**GROUP MEMBERS**

**PATRICK GICHINGA - C025-01-0966/2015**

**PETER KAORA - C025-01-1636/2015**

**JOSHUA MAKORI - C025-01-0985/2015**

**GEORGE KITONGA - C025-01-0951/2015**

**JACKSON MWANGE - C025-01-09/2015**

**COURSE : BSc. INFORMATION TECHNOLOGY**

**UNIT CODE: ICS 2403**

**UNIT NAME: DISTRIBUTED SYSTEMS**

**TASK : ASSIGNMENT 2 & 3**

**DATE: 30th/7/2018**

**ASSIGNMENT 2**

1. **Question 1**
2. **Give an outline of the server implementation showing how the operation *getRequest* and *sendReply* are used by a server that creates a new thread to execute each client request. *(5 marks)***
3. **Indicate how the server will copy the requestId from the request message into the reply message and how it will obtain the client IP adress and port. *(5 marks)***

*class Server{*  
*private int serverPort = 8888;*  
*public static int messageLength = 1000;*  
*DatagramSocket mySocket;*  
*public Server(){*  
*mySocket = new DatagramSocket(serverPort);*  
*while(true){*  
*byte [] request = getRequest();*  
*Worker w = new Worker(request);*  
*}*  
*}*  
*public byte [] getRequest(){*  
//as above}  
*public void sendReply(byte[]reply, InetAddress clientHost, int clientPort){*  
// as above}  
*}*  
*class Worker extends Thread {*  
*InetAddress clientHost;*  
*int clientPort;*  
*int requestId;*  
*byte [] request;*  
*public Worker(request){*  
// extract fields of message into instance variables  
*}*  
*public void run(){*  
*try{*  
*req = request.unmarshal();*  
*byte [] args = req.getArgs();*  
//unmarshall args, execute operation,  
// get results marshalled as array of bytes in result

*RequestMessage rm = new RequestMessage( requestId, result);*  
*reply = rm.marshal();*  
*sendReply(reply, clientHost, clientPort );*  
*}catch {*... *}*  
*}*  
*}*

1. **Question 2**
2. **Why might the number of messages exchanged in a protocol be more significant to performance than the total amount of data sent? *(5 marks)***

The time for the exchange of a message = A + B\* length, where A is the fixed processing overhead and B is the rate of transmission. A is large because it represents significant processing at both sender and receiver; the sending of data involves a system call; and the arrival of a message is announced by an interrupt which must be handled and the receiving process is scheduled. Protocols that involve several rounds of messages tend to be expensive because of paying the A cost for every message.

1. **Design a variant of the RRA protocol in which the acknowledgement is piggy-backed on, that is, transmitted in the same message as, the next request where appropriate, and otherwise sent as a separate message. *Hint: use an extra timer in the client. (5 marks)***

The new version of RRA has:

|  |  |
| --- | --- |
| **Client** | **Server** |
| 1. cancel any outstanding acknowledgement on a timer |  |
| 1. send Request |  |
|  | 1. receive Request |
|  | 1. send Reply |
| 1. receive Reply |  |
| 1. set timer to send acknowledgement after delay T |  |
|  | 1. receive Acknowledgement |

The client always sends an acknowledgement, but it is piggy-backed on the next request if one arises in the next T seconds. It sends a separate acknowledgement if no request arises. Each time the server receives a request or an acknowledgement message from a client, it discards any reply message saved for that client.

1. **Question 3**
2. **A router separating process *p* from two others, *q* and *r*, fails immediately after p   
   initiates the multicasting of message *m.* If the group communication system is view-synchronous, explain what happens to *p* next. *(5 marks)***

Process *p* must receive a new group view containing only itself, and it must receive the message it sent. The question is: in what order should these events be delivered to *p*?.

If *p* received the message first, then that would tell *p* that *q* and *r* received the message; but the question implies that they did not receive it. So *p* must receive the group view first.

1. **You are giving a group communication system with a totally ordered multicast operation, and a failure detector. Is it possible to construct view-synchronous group communication from these components alone? *(5 marks)***

If the multicast is reliable, yes.

