UNIVERSIDADE FEDERAL DO ABC

REDES DE COMPUTADORES

EXERCÍCIO PROGRAMÁTICO

Gabriel Murakami Alves 11071916

Santo André
2020

1. Formato das mensagens transferidas

```java
import java.io.Serializable;

public class Packet implements Serializable {

    public int id;
    public String msg;

    public Packet(int id, String msg) {
        super();
        this.id = id;
        this.msg = msg;
    }

    public int getId() {
        return id;
    }

    public void setId(int setId) {
        this.id = setId;
    }

    public String getMsg() {
        return msg;
    }
}
```

Imagem 1: Classe Packet

```java
import java.io.Serializable;

public class Ack implements Serializable {

    public int id;

    public Ack(int id) {
        super();
        this.id = id;
    }

    public int getId() {
        return id;
    }
}
```

Imagem 2: Classe Ack

O corpo da mensagem enviada foi montado utilizando a classe Packet (Imagem 1) com os atributos *id* e *msg*. Representando, respectivamente, o número de sequência do pacote e o seu conteúdo.

Os pacotes de confirmação, os Ack's, foram estruturados na classe Ack (Imagem 2). Onde basicamente temos somente o atributo id para identificar o pacote que estamos confirmando.

Como a comunicação é feita em bytes, as classes indicadas foram serializadas e transformadas em *byteArray* para que pudessem ser enviadas através de um datagrama. Sendo deserializadas quando precisassem ser lidas.

```
ByteArrayOutputStream bos = new ByteArrayOutputStream();
ObjectOutputStream out = new ObjectOutputStream(bos);
out.writeObject(packet);
data = bos.toByteArray();
```

Imagem 3: Serialização do pacote para envio (lado do cliente)

```
ByteArrayOutputStream bos = new ByteArrayOutputStream();
ObjectOutputStream out = new ObjectOutputStream(bos);
out.writeObject(ack);
sendBuff = bos.toByteArray();
```

Imagem 4: Serialização do Ack (lado do servidor)

```
ByteArrayInputStream b = new ByteArrayInputStream(recPacket.getData());
ObjectInputStream o = new ObjectInputStream(b);
Ack ack = (Ack) o.readObject();
```

Imagem 5: Deserialização do Ack recebido (lado do cliente)

```
ByteArrayInputStream b = new ByteArrayInputStream(recPacket.getData());
ObjectInputStream o = new ObjectInputStream(b);
Packet packet = (Packet) o.readObject();
```

Imagem 6: Deserialização do pacote recebido (lado do servidor)

2. Mensagem lenta

```
if(slowPacketId == i){
    Thread.currentThread().sleep(3000);
}
```

Imagem 7: Atrasando o pacote

Para enviar o pacote de forma "lenta", a thread que está sendo usada no momento é congelada por um tempo predeterminado. Dessa forma, o cliente irá demorar para receber a confirmação. No servidor, o tratamento foi igual aos outros casos.

3. Mensagem perdida

```
if (i==lostPacketId && lost) {
    lost = false;
    continue;
}
else if(i==WINDOW_SIZE && !lost) {
    i = lostPacketId;
    lost = true;
}
```

Imagem 8: Pacote perdido

Para simular a perda do pacote com o id predefinido em "lostPacketId", é verificado se o laço se encontra na iteração desejada. Se sim, a iteração indicada é pulada e é sinalizado que não estamos mais na janela de perda. Caso a iteração tenha chegado ao fim da janela, o laço é retornado para a posição do pacote perdido, reenviando os pacotes a partir do mesmo.

Para este caso não precisou de alterações no servidor, pois o tratamento para mensagem duplicada e fora de ordem já garantem a resposta adequada à falta de um pacote em uma determinada posição.

4. Mensagem fora de ordem

```
if(wrongOrder) {
    if(i==wrongOrderId) {
        packet.setId(wrongOrderId + 1);
    }
    else if(i==wrongOrderId + 1) {
        packet.setId(wrongOrderId);
    }
}
```

Imagem 9: Definindo ordem incorreta

Inverte o id de dois pacotes para enviá-los fora de ordem com base em um "wrongOrderId" predefinido. Verificando e utilizando a variável "wrongOrder" para sinalizar se será mandado fora de ordem ou não.

```
if(wrongOrder && i == WINDOW_SIZE - 1) {
    i = wrongOrderId - 1;
    wrongOrder = false;
}
```

