Nome: Wesley Axel de Barros **Ra: 11201722424**

UFABC – Universidade Federal do ABC – Primeiro quadrimestre 2020.

Redes de Computadores - versão ECE

Exercício programático em Java.

Introdução:

O exercício programático em Java da disciplina de redes de computadores versão ECE, do primeiro quadrimestre de 2020, propõe que construíssemos um protocolo com paralelismo de transporte confiável, executado acima do UDP, implementando os conceitos de go-back-n ou de repetição seletiva. Este protocolo seria utilizado para enviar mensagens de um computador para outro simulando um servidor e um cliente, executados em maquinas diferentes, com diferentes tratativas para as mensagens que podem ser enviadas normais, ou com atrasos, duplicidades, percas ou fora de ordem.

Formato da mensagem e explicação básica do programa:

Com esta proposta, desenvolvi esse protocolo utilizando o conceito de **go-back-n n**a linguagem **Java**. O protocolo se baseia em um cliente e um servidor, sendo o servidor responsável por apenas receber as mensagens e o cliente apenas enviar.

Este protocolo desenvolvido, se faz a transmissão de pacotes através do UDP, e está programado para enviar mensagens em duas maquinas com I.P. diferentes em uma **rede local**. Os testes foram realizados em duas maquinas diferentes cada uma com o arquivo servidor ou cliente compilado e executado. Se tentar executar em uma mesma máquina o servidor e o cliente com o mesmo I.P. o seguinte erro é apresentado: "**Erro Address already in use: Cannot bind**".

O que é enviado, a "mensagem", se trata de uma String de texto digitada no cliente pelo usuário, que é destrinchada em cada caractere de texto, e armazenado cada um dos caractere em um objeto chamado "pacote" que possui o caractere que será enviado, um número de sequência(gerado aleatoriamente em um número inteiro de 0 a 100) e se o mesmo é o último da mensagem ou não. Sendo esses pacotes os que serão serializados em bytes e enviados através do protocolo. Para armazenamento em buffer desses pacotes, os mesmos são armazenados em um array de até 1000 objetos de classe "pacote". Ou seja, fazendo com que a mensagem tenha limite de tamanho de 1000 caracteres.

A mensagem foi escolhida dessa maneira, sendo dividida em caracteres e cada caractere sendo um pacote diferente, para termos uma quantidade significativa de pacotes para podermos simular com mais facilidade, as características marcantes do protocolo UDP, que são as percas de pacote, pacotes fora de ordem, e etc.

Se tratando do cliente, o mesmo começa perguntando o I.P do servidor e a porta utilizada para fazer a conexão UDP que será utilizada para enviar a mensagem, após informado, no console é impresso uma tabela de opções de mensagens que podem ser enviadas, junto com a opção de finalizar o programa e fechar o mesmo. As opções são para mensagem normal, mensagem lenta, mensagem perdida, mensagem desorganizada e mensagem duplicada (Será explicado depois cada tratativa). Caso aconteça qualquer erro ao enviar a mensagem, será lançado uma exceção detalhando o problema na conexão do socket do servidor.

Se tratando do servidor, o mesmo começa perguntando qual a porta que será iniciado uma conexão no socket através do protocolo UDP com a porta perguntada e o IP local da máquina. Caso não seja possível iniciar a conexão no socket, é informado o problema. Se o mesmo conseguir abrir a conexão no socket, é criado um buffer com limite de 1000 caracteres, e o mesmo fica em aguardo até chegar qualquer protocolo neste socket. Ao chegar qualquer pacote é começado o recebimento e armazenamento dos pacotes no buffer e o envio dos pacotes ACK de confirmação de chegada de pacotes. É realizado uma tratativa nesses pacotes, para checar duplicidade ou se estão fora de ordem. Feito a checagem do último pacote, os pacotes são retirados do buffer e a mensagem é reconstruída e impressa no terminal, e é perguntado se o servidor deve finalizar ou limpar o buffer e começar novamente o recebimento de mensagens.

Explicação em alto nível das mensagens enviadas:

Temos a classe pacote, que é um objeto que contém 3 atributos, um char data, um inteiro que é o número de sequência e um booleano que indica se o objeto iniciado é o último no array de buffer.