Then we can solve consensus. In particular, we can decide, for each message, the view of the group to deliver it to. Since both messages and new group views can be totally ordered, the resultant communication will be view-synchronous. If the multicast is unreliable, then we do not have a way of ensuring the consistency of view delivery to all of the processes involved.

**TASK: ASSIGNMENT 3**

1. **Is it conceivably useful for a port to have several receivers?**

If several processes share a port, then it must be possible for all of the messages that arrive on that port to be received and processed independently by those processes.  
Processes do not usually share data, but sharing a port would require access to common data representing the messages in the queue at the port. In addition, the queue structure would be complicated by the fact that each process has its own idea of the front of the queue and when the queue is empty.  
A port group may be used to allow several processes to receive the same message.

1. **A server creates a port which it uses to receive requests from clients. Discuss the design issues concerning the relationship between the name of this port and the names used by clients.**

The main design issues for locating server ports are:

1. How does a client know what port and IP address to use to reach a service?

The options are:

1. Use a name server/binder to map the textual name of each service to its port.
2. Each service uses well-known location-independent port id, which avoids a look up at a name server.
3. The operating system still has to look up the whereabouts of the server, but the answer may be cached locally.
4. How can different servers offer the service at different times?

Location independent port identifiers allow the service to have the same port at different locations. If a binder is used, the client needs to re-consult the client to find the new location.

1. Efficiency of access to ports and local identifiers.

Sometimes operating systems allow processes to use efficient local names to refer to ports. This becomes an issue when a server creates a non-public port for a particular client to send messages to, because the local name is meaningless to the client and must be translated to a global identifier for use by the client.

1. **Use the programs in Figure 1 and Figure 2 to make a test kit to determine the conditions in which datagrams are sometimes dropped. Hint: the client program should be able to vary the number of messages sent and their size; the server should detect when a message from a particular client is missed.**

*import java.net.\*;*   
*import java.io.\**

*public class UDPClient{*

*public static void main(String args[]){*

*//args give message contents and server hostname*   
*try{*

*DatagramSocket aSocket = new DatagramSocket();*   
*byte[] m = args[0].getBytes();*   
*InetAddress aHost = InetAddress.getByName(args[1]);*   
*int serverPort = 6789;*   
*DatagramPacket request =*   
*new DatagramPacket(m,args[0].length(), aHost,serverPort);*   
*aSocket.send(request);*   
*byte[] buffer = new byte[1000];*   
*DatagramPacket reply =*   
*new DatagramPacket(buffer,buffer.length);*   
*aSocket.receive(reply);*   
*System.out.println("Reply:"+new String(reply.getData()));*   
*aSocket.close();*

*}catch(SocketException e)*   
*{System.out.println("Socket:"+e.getMessage());*   
*}catch(IOException e){System.out.println("IO:"+e.getMessage();}*

*}*

*}*

**Figure 1: UDP client sends a message to the server and gets a reply**

*import java.net.\**   
*import java.io.\**

*public class UDPServer{*

*public static void main(String args[]){*

*try{*

*DatagramSocket aSocket = new DatagramSocket(6789);*   
*byte[] buffer = new byte[1000];*   
*while(true){*

*DatagramPacket request =*   
*new datagramPacket(buffer, buffer.length);*   
*aSocket.receive(request);*   
*DatagramPacket reply =*   
*new DatagramPacket(request.getData(),*   
*request.getLength(), request.getAddress(),*   
*request.getPort());*   
*aSocket.send(reply);*

*}*

*}catch(SocketException e)*   
*{System.out.println("Socket:" + e.getMessage());*   
*}catch(IOException e)*   
*{System.out.println("IO:" + e.getMessage());}*

*}*

*}*

**Figure 2: UDP server repeatedly receives a request and sends it back to the client**

For a test of this type, one process sends and another receives.

Modify the program in Figure 1 so that the program arguments specify

1. the server’s hostname
2. the server ports
3. the number, *n* of messages to be sent and
4. the length, *l* of the messages.

