**2021**

**QUESTION 1**

**a)**

Object/service based:

* This is suitable as the players would act as objects communicating with the centralised server ‘GameServer’. The clients/objects would make requests to the game server and the server would wait for the requests and reply to the players. One of the players, which hosts as the GameServer, acts as both the client and the server.

Peer-to-peer:

* This architecture is based around the idea that all peers are the same and have the same functionality. This is not a suitable architecture since here the peers would have to communicate with each other directly, without a centralised GameServer with a map.

Event-based architecture:

* This is suitable since the players can subscribe to updates from the GameServer object. The GameServer would publish the the event of an updated map to whoever has subscribed to it.

**b)**

public class PlayerImpl extends UnicastRemoteObject implements Player{  
  
 Map localMap;  
 Player nextPlayer;  
 GameServer gameServer;  
  
 PlayerImpl() throws RemoteException {  
 super();  
 }  
  
 public static void main(String[] args){  
 GameServer gameServer;  
 try{  
 // get the URI of the game (assuming game is a server)  
 String gameURI = args[0];  
 gameServer = (GameServer) Naming.*lookup*(gameURI);  
  
 } catch (RemoteException e){  
 try{  
 // if no gameServer exists, create it and bind it to local registry  
 gameServer = new GameServerImpl();  
 Naming.*rebind*("rmi://localhost/GameServer/", gameServer);  
  
 } catch(RemoteException e){  
 System.*out*.println(e.getMessage());  
 return;  
 }  
 }  
  
 // create the player object and bind to register with the GameServer  
 try{  
 Player player = new PlayerImpl();  
 gameServer.register(player);  
 ((PlayerImpl) player).gameServer = gameServer;  
  
 } catch(RemoteException e){  
 System.*out*.println(e.getMessage());  
 }  
 }  
  
 @Override  
 public void update(Map m) throws RemoteException {  
 localMap = m;  
 }  
  
 @Override  
 public void setNext(Player p) throws RemoteException {  
 nextPlayer = p;  
 }  
  
 @Override  
 public void yourTurn() throws RemoteException {  
  
 /// code for the player to make a move////  
 Move m = new Move();  
 gameServer.play(m);  
  
 // call next player to make a move  
 nextPlayer.yourTurn();  
 }  
}

c)

We could implement the bully election algorithm:

* Each player would hold an ID, corresponding to thei bandwidth connection to a host with IP address 146.169.14.1
* When a player detects that the GameServer objecy as failed, it will message all the players with a higher bandwidth than itself.
  + If a player receives a message with a higher bandwidth than itself, it will broadcast the same message to all the players that have a higher bandwidth than itself.
  + If a player receives a message with a lower bandwidth than itself, it will reply back to the sender indicating that it’s still alive.
* If a player receives an answer from another player with a higher bandwidth, it waits certain time for a coordinator message. If none received, it starts the election again
* If a player receives no answers from players with a higher bandwidth, it assumes they are idle and declares itself a coordinator.
* The new leader broadcasts a coordinator message to all the players, indicating that it’s the new GameServer
* All the players then update their reerences of GameServer

à each player must keep a field of their ID (bandwidth to the IP address) and a Boolean indicating if they are the current leader:

public void startElection() {

isLeader = false;

List<Player> higherPlayers = getHigherPlayers();

if (higherPlayers.isEmpty()) {

// This player is the highest, become leader

becomeLeader();

} else {

// Send election message to higher players

ElectionMessage message = new ElectionMessage(id);

for (Player p : higherPlayers) {

p.handleElectionMessage(message);

}

}

}

public void handleElectionMessage(ElectionMessage message) {

if (message.getSenderId() > id) {

// Forward election message to higher players

for (Player p : getHigherPlayers(message.getSenderId())) {

p.handleElectionMessage(message);

}

} else {

// Send response message to lower player

ResponseMessage response = new ResponseMessage(id);

Player sender = getPlayerById(message.getSenderId());

sender.handleResponseMessage(response);

}

}

public void handleResponseMessage(ResponseMessage message) {

// Do nothing, just indicate that this player is still alive

}

public void becomeLeader() {

isLeader = true;

broadcastCoordinatorMessage();

}

public void broadcastCoordinatorMessage() {

CoordinatorMessage message = new CoordinatorMessage();

for (Player p : getAllPlayers()) {

p.handleCoordinatorMessage(message);

}

}

public void handleCoordinatorMessage(CoordinatorMessage message) {

gameServer = this;

**d)**

Message1:

* The player sends a message to the game server which is encrypted by the game server’s public key. The game server can decrypt this message using its private key and see that the message is from P. The nonce indicates that the message is fresh.

Message2:

* The game server sends a message to the player which is encrypted with the player’s public key. The player can decrypt it with its private key and see the origina nonce that it sent to the game server, as well as a nonce from the game server. This allows the player to authenticate the game server since it sees the nonce and knows that the game server must have decrypted it using its private key. The nonce from the game server also indicates freshness.

Message3:

* The player sends a message to the game server with the game server’s nonce. Only the game server can decrypt this using its private key and see the nonce it had sent. It therefore authenticates the player as it knows the player only could have determined the nonce by decrypting with its private key.

**Not sure about this:**

S1 could pretend to be P1 by:

* S1 could intercept message 1 and send it to GameServer S2, pretending to be P1.
* S2 would reply to S1 with the two nonces, encrypted by the public key of P1.
* S1 does not need to decrypt this message – it could just forward this message to P1 and pretend to be S2.
* P1 would then reply to S1 (thinking it is S2), with a message containing the nonce encrypted with S2’s public key.
* S1 would then forward this message to S2 – S2 would decrypt it using its private key and see a nonce from P1.