Mensagens normais:

Para mensagens normais, na classe **cliente**, é criado um array de objetos da classe pacote através de um método **criaPacote(),** que inicia uma String e scannea a String digitada pelo usuário. Essa String terá que ter menos 1000 caracteres, se não o processo não continua até o usuário digitar algo com menos de 1000 caracteres. Com a String digitada e pronta, é criado um array de objetos pacote, e iniciados cada um com um número de sequência, sendo o primeiro número de sequência do primeiro pacote um número gerado aleatoriamente entre 1 e 100, e um caractere da String digitada. O último pacote recebera a flag boolean de último pacote do array. Este array será devolvida ao método principal, e flag necessárias para o envio serão iniciadas com os valores dependendo do buffer criado. Sendo elas o ultimoseq (Último número de sequência enviado), ackesperado(Pacote ack esperado após o envio do mesmo), essas duas com o valor do primeiro número de sequência do array no buffer, e o ultimopossivel(Último número de sequência possível), iniciado com o valor do último número de sequência do array para determinar o termino do envio. É iniciado dois inteiros, “i” e “u”, com valor zero, que serão incrementados conforme o envio dos pacotes e recebimento de pacotes ack.

É criado um laço que será quebrado no termino de envio dos pacotes. Dentro desse laço existe um outro, que é o laço principal de envio, que fica em um loop de envio, determinado por duas condições, se o valor do ultimoseq – ackesperado for menor que o tamanho da janela (valor inteiro para determinar quantidade de pacotes enviados antes do recebimento do primeiro ack) e se o contador i, que percorrera o índice do array no buffer, for menor que o tamanho do buffer. Dentro deste laço, é serializado em bytes um objeto pacote no array de pacotes que é o buffer em bytes, e após isso criado um DatagramPacket, que é um pacote UDP em formato java que será enviado através do DatagramSocket. Com isso, o DatagramPacket é enviado, a flag ultimoseq é incrementada em um, e o contador i em um também. Para garantir que a ordem dos pacotes seja respeitada ao chegar no servidor, foi adicionado um temporizador entre os envios. Na rede utilizada em casa, foi necessário determinar em 2 segundos. Testei em outra rede, e com valores menores, a ordem era respeitada.

Fora do laço de envio principal, é criado um array de bites vazios de tamanho 83 bytes, que é o tamanho máximo que um pacote ack enviado possa ter, e um DatagramPacket, para o pacote ack de confirmação de envio de pacote. Logo dentro de uma estrutura try, é criado um temporizador para receber o pacote ack de tempo determinado em 1000 ms. Se o pacote ack for recebido com

sucesso, o mesmo é deserializado de bytes em um objeto Pacote. Se o valor do número de sequência desse pacote ack for igual a flag de ackesperado, o índice de ackesperado aumenta e o contador u aumenta. Caso o valor do número de sequência desse pacote ack seja igual ao ultimopossivel, é lançado um comando break para sair do laço principal de envio. Caso o temporizador de recebimento do pacote ack se esgote, é lançado uma exceção, e todos os pacotes antes do último pacote ack com valor de número de sequência recebido, através de um laço for, que irá serializar e enviar os pacotes respeitando essa condição.