Imagem 10: Retornando o laço

Retorna o laço para a posição do primeiro pacote enviado fora de ordem e manda novamente os pacotes que não receberam ACK pelo servidor por estarem na ordem incorreta.

```
if(buffer > last && last == packet.getId()) {
    System.out.println("Mensagem ID:" + packet.getId() + " '" + packet.getMsg() + "' recebida de forma duplicada\n");
}
else if(packet.getId() > buffer && !wrongOrder) {
    System.out.println("Mensagem ID:" + packet.getId() + " '" + packet.getMsg() + "' recebida fora de ordem\n");
    wrongOrder = true;
}
else {
    System.out.println("Mensagem ID:" + packet.getId() + " '" + packet.getMsg() + "' recebida\n");
}
```

Imagem 11: Verificação da sequência e último

No lado do servidor é verificado se o id do pacote recebido é maior do que a sequência que ele estava esperando, printando a mensagem de fora de ordem e sinalizando que "wrongOrder" é verdadeiro para ser utilizado mais a frente no ajuste do buffer.

```
int ackId = last;
if(!wrongOrder && buffer == packet.getId()) {
    last = buffer;
    ackId = buffer;
    buffer += 1;
}
```

Imagem 12: Definindo o buffer

Como indicado na Imagem x, o pacote esperado (buffer) só é atualizado se não estivermos em um cenário de ordem incorreta.

```
/* Sinaliza que não está mais na janela de ordem incorreta */
if (wrongOrder && packet.getId() == WINDOW_SIZE-1) {
    wrongOrder = false;
}
```

Imagem 13: Sinalizando que retornou à janela sem erro

Chegando ao fim da janela, é sinalizado que o servidor pode voltar a atualizar o buffer normalmente, pois as mensagens que foram enviadas fora de ordem começarão a ser reenviadas pelo cliente.

5. Mensagem duplicada

```
if(i==duplicatedPacketId && duplicate) {
    packet.setId(duplicatedPacketId - 1);
    i--;
    duplicate = false;
}
```

Imagem 14: Definindo pacote duplicado

No cliente, para mandar a mensagem duplicada, ao chegar em um id predefinido por "duplicatedPacketId", o laço é retornado para a iteração anterior e é enviado o pacote com o mesmo id novamente. Definindo também a variável “duplicate” como “false”, sinalizando que a ação de duplicação já foi executada.

```
else if(packet.getId() > buffer && !wrongOrder) {
    System.out.println("Mensagem ID:" + packet.getId() + " '" + packet.getMsg() + "' recebida fora de ordem\n");
    wrongOrder = true;
}
```

Imagem 15: Verificando a ordem

No servidor, por sua vez, utilizando uma variável “last” para guardar o id da última mensagem recebida com sucesso, é verificado o id da mensagem atual em relação ao id da última mensagem recebida. Se forem iguais então sabe-se que a mensagem é duplicada.

6. Buffer

O buffer foi utilizado para definir o número de sequência que é esperado pelo servidor, garantindo que as condições que estão sendo pedidas possam ser atendidas e verificadas adequadamente.

7. Código Fonte

7.1. Ack.java

```
import java.io.Serializable;


public class Ack implements Serializable {
  public int id;
  public Ack(int id) {
    super();
    this.id = id;
 }
  public int getId() {
    return id;
 }
}
```

7.2. Packet.java

```
import java.io.Serializable;

public class Packet implements Serializable {

 public int id;
 public String msg;

 public Packet(int id, String msg) {
   super();
   this.id = id;
   this.msg = msg;
 }

 public int getId() {
   return id;
 }

 public void setId(int setId) {
   this.id = setId;
 }

 public String getMsg() {
   return msg;
 }
}
```