1. P --> S1: {NP, P}KS1

2. S1 --> S2: {NP, P}KS2

3. S2 --> S1: {NP, NS2}KP

4. S1 --> P: {NP, NS2}KP

5. P --> S1: {NS2}KS1

6. S1 --> S2: {NS2}KS2

**Also not sure:**

Modification to the protocol:

* We could include timestamps to ensure that the message is not replayed again
* Or could use a session key that can only be decrypted by each entity’s private key. Therefore S1 would not be able to access the session key between P1 and S1

QUESTION 2

a)

W1:

IP address: **11000001.00000000.00000000.00001**010

Network mask: **11111111.11111111.11111111.11111**000

Gateway: 11000001.00000000.00000000.00001100

i)

Based on W1’s network mask, the subnet part of the network is 29 bits long and the host part is 3 bits. Therefore, we can allocate up to 8 addresses in each subnet.

A possible configuration for W2:

IP address: **11000001.00000000.00000000.00001**011 OR 193.0.0.11

Network mask: **11111111.11111111.11111111.11111**000

Gateway: 11000001.00000000.00000000.00001100

The network mask is the same as for W1, since it is within the same network. Th default gateway is also the same as W1 since they share the same subnet.

The subnet with W3 and W4 will have the same network mask since they are on the same network, but they will have a different subnet part. They will, however, share the same gateway.

A possible configuration for W3:

IP address: **11000001.00000000.00000000.00000**010 OR 193.0.0.2

Network mask: **11111111.11111111.11111111.11111**000

Gateway: 11000001.00000000.00000000.00000100 OR 193.0.0.4

Here we are following the configuration of W1 by assigning the gateway to the 5th available address of the subnet and the W3 to the 3nd available address of the subnet.

Similarly for W4:

IP address: **11000001.00000000.00000000.00000**011 OR 193.0.0.3

Network mask: **11111111.11111111.11111111.11111**000

Gateway: 11000001.00000000.00000000.00000100 OR 193.0.0.4

ii)

For each subnet, we have 3 bits available to assigns hosts to, so 8 addresses in total per subnet. Assuming that each subnet has 2 reserved addresses (all 1s for broadcasting and all 0s for the subnet itself). Therefore, there are 4 free addresses in each subnet to add workstations to, 8 in total.

iii)

|  |  |
| --- | --- |
| Source IP Address | Router Interface |
| 193.0.0.8/29 | Eth1 |
| 193.0.0.0/29 | Eth2 |
| Default | Eth3 |

iv)

**Not sure:**

W1 could send the IP datagram to the broadcast address of the subnet. This would be passed down to the datalink layer with the MAC address of the router as the destination. The switch would pass it over to the router, which would extract the IP datagram and see the IP destination is the broadcast address of the subnet including W1 and W2. The router would then proceed to broadcast the message to everyone in subnet 1.

b)

1. The process on W1 would initiate the UDP communication by sending a message to W2’s IP address, using port number 4444 as the destination port.
2. The transport layer on W1 would encapsulate the data into a UDP segment. The UDP segment would have the source port number of W1, and the destination port number would be 4444.
3. The network layer on W1 would see that the destination IP address is on the same subnet, therefore it should be passed onto W2 straight away without going through the router. Assuming that W1 does not know the MAC address of W2, it would use the ARP protocol:
   1. W1 would broadcast an ARP query containing W2’s IP address.
   2. W2 would reply to W1 with an ARP response, giving its MAC address
   3. W1 would add W2’s MAC address to its ARP table
4. The network layer would encapsulate the UDP segment into an IP packet. The source IP address would be the IP address of W1 and the destination IP address would be the IP address of W2.
5. The network layer would see that 3000 bytes is too much for the MTU of 1500 and would down the original datagram into fragments:
   1. Since the datagram header is 20 bytes, the first fragment would contain 1480 of the bytes, with the offset field set to 0 and more fragments flag set to 1.
   2. Similarly, the second fragment would contain another 1480 bytes with the offset field set to 185 (1480/8)
   3. The third fragment would contain the rest of bytes (40 bytes) with the offset field set to 370 and tge flag set to 0.
6. The network layer would pass the fragment down to the datalink layer, which would create the frame. The frame would contain a header, trailer and the IP packet payload. The source MAC address would be W1’s MAC address and the destination MAC address would be W2’s MAC address.
7. The frame would be transmitted over the network to the switch, which would use the destination MAC address to forward the frame directly to W2.
8. The data-link layer on W2 would receive the frame, check for errors and extract the IP packet from it.
9. The network layer on W2 would see that the received packet is a fragment of an IP datagram, so it waits for other fragments. Once all fragments have been received, it reassembles them and then pass the packet up to the transport layer.
10. The transport layer on W2 would extract the UDP segment from the packet and pass it to the receiving process on port number 4444. The process would receive the 3000 bytes of data.

c)

Don’t think we covered this

d)

|  |  |
| --- | --- |
| **IP Addresses** | **MAC Addresses** |
| On the network layer | On the datalink layer |
| 32 bits long | 48 bits long |
| Enable routing of IP packets between different networks | Enable communication of between devices physically adjacent over a link |
| Assigned to hosts by the network server (DHCP server) | Assigned by the manufacturer |
| Ip addresses are a hierarchical namespace (contain a subnet and a host part) | Not hierarchical |

Both are used to identify which interface/port of a switch or a router to forward the message from.

If we could accommodate for different address sizes, there would be many more address sizes available, so might be able to identify a specific host with only one address. In this case, it would be preferrable to use IP addresses since they are hierarchical and would allow for more flexible and dynamic routing.