Na classe **servidor**, após o mesmo ter iniciado seu DatagramSocket com sucesso com a porta indicada pelo usuário e o ip local da máquina, é criado e iniciado um array de objetos Pacote, com valores nulos para servir como buffer, uma flag booleana finaliza, que determinara quando o laço seguinte terá que terminar, e dois inteiros, um contador "i" e "primeiroseq" com valor 0, após isso é iniciado um laço que irá receber os pacotes. Dentro desse laço, é criado um array de bytes para armazenar o pacote que estará chegando, e após isso criado um DatagramPacket que ficara no aguardo para receber o pacote UDP enviado pelo cliente, que será armazenado no array de bytes criado anteriormente. Após isso é criado uma estrutura try, que irá verificar se o pacote é o primeiro a ser recebido e imprimira uma mensagem de aguardo, e caso não seja, será iniciado um temporizador de 10 segundos de recebimento em cada pacote. Após isso o comando do Socket de recebimento é lançado e o pacote recebido, o mesmo é deserializado de bytes em um objeto pacote, e caso seja o primeiro, vai ser armazenado no buffer sem verificar duplicidade e o inteiro primeiroseq recebera o valor de sequência desse pacote e o contador "i" incrementado em um. Caso não seja o primeiro pacote, o mesmo terá seu número de sequência analisado em duas condições, se o mesmo é maior que o número de sequência armazenado anteriormente ao que ele será armazenado através do contador "i" e se o mesmo é igual ao valor esperado do número de sequência armazenado anteriormente, que é o mesmo incrementado em um. Caso o número de sequência passe nessas condições, o pacote é armazenado no buffer, o contador "i" incrementado em um e caso o mesmo tenha sua flag boolena UltimoPacote definida com valor true, a flag "finaliza" é definida com valor true. Caso esse número de sequência, não passes nessas condições, o mesmo é verificado se é igual ao valor do primeiro número de sequência armazenado no inteiro "primeiroseq". Caso seja é impresso uma mensagem de **possível mensagem duplicada.** Caso o número de sequência do pacote não passe por essas condições, é impresso uma mensagem de **possível mensagem fora de ordem**. Independente se o pacote foi armazenado ou não, no processo de envio de pacotes ack, é iniciado um objeto pacote ack, com apenas o número de sequência do pacote recebido. Esse pacote ack é serializado em bytes, determinado em um DatagramPacket, e enviado de volta para o cliente. Após isso, caso a flag "finaliza" tenha valor true, é impresso a mensagem de conexão finalizada, o buffer é passado para um método chamado **reformulaString()**, que utiliza o String Builder com o array de caracteres obtido no buffer para formar uma String que é a mensagem passada, e devolve essa String para essa ser impressa novamente.

Mensagens Lentas

Para as mensagens lentas, o programa funciona semelhante ao modo de mandar mensagens normais com algumas diferenças:

No cliente, no laço principal de envio com a janela deslizante, é uma condicional por probabilidade aleatório, através de uma variável double, que recebe um valor aleatório entre 0 e 100, indicando que se o valor for maior que 10, o processo de envio do pacote é normal, e se for menor que 10, será adicionado um temporizador de 10 segundos, que apenas após esses 10 segundos, que o pacote será enviado.

No servidor, como já tratado na mensagem normal, há uma exceção a ser lançada caso o pacote não seja o primeiro e demore mais de 10 segundos para chegar, que irá imprimir que um possível há um possível pacote lento.

Mensagens perdidas:

Para as mensagens perdidas, o programa funciona semelhante ao modo de mandar mensagens normais com algumas diferenças:

No cliente, no laço principal de envio com a janela deslizante, é uma condicional por probabilidade aleatório, através de uma variável double, que recebe um valor aleatório entre 0 e 100, indicando que se o valor for maior que 10, o processo de envio do pacote é normal, e se for menor que 10, o pacote com o índice "i" no momento, não será enviado, e os contadores "ultimoseq" e "i" não serão incrementados em um.

No servidor, como não há pacote a ser recebido, o mesmo ficara aguardando o próximo.

Mensagens fora de ordem:

Para as mensagens fora de ordem, o programa funciona semelhante ao modo de mandar mensagens normais com algumas diferenças:

No cliente, no laço principal de envio com a janela deslizante, é uma condicional por probabilidade aleatório, através de uma variável double, que recebe um valor aleatório entre 0 e 100, indicando que se o valor for maior que 10, o processo de envio do pacote é normal, e se for menor que 10, um pacote com índice de uma variável int rand, que recebera um valor aleatório entre 0 e o tamanho máximo do array de buffer, será enviado para o servidor e os contadores "ultimoseq" e "i" não serão incrementados em um.