7.3. UDPServer.java

```
import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;

public class UDPServer {

 public static void main(String[] args) throws Exception {
   int buffer = 0;
   int last = 0;
   int WINDOW_SIZE = 5;

   DatagramSocket serverSocket = new DatagramSocket(9876);

   boolean wrongOrder = false;

   while (true) {

     /* Aguarda a chegada de um pacote do cliente */
     byte[] recBuffer = new byte[1024];
     DatagramPacket recPacket = new DatagramPacket(recBuffer,
recBuffer.length);
     System.out.println("Esperando pacote do cliente...");
     serverSocket.receive(recPacket);

     /* Deserializa o pacote recebido */
     ByteArrayInputStream b = new
ByteArrayInputStream(recPacket.getData());
```

```
      ObjectInputStream o = new ObjectInputStream(b);
      Packet packet = (Packet) o.readObject();

      if(packet.getMsg().equals("reset")){
        System.out.println("Resetando as variáveis do
servidor...");
        System.out.println("================================\n");
        buffer = 0;
        last = 0;
      }
      else {
        /* Verifica a duplicidade do pacote recebido */
        if(buffer > last && last == packet.getId()) {
          System.out.println("Mensagem ID:" + packet.getId() + "
'" + packet.getMsg() + "' recebida de forma DUPLICADA\n");
        }
        else if(packet.getId() > buffer && !wrongOrder) {
          System.out.println("Mensagem ID:" + packet.getId() + "
'" + packet.getMsg() + "' recebida FORA DE ORDEM\n");
          wrongOrder = true;
        }
        else {
          System.out.println("Mensagem ID:" + packet.getId() + "
'" + packet.getMsg() + "' recebida\n");
        }

        /* Verifica a confirmação correta para o pacote recebido
*/
        int ackId = last;
        if(!wrongOrder && buffer == packet.getId()) {
          last = buffer;
          ackId = buffer;
          buffer += 1;
```

```
        }
        /* Serializa o pacote de confirmação adequado */
        byte[] sendBuff = new byte[1024];
        Ack ack = new Ack(ackId);
        ByteArrayOutputStream bos = new ByteArrayOutputStream();
        ObjectOutputStream out = new ObjectOutputStream(bos);
        out.writeObject(ack);
        sendBuff = bos.toByteArray();

        /* Obtem a porta e IP para enviar a confirmação */
        InetAddress IPAddress = recPacket.getAddress();
        int port = recPacket.getPort();

        /* Envia o pacote de confirmação para o cliente */
        DatagramPacket sendPacket =  new DatagramPacket(sendBuff,
sendBuff.length, IPAddress, port);
        serverSocket.send(sendPacket);

        /* Sinaliza que não está mais na janela de ordem incorreta
*/
        if (wrongOrder && packet.getId() == WINDOW_SIZE-1) {
          wrongOrder = false;
        }
      }
    }
  }
}
```

