DISTRIBUTED COMPUTING SYSTEMS

Sockets and RPC

DIRECT MESSAGE TRANSMISSION: SOCKETS

DIRECT MESSAGE TRANSMISSION: SOCKETS

Output Uses transport layer directly in the form of Middleware.



- A socket is an abstract object that represents the endpoint of the connection.
- TCP/IP socket is a combination of IP address and port number, for example, 10.10.10.10: 80.
- Socket interface first appeared in BSD Unix.

BERKELEY SOCKETS API (1)

Socket primitives for TCP/IP.

Primitive	Meaning
Socket	Create a new communication endpoint
Bind	Attach a local address to a socket
Listen	Announce willingness to accept connections
Accept	Block caller until a connection request arrives
Connect	Actively attempt to establish a connection
Send	Send some data over the connection
Receive	Receive some data over the connection
Close	Release the connection

BERKELEY SOCKETS (2)



SOCKET IMPLEMENTATION EXAMPLE

- C # supports two types of network connections:
- Server using the TcpListener class objects;
- the client implemented by using objects of the TcpClient class.

TCPLISTENER AND TCPCLIENT OBJECTS

- An object of TcpListener class allows only to listen to a specific port on your computer.
- Any processes of data transmission via this socket are carried out using the TcpClient object.
- The AcceptTcpClient() method of the TcpListener class returns the TcpClient object that provides the listening port.

SERVER EXAMPLE

```
using System.Net;
using System.Net.Sockets;
Int32 port = 13000;
IPAddress localAddr = IPAddress. Parse
   ("127.0.0.1");
TcpListener server = new TcpListener (localAddr,
   port);
server.Start ();
//Start listening on port
TcpClient client = server.AcceptTcpClient ();
//After connection create message flow
```

NetworkStream stream = client.Getstream();

MESSAGING

Writing messages

```
Byte [] bytes = new Byte
[256];
```

```
String data = "text";
```

```
bytes =
   System.Text.Encoding.UTF.
   GetBytes (data);
```

```
stream.Write (bytes, 0,
    bytes.Length);
```

Reading messages

```
Byte [] bytes = new Byte
[256];
```

String data = null;

```
int i = stream.Read (bytes,
    0, bytes.Length);
```

data = system.text. encoding.UTF8.GetString (bytes, 0, i);

REMOTE PROCEDURE CALL RPC-REMOTE PROCEDURE CALL RMI-REMOTE METHOD INVOCATION

RPC TECHNOLOGY

Remote procedure call is a technology that allows computer programs to call the function or procedure in a different address space.



© GLEB RADCHENKO

THE STACK WHEN CALLING LOCAL PROCEDURES



RPC IMPLEMENTATION

- The idea: remote procedure call "transparent" for the local process
- Instead of the local procedure we use the "client stub".
- It is called as a local procedure, but instead of execution it sends a message the remote machine.

REMOTE PROCEDURE CALL

The server machine (3) Procedure Procedure Client (1) Procedure call call process (4) Result Server Client stub (6) Result stub Client Server kernel kernel (2) Message-call

Client machine

14

© GLEB RADCHENKO

(5) Message-result

RPC PSEUDO CODE

Client	Server	
<pre>main { myType a = remoteProcedure (arg1, arg2); } (6) Result (1) Procedure call (2) Message call</pre>	<pre>byte[] serverStab (string name, byte[] mess) { witch name: case "remoteProcedure": int a, b; decRemoteProcedure (mess, &a, &b); myType res = remoteProcedure (a, b); byte [] response = encRemoteProcedureResponce (res); return response; case (3) Procedure call MyType remoteProcedure (arg1, arg2) { return process(arg1, arg2); } </pre>	
<pre>myType remoteProcedure (int arg1, int arg2) { byte [] mess, response; string name = "remoteProcedure"; string addr = "remote.host:1122"; mess = encRemoteProcedure (arg1, arg2); (5) Message- response = response</pre>		
<i>callRemoteProcedure</i> (addr, name, mess); return <i>decRemoteProcedure</i> Responce (response); }		

STAGES OF THE RPC

			↑
Client stut		Call the server Place the parameters onto the stack Unpack the parameters	 The server stub
Kernel		Switch to the server stub context Copy the message to the server stub Determine whether the stub is waiting Determine the stab, which send the packet Validate a package Interrupt the process	Kernel
	Kernel	Kernel	Kernel

REMOTE METHOD INVOCATION

In terms of OOP the Remote Method Invocation (RMI) concept was implemented.

- RMI allows to provide transparent access to the methods of remote objects, providing
 - delivery of parameters of the invoked method,
 - message to the remote object to execute the method
 - and the transfer of a return values back to the client





- The remote object is a collection of some data that determine its State. This State can be changed by calling some of his methods.
- Methods and fields of an object that can be used via remote calls, are available through the external interface of the objects class.

19 REMOTE PROCEDURE VS REMOTE OBJECT



A PROXY AND A SCELETON

- A client stub that invokes a remote object is called the proxy.
- Proxy implements the same interface as the remote object.
- The server-side stub is called the skeleton (in Java RMI)
- The skeleton is associated with a specific instance of the remote object and invokes the method with the desired settings

REMOTE OBJECT USAGE



REMOTE METHOD INVOCATION RMI

Interface:

```
public interface ProductCatalogue extends java.rmi.Remote
```

```
ProductDescription[] searchProduct(String productType) throws java.rmi.RemoteException;
Product provideProduct(ProductDescription d) throws java.rmi.RemoteException;
int deleteProduct(ProductDescription d) throws java.rmi.RemoteException;
int updateProduct(Product p) throws java.rmi.RemoteException;
...
```

Server – interface realization:

```
public class ProductCatalogueImpl extends java.rmi.server.UnicastRemoteObject
implements ProductCatalogue
```

```
public ProductCatalogueImpl() throws java.rmi.RemoteException
{    super(); }
public ProductDescription[] searchProduct(String productType)
throws java.rmi.RemoteException
```

```
ProductDescription[] desc = ProductCatalogue.getDescriptionByType(productType);
return desc;
```

```
© GLEB RADCHENKO
```

{

}

ł

REMOTE METHOD INVOCATION RMI

Server realization:

```
public class ProductCatalogueServer {
    public ProductCatalogueServer() {
        try {
            ProductCatalogue c = new ProductCatalogueImpl();
            Naming.rebind("rmi://localhost:1099/ProductCatalogueService", c);
        }
        catch (Exception e) {...}
    }
    public static void main(String args[]) {
        new ProductCatalogueServer();
    }
```

Client Realization:

```
public class ProductCatalogueClient {
    public static void main(String[] args)
    {
        try {
            ProductCatalogue c= (ProductCatalogue)Naming.lookup(
                "rmi://hostname/ProductCatalogueService");
            System.out.println( c.searchProduct("book");
        }
    catch (Exception e) {...}
    }
    GLEB RADCHENKO
```

}