If the arguments are not suitable, the program should exit immediately. The program should open a datagram socket and then send *n* UDP datagram messages to the server. Message *i* should contain the integer *i* in the first four bytes and the character ‘\*’ in the remaining l-4  
bytes. It does not attempt to receive any messages.  
Take a copy of the program in Figure 2 and modify it so that the program argument specifies the server port.  
The program should open a socket on the given port and then repeatedly receive a datagram message. It should check the number in each message and report whenever there is a gap in the sequence of numbers in the messages received from a particular client.  
Run these two programs on a pair of computers and try to find out the conditions in which datagrams are dropped, e.g. size of message, number of clients.

1. **Use the program in Figure 1 to make a client program that repeatedly reads a line of input from the user, sends it to the server in a UDP datagram message, then receives a message from the server. The client sets a timeout on its socket so that it can inform the user when the server does not reply. Test this client program with the server in Figure 2.**

The program is as Figure 2 with the following amendments:  
DatagramSocket aSocket = new DatagramSocket();  
aSocket.setSoTimeout(3000);// in milliseconds  
while (// not eof) {  
try{  
// get user’s input and put in request  
.....  
aSocket.send(request);  
........  
aSocket.receive(reply);  
}catch (InterruptedIOException e){System.out.println("server not responding”);

}

1. **Modify the programs in Figure 3 and Figure 4 so that the client repeatedly takes a line of user's input and writes it to the stream and the server reads repeatedly from the stream, printing out the result of each read. Make a comparison between sending data in UDP datagram and over a stream.**

*import java.net.\*;*   
*import java.io.\*;*

*public class TCPClient {*

*public static void main (String args[]) {*   
*//arguments supply message and hostname of destination*

*try{*

*int serverPort = 7896;*   
*Socket s = new Socket(args[1],serverPort);*   
*DataInputStream in = new DataInputStream(s.getInputStream());*   
*DataOutputStream out = new DataOutputStream(s.getOutputStream());*   
*out.writeUTF(args[0]);*   
*String data = in.readUTF();*   
*System.out.println("Received:"+data);*   
*s.close();*

*}catch(EOFException e){System.out.println("EOF:" + e.getMessage());*   
*}catch(IOException e){System.out.println("IO:" + e.getMessage());}*

*}*

*}*

**Figure 3: TCP client makes connection to server, sends request and receives reply.**

*import java.net.\*;*   
*import java.io.\*;*

*public class TCPServer {*

*public static void main(String args[]){*

*try{*

*int serverPort = 7896;*   
*ServerSocket listenSocket = new ServerSocket(serverPort);*   
*while(true) {*

*Socket clientSocket = listenSocket.accept();*   
*Connection c = new Connection(clientSocket);*

*}*

*}catch(IOException e)*   
*{System.out.println("Listen:" + e.getMessage());}*

*}*

*}*

*class Connection extends Thread {*

*DataInputStream in;*   
*DataOutputStream out;*   
*Socket clientSocket;*   
*public Connection (Socket aClientSocket) {*

*try{*

*clientSocket = aClientSocket;*   
*in = new DataInputStream(clientSocket.getInputStream());*   
*out = new DataOutputStream(clientSocket.getOutputStream());*   
*this.start();*

*}catch(IOException e){System.out.println("Connection:" + e.getMessage());}*

*}*   
*public void run(){*

*try{*

*//an echo server*   
*String data = in.readUTF();*   
*out.writeUTF(data);*   
*clientSocket.close();*

*}catch(EOFException e){System.out.println("EOF:" + e.getMessage());*   
*}catch(IOException e){System.out.println("IO:" + e.get.Message());}*

*}*

*}*

The changes to the two programs are straightforward. Not all sends go immediately and that receives must match the data sent.  
For the comparison. In both cases, a sequence of bytes is transmitted from a sender to a receiver. In the case of a message the sender first constructs the sequence of bytes and then transmits it to the receiver which receives it as a whole. In the case of a stream, the sender transmits the bytes whenever they are ready and the receiver collects the bytes from the stream as they arrive.

**Figure 4: TCP server makes a connection for each client and then echoes the client's request**

1. **Use the program developed in Question 5 to the test the effect on the sender when the receiver crashes and vice-versa**

Run them both for a while and then kill first one and then the other. When the reader process crashes, the writer gets IOException - broken pipe. When writer process crashes, the reader gets EOF exception.