7.4. UDPClient.java

```
import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
```

```java
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;
import java.util.Scanner;

public class UDPClient {

 public static void main(String[] args) throws Exception {

   int WINDOW_SIZE = 5;
   boolean running = true;
   InetAddress IPAddress = InetAddress.getByName("127.0.0.1");

   while(running){
     DatagramSocket clientSocket = new DatagramSocket();
     Scanner sc = new Scanner(System.in);
     System.out.println("Opções:");
     System.out.println("1) Envio normal");
     System.out.println("2) Envio com perda");
     System.out.println("3) Envio fora de ordem");
     System.out.println("4) Envio duplicado");
     System.out.println("5) Envio lento");
     System.out.println("6) Sair");
     System.out.print("Insira o número da opção escolhida: ");
     int op = sc.nextInt();
     System.out.println("================================");

     switch(op) {
     case 1:
       System.out.println("Normal");
       System.out.println("================================");
```

```
      for(int i=0; i<WINDOW_SIZE; i++) {
        /* Define a mensagem a ser enviada */
        byte[] data = new byte[1024];
        String msg = "Bom dia";
        Packet packet = new Packet(i, msg);

        /* Serializa o pacote e o transforma em byteArray */
        ByteArrayOutputStream bos = new ByteArrayOutputStream();
        ObjectOutputStream out = new ObjectOutputStream(bos);
        out.writeObject(packet);
        data = bos.toByteArray();

        /* Envia o pacote */
        DatagramPacket sendPacket = new DatagramPacket(data,
data.length, IPAddress, 9876);
        clientSocket.send(sendPacket);
        System.out.println("Mensagem '" + packet.getMsg() + "'
enviada com ID " + i);

        /* Aguarda o recebimente da confirmação do server */
        byte[] recBuffer = new byte[1024];
        DatagramPacket recPacket = new DatagramPacket(recBuffer,
recBuffer.length);
        clientSocket.receive(recPacket);

        /* Deserializa a confirmação do servidor */
        ByteArrayInputStream b = new
ByteArrayInputStream(recPacket.getData());
        ObjectInputStream o = new ObjectInputStream(b);
        Ack ack = (Ack) o.readObject();
        System.out.println("Confirmação de mensagem ID: " +
ack.getId() + " recebida pelo servidor\n");
```

```
      }
      resetServer(clientSocket, IPAddress);
      clientSocket.close();

      break;
    case 2:
      System.out.println("Perda");
      System.out.println("================================");

      int lostPacketId = 3;
      boolean lost = true;

      for(int i=0; i<WINDOW_SIZE+lostPacketId; i++) {

        /* Simular a perda do pacote 3 pulando sua iteração */
        if (i==lostPacketId && lost) {
          lost = false;
          continue;
        }
        else if(i==WINDOW_SIZE && !lost) {
          i = lostPacketId;
          lost = true;
        }

        /* Define a mensagem a ser enviada */
        byte[] data = new byte[1024];
        String msg = "Bom dia";
        Packet packet = new Packet(i, msg);

        /* Serializa o pacote e o transforma em byteArray */
        ByteArrayOutputStream bos = new ByteArrayOutputStream();
        ObjectOutputStream out = new ObjectOutputStream(bos);
        out.writeObject(packet);
```

```
            data = bos.toByteArray();

            /* Envia o pacote */
            DatagramPacket sendPacket = new DatagramPacket(data,
data.length, IPAddress, 9876);
            clientSocket.send(sendPacket);
            System.out.println("Mensagem '" + packet.getMsg() + "'
enviada com ID " + i);

            /* Aguarda o recebimente da confirmação do server */
            byte[] recBuffer = new byte[1024];
            DatagramPacket recPacket = new DatagramPacket(recBuffer,
recBuffer.length);
            clientSocket.receive(recPacket);

            /* Deserializa a confirmação do servidor */
            ByteArrayInputStream b = new
ByteArrayInputStream(recPacket.getData());
            ObjectInputStream o = new ObjectInputStream(b);
            Ack ack = (Ack) o.readObject();
            System.out.println("Confirmação de mensagem ID: " +
ack.getId() + " recebida pelo servidor\n");
        }
        resetServer(clientSocket, IPAddress);
        clientSocket.close();

        break;
      case 3:
        System.out.println("Fora de Ordem");
        System.out.println("================================");

        boolean wrongOrder = true;
        int wrongOrderId = 2;
```

```
      for(int i=0; i<WINDOW_SIZE+wrongOrderId; i++) {
        /* Define a mensagem a ser enviada */
        byte[] data = new byte[1024];
        String msg = "Bom dia";

        Packet packet = new Packet(i, msg);

        /* Reseta o for para a posição do pacote enviado fora de
ordem */
        if(wrongOrder && i == WINDOW_SIZE - 1) {
          i = wrongOrderId - 1;
          wrongOrder = false;
        }

        /* Envia o dois pacotes fora de ordem */
        if(wrongOrder) {
          if(i==wrongOrderId) {
            packet.setId(wrongOrderId + 1);
          }
          else if(i==wrongOrderId + 1) {
            packet.setId(wrongOrderId);
          }
        }

        /* Serializa o pacote e o transforma em byteArray */
        ByteArrayOutputStream bos = new ByteArrayOutputStream();
        ObjectOutputStream out = new ObjectOutputStream(bos);
        out.writeObject(packet);
        data = bos.toByteArray();

        /* Envia o pacote */
```

```
          DatagramPacket sendPacket = new DatagramPacket(data,
data.length, IPAddress, 9876);
          clientSocket.send(sendPacket);

          if(wrongOrder && ( i==wrongOrderId || i==wrongOrderId +