No servidor, como o pacote é aleatório, ele entre nas 3 possíveis condições; com o seu número de sequência, se o mesmo é maior que o número de sequência armazenado anteriormente ao que ele será armazenado através do contador "i" e se o mesmo é igual ao valor esperado do número de sequência armazenado anteriormente, que é o mesmo incrementado em um, o pacote é armazenado no buffer, o contador "i" incrementado em um e caso o mesmo tenha sua flag boolean UltimoPacote como true, a flag "finaliza" é definida com valor true. Caso esse número de sequência, não passes nessas condições, o mesmo é verificado se é igual ao valor do primeiro número de sequência armazenado no inteiro "primeiroseq". Caso seja é impresso uma mensagem de **possível mensagem duplicada.** Caso o número de sequência do pacote não passe por essas condições, é impresso uma mensagem de **possível mensagem fora de ordem**.

Mensagens duplicadas:

Para as mensagens duplicadas, o programa funciona semelhante ao modo de mandar mensagens normais com algumas diferenças:

No cliente, o laço principal de enviar pacotes entra de um outro laço "for" de contador 2, que irá repetir o processo principal de enviar os pacotes duas vezes. Porém antes dos processos e após a criação do pacote, o ultimi pacote tem seu atributo boolean "Ultimo" definido como false para o primeiro envio, e para o segundo envio, definido como true, para a finalização de envio.

No servidor, como o primeiro pacote será enviado novamente, o mesmo irá imprimir que há uma possível mensagem duplicada a ser recebida.

Explicação em alto nível do consumo de buffer:

O buffer no protocolo desenvolvido, se trata de um array de objetos da classe “pacote”, que é criado após a abertura com sucesso do DatagramSocket, e o mesmo é preenchido conforme o recebimento de pacotes e esvaziado conforme o último pacote seja recebido e a mensagem exibida. Tem limite de 1000 objetos de classe “pacote”, ou seja, a mensagem transmitida terá limite de 1000 caractere.

Como compilar:

Arquivos do projeto:

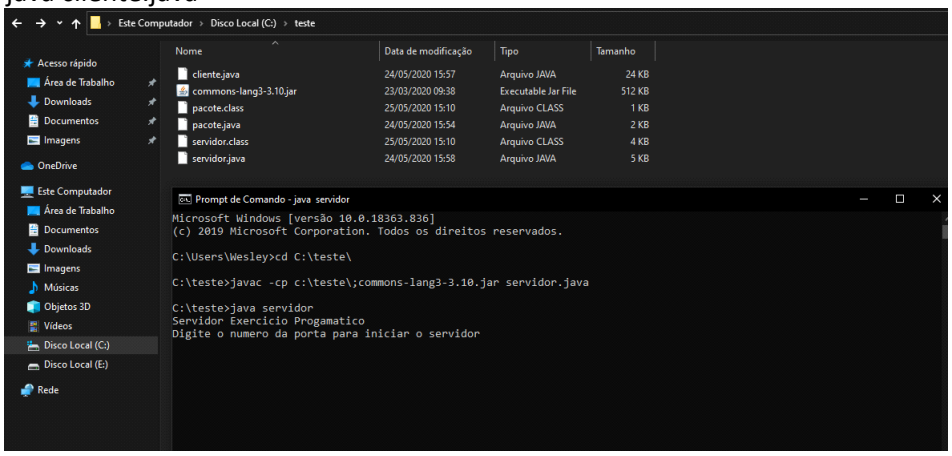
- servidor.java
- cliente.java
- Pacote.java
- commons-lang3-3.10.jar

Tendo todos os arquivos juntos dentro de uma pasta, é necessário lançar os seguintes comandos no terminal:

- javac -cp *Diretório da pasta*\;commons-lang3-3.10.jar servidor.java
- javac -cp *Diretório da pasta*\;commons-lang3-3.10.jar cliente.java

E para executar os mesmos

- java servidor.java
- java cliente.java



Referencias:

https://www.geeksforgeeks.org/working-udp-datagramsockets-java/- - MarkShanks – Acessado Maio/2020.

https://www.ccs-labs.org/teaching/rn/animations/gbn_sr/ - Johannes Kessler 2012- Acessado Maio/2020.

https://docs.oracle.com/javase/7/docs/api/java/net/DatagramSocket.html - Oracle – Acessado Maio/2020.

https://stackoverflow.com/questions/10298480/getlocaladdress-returning-0-0-0-0 - StackOverFlow – Acessado Maio/2020. - Acessado devido a questão de conectar em duas maquinas diferentes.