas foi recusado<<<")
        socket_servidor.sendto(resposta_negação.encode(), (endereco_cliente))
```

Quando o jogador responde a pergunta e ainda está dentro do tempo limite ele fica esperando a resposta do servidor confirmando se ele acertou, errou ou não respondeu a pergunta, caso o jogador acerte ele soma **25** pontos, se errar subtrai **-5** pontos e se caso o tempo for esgotado e o mesmo não tiver respondido subtrai **-1** ponto.

```
for k, v in dic_resposta.items():
    if v == perguntas_e_respostas[indicesP[contador]][1]:

        print(f"O(A) jogador(a) {participantes[k][0].decode()} acertou a resposta")
        participantes[k][1] += 25
        resposta_1_cliente = str.encode(f"Parabéns! Você acertou a resposta e ganhou 25 pontos.
                                        Sua pontuação atual é {participantes[k][1]}")

        socket_servidor.sendto(resposta_1_cliente, (k))
    else:
        if v == "nao respondeu":
            participantes[k][1] -= 1
        else:
            participantes[k][1] -= 5

        print(
            f"O(A) jogador(a) {participantes[k][0].decode()} errou a resposta")

        resposta_1_cliente = str.encode(
            f"Infelizmente, você errou a resposta e perdeu 5 pontos. Sua pontuação atual é {participantes[k][1]}\n")
        socket_servidor.sendto(resposta_1_cliente, (k))
```

A quantidade de perguntas é delimitada pelo seguinte trecho de código, no qual, enquanto as variáveis **valor** e **contador** não forem iguais a **5**, a **Thread** será chamada novamente, fazendo com que o servidor envie a próxima pergunta para os jogadores.

```
valor += 1
contador += 1

if valor != 5 and contador != 5:  #Definição de quantidade de iniciarPerguntas

    Thread(target=pergutaResposta, args=(participantes,
           perguntas_e_respostas, valor, contador, indicesP)).start()
```

Quando as variáveis “**valor**” e “**contador**" forem iguais a **5,** o servidor envia para todos os clientes a resposta/mensagem **“500” e** quando todos clientes recebem a mensagem, a partida é finalizada.

```
resposta = "500"
resposta_cliente = str.encode(resposta)
for x, y in dic_resposta.items():
    socket_servidor.sendto(resposta_cliente, x)
```

Com o fim da partida, o servidor organiza o ranking em ordem decrescente, com a pontuação dos jogadores participantes e, para isso, é usado o algoritmo **quicksort** para ordenar o ranking e, logo após de ordenado, o servidor envia envia as informações para todos os jogadores e, por fim, o servidor fecha.

```python
def quicksort(vetor, indiceinicial=0, indiceparada=None):
    if indiceparada == None:
        indiceparada = len(vetor) - 1

    if indiceinicial < indiceparada:
        pivor = particao(vetor, indiceinicial, indiceparada)
        quicksort(vetor, indiceinicial, pivor - 1)
        quicksort(vetor, pivor + 1, indiceparada)


def particao(vetor, indiceinicial, indiceparada):
    indice_de_inicio = indiceinicial
    pivor = vetor[indiceparada][1]
    for i in range(indiceinicial, indiceparada):
        if vetor[i][1] >= pivor:
            vetor[indice_de_inicio], vetor[i] = vetor[i], vetor[indice_de_inicio]
            indice_de_inicio += 1

    else:
        vetor[indice_de_inicio], vetor[indiceparada] = vetor[indiceparada], vetor[indice_de_inicio]

    return indice_de_inicio
```

```python
listao = []
for x in participantes.values():
    listao.append(x)

ordena_listao = quicksort(listao)

totalJogadores = str(qtd_clientes).encode()
for p in participantes.keys():
    socket_servidor.sendto(totalJogadores, p)

for x in listao:
    for y in participantes.keys():
        classificacao = str.encode(
            f"A pontuação do(a) jogardor(a) {x[0].decode()} foi de: {x[1]} pontos.")
        socket_servidor.sendto(classificacao, y)
socket_servidor.close()
```