1)) {
            System.out.println("Mensagem '" + packet.getMsg() + "'
enviada de FORA DE ORDEM com ID " + packet.getId());
          }
          else {
            System.out.println("Mensagem '" + packet.getMsg() + "'
enviada com ID " + packet.getId());
          }

          /* Aguarda o recebimente da confirmação do server */
          byte[] recBuffer = new byte[1024];
          DatagramPacket recPacket = new DatagramPacket(recBuffer,
recBuffer.length);
          clientSocket.receive(recPacket);

          /* Deserializa a confirmação do servidor */
          ByteArrayInputStream b = new
ByteArrayInputStream(recPacket.getData());
          ObjectInputStream o = new ObjectInputStream(b);
          Ack ack = (Ack) o.readObject();
          System.out.println("Confirmação de mensagem ID: " +
ack.getId() + " recebida pelo servidor\n");
        }
        resetServer(clientSocket, IPAddress);
        clientSocket.close();

        break;
      case 4:
```

```
        System.out.println("Duplicado");
        System.out.println("==============================");

        int duplicatedPacketId = 2;
        boolean duplicate = true;

        for(int i=0; i<WINDOW_SIZE; i++) {
          /* Define a mensagem a ser enviada */
          byte[] data = new byte[1024];
          String msg = "Bom dia";

          Packet packet = new Packet(i, msg);

          /* Força o pacote com o ID definido a ir duplicado */
          if(i==duplicatedPacketId && duplicate) {
            packet.setId(duplicatedPacketId - 1);
            i--;
            duplicate = false;
          }

          /* Serializa o pacote e o transforma em byteArray */
          ByteArrayOutputStream bos = new ByteArrayOutputStream();
          ObjectOutputStream out = new ObjectOutputStream(bos);
          out.writeObject(packet);
          data = bos.toByteArray();

          /* Envia o pacote */
          DatagramPacket sendPacket = new DatagramPacket(data,
data.length, IPAddress, 9876);
          clientSocket.send(sendPacket);

          if(i==1 && !duplicate) {
```

```
            System.out.println("Mensagem '" + packet.getMsg() + "'
enviada de forma DUPLICADA com ID " + i);
          }
          else {
            System.out.println("Mensagem '" + packet.getMsg() + "'
enviada com ID " + i);
          }

          /* Aguarda o recebimente da confirmação do server */
          byte[] recBuffer = new byte[1024];
          DatagramPacket recPacket = new DatagramPacket(recBuffer,
recBuffer.length);
          clientSocket.receive(recPacket);

          /* Deserializa a confirmação do servidor */
          ByteArrayInputStream b = new
ByteArrayInputStream(recPacket.getData());
          ObjectInputStream o = new ObjectInputStream(b);
          Ack ack = (Ack) o.readObject();
          System.out.println("Confirmação de mensagem ID: " +
ack.getId() + " recebida pelo servidor\n");
        }
        resetServer(clientSocket, IPAddress);
        clientSocket.close();

        break;
      case 5:
        System.out.println("Lento");
        System.out.println("================================");

        int slowPacketId = 3;

        for(int i=0; i<WINDOW_SIZE; i++) {
```

```
          /* Define a mensagem a ser enviada */
          byte[] data = new byte[1024];
          String msg = "Bom dia";
          Packet packet = new Packet(i, msg);

          /* Serializa o pacote e o transforma em byteArray */
          ByteArrayOutputStream bos = new ByteArrayOutputStream();
          ObjectOutputStream out = new ObjectOutputStream(bos);
          out.writeObject(packet);
          data = bos.toByteArray();

          /* Envia o pacote */
          DatagramPacket sendPacket = new DatagramPacket(data,
data.length, IPAddress, 9876);
          clientSocket.send(sendPacket);

          if(slowPacketId == i){
            System.out.println("Mensagem '" + packet.getMsg() + "'
enviada de forma LENTA com ID " + i);
          }
          else{
            System.out.println("Mensagem '" + packet.getMsg() + "'
enviada com ID " + i);
          }

          /* Aguarda o recebimente da confirmação do server */
          byte[] recBuffer = new byte[1024];
          DatagramPacket recPacket = new DatagramPacket(recBuffer,
recBuffer.length);
          clientSocket.receive(recPacket);

          if(slowPacketId == i){
            Thread.currentThread().sleep(3000);
```

```
            }

            /* Deserializa a confirmação do servidor */
            ByteArrayInputStream b = new
ByteArrayInputStream(recPacket.getData());
            ObjectInputStream o = new ObjectInputStream(b);
            Ack ack = (Ack) o.readObject();
            System.out.println("Confirmação de mensagem ID: " +
ack.getId() + " recebida pelo servidor\n");
          }
          resetServer(clientSocket, IPAddress);
          clientSocket.close();

          break;
        case 6:
          System.out.println("Encerrando o cliente...");
          running = false;
          resetServer(clientSocket, IPAddress);

          break;
        }
      }
    }

    private static void resetServer(DatagramSocket clientSocket,
InetAddress IPAddress) throws Exception {
      byte[] data = new byte[1024];
      String msg = "reset";
      Packet packet = new Packet(0, msg);

      /* Serializa o pacote e o transforma em byteArray */
      ByteArrayOutputStream bos = new ByteArrayOutputStream();
      ObjectOutputStream out = new ObjectOutputStream(bos);
```

```
   out.writeObject(packet);
   data = bos.toByteArray();

   /* Envia o pacote para resetar as varáveis do servidor */
   DatagramPacket sendPacket = new DatagramPacket(data,
data.length, IPAddress, 9876);
   clientSocket.send(sendPacket);
 }